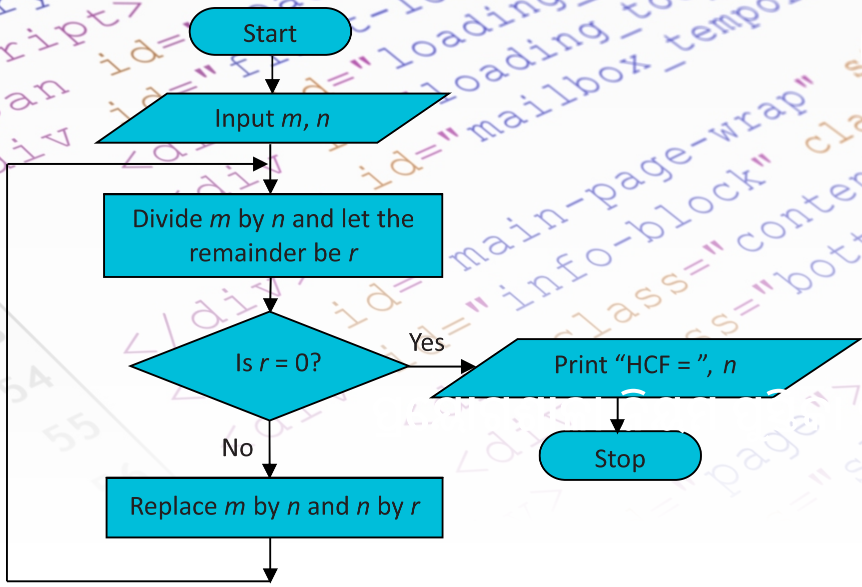
AICTE ଅନୁମୋଦିତ ନମୁନା ପାଠ୍ୟକ୍ରମ ଅନୁସାରେ

ପରିଣାମ ଆଧାରିତ ଶିକ୍ଷା ସହିତ ସମାନ୍ତରାଳ

ଜାତୀୟ ଶିକ୍ଷାନୀତି -୨୦୨୦

UNDERGRADUATE COURSE BOOK



ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ

ପ୍ରୟୋଗଶାଳା ନିୟମାବଳି ସହିତ

ମୂଳ ଇଂରାଜୀ ରଚନା

ଆର ଏସ ସଲରିୟା

ଓଡିଆ ଅନୁବାଦ

ସତ୍ୟନାରାୟଣ ଦାସ

ସଚ୍ଚିକାନ୍ତ ଦାଶ

ସମସ୍ୟା ସମାଧାନପାଇଁ
ପ୍ରୋଗ୍ରାମିଂ
Programming for problem Solving

ପ୍ରୟୋଗଶାଳା ନିୟମାବଳି ସହିତ

ମୂଳ ଇଂରାଜୀ ରଚନା:

ଆର ଏସ ସଲରିୟା

ଓଡିଆ ଅନୁବାଦ:

ସତ୍ୟ ନାରାୟଣ ଦାସ

ସଚ୍ଚିକାନ୍ତ ଦାଶ

Published by:
Institute of Odia Studies and Research.
3R9, BJB Nagar, Bhubaneswar, Odisha

ISBN: 978-93-91638-07-8

Book Code: UG003OD

Samasya Samadhanapain Programming

[Programming for problem Solving]

by

[English Edition]
R.S Salaria

[Odia Edition]
Dr. Satyanarayan Dash
Dr. Sachikanta Dash

First Edition: 2022

Published by:
Institute of Odia Studies and Research.
3 R 9, BJB Nagar, Bhubaneswar, Odisha
Email: iosrodishaaicte@gmail.com
Phone: 0674-3574404, Mob: 9090465758

ଓଡିଆ ଅନୁବାଦ:
ସତ୍ୟ ନାରାୟଣ ଦାସ
ସହଯୋଗୀ ଅଧ୍ୟାପକ, କମ୍ପ୍ୟୁଟର ବିଜ୍ଞାନ ଏବଂ ଯାନ୍ତ୍ରିକବିଦ୍ୟା ବିଭାଗ
ଜି.ଆଇ.ଇ.ଟି ବିଶ୍ୱବିଦ୍ୟାଳୟ, ଗୁଣୁପୁର, ରାୟଗଡ଼ା, ଓଡ଼ିଶା
ଡକ୍ଟର ସଚ୍ଚିକାନ୍ତ ଦାଶ
ସହଯୋଗୀ ଅଧ୍ୟାପକ, କମ୍ପ୍ୟୁଟର ବିଜ୍ଞାନ ଏବଂ ଯାନ୍ତ୍ରିକବିଦ୍ୟା ବିଭାଗ
ଜି.ଆଇ.ଇ.ଟି ବିଶ୍ୱବିଦ୍ୟାଳୟ, ଗୁଣୁପୁର, ରାୟଗଡ଼ା, ଓଡ଼ିଶା

ଓଡିଆ ଭାଷା ବିଶେଷଜ୍ଞ:
ଡକ୍ଟର ସୁବ୍ରତ କୁମାର ପୃଷ୍ଟି
ସଦସ୍ୟ ସଚିବ, ଓଡିଆ ଅଧ୍ୟୟନ ଓ ଗବେଷଣା ସଂସ୍ଥା, ଭୁବନେଶ୍ୱର, ଓଡିଶା

ବିଷୟ ସମୀକ୍ଷକ:
ପ୍ରଫେସର ଅଜିତ କୁମାର ନାୟକ
ବିଭାଗୀୟ ମୁଖ୍ୟ, କମ୍ପ୍ୟୁଟର ବିଜ୍ଞାନ ଏବଂ ସୂଚନା ପ୍ରଯୁକ୍ତି ବିଭାଗ
ଶିକ୍ଷା ଓ ଅନୁସନ୍ଧାନ ବିଶ୍ୱବିଦ୍ୟାଳୟ, ଭୁବନେଶ୍ୱର

ଓଡିଆ ଭାଷା ସମୀକ୍ଷକ:
ପ୍ରଫେସର ନଟବର ଶତପଥୀ
ପ୍ରାକ୍ତନ ବିଭାଗୀୟ ମୁଖ୍ୟ, ସ୍ନାତକୋତ୍ତର ଓଡିଆ ବିଭାଗ
ରେଭେନ୍ସା ବିଶ୍ୱବିଦ୍ୟାଳୟ, କଟକ, ଓଡ଼ିଶା

Printed in India by: BK Publications Pvt.Ltd, Bhubaneswar,

Copyright © Reserved by AICTE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior permission of the publisher.

This book is sold subject to the condition that it shall not, by way of trade, be lent, re-sold, hired out or otherwise disposed of without the publisher's consent, in any form of binding or cover other than that in which it is published.

Disclaimer: The website links provided by the author in this book are placed for informational, educational & reference purpose only. The Publisher do not endorse these website links or the views of the speaker/ content of the said weblinks. In case of any dispute, all legal matters to be settled under New Delhi Jurisdiction only.

ISBN 978-93-91638-07-8

9 789391 638078

प्रो. अनिल डी. सहस्रबुद्धे
अध्यक्ष
Prof. Anil D. Sahasrabudhe
Chairman

सत्यमेव जयते

अखिल भारतीय तकनीकी शिक्षा परिषद्
(भारत सरकार का एक सांविधिक निकाय)
(शिक्षा मंत्रालय, भारत सरकार)
नेल्सन मंडेला मार्ग, वसंत कुंज, नई दिल्ली-110070
दूरभाष : 011-26131498
ई-मेल : chairman@aicte-india.org

ALL INDIA COUNCIL FOR TECHNICAL EDUCATION
(A STATUTORY BODY OF THE GOVT. OF INDIA)
(Ministry of Education, Govt. of India)
Nelson Mandela Marg, Vasant Kunj, New Delhi-110070
Phone : 011-26131498
E-mail : chairman@aicte-india.org

ପ୍ରାକ୍‌କଥନ

ଶହଶହ ବର୍ଷ ଧରି ମାନବଜାତି ଓ ସମାଜର ଅଗ୍ରଗତି ଏବଂ ବିସ୍ତାରରେ ଇଞ୍ଜିନିୟରିଂ ଏକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଭୂମିକା ଗ୍ରହଣ କରିଛି। ଭାରତୀୟ ଉପମହାଦେଶରେ ସୃଷ୍ଟ ଇଞ୍ଜିନିୟରିଂ ଚିନ୍ତାଧାରା ଜଗତ ଉପରେ ଏକ ଚିନ୍ତାଶୀଳ ପ୍ରଭାବ ପକାଇଛି।

ଅଖିଳ ଭାରତୀୟ ବୈଷୟିକ ଶିକ୍ଷା ପରିଷଦ [All India Council for technical Education (AICTE)] 1987ରେ ଆରମ୍ଭ ହେବା ପରଠାରୁ ବୈଷୟିକ ଶିକ୍ଷାର ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କୁ ସାହାଯ୍ୟ କରିବାରେ ସର୍ବଦା ଅଗ୍ରଣୀ ଭୂମିକା ଗ୍ରହଣ କରି ଆସିଛି। AICTEର ଲକ୍ଷ୍ୟ ହେଉଛି ଗୁଣାତ୍ମକ ବୈଷୟିକ ଶିକ୍ଷାକୁ ପ୍ରୋତ୍ସାହିତ କରିବା ଏବଂ ଏହା ମାଧ୍ୟମରେ ଶିଳ୍ପ ଜଗତକୁ ଶୀର୍ଷ ସ୍ଥାନରେ ପହଞ୍ଚାଇବା। ସର୍ବୋପରି ଆମର ପ୍ରିୟ ମାତୃଭୂମି ଭାରତକୁ ଏକ ଆଧୁନିକ ବିକଶିତ ରାଷ୍ଟ୍ରରେ ପରିଣତ କରିବା। ଏଠାରେ ଉଲ୍ଲେଖ କରିବା ଅନୁଚିତ ହେବ ନାହିଁ ଯେ ଯନ୍ତ୍ରୀମାନେ ଆଧୁନିକ ସମାଜର ମେରୁଦଣ୍ଡ। ଉତ୍ତମ ଯନ୍ତ୍ରୀ ଅର୍ଥାତ୍ ଉତ୍ତମ ଶିଳ୍ପ ଏବଂ ଉତ୍ତମ ରାଷ୍ଟ୍ର।

ଜାତୀୟ ଶିକ୍ଷାନୀତି-୨୦୨୦ର ପରିକଳ୍ପନା ହେଉଛି ବିଦ୍ୟାର୍ଥୀଙ୍କୁ ଭାରତର ସାମ୍ବିଧାନିକ ବ୍ୟବସ୍ଥା ଅନୁଯାୟୀ ନିଜ ଭାଷାରେ ଶିକ୍ଷା ପ୍ରଦାନ କରିବା, ଯାହାଦ୍ୱାରା ପ୍ରତ୍ୟେକ ଛାତ୍ରଛାତ୍ରୀ ଜ୍ଞାନାର୍ଜନରେ ଯଥେଷ୍ଟ ସମର୍ଥ ଓ ଦକ୍ଷ ହୋଇପାରିବେ ଏବଂ ଜାତୀୟ ଅଭିବୃଦ୍ଧି ତଥା ବିକାଶ ଦିଗରେ ସହଯୋଗ କରିପାରିବେ।

ଗତ କିଛି ବର୍ଷରୁ AICTE ନିରନ୍ତର ଭାବରେ କାର୍ଯ୍ୟ କରୁଥିବା କ୍ଷେତ୍ରଗୁଡିକ ମଧ୍ୟରୁ ଗୋଟିଏ ହେଉଛି ଏହାର ସମସ୍ତ ଇଞ୍ଜିନିୟରିଂ ବିଦ୍ୟାର୍ଥୀଙ୍କୁ ଭାରତୀୟ ଭାଷାରେ ଉଚ୍ଚମାନ ତଥା ଉଚିତ ମୂଲ୍ୟର ଆନ୍ତର୍ଜାତୀୟ ପୁସ୍ତକ ପ୍ରଦାନ କରିବା। ଏହି ପୁସ୍ତକଗୁଡ଼ିକ କେବଳ ସହଜ ଭାଷା, ବାସ୍ତବ ଜୀବନ ସମ୍ବଳିତ ଉଦାହରଣ, ସମୃଦ୍ଧ ବିଷୟବସ୍ତୁକୁ ଧ୍ୟାନରେ ରଖି ପ୍ରସ୍ତୁତ କରାଯାଇନାହିଁ ବରଂ ଏହି ନିତିଦିନିଆ ପରିବର୍ତ୍ତନଶୀଳ ଦୁନିଆରେ ଶିଳ୍ପର ଆବଶ୍ୟକତାକୁ ମଧ୍ୟ ଦୃଷ୍ଟିରେ ରଖାଯାଇ ପ୍ରସ୍ତୁତ କରାଯାଇଛି। ଇଞ୍ଜିନିୟରିଂ ଏବଂ ଟେକ୍ନୋଲୋଜି- 2018ର AICTE ମଡେଲ୍ ପାଠ୍ୟକ୍ରମ ଅନୁଯାୟୀ ଏହି ପୁସ୍ତକଗୁଡ଼ିକ ପ୍ରସ୍ତୁତ ହୋଇଛି।

ବହୁ ଜ୍ଞାନ ଏବଂ ଅଭିଜ୍ଞତା ଥିବା ସମଗ୍ର ଭାରତବର୍ଷର କୋଣ ଅନୁକୋଣର ବିଶିଷ୍ଟ ପ୍ରଫେସରମାନେ ଶିକ୍ଷାବ୍ୟବସ୍ଥାର ସୁଲଭତା ଓ ସୁଗମତାପାଇଁ ଏହି ପୁସ୍ତକଗୁଡ଼ିକ ଲେଖିଛନ୍ତି। AICTE ନିଶ୍ଚିତ ଯେ ସମୃଦ୍ଧ ବିଷୟବସ୍ତୁ ସହିତ ଏହି ପୁସ୍ତକଗୁଡ଼ିକ ବୈଷୟିକ ଶିକ୍ଷାର ବିଦ୍ୟାର୍ଥୀମାନଙ୍କୁ ସହଜରେ ଗୁଣାତ୍ମକ ଜ୍ଞାନ ପ୍ରଦାନରେ ସହାୟକ ହେବ।

ଇଞ୍ଜିନିୟରିଂ ବିଷୟଗୁଡ଼ିକୁ ଅଧିକ ପ୍ରାଞ୍ଜଳ କରିବାରେ ପ୍ରୟାସ କରିଥିବା ମୂଳ ଲେଖକ, ସଂଯୋଜକ ଏବଂ ଅନୁବାଦକଙ୍କ କଠିନ ପରିଶ୍ରମକୁ AICTE ପ୍ରଶଂସା କରୁଛି।

(Anil D. Sahasrabudhe)

ଅନୁବାଦକଙ୍କ କଲମରୁ...

ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ ନିଃସନ୍ଦେହରେ ସବୁଠାରୁ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ବିଷୟ ମଧ୍ୟରୁ ଅନ୍ୟତମ। ଏହି ବିଷୟର ଉଦ୍ଦେଶ୍ୟ ହେଉଛି ସମସ୍ୟା ସମାଧାନ କୌଶଳ ଏବଂ ସେମାନଙ୍କ କାର୍ଯ୍ୟକାରିତାପାଇଁ କମ୍ପ୍ୟୁଟର ବା C ଭାଷାରେ ପ୍ରୋଗ୍ରାମ ସୃଷ୍ଟି କରିବାର ଦକ୍ଷତା ବିକାଶ କରିବା। ସାରା ଭାରତବର୍ଷର "ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ" ପାଠ୍ୟକ୍ରମ ଅଖିଳ ଭାରତୀୟ ବୈଷୟିକ ଶିକ୍ଷା ପରିଷଦ (AICTE) ମଡେଲ୍ ପାଠ୍ୟକ୍ରମ ଅନୁଯାୟୀ ସମସ୍ତ ଇଞ୍ଜିନିୟରିଂ ବିଭାଗରେ ପ୍ରଥମ ବର୍ଷର ଛାତ୍ରମାନଙ୍କପାଇଁ ପ୍ରସ୍ତୁତ କରାଯାଇଛି। ପ୍ରକୃତରେ ଏହି ଶିକ୍ଷା ପ୍ରାୟତଃ ଅଭ୍ୟାସଦ୍ୱାରା ଶିଖାଯାଇଥାଏ।

ଆମେ ସମସ୍ତେ ଅବଗତ ଯେ, ଜାତୀୟ ଶିକ୍ଷାନୀତି ୨୦୨୦ରେ ଆଞ୍ଚଳକ ଭାଷାର ଶିକ୍ଷାଦାନକୁ ଅଧିକ ଗୁରୁତ୍ୱ ପ୍ରଦାନ କରାଯାଇଛି। ସେଥିପାଇଁ ସମସ୍ତ ଇଞ୍ଜିନିୟରିଂ ପୁସ୍ତକକୁ ଆଞ୍ଚଳିକ ଭାଷାର ଅନୁବାଦ କରିବାପାଇଁ ଅଖିଳ ଭାରତୀୟ ବୈଷୟିକ ଶିକ୍ଷା ପରିଷଦର ଅଭିନବ ପ୍ରୟାସ ପ୍ରଶଂସନୀୟ। ଏହି ବିଷୟର ଓଡିଆ ଭାଷାରେ ଅନୁବାଦ କରିବାପାଇଁ ଆମକୁ ସୁଯୋଗ ଦେଇଥିବାରୁ ଆମେ ଓଡିଆ ଅଧ୍ୟୟନ ଓ ଗବେଷଣା ସଂସ୍ଥାକୁ ଅଶେଷ ଧନ୍ୟବାଦ ଜ୍ଞାପନ କରୁଛୁ। ବିଶେଷ କରି ଅଖିଳ ଭାରତୀୟ ବୈଷୟିକ ଶିକ୍ଷା ପରିଷଦର ଅଧ୍ୟକ୍ଷ ପ୍ରଫେସର ଅନିଲ ଡି. ସହସ୍ରବୁଦ୍ଧେ, ଉପାଧ୍ୟକ୍ଷ ପ୍ରଫେସର ଏମ୍. ପି. ପୁନିଆ, ସଦସ୍ୟ-ସଚିବ ପ୍ରଫେସର ରାଜୀବ କୁମାର, ମୁଖ୍ୟ ସମନ୍ୱୟ ଅଧିକାରୀ ବୁଦ୍ଧ ଚନ୍ଦ୍ରଶେଖର, ନିର୍ଦ୍ଦେଶକ ଡକ୍ଟର ଅମିତ ଶ୍ରୀବାସ୍ତବ, ପ୍ରାକ୍ତନ ନିର୍ଦ୍ଦେଶକ କର୍ଣ୍ଣେଲ ବି ଭେଙ୍କଟ ଏବଂ ଓଡିଆ ଅଧ୍ୟୟନ ଓ ଗବେଷଣା ସଂସ୍ଥାର ଅଧ୍ୟକ୍ଷ ଆତ୍ମାସଡର ଅବସର ବେଉରିଆ, ଉପାଧ୍ୟକ୍ଷ ନଟବର ଶତପଥୀ (ଶିକ୍ଷା ଓ ଗବେଷଣା), ଉପାଧ୍ୟକ୍ଷ ଡକ୍ଟର ସିପ୍ରା ମଲ୍ଲିକ (ପରିଚାଳନା), ନିର୍ଦ୍ଦେଶକ ପ୍ରଫେସର ବ୍ରଜେନ୍ଦ୍ର ପ୍ରସାଦ ଦାସ ଏବଂ ଏହି କାର୍ଯକ୍ରମର ମୁଖ୍ୟ ଥିବା ସଦସ୍ୟ ସଚିବ ଡକ୍ଟର ସୁବ୍ରତ କୁମାର ପୃଷ୍ଟିଙ୍କୁ କୃତଜ୍ଞତା ଜଣାଉଛୁ।

ମୂଳ ଇଂରାଜୀ ପୁସ୍ତକର ଓଡିଆ ଅନୁବାଦପାଇଁ ଓଡିଆ ଅଧ୍ୟୟନ ଓ ଗବେଷଣା ସଂସ୍ଥା ପକ୍ଷରୁ ଆୟୋଜିତ ସାକ୍ଷାତକାରଦ୍ୱାରା ମନୋନୀତ ହେବା ପରେ ଓଡିଆଭାଷରେ ବୈଷୟିକ ପୁସ୍ତକ ରଚନା କରିବା ପୂର୍ବରୁ ପ୍ରତ୍ୟେକ ବିଷୟବସ୍ତୁକୁ ଭଲ ଭାବରେ ବୁଝିବା ସହିତ ପ୍ରତ୍ୟେକ ବାକ୍ୟ ଯେମିତି ଉଦ୍ଦିଷ୍ଟ ଅର୍ଥକୁ ପ୍ରତିପାଦନ କରିବ, ସେଥିପାଇଁ ଆମେ ଅଧିକ ଯତ୍ନବାନ ହୋଇଥିଲୁ। ବିଷୟ ସମୀକ୍ଷକ ଶିକ୍ଷା ଓ ଅନୁସନ୍ଧାନ ବିଶ୍ୱବିଦ୍ୟାଳୟର କମ୍ପ୍ୟୁଟରବିଜ୍ଞାନ ଏବଂ ସୂଚନା ପ୍ରଯୁକ୍ତି ବିଦ୍ୟା ବିଭାଗର ମୁଖ୍ୟ ପ୍ରଫେସର ଅଜିତ କୁମାର ନାୟକ, ଭାଷ ବିଶେଷଜ୍ଞ ଡକ୍ଟର ସୁବ୍ରତ କୁମାର ପୃଷ୍ଟି ଏବଂ ଭାଷା ସମୀକ୍ଷକ ପ୍ରଫେସର ନଟବର ଶତପଥୀଙ୍କ ଉଚ୍ଚକୋଟୀର ସମୀକ୍ଷା ଏବଂ ପରାମର୍ଶ ଏହି ଅନୁବାଦ କାର୍ଯ୍ୟକୁ ଋଦ୍ଧିମନ୍ତ ଏବଂ ସୁଚାରୁ ରୂପେ ସମ୍ପାଦନ କରିବାରେ ବିଶେଷ ସହାୟକ ହୋଇଛି। ସେଥିପାଇଁ ଆମେ ସେମାନଙ୍କ ନିକଟରେ ଚିରକୃତଜ୍ଞ। ଏଥି ସହିତ ଆମର କର୍ମକ୍ଷେତ୍ର ଥିବା ଜିଁ.ଆଇ.ଇ.ଟି ବିଶ୍ୱବିଦ୍ୟାଳୟ କର୍ତ୍ତୃପକ୍ଷଙ୍କ ସହଯୋଗପାଇଁ ସେମାନଙ୍କ ନିକଟରେ ଚିର ଋଣୀ। ଏହି କାର୍ଯ୍ୟର ସଫଳ ରୂପାୟନପାଇଁ ଆମେ ଆମର ପରିବାର ପ୍ରତି ବିଶେଷ କୃତଜ୍ଞତା ଜ୍ଞାପନ କରୁଛୁ ।

ସମଗ୍ର ଅନୁବାଦ "ଲେଖ ଓଡିଆ" ସଫ୍ଟୱେୟାର ସାହାଯ୍ୟରେ ପ୍ରସ୍ତୁତ କରାଯାଇଛି। ସମସ୍ୟା ସମାଧାନ ପ୍ରକୃତରେ ଏକ ଆତ୍ମ-ନିର୍ଦ୍ଦେଶିତ ପ୍ରକ୍ରିୟା। ଏହା କୌତୁହଳ, ନମନୀୟତା, ଯତ୍ନଶୀଳ ପର୍ଯ୍ୟବେକ୍ଷଣ ତଥା ବିଶ୍ଳେଷଣ ଏବଂ ଏକ ଧାରଣାଗତ ଢାଞ୍ଚା ଆବଶ୍ୟକ କରେ ଯାହା ସମୟ ସହିତ ଧୀରେ ବଢିଥାଏ। ତେଣୁ ଏହି ପୁସ୍ତକଟିରେ ସେସବୁ ଦିଗକୁ ଧ୍ୟାନ ରଖି ଲେଖାଯାଇଛି

ଅନୁବାଦକୁ ତ୍ରୁଟି ବିହୀନ କରିବାପାଇଁ ଏଠାରେ ବିଶେଷ ଦୃଷ୍ଟି ଦିଆଯାଇଛି। ତଥାପି ଯଦି ପାଠକଙ୍କଦ୍ୱାରା କିଛି ତ୍ରୁଟି ପରିଲକ୍ଷିତ ହୁଏ ତା'ହେଲେ ଅନୁବାଦକଙ୍କ ଦୃଷ୍ଟି ଆକର୍ଷଣ ଏକାନ୍ତ କାମ୍ୟ। ଏହି ଅନୁବାଦିତ ପୁସ୍ତକର ଉନ୍ନତିପାଇଁ ଆପଣଙ୍କ ପରାମର୍ଶ ଅତ୍ୟନ୍ତ ସ୍ୱାଗତଯୋଗ୍ୟ।

ଡକ୍ଟର ସତ୍ୟ ନାରାୟଣ ଦାସ

ଡକ୍ଟର ଶଚ୍ଚିକାନ୍ତ ଦାଶ

କୃତଜ୍ଞତା

ଅଖିଳ ଭାରତୀୟ ବୈଷୟିକ ଶିକ୍ଷା ପରିଷଦ(AICTE)ର କର୍ତ୍ତୃପକ୍ଷ, ବିଶେଷ କରି ଅଧ୍ୟକ୍ଷ ପ୍ରଫେସର ଅନିଲ ଡି ସହସ୍ରବୁଧେ, ଉପାଧ୍ୟକ୍ଷ ପ୍ରଫେସର ମହାବୀର ପ୍ରସାଦ ପୁନିଆ, ସଦସ୍ୟ-ସଚିବ ପ୍ରଫେସର ରାଜୀବ କୁମାର, ଏବଂ ଅଧ୍ୟାପକ ବିକାଶ ସେଲର ନିର୍ଦ୍ଦେଶକ କର୍ଣ୍ଣେଲ ବି ଭେଙ୍କଟଙ୍କୁ ଯାନ୍ତ୍ରିକ ଏବଂ ପ୍ରଦ୍ୟୋଗିକ ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କପାଇଁ **"ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ"**ଏବଂ ଅନ୍ୟାନ୍ୟ ବୈଷୟିକ ପୁସ୍ତକ ଅତି ଯତ୍ନଶୀଳ ଭାବରେ ପ୍ରକାଶ କରିବାକୁ ଯୋଜନା ଏବଂ କାର୍ଯ୍ୟକାରୀ କରିଥିବା ଯୋଗୁଁ ଲେଖକବୃନ୍ଦ ସେମାନଙ୍କର କୃତଜ୍ଞତା ପ୍ରକାଶ କରୁଛୁ। ଏହାକୁ ଛାତ୍ରମାନଙ୍କ ସହଜସାଧ କରିବା ଏବଂ କଳାତ୍ମକ ଢଙ୍ଗରେ ଏକ ଉତ୍ତମ ଆକୃତି ଦେବାପାଇଁ ଆମେ ପୁସ୍ତକର ସମୀକ୍ଷକଙ୍କ, ମୂଲ୍ୟବାନ ଅବଦାନକୁ ଆନ୍ତରିକତାର ସହ ସ୍ୱୀକାର କରୁଛୁ । ବାସ୍ତବରେ, ସେ **"ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ"** ପୁସ୍ତକର ସମୀକ୍ଷକ ଭାବରେ ଆରମ୍ଭରୁ ଆମକୁ ମାର୍ଗଦର୍ଶନ କରିଥିଲେ, ଯେଉଁଠାରେ ଆବଶ୍ୟକ ହୁଏ ସେଠାରେ ଅନେକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ପରାମର୍ଶ ପ୍ରଦାନ କରିଥିଲେ ଏବଂ ତାଙ୍କର ସମାଲୋଚନାମୂଳକ ମନ୍ତବ୍ୟ ମଧ୍ୟ ସର୍ବତ୍ର ଦେଇଥିଲେ। ଏହି ପୁସ୍ତକଟି ଏଆଇସିଟିଇର ସଦସ୍ୟ, ବିଶେଷଜ୍ଞ ଏବଂ ଲେଖକଙ୍କ ବିଭିନ୍ନ ପରାମର୍ଶର ଫଳାଫଳ। ସେମାନେ ଆମ ଦେଶରେ ଇଞ୍ଜିନିୟରିଂ ଶିକ୍ଷାକୁ ଆହୁରି ବିକଶିତ କରିବାକୁ ସେମାନଙ୍କର ସୁଚିନ୍ତିତ ମତାମତ ରଖିଥିଲେ। ଏହି କ୍ଷେତ୍ରରେ, ଯୋଗଦାନକାରୀ ଏବଂ ବିଭିନ୍ନ କର୍ମୀମାନଙ୍କପାଇଁ ସ୍ୱୀକୃତି ଦିଆଯାଇଛି ଯାହାର ପ୍ରକାଶିତ ପୁସ୍ତକ, ସମୀକ୍ଷା ପ୍ରବନ୍ଧ, ପତ୍ରିକା, ଫଟୋଗ୍ରାଫ୍, ଟିପ୍ପଣୀ, ରେଫରେନ୍ସ ଏବଂ ଅନ୍ୟାନ୍ୟ ମୂଲ୍ୟବାନ ସୂଚନା, ଯାହା ପୁସ୍ତକ ଲେଖିବା ସମୟରେ ଆମକୁ ସମୃଦ୍ଧ କରିଥିଲା। ଶେଷରେ, ଆମେ ନୂଆଦିଲ୍ଲୀସ୍ଥିତ ଖାନ୍ନା ପବ୍ଲିସିଂ ହାଉସର ପ୍ରକାଶକ ଶ୍ରୀ ମୁକୁଲ ସେଠଙ୍କୁ ଆନ୍ତରିକତାର ସହ ଧନ୍ୟବାଦ ଜଣାଇବାକୁ ଚାହୁଁଛୁ, ଯାହାର ସମସ୍ତ କର୍ମୀମାନେ ପ୍ରକାଶନର ସମସ୍ତ ଦିଗରେ ସହଯୋଗ କରିବାକୁ ସର୍ବଦା ପ୍ରସ୍ତୁତ ଥିଲେ, ଯାହା ଏକ ଅଦଭୂତ ଅଭିଜ୍ଞତା ଅଟେ।

ଲେଖକ
ଆର. ଏସ୍ ସାଲାରିଆ

ମୁଖବନ୍ଧ

"ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ" ନିଃସନ୍ଦେହରେ ସବୁଠାରୁ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ କୌଶଳ ମଧ୍ୟରୁ ଗୋଟିଏ; ଦକ୍ଷ କୋଡ୍ ଲେଖିବା, ପ୍ରଭାବଶାଳୀ ଯୋଗାଯୋଗ, ଏକ ଦଳ ସହିତ କାର୍ଯ୍ୟ କରିବା ଏବଂ ଅନ୍ୟ ଅନେକ ଦକ୍ଷତା ମଧ୍ୟ ଅତ୍ୟନ୍ତ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ। କୌଣସି ଗୋଟିଏ କୌଶଳ ସବୁଠାରୁ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ବୋଲି କହିବା ଅସମ୍ଭବ। "ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ" ବିଷୟର ଉଦ୍ଦେଶ୍ୟ ହେଉଛି ସମସ୍ୟା ସମାଧାନ କୌଶଳ ଏବଂ ସେମାନଙ୍କ କାର୍ଯ୍ୟକାରିତାପାଇଁ C ଭାଷାରେ ପ୍ରୋଗ୍ରାମସୃଷ୍ଟି କରିବାର ଦକ୍ଷତା ବିକାଶ କରିବା ।

ଇଞ୍ଜିନିୟରିଂ ଏବଂ ପ୍ରଯୁକ୍ତି ବିଦ୍ୟା (ବିଇ/ବିଟେକ୍)ରେ ସ୍ନାତକ କାର୍ଯ୍ୟକ୍ରମର ସମସ୍ତ ଶାଖାର ପ୍ରଥମ ବର୍ଷର ଛାତ୍ର ଛାତ୍ରୀ ମାନଙ୍କପାଇଁ ଏଆଇସିଟିଇର ମଡେଲ୍ ପାଠ୍ୟକ୍ରମ ଅନୁଯାୟୀ ଆପଣଙ୍କ ହାତରେ ଥିବା ପାଠ୍ୟପୁସ୍ତକ ପ୍ରସ୍ତୁତ କରାଯାଇଛି । ପ୍ରୋଗ୍ରାମିଂ ପୁସ୍ତକଗୁଡ଼ିକ ସମସ୍ୟା ସମାଧାନ ଶିଖାଏ । ଦୁର୍ଭାଗ୍ୟବଶତଃ ପ୍ରକୃତରେ କ'ଣ ଶିକ୍ଷା ଦିଆଯାଇପାରିବ ତାହା ଉପରେ ସୀମିତତା ଅଛି । ଏହା ପ୍ରାୟତଃ ଅଭ୍ୟାସଦ୍ୱାରା ଶିଖା ଯାଇଥାଏ ।

ମୁଁ ଯେଉଁ କଥା କହିବାକୁ ଚାହୁଁଛି ତାହା ହେଉଛି ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କୁ ସମସ୍ୟା ସମାଧାନ ପ୍ରକ୍ରିୟାର କାର୍ଯ୍ୟକାରିତା ଦେଖିବା । ଉଦାହରଣ ସ୍ୱରୂପ, ଏକ ସର୍ଟିଂ ଆଲଗୋରିଦମ୍ ପ୍ରସ୍ତୁତ କରିବା ହେଉଛି ଏକ "ସମସ୍ୟା"ର ମୌଳିକ ଉଦାହରଣ ଯାହା "ସମାଧାନ" ହେବା ଆବଶ୍ୟକ । ବିଭିନ୍ନ ଆଲଗୋରିଦମ୍ କିପରି କାର୍ଯ୍ୟକାରୀ କରିବେ ତାହା ବୁଝିବା ଏବଂ ସର୍ଟ କରିବାପାଇଁ ସର୍ବୋତ୍ତମ ରଣନୀତି କିପରି ଚୟନ କରିବେ ତାହା ଆପଣଙ୍କୁ ଏକ ପ୍ରାଥମିକ ଉପାୟରେ ସମସ୍ୟାର ସମାଧାନ ଶିଖିବାରେ ସାହାଯ୍ୟ କରେ। ଏହି ପ୍ରକାରର ବିଶ୍ଳେଷଣରେ କ'ଣ ଅନ୍ତର୍ଭୁକ୍ତ ତାହାର ଯତ୍ନର ସହିତ ପରୀକ୍ଷା ଆପଣଙ୍କ ସମସ୍ୟା ସମାଧାନ ପ୍ରକ୍ରିୟାରେ ଉନ୍ନତି ଆଣିବ। ଦୁର୍ଭାଗ୍ୟବଶତଃ, ଅଧିକାଂଶ ବିଦ୍ୟାର୍ଥୀ କେବଳ ଆଲଗୋରିଦମ୍ ଶିଖନ୍ତି ଏବଂ ଅଭ୍ୟାସ ସମାପ୍ତ କରନ୍ତି, ଏବଂ ଏହାଠାରୁ ଗଭୀର ଚିନ୍ତା କରନ୍ତି ନାହିଁ ।

ଯଦି ଆପଣ କେବଳ ଶ୍ରେଣୀରେ ନିଆଯାଇଥିବା ପୁସ୍ତକ କିମ୍ବା ନୋଟ୍ ପଢନ୍ତି ଏବଂ ସମାଧାନ କାର୍ଯ୍ୟକାରୀ କରନ୍ତି, ତେବେ ଆପଣ ସମସ୍ୟାର ସମାଧାନ ଶିଖୁନାହାଁନ୍ତି । ଏକ ଅଧିକ ପ୍ରଭାବଶାଳୀ ପଦ୍ଧତି ହେଉଛି ସମସ୍ୟା ପଢିବା, ତା'ପରେ ପୁସ୍ତକ/ନୋଟ୍ ବନ୍ଦ କରିବା ଏବଂ ଏକ ସମାଧାନ ଆଣିବାକୁ ଚେଷ୍ଟା କରିବା । ନିଜେ ଏକ ସମାଧାନ ସୃଷ୍ଟି କରିବା ପରେ, ପୂର୍ବକୁ ଯାଆନ୍ତୁ ଏବଂ ପୁସ୍ତକ/ନୋଟରେ ଯାହା ଲେଖାଯାଇଛି ତାହା ସହିତ ଆପଣଙ୍କ ଫଳାଫଳକୁ ତୁଳନା କରନ୍ତୁ । ତା'ପରେ ଆପଣ ସମସ୍ୟାର ସମାଧାନ କିପରି କରିବେ ଶିଖନ୍ତୁ ।

ସମସ୍ୟା ସମାଧାନ ପ୍ରକୃତରେ ଏକ ଆତ୍ମ-ନିର୍ଦ୍ଦେଶିତ ପ୍ରକ୍ରିୟା। ଏହା କୌତୁହଳ, ନମନୀୟତା, ଯତ୍ନଶୀଳ ପର୍ଯ୍ୟବେକ୍ଷଣ ଏବଂ ବିଶ୍ଳେଷଣ ଏବଂ ଏକ ଧାରଣାଗତ ଢାଞ୍ଚା ଆବଶ୍ୟକ କରେ ଯାହା ସମୟ ସହିତ ଧୀରେ ଧୀରେ ବଢିଥାଏ। ସେହି ଜିନିଷଗୁଡ଼ିକ ମଧ୍ୟରୁ, କେବଳ ଗୋଟିଏ ଯାହା ଶିକ୍ଷା ଦିଆଯାଇପାରିବ ତାହା ହେଉଛି ଧାରଣାଢାଞ୍ଚା, ତେଣୁ ପୁସ୍ତକ ଏବଂ ଶ୍ରେଣୀଶିକ୍ଷା ଗୁଡ଼ିକ ତାହା ପ୍ରଦାନ କରନ୍ତି । ବାକି ଗୁଡ଼ିକ ଆପଣଙ୍କ ଉପରେ ନିର୍ଭର କରେ ।

ଆପଣଙ୍କ ହାତରେ ଥିବା ପୁସ୍ତକ ଆପଣଙ୍କୁ ଏକ ଭଲ ସମସ୍ୟା ସମାଧାନକାରୀ ଏବଂ ଭଲ ପ୍ରୋଗ୍ରାମର୍ ହେବାପାଇଁ ଆବଶ୍ୟକ କରୁଥିବା ସମସ୍ତ ଉପାଦାନ ପ୍ରଦାନ କରିବ ।ଏହି ପୁସ୍ତକ ପଢିବାକୁ ଉପଭୋଗ କରନ୍ତୁ, ଏବଂ ଆପଣଙ୍କର ମୁକ୍ତ ଏବଂ ଖୋଲା ମନ୍ତବ୍ୟ, ପରାମର୍ଶ, ଏବଂ ସକରାତ୍ମକ ସମାଲୋଚନା ପଠାଇବାକୁ ମୁକ୍ତ ଅନୁଭବ କରନ୍ତୁ । ଆପଣଙ୍କର ମୂଲ୍ୟବାନ ମନ୍ତବ୍ୟ ମୋତେ ଆପଣଙ୍କର ଆଶା ପୂରଣ କରିବାପାଇଁ ପୁସ୍ତକକୁ ଉନ୍ନତ ଏବଂ ସୂକ୍ଷ୍ମ କରିବାରେ ସାହାଯ୍ୟ କରିବ ।

ଶେଷରେ ମୁଁ ଏତିକି କହିବାକୁ ଚାହେଁ, ପୁସ୍ତକ ଲେଖିବା ସମୟରେ, ତ୍ରୁଟି (ଭୁଲ୍ ପ୍ରିଣ୍ଟ ଏବଂ/କିମ୍ବା ଭୁଲ୍) ଏଡାଇବାପାଇଁ ଉପଯୁକ୍ତ ଯତ୍ନ ନିଆଯାଇଛି, ତଥାପି ସିଦ୍ଧତା ଦାବି କରିବା କଷ୍ଟକର । ଯଦି କୌଣସି ତ୍ରୁଟି ସୂଚିତ କରାଯାଏ ତେବେ ମୁଁ ପାଠକମାନଙ୍କ ପ୍ରତି କୃତଜ୍ଞ ହେବି ।

ଆର. ଏସ୍ ସାଲାରିଆ

ଫଳାଫଳ ଆଧାରିତ ଶିକ୍ଷା

ଏକ ଫଳାଫଳ ଆଧାରିତ ଶିକ୍ଷା(outcome based education)ର କାର୍ଯ୍ୟକାରିତାପାଇଁ ପ୍ରାଥମିକ ଆବଶ୍ୟକତା ହେଉଛି, ଏକ ଫଳାଫଳ ଆଧାରିତ ପାଠ୍ୟକ୍ରମ(outcome based curriculum)ର ବିକଶିତ କରିବା ଏବଂ ଶିକ୍ଷା ବ୍ୟବସ୍ଥାରେ ଏକ ଫଳାଫଳ ଆଧାରିତ ମୂଲ୍ୟାଙ୍କନ(outcome based assessment) ଅନ୍ତର୍ଭୁକ୍ତ କରିବା। ଫଳାଫଳ ଆଧାରିତ ମୂଲ୍ୟାଙ୍କନ ମାଧ୍ୟମରେ ମୂଲ୍ୟାଙ୍କନକାରୀମାନେ(evaluators) ମୂଲ୍ୟାଙ୍କନ କରିବାକୁ ସକ୍ଷମ ହେବେ ଯେ, ଛାତ୍ରମାନେ ନିଶ୍ଚିତ ମାନକ, ନିର୍ଦ୍ଦିଷ୍ଟ ଏବଂ ପରିମାପଯୋଗ୍ୟ ଫଳାଫଳ ହାସଲ କରିଛନ୍ତି କି ନାହିଁ। ଫଳାଫଳ ଆଧାରିତ ଶିକ୍ଷାର ଉପଯୁକ୍ତ ଅନ୍ତର୍ଭୁକ୍ତୀକରଣ ସହିତ କୌଣସି ସ୍ତରରେ ସମର୍ପଣ ନକରି, ସମସ୍ତ ଶିକ୍ଷାର୍ଥୀଙ୍କପାଇଁ ସର୍ବନିମ୍ନ ମାନକ ହାସଲ କରିବାକୁ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ପ୍ରତିବଦ୍ଧତା ରହିବ। ଫଳାଫଳ ଆଧାରିତ ଶିକ୍ଷା ସହାୟତାରେ ଚାଲୁଥିବା କାର୍ଯ୍ୟକ୍ରମ ଶେଷରେ, ଜଣେ ଛାତ୍ର ନିମ୍ନଲିଖିତ ଫଳାଫଳରେ ପହଞ୍ଚିବାକୁ ସକ୍ଷମ ହେବ:

PO-1: ଇଞ୍ଜିନିୟରିଂ ଜ୍ଞାନ(Engineering knowledge): ଜଟିଳ ଇଞ୍ଜିନିୟରିଂ ସମସ୍ୟାର ସମାଧାନପାଇଁ ଗଣିତ, ବିଜ୍ଞାନ, ଇଞ୍ଜିନିୟରିଂ ମୌଳିକତା ଏବଂ ଏକ ଇଞ୍ଜିନିୟରିଂ ବିଶେଷତା ବିଷୟରେ ଜ୍ଞାନ ପ୍ରୟୋଗ କରନ୍ତୁ।

PO-2: ସମସ୍ୟା ବିଶ୍ଳେଷଣ(Problem analysis): ଗଣିତ, ପ୍ରାକୃତିକ ବିଜ୍ଞାନ ଏବଂ ଇଞ୍ଜିନିୟରିଂ ବିଜ୍ଞାନର ପ୍ରଥମ ନୀତି ବ୍ୟବହାର କରି ପ୍ରମାଣିତ ସିଦ୍ଧାନ୍ତରେ ପହଞ୍ଚିବାପାଇଁ ଚିହ୍ନଟ, ସୂତ୍ରପ୍ରସ୍ତୁତ, ଅନୁସନ୍ଧାନ ସାହିତ୍ୟ ସମୀକ୍ଷା ଏବଂ ଜଟିଳ ଇଞ୍ଜିନିୟରିଂ ସମସ୍ୟାଗୁଡ଼ିକୁ ବିଶ୍ଳେଷଣ କରନ୍ତୁ ।

PO-3: ସମାଧାନର ପରିକଳ୍ପନା / ବିକାଶ(Design/development of solutions): ଜଟିଳ ଇଞ୍ଜିନିୟରିଂ ସମସ୍ୟାର ପରିକଳ୍ପନା ଏବଂ ସିଷ୍ଟମ୍ ଉପାଦାନର ପରିକଳ୍ପନା କିମ୍ବା ପ୍ରକ୍ରିୟା ଯାହା ଜନସ୍ୱାସ୍ଥ୍ୟ ଓ ନିରାପତ୍ତା ଏବଂ ସାଂସ୍କୃତିକ, ସାମାଜିକ ଏବଂ ପରିବେଶ ବିଚାରପାଇଁ ଉପଯୁକ୍ତ ବିଚାର ସହିତ ନିର୍ଦ୍ଦିଷ୍ଟ ଆବଶ୍ୟକତାପୂରଣ କରିବାର ପରିକଳ୍ପନା।

PO-4: ଜଟିଳ ସମସ୍ୟାର ଅନୁସନ୍ଧାନ କରିବା(Conduct investigations of complex problems): ବିଧିମାନ୍ୟ ସିଦ୍ଧାନ୍ତ ପ୍ରଦାନ କରିବାପାଇଁ, ପରୀକ୍ଷଣର ପରିକଳ୍ପନା, ବ୍ୟାଖ୍ୟା , ଡାଟାର ବିଶ୍ଳେଷଣ ଏବଂ ସୂଚନାର ସଂଶ୍ଳେଷଣ ସହିତ ଗବେଷଣା-ଆଧାରିତ ଜ୍ଞାନ ଏବଂ ଗବେଷଣା ପଦ୍ଧତିର ବ୍ୟବହାର।

PO-5: ଆଧୁନିକ ଉପକରଣ ବ୍ୟବହାର(Modern tool usage): ସୀମିତତାର ବୁଝାମଣା ସହିତ, ଜଟିଳ ଇଞ୍ଜିନିୟରିଂ କାର୍ଯ୍ୟକଳାପଗୁଡ଼ିକୁ, ପ୍ରାକ୍ ଉକ୍ତି ଏବଂ ପ୍ରତିରୂପାୟଣ ଅନ୍ତର୍ଭୁକ୍ତ କରି, ଉପଯୁକ୍ତ କୌଶଳ, ଉତ୍ସ, ଏବଂ ଆଧୁନିକ ଇଞ୍ଜିନିୟରିଂ ଏବଂ ଆଇଟି ଉପକରଣର ସୃଷ୍ଟି, ଚୟନ ଏବଂ ପ୍ରୟୋଗ କରନ୍ତୁ।

PO-6 : ଇଞ୍ଜିନିୟର ଏବଂ ସମାଜ(The engineer and society): ସାମାଜିକ, ସ୍ୱାସ୍ଥ୍ୟ, ନିରାପତ୍ତା, ଆଇନଗତ ଏବଂ ସାଂସ୍କୃତିକ ପ୍ରସଙ୍ଗ ଏବଂ ବୃତ୍ତିଗତ ବୈଷୟିକ ଅଭ୍ୟାସପାଇଁ ପ୍ରାସଙ୍ଗିକ ଦାୟିତ୍ୱଆକଳନ କରିବାପାଇଁ ପ୍ରାସଙ୍ଗିକ ଜ୍ଞାନ ଦ୍ୱାରା ସୂଚିତ ଯୁକ୍ତି ପ୍ରୟୋଗ କରନ୍ତୁ।

PO-7: ପରିବେଶ ଏବଂ ସ୍ଥାୟୀତ୍ୱ(Environment and sustainability): ସାମାଜିକ ଏବଂ ପରିବେଶ ପ୍ରସଙ୍ଗରେ ବୃତ୍ତିଗତ ଇଞ୍ଜିନିୟରିଂ ସମାଧାନର ପ୍ରଭାବକୁ ବୁଝନ୍ତୁ ଏବଂ ଜ୍ଞାନର ଏବଂ ନିରନ୍ତର ବିକାଶର ଆବଶ୍ୟକତାକୁ ପ୍ରଦର୍ଶନ କରନ୍ତୁ।

PO-8 : ନୈତିକତା(Ethics): ନୈତିକ ନୀତି ପ୍ରୟୋଗ କରନ୍ତୁ ଏବଂ ବୃତ୍ତିଗତ ନୈତିକତା ଏବଂ ଦାୟିତ୍ୱ ଏବଂ ଇଞ୍ଜିନିୟରିଂ ଅଭ୍ୟାସର ମାନଦଣ୍ଡ ପ୍ରତି ପ୍ରତିବଦ୍ଧ ହୁଅନ୍ତୁ।

PO-9: ବ୍ୟକ୍ତିଗତ ଏବଂ ଦଳଗତ କାର୍ଯ୍ୟ(Individual and team work): ବିବିଧ ଦଳରେ ଏବଂ ବହୁମୁଖୀ ପୃଷ୍ଠଭୂମିରେ, ଜଣେ ବ୍ୟକ୍ତିଗତ ଭାବରେ, ଏବଂ ସଦସ୍ୟ ଭାବରେ କିମ୍ବା ମୁଖ୍ୟ ଭାବରେ ପ୍ରଭାବଶାଳୀ ଭାବରେ କାର୍ଯ୍ୟ କରନ୍ତୁ ।

PO-10: ଯୋଗାଯୋଗ(Communication): ଇଞ୍ଜିନିୟରିଂ ସମ୍ପ୍ରଦାୟ ଏବଂ ସମାଜ ସହିତ ଜଟିଳ ଇଞ୍ଜିନିୟରିଂ କାର୍ଯ୍ୟକଳାପ ଉପରେ ପ୍ରଭାବଶାଳୀ ଭାବରେ ଯୋଗାଯୋଗ କରନ୍ତୁ, ଯେପରିକି, ପ୍ରଭାବଶାଳୀ ବିବୃତି ଲେଖିବା ଏବଂ ଦସ୍ତାବିଜ ରୂପରେଖ କରିବା ଏବଂ ବୁଝିବାରେ ସକ୍ଷମ ହେବା, ପ୍ରଭାବଶାଳୀ ଉପସ୍ଥାପନା କରିବା, ଏବଂ ସ୍ପଷ୍ଟ ନିର୍ଦ୍ଦେଶାବଳୀ ଦେବା ଏବଂ ଗ୍ରହଣ କରିବା।

PO-11 : ପ୍ରକଳ୍ପ ପରିଚାଳନା ଏବଂ ଅର୍ଥ(Project management and finance): ଇଞ୍ଜିନିୟରିଂ ଏବଂ ପରିଚାଳନା ନୀତିବିଷୟରେ ଜ୍ଞାନ ଏବଂ ବୁଝାମଣା ପ୍ରଦର୍ଶନ କରନ୍ତୁ ଏବଂ ଏଗୁଡିକୁ ଜଣଙ୍କର ନିଜକାର୍ଯ୍ୟରେ, ପ୍ରକଳ୍ପ ପରିଚାଳନା କରିବାକୁ ଏବଂ ବହୁମୁଖୀ ପରିବେଶରେ ଏକ ଦଳର ସଦସ୍ୟଭାବେ ଏବଂ ମୁଖ୍ୟଭାବେ ପ୍ରୟୋଗ କରନ୍ତୁ।

PO-12: ଜୀବନବ୍ୟାପୀ ଶିକ୍ଷା(Life-long learning): ଆବଶ୍ୟକତାକୁ ଚିହ୍ନନ୍ତୁ ଏବଂ ବୈଷୟିକ ପରିବର୍ତ୍ତନର ବ୍ୟାପକ ପରିପ୍ରେକ୍ଷୀରେ ସ୍ୱାଧୀନଭାବେ ଏବଂ ଜୀବନବ୍ୟାପି ଶିକ୍ଷାରେ ଜଡିତ ହେବାର ପ୍ରସ୍ତୁତି ଏବଂ ଦକ୍ଷତାକୁ ଅର୍ଜନ କରନ୍ତୁ ।

ପାଠ୍ୟକ୍ରମ

ଇଞ୍ଜିନିୟରିଂ ଏବଂ ଟେକ୍ନୋଲୋଜିରେ ପ୍ରଥମ ବର୍ଷର ସ୍ନାତକ ଡିଗ୍ରୀ ପାଠ୍ୟକ୍ରମପାଇଁ ଏଆଇସିଟିଇ ମଡେଲ୍ ପାଠ୍ୟକ୍ରମ

କୋର୍ସ କୋଡ୍	ବିଏସସି103				
ବର୍ଗ	ମୌଳିକ ବିଜ୍ଞାନ ପାଠ୍ୟକ୍ରମ				
ପାଠ୍ୟକ୍ରମ ଶୀର୍ଷକ	ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ				
ଯୋଜନା ଏବଂ କ୍ରେଡିଟ୍	L	T	P	କ୍ରେଡିଟ୍	ସେମିଷ୍ଟାର-୧/୨
	3	0	3	5.0	

ସିଦ୍ଧାନ୍ତ: ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ [ଏଲ୍ : 3; ଟି:0; ପି : 0 (3 କ୍ରେଡିଟ୍)]

ବିସ୍ତୃତ ବିଷୟବସ୍ତୁ:

ଅଧ୍ୟାୟ 1: ପ୍ରୋଗ୍ରାମିଂର ପରିଚୟ

ଏକ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ଉପାଦାନଗୁଡିକର ପରିଚୟ (ଡିସ୍କ, ମେମୋରୀ, ପ୍ରୋସେସର୍, ଯେଉଁଠାରେ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଷ୍ଟୋର୍ ଏବଂ କାର୍ଯ୍ୟକାରୀ କରାଯାଏ, ଅପରେଟିଂ ସିଷ୍ଟମ୍, ସଂକଳକ ଇତ୍ୟାଦି)

ଆଲଗୋରିଦମ୍ ର ଧାରଣା: ଯୁକ୍ତିଯୁକ୍ତ ଏବଂ ସଂଖ୍ୟାଗତ ସମସ୍ୟାର ସମାଧାନପାଇଁ ପଦକ୍ଷେପ। ଆଲଗୋରିଦମ୍ ର ପ୍ରତିନିଧିତ୍ୱ: ଉଦାହରଣ ସହିତ ଫ୍ଲୋଚାର୍ଟ/ ସ୍ୟୁଡୋକୋଡ୍

ଆଲଗୋରିଦମ୍ ଠାରୁ ପ୍ରୋଗ୍ରାମ୍ ପର୍ଯ୍ୟନ୍ତ; ଉତ୍ସ କୋଡ୍, ଚଳ (ଡାଟା ପ୍ରକାର ସହିତ) ଚଳ ଏବଂ ମେମୋରୀ ଅବସ୍ଥାନ, ସଂକଳନ, ବସ୍ତୁ ଏବଂ କାର୍ଯ୍ୟକାରୀ କୋଡ୍ ରେ ସିଣ୍ଟାକ୍ସ ଏବଂ ଯୁକ୍ତିଯୁକ୍ତ ତ୍ରୁଟି

ଅଧ୍ୟାୟ 2: ଗଣିତ ଅଭିବ୍ୟକ୍ତି ଏବଂ ଅଗ୍ରାଧିକାର

ଅଧ୍ୟାୟ 3: ସର୍ତ୍ତମୂଳକ ଶାଖା ଏବଂ ଲୁପ୍

ସର୍ତ୍ତାବଳୀ ଏବଂ ଫଳସ୍ୱରୂପ ଶାଖାର ଲେଖା ଏବଂ ମୂଲ୍ୟାଙ୍କନ

ପୁନରାବୃତ୍ତି ଏବଂ ଲୁପ୍

ଅଧ୍ୟାୟ 4: ଆରେ

ଆରେସ୍ (1-ଡି, 2-ଡି), ଚରିତ୍ର ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍

ଅଧ୍ୟାୟ 5: ମୌଳିକ ଆଲଗୋରିଦମ୍

ସନ୍ଧାନ, ମୌଳିକ ସର୍ଟିଂ ଆଲଗୋରିଦମ୍ (ବବଲ୍, ସନ୍ନିବେଶ ଏବଂ ଚୟନ), ସମୀକରଣର ମୂଳ ଖୋଜିବା, ଉଦାହରଣ ପ୍ରୋଗ୍ରାମ୍ ମାଧ୍ୟମରେ ଜଟିଳତାର କ୍ରମର ଧାରଣା (କୌଣସି ଆନୁଷ୍ଠାନିକ ସଂଜ୍ଞା ଆବଶ୍ୟକ ନାହିଁ)

ଅଧ୍ୟାୟ 6: ପ୍ରକାର୍ଯ୍ୟ

ପ୍ରକାର୍ଯ୍ୟଗୁଡିକ (ଲାଇବ୍ରେରୀରେ ନିର୍ମିତ ବ୍ୟବହାର କରିବା ଅନ୍ତର୍ଭୁକ୍ତ), ପ୍ରକାର୍ଯ୍ୟଗୁଡିକରେ ପାସ୍ କରୁଥିବା ପାରାମିଟର୍, ମୂଲ୍ୟଦ୍ୱାରା କଲ୍,

ପ୍ରକାର୍ଯ୍ୟଗୁଡିକୁ ଆରେ ପାସ୍ କରିବା: ରେଫରେନ୍ସଦ୍ୱାରା କଲ୍ର ଧାରଣା

ଅଧ୍ୟାୟ 7: ପୁନରାବୃତ୍ତି

ସମସ୍ୟାର ସମାଧାନର ଏକ ଭିନ୍ନ ଉପାୟ ଭାବରେ ପୁନରାବୃତ୍ତି କରନ୍ତୁ। ଉଦାହରଣ ପ୍ରୋଗ୍ରାମଗୁଡିକ, ଯେପରିକି ଫାଇଣ୍ଡିଂ ଫ୍ୟାକ୍ଟୋରିଆଲ୍, ଫିବୋନାକି ସିରିଜ୍, ଏକରମେନ୍ ପ୍ରକାର୍ଯ୍ୟ ଇତ୍ୟାଦି, ଦ୍ରୁତ ସର୍ଟ କିମ୍ବା ମିଶ୍ରଣ ସର୍ଟ

ଅଧ୍ୟାୟ 8: ଷ୍ଟ୍ରକଚର

ଷ୍ଟ୍ରକଚର, ବ୍ୟାଖ୍ୟା କାରୀ ଷ୍ଟ୍ରକଚର ଏବଂ ଷ୍ଟ୍ରକଚରର ଆରେ

ଅଧ୍ୟାୟ 9: ସୁଚକ

ପଏଣ୍ଟରର ଧାରଣା, ସୁଚକ ବ୍ୟାଖ୍ୟା କରିବା, ଆତ୍ମ-ରେଫରେଣ୍ଟାଲ୍ ଷ୍ଟ୍ରକଚରରେ ସୁଚକ ବ୍ୟବହାର, ଲିଙ୍କ୍ ହୋଇଥିବା ତାଲିକାର ଧାରଣା (କୌଣସି କାର୍ଯ୍ୟକାରିତା ନାହିଁ)

ଅଧ୍ୟାୟ 10: ଫାଇଲ୍ ହ୍ୟାଣ୍ଡଲିଂ (କେବଳ ଯଦି ସମୟ ଉପଲବ୍ଧ ହୁଏ, ଅନ୍ୟଥା ଲ୍ୟାବ୍ ର ଏକ ଅଂଶ ଭାବରେ କରାଯିବା ଉଚିତ୍)

ପରୀକ୍ଷାଗାର: ସମସ୍ୟା ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ [ଏଲ୍ : 0; ଟି:0 ; ପି : 4 (2 କ୍ରେଡିଟ୍)]

ପରୀକ୍ଷଣଗୁଡିକର ପ୍ରସ୍ତାବିତ ତାଲିକା:

[ଦିଆଯାଇଥିବା ସମସ୍ୟାପାଇଁ କାର୍ଯ୍ୟକାରୀ ହେବାକୁ ଥିବା ଆଭିମୁଖ୍ୟ କିମ୍ବା ଆଲଗୋରିଦମ୍ ବ୍ୟାଖ୍ୟା କରିବାକୁ ଏକ ଟ୍ୟୁଟୋରିଆଲ୍ ପୂର୍ବରୁ କିମ୍ବା ତା'ପରେ ପରୀକ୍ଷାଗାର କରାଯିବା ଉଚିତ୍।]

ଟ୍ୟୁଟୋରିଆଲ୍ 1: କମ୍ପ୍ୟୁଟର୍ ବ୍ୟବହାର କରି ସମସ୍ୟା ସମାଧାନ ହେଉଛି:

ଲ୍ୟାବ1: ପ୍ରୋଗ୍ରାମିଂ ପରିବେଶ ସହିତ ପରିଚିତ

ଟ୍ୟୁଟୋରିଆଲ୍ 2: ଚଳ ପ୍ରକାର ଏବଂ ପ୍ରକାର ରୂପାନ୍ତରଣ:

ଲ୍ୟାବ୍ 2: ଗଣିତ ଅଭିବ୍ୟକ୍ତି ବ୍ୟବହାର କରି ସରଳ ଗଣନାସମସ୍ୟା

ଟ୍ୟୁଟୋରିଆଲ୍ 3: ଶାଖା ଏବଂ ଯୁକ୍ତିଯୁକ୍ତ ଅଭିବ୍ୟକ୍ତି:

ଲ୍ୟାବ୍ 3: ଯଦି-ସେତେବେଳେ ଅନ୍ୟ ଷ୍ଟ୍ରକଚର ସହିତ ଜଡିତ ସମସ୍ୟା

ଟ୍ୟୁଟୋରିଆଲ୍ 4: ଲୁପ୍, ସମୟରେ ଏବଂ ଲୁପ୍ପାଇଁ:

ଲ୍ୟାବ୍ 4: ଇଟେରେଟିଭ୍ ସମସ୍ୟା ଉଦାହରଣ, ସିରିଜ୍ ର ସମଷ୍ଟି

ଟ୍ୟୁଟୋରିଆଲ୍ 5: 1ଡି ଆରେସ୍: ସନ୍ଧାନ, ସର୍ଟିଂ:

ଲ୍ୟାବ୍ 5: 1ଡି ଆରେ ହେରଫେର

ଟ୍ୟୁଟୋରିଆଲ୍ 6: 2ଡି ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍

ଲ୍ୟାବ୍ 6: ମ୍ୟାଟ୍ରିକ୍ସ ସମସ୍ୟା, ଷ୍ଟ୍ରିଙ୍ଗ୍ ସଞ୍ଚାଳନ

ଟ୍ୟୁଟୋରିଆଲ୍ 7: ପ୍ରକାର୍ଯ୍ୟଗୁଡିକ, ମୂଲ୍ୟ ଅନୁଯାୟୀ କଲ୍ କରନ୍ତୁ:

ଲ୍ୟାବ୍ 7: ସରଳ ପ୍ରକାର୍ଯ୍ୟଗୁଡିକ

ଟ୍ୟୁଟୋରିଆଲ୍ 8 &9: ସଂଖ୍ୟାଗତ ପଦ୍ଧତି (ରୁଟ୍ ସନ୍ଧାନ, ସଂଖ୍ୟାଗତ ଭିନ୍ନତା, ସଂଖ୍ୟାଗତ ଏକୀକରଣ):

ଲ୍ୟାବ୍ 8 ଏବଂ 9: ସଂଖ୍ୟାଗତ ପଦ୍ଧତି ସମସ୍ୟାର ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ

ଟ୍ୟୁଟୋରିଆଲ୍ 10: ପୁନରାବୃତ୍ତି, ପୁନରାବୃତ୍ତି କଲ୍ ର ଗଠନ

ଲ୍ୟାବ୍ 10: ପୁନରାବୃତ୍ତି ପ୍ରକାର୍ଯ୍ୟଗୁଡିକ

ଟ୍ୟୁଟୋରିଆଲ୍ 11: ସୁଚକ, ଷ୍ଟ୍ରକଚର ଏବଂ ଗତିଶୀଳ ମେମୋରୀ ଆବଣ୍ଟନ

ଲ୍ୟାବ୍ 11: ସୁଚକ ଏବଂ ଷ୍ଟ୍ରକଚରଗୁଡିକ

ଟ୍ୟୁଟୋରିଆଲ୍ 12: ଫାଇଲ୍ ପରିଚାଳନା:

ଲ୍ୟାବ୍ 12: ଫାଇଲ୍ ସଞ୍ଚାଳନ

ପାଠ୍ୟକ୍ରମ ଫଳାଫଳ

ପାଠ୍ୟକ୍ରମ ସମାପ୍ତ ହେବା ପରେ ଛାତ୍ରଛାତ୍ରୀମାନେ ନିମ୍ନଲିଖିତ ବିଷୟ ଶିକ୍ଷାରେ ସକ୍ଷମ ହେବେ:

CO-1: ଗଣିତ ଏବଂ ଯୁକ୍ତିଯୁକ୍ତ ସମସ୍ୟାପାଇଁ ସରଳ ଆଲଗୋରିଦମ୍ ପ୍ରସ୍ତୁତ କରିବା

CO -2: ପ୍ରୋଗ୍ରାମଗୁଡ଼ିକୁ ଆଲଗୋରିଦମ୍ ଅନୁବାଦ କରିବାକୁ (C ଭାଷାରେ)

CO-3: ପ୍ରୋଗ୍ରାମଗୁଡ଼ିକ ପରୀକ୍ଷା ଏବଂ କାର୍ଯ୍ୟକାରୀ କରିବାକୁ ଏବଂ ସିଣ୍ଟାକ୍ସ ଏବଂ ଯୁକ୍ତିଯୁକ୍ତ ତ୍ରୁଟିଗୁଡ଼ିକ ସଂଶୋଧନ କରନ୍ତୁ

CO-4: ସର୍ତ୍ତମୂଳକ ଶାଖା, ପୁନରାବୃତ୍ତି ଏବଂ ପୁନରାବୃତ୍ତି କାର୍ଯ୍ୟକାରୀ କରିବାକୁ

CO-5: କାର୍ଯ୍ୟରେ ଏକ ସମସ୍ୟାକୁ ବିଘଟନ କରିବା ଏବଂ ବିଭାଜନ ଏବଂ ବିଜୟ ଆଭିମୁଖ୍ୟ ବ୍ୟବହାର କରି ଏକ ସମ୍ପୂର୍ଣ୍ଣ ପ୍ରୋଗ୍ରାମ୍ ସଂଶ୍ଳେଷଣ କରିବା

CO-6: ଆଲଗୋରିଦମ୍ ଏବଂ ପ୍ରୋଗ୍ରାମ୍ ପ୍ରସ୍ତୁତ କରିବାକୁ ଆରେ, ସୂଚକ ଏବଂ ଷ୍ଟ୍ରକଚର ବ୍ୟବହାର କରିବାକୁ

CO-7: ମ୍ୟାଟ୍ରିକ୍ସ ଯୋଗ ଏବଂ ଗୁଣନ ସମସ୍ୟାର ସମାଧାନ ଏବଂ ସନ୍ଧାନ ଏବଂ ସଜାଡ଼ିବା ସମସ୍ୟାର ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ ପ୍ରୟୋଗ କରିବା

CO-8: ସରଳ ସଂଖ୍ୟାଗତ ପଦ୍ଧତି ସମସ୍ୟାର ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମିଂ ପ୍ରୟୋଗ କରିବା, ଯଥା କାର୍ଯ୍ୟର ମୂଳ ସନ୍ଧାନ, କାର୍ଯ୍ୟର ଭିନ୍ନତା ଏବଂ ସରଳ ଏକୀକରଣ

ପାଠ୍ୟକ୍ରମ ଫଳାଫଳ (CO)	ପ୍ରୋଗ୍ରାମ୍ ଫଳାଫଳଗୁଡ଼ିକ (PO) ସହ ଆଶା କରାଯାଉଥିବା ମ୍ୟାପିଂ (1- ଦୁର୍ବଳ ସହବନ୍ଧନ; 2- ମଧ୍ୟମ ସହବନ୍ଧନ; 3- ଦୃଢ଼ ସହବନ୍ଧନ)											
	PO-1	**PO-2**	**PO-3**	**PO-4**	**PO-5**	**PO-6**	**PO-7**	**PO-8**	**PO-9**	**PO-10**	**PO-11**	**PO-12**
CO-1	3	3	3	-	-	-	-	-	-	-	-	-
CO-2	3	-	1	-	-	-	-	-	-	-	-	-
CO-3	3	-	1	-	3	-	-	-	-	-	-	-
CO-4	3	-	-	-	-	-	-	-	-	-	-	-
CO-5	3	3	1	2	-	-	-	-	-	-	-	-
CO-6	3	-	1	-	-	-	-	-	-	-	-	-
CO-7	3	2	1	1	-	-	-	-	-	-	-	-
CO-8	3	2	1	1	-	-	-	-	-	-	-	-

ସଂକ୍ଷେପଣ ଓ ପ୍ରତୀକ

ସଂକ୍ଷିପ୍ତ ନାମର ତାଲିକା

ସଂକ୍ଷିପ୍ତ ନାମ	ବିସ୍ତୃତ ନାମ
ଏଏଲୟୁ (ALU)	ଗଣିତ ଏବଂ ତର୍କ ଏକକ
ଏଏନଏସଆଇ(ANSI)	ଆମେରିକୀୟ ଜାତୀୟ ମାନକ ପ୍ରତିଷ୍ଠାନ
ଏଏସସିଆଇଆଇ(ASCII)	ସୂଚନା ଅଦଳବଦଳପାଇଁ ଆମେରିକୀୟ ମାନକ କୋଡ୍
ବାୟୋସ୍(BIOS)	ମୌଳିକ ଇନପୁଟ୍ ଆଉଟପୁଟ୍ ସିଷ୍ଟମ୍
ବାୟୋସ୍(BIOS)	ମୌଳିକ ଇନପୁଟ୍ ଆଉଟପୁଟ୍ ସିଷ୍ଟମ୍
ବିଏମଆଇ(BMI)	ବଡି ମାସ୍ ଇଣ୍ଡେକ୍ସ
ବିଓଡିଏମଏଏସ(BODMAS)	ବ୍ରାକେଟ୍, ଅଫ୍, ଡିଭିଜନ୍, ମଲ୍ଟିପ୍ଲିକେସନ୍, ଯୋଗ, ଏବଂ ଉପଟ୍ରାକ୍ସନ୍
ବସ୍(BOSS)	ଭାରତ ଅପରେଟିଂ ସିଷ୍ଟମ୍ ସମାଧାନ
ସିପିୟୁ(CPU)	କେନ୍ଦ୍ରୀୟ ପ୍ରକ୍ରିୟାକରଣ ଏକକ
ଇଓଏଫ୍(EOF)	ଫାଇଲ୍ ଶେଷ
ଜିସିଡି(GCD)	ସର୍ବଶ୍ରେଷ୍ଠ ସାଧାରଣ ଡିଭିଜର
ଗୁଇ(GUI)	ଗ୍ରାଫିକାଲ୍ ଉପଭୋକ୍ତା ଇଣ୍ଟରଫେସ୍
ଏଚସିଏଫ(HCF)	ସର୍ବୋଚ୍ଚ ସାଧାରଣ କାରକ
ଏଚଏଲଏଲ(HLL)	ଉଚ୍ଚସ୍ତରୀୟ ଭାଷା
ଆଇଡିଇ(IDE)	ସମନ୍ୱିତ ବିକାଶ ପରିବେଶ
ଆଇପିଓ(IPO)	ଇନପୁଟ୍-ପ୍ରକ୍ରିୟା-ଆଉଟପୁଟ୍
ଓଏସ୍(OS)	ଅପରେଟିଂ ସିଷ୍ଟମ୍
ପିଡିଏଲ(PDL)	ପ୍ରୋଗ୍ରାମ୍ ଡିଜାଇନ୍ ଭାଷା
ପୋଷ୍ଟ(POST)	ଆତ୍ମ ପରୀକ୍ଷାରେ ଶକ୍ତି
ରାମ୍(RAM)	ଅନିୟମିତ ଆକସେସ୍ ମେମୋରୀ
ରୋମ୍(ROM)	କେବଳ ମେମୋରୀ ପଢନ୍ତୁ
ଭିଡିୟୁ(VDU)	ଭିଜୁଆଲ୍ ପ୍ରଦର୍ଶନ ଏକକ
ଭିଏଲଏସଆଇ(VLSI)	ବହୁତ ବଡ଼ ଧରଣର ଏକୀକରଣ

ଚିତ୍ର ସୂଚୀ

ଅଧ୍ୟାୟ-1: ପ୍ରୋଗ୍ରାମିଂ ପରିଚୟ

ଚିତ୍ର . 1.1: ଏକ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ଚାଳନ ଅଂଗସମୂହ 3
ଚିତ୍ର. 1.2: ସାଧାରଣ ଇନପୁଟ୍ ଯନ୍ତ୍ରାଂଶ 4
ଚିତ୍ର. 1.3: ସାଧାରଣ ଆଉଟପୁଟ୍ ଯନ୍ତ୍ରାଂଶ 5
ଚିତ୍ର . 1.4: ମେମୋରୀ ଉପକରଣ 5
ଚିତ୍ର. 1.5: ସାଧାରଣ ଭଣ୍ଡାରଣ ଯନ୍ତ୍ରାଂଶ ସମୂହ 6
ଚିତ୍ର . 1.6: ଏକ ବ୍ୟକ୍ତିଗତ କମ୍ପ୍ୟୁଟର ଚାଳନର ଚିତ୍ର 7
ଚିତ୍ର. 1.7: କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ସ୍ତରଯୁକ୍ତ ସ୍ଥାପତ୍ୟ 9
ଚିତ୍ର 1.8: ଅପରେଟିଂ ସିଷ୍ଟମ୍ ପରିଚାଳନା କରୁଥିବା କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ବିଭିନ୍ନ ସାଧନସମୂହ 9
ଚିତ୍ର . 1.9: ଏକ ଦ୍ୱିଘାତୀ ସମୀକରଣର ମୂଳ ପ୍ରକୃତି ଜାଣିବାପାଇଁ ଫ୍ଲୋ ଚାର୍ଟ 15
ଚିତ୍ର. 1.10: କ୍ରମ ସଂରଚନାପାଇଁ ସ୍ୟୁଡୋ କୋଡ୍ ଏବଂ ଫ୍ଲୋଚାର୍ଟ 17
ଚିତ୍ର. 1.11: If... Endifପାଇଁ ଚୟନ ସଂରଚନାର ସ୍ୟୁଡୋ କୋଡ୍ ଏବଂ ଫ୍ଲୋଚାର୍ଟ 17
ଚିତ୍ର. 1.12: If . . . Else . . . Endifପାଇଁ ଚୟନ ସଂରଚନାର ସ୍ୟୁଡୋକୋଡ୍ ଏବଂ ଫ୍ଲୋଚାର୍ଟ 18
ଚିତ୍ର. 1.13: Else If ladder ସିଡ଼ିର ସିଣ୍ଟାକ୍ସ 18
ଚିତ୍ର. 1.14: Else If ladder ସିଡ଼ିର ଲଜିକ ପ୍ରବାହ 19
ଚିତ୍ର. 1.15: while....Endwhile ସଂରଚନାପାଇଁ ସ୍ୟୁଡୋକୋଡ୍ ଏବଂ ଫ୍ଲୋଚାର୍ଟ 19
ଚିତ୍ର. 1.16: Do ... While ସଂରଚନାପାଇଁ ସ୍ୟୁଡୋକୋଡ୍ ଏବଂ ଫ୍ଲୋଚାର୍ଟ 20
ଚିତ୍ର. 1.17: ଦୁଇଟି ଚଳ ଅଦଳବଦଳ କରିବାକୁ ଫ୍ଲୋଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ 22
ଚିତ୍ର. 1.18: ଦିଆଯାଇଥିବା ସଂଖ୍ୟା ଯୁଗ୍ମ କି ଅଯୁଗ୍ମ ପରୀକ୍ଷା କରିବାକୁ ଫ୍ଲୋଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ 23
ଚିତ୍ର. 1.19: ତିନୋଟି ସଂଖ୍ୟା ମଧ୍ୟରୁ ସର୍ବବୃହତ୍ ଟିକୁ ଖୋଜିବାପାଇଁ ଫ୍ଲୋଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ 24
ଚିତ୍ର. 1.20: କମିଶନ ହିସାବପାଇଁ ଫ୍ଲୋଚାର୍ଟ 26
ଚିତ୍ର. 1.21: ଏକ ସଂଖ୍ୟାର ଅଙ୍କଗୁଡ଼ିକର ସମଷ୍ଟି ପାଇବାପାଇଁ ଫ୍ଲୋଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ 27
ଚିତ୍ର. 1.22: ଦିଆଯାଇଥିବା ସଂଖ୍ୟା n ଟି ପାଲିଣ୍ଡ୍ରୋମ୍ କି ନୁହେଁ ଜାଣିବାକୁ ଫ୍ଲୋଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ 28
ଚିତ୍ର. 1.23: ଦିଆଯାଇଥିବା ସଂଖ୍ୟା n ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ କି ନୁହେଁଯାଞ୍ଚ କରିବାକୁ ଫ୍ଲୋଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ 29
ଚିତ୍ର. 1.24: ନମ୍ବର୍ n ମୌଳିକ କି ନୁହେଁ ଯାଞ୍ଚ କରିବାକୁ ଫ୍ଲୋଚାର୍ଟ 30
ଚିତ୍ର. 1.25: ଗ.ସା.ଗୁ (HCF/GCD) ଗଣନା ପ୍ରକ୍ରିୟାର ଚିତ୍ରଣ 31
ଚିତ୍ର. 1.26: ଦୁଇଟି ସଂଖ୍ୟାର ଏଚସିଏଫ୍ ଗଣନା କରିବାକୁ ଫ୍ଲୋଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ 32
ଚିତ୍ର. 1.27: ଫିବୋନାକି କ୍ରମର ପ୍ରଥମ n ଟି ପଦ ପ୍ରିଣ୍ଟ କରିବାକୁ ଫ୍ଲୋଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ 33
ଚିତ୍ର 1.28: ଏକ C ପ୍ରୋଗ୍ରାମର ସାଧାରଣ ଗଠନ 34

ଚିତ୍ର . 1.29: ସଂକଳନ ପ୍ରକ୍ରିୟା 40
ଚିତ୍ର . 1.30: ସିଣ୍ଟେକ୍ସ ତ୍ରୁଟି ସୂଚିତ କରୁଥିବା ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍ ସ୍କ୍ରିନ୍ ସଟ୍ 41
ଚିତ୍ର . 1.31: ସଂକଳନ ପ୍ରକ୍ରିୟାର ସଫଳତା ସୂଚିତ କରୁଥିବା ଟର୍ବୋ C/C++ ସଙ୍କଳକ ର ସ୍କ୍ରିନ୍ ସଟ୍ 41
ଚିତ୍ର . 1.32: ସଂକଳନ ପ୍ରକ୍ରିୟାର ସଫଳତା ସୂଚିତ କରୁଥିବା ଟର୍ବୋ C/C++ ସଙ୍କଳକ ର ସ୍କ୍ରିନ୍ ସଟ୍ 42
ଚିତ୍ର. 1.33: ପ୍ରୋଗ୍ରାମ୍ ଆଉଟପୁଟ୍ ଦେଖାଉଥିବା ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍ ର ବ୍ୟବହାରକାରୀ-ସ୍କ୍ରିନ୍ 42

ଅଧ୍ୟାୟ-3: ସର୍ତ୍ତମୂଳକ ଶାଖା ଏବଂ ଲୁପ୍

ଚିତ୍ର. 3.1: if ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍ 99
ଚିତ୍ର. 3.2: if-else ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍ 101
ଚିତ୍ର . 3.3: ଇଫ୍-ଏଲ୍ସ ସିଡ଼ି କୋଡ୍ 104
ଚିତ୍ର. 3.4: if-elseif ସିଡ଼ି ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ 104
ଚିତ୍ର. 3.5: ବିବୃତ୍ତିର କୋଡ୍ ସୁଇଚ୍ କରନ୍ତୁ 105
ଚିତ୍ର. 3.6: switch ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ 106
ଚିତ୍ର. 3.7: ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍ପାଇଁ ବିବୃତ୍ତି 114
ଚିତ୍ର. 3.8: while ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍ 115
ଚିତ୍ର. 3.9: do - while ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍ 118
ଚିତ୍ର . 3.10: switch ବିବୃତ୍ତି ଭିତରେ break ବିବୃତ୍ତିର କାର୍ଯ୍ୟ 122
ଚିତ୍ର . 3.11: for, while ଏବଂ do-while ବିବୃତ୍ତି ଭିତରେ break ବିବୃତ୍ତିର କାର୍ଯ୍ୟ 123
ଚିତ୍ର . 3.12: for, while ଏବଂ do-while ବିବୃତ୍ତି ଭିତରେ continue ବିବୃତ୍ତିର କାର୍ଯ୍ୟ କାରୀତା 124
ଚିତ୍ର . 3.13: ଲମ୍ବା ବିଭାଜନ ବ୍ୟବହାର କରି GCDପାଇଁ ଗଣନା ପ୍ରକ୍ରିୟାର ଚିତ୍ର 129

ଅଧ୍ୟାୟ-4: ଆରେ

ଚିତ୍ର . 4.1: ମାର୍କ ଆରେ 153
ଚିତ୍ର . 4.2: 1-D ଆରେର ଘୋଷଣା 154
ଚିତ୍ର . 4.3: ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ 161
ଚିତ୍ର. 4.4: ମେମୋରିରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଷ୍ଟୋର୍ କରିବା 169
ଚିତ୍ର . 4.5: ବର୍ଣ୍ଣ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍ ର ଷ୍ଟୋରେଜ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ 170
ଚିତ୍ର. 4.6: ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଏବଂ ଏକ ଅକ୍ଷର ଆରେ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ 170
ଚିତ୍ର. 4.7: ଆରେର ଏକ ଅଂଶରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଷ୍ଟୋର୍ ହୋଇଛି 170

ଅଧ୍ୟାୟ-5: ମୌଳିକ ଆଲଗୋରିଦମ୍

ଚିତ୍ର . 5.1: ବବଲ୍ ସର୍ଟ (Bubble sort) ପଦ୍ଧତିର ଚିତ୍ରଣ 204

ଚିତ୍ର . 5.2: ସିଲେକସନ ସର୍ଟ ପଦ୍ଧତିର ଚିତ୍ରଣ 207
ଚିତ୍ର. 5.3: ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ପଦ୍ଧତିର ଚିତ୍ରଣ 211
ଚିତ୍ର . 5.4: ଦ୍ୱିଭାଜନ ପଦ୍ଧତିଦ୍ୱାରା ମୂଳ ଆନୁମାନିକତା 213
ଚିତ୍ର . 5.5: ଏକ ପ୍ରକାର୍ଯ୍ୟ f(x)ପାଇଁ ଆନୁମାନିକ ପଲିନୋମିଆଲ୍ p(x) 222
ଚିତ୍ର . 5.6: ଛାୟାଙ୍କିତ କ୍ଷେତ୍ରଦ୍ୱାରା ପ୍ରତିନିଧିତ୍ୱ କରୁଥିବା ସୀମାଯୁକ୍ତ ସମାକଳ 222
ଚିତ୍ର . 5.7: ଟ୍ରାପିଜୋଇଡାଲ୍ ନିୟମଦ୍ୱାରା କ୍ଷେତ୍ରଫଳର ଆନୁମାନିକତା 223

ଅଧ୍ୟାୟ-6: ଫଙ୍କସନ

ଚିତ୍ର . 6.1: ଏକ ବହୁ-ପ୍ରକାର୍ଯ୍ୟ ପ୍ରୋଗ୍ରାମର ପଦାନୁକ୍ରମିକ ସଂଗଠନ 229
ଚିତ୍ର 6.2: ପ୍ରକାର୍ଯ୍ୟ ଘୋଷଣାର ବିଧି (Syntax) 231
ଚିତ୍ର. 6.3: ପ୍ରକାର୍ଯ୍ୟ ସଂଜ୍ଞା ଲେଖିବାର ବିଧି (Syntax) 232
ଚିତ୍ର 6.4: ଏକ ପ୍ରକାର୍ଯ୍ୟରେ ଚର ଭାବରେ ଏକ- ପରିସର ଆରେ ଗୃହୀତ କରିବା 242
ଚିତ୍ର 6.5: ଏକ ପ୍ରକାର୍ଯ୍ୟପାଇଁ ଚର ଭାବରେ ଦୁଇ - ପରିସର ଆରେ ଗୃହୀତ କରିବା 243

ଅଧ୍ୟାୟ-7: ରିକର୍ସନ୍

ଚିତ୍ର . 7.1: ଆରେ ବିଭାଜନର ଉଦାହରଣ 269
ଚିତ୍ର . 7.2: ମର୍ଜ ସର୍ଟ କ୍ରମାଗତ ପଦକ୍ଷେପ ଚିତ୍ରଣ 272

ଅଧ୍ୟାୟ-8: ଷ୍ଟ୍ରକଚର

ଚିତ୍ର. 8.1: ଏକ ଟ୍ୟାଗ୍ ଯୁକ୍ତ ଷ୍ଟ୍ରକଚର ବ୍ୟାଖ୍ୟା 291
ଚିତ୍ର 8.2: ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକଚର ଘୋଷଣା 292
ଚିତ୍ର. 8.3: ଫଙ୍କ୍ସନକୁ ଷ୍ଟ୍ରକ୍‌ଚର ପ୍ରେରଣ 301
ଚିତ୍ର. 8.4: ଷ୍ଟ୍ରକ୍‌ଚର ଫେରସ୍ତ କରୁଥିବା ଏକ ଫଙ୍କ୍ସନ 302

ଅଧ୍ୟାୟ-9: ପଏଁଟର

ଚିତ୍ର . 9.1: ମେମୋରୀ ସଂଗଠନ 322
ଚିତ୍ର . 9.2: ମେମୋରୀରେ ଏକ ଚଳର ଉପସ୍ଥାପନା 322
ଚିତ୍ର. 9.3: ପଏଁଟର ଏକ ଚଳ ଭାବରେ 322
ଚିତ୍ର . 9.4: 4 ଟି ନୋଡ୍ ସହିତ ଇଣ୍ଟିଜର୍ ମୂଲ୍ୟର (Linear linked list) 328

ଅଧ୍ୟାୟ -10: ଫାଇଲ୍ ପରିଚାଳନା

ଚିତ୍ର. 10.1: ଫାଇଲ ଗୁଡ଼ିକରେ ତଥ୍ୟ ସଂରକ୍ଷଣର ଚିତ୍ରଣ 345

ଟେବୁଲଗୁଡିକର ତାଲିକା

ଅଧ୍ୟାୟ-1: ପ୍ରୋଗ୍ରାମିଂ ପରିଚୟ

ଟେବୁଲ୍ 1.1: ବିଭିନ୍ନ ଫ୍ଲୋଚାର୍ଟ ପ୍ରତୀକ ଏବଂ ସେମାନଙ୍କର ସଂକ୍ଷିପ୍ତ ବର୍ଣ୍ଣନା 15

ଟେବୁଲ୍ 1.2: Cରେ କୀୱାର୍ଡଗୁଡିକ 47

ଟେବୁଲ୍ 1.3: ସାଧାରଣ ଏସ୍କେପ ସିକ୍ୱେଂସ 48

ଟେବୁଲ୍ 1.4: କିଛି ବିରାମ ଚିହ୍ନ ଏବଂ ସେମାନଙ୍କର ବର୍ଣ୍ଣନା 48

ଟେବୁଲ୍ 1.5: ବିଲ୍ଟ-ଇନ୍ ଡାଟା ଟାଇପ୍ 50

ଟେବୁଲ୍ 1.6: ପୂର୍ଣସଂଖ୍ୟା ନମ୍ବରର ପ୍ରକାର 51

ଟେବୁଲ୍ 1.7: ବାସ୍ତବ ସଂଖ୍ୟାର ଡାଟା ଟାଇପ୍ 51

ଟେବୁଲ୍ 1.8: I/O ଫଙ୍କ୍ସନଗୁଡ଼ିକ 55

ଟେବୁଲ୍ 1.9: ସାଧାରଣତଃ ବ୍ୟବହୃତ ଫର୍ମାଟ୍ ସ୍ପେସିଫାୟର ତାଲିକା 57

ଅଧ୍ୟାୟ -2: ଗାଣିତିକ ପରିପ୍ରକାଶ ଏବଂ ଅପରେଟର୍ ପ୍ରାଥମିକତା

ଟେବୁଲ୍ 2.1: ବାଇନାରୀ ଗଣିତ ଅପରେଟର୍ 70

ଟେବୁଲ୍ 2.2: ସମ୍ବନ୍ଧୀୟ (ତୁଳନା) ଅପରେଟର୍ 71

ଟେବୁଲ୍ 2.3: ଲଜିକାଲ୍ ଅପରେଟର୍ 72

ଟେବୁଲ୍ 2.4: ବିଟୱାଇଜ୍ ଅପରେଟର୍ 73

ଟେବୁଲ୍ 2.5: sizeof ଅପରେଟର୍ ର ବ୍ୟବହାର 75

ଟେବୁଲ୍ 2.6: ଠିକଣା (&) ଅପରେଟର୍ ର ବ୍ୟବହାର 76

ଟେବୁଲ୍ 2.7: ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍ 78

ଟେବୁଲ୍ 2.8: ନମୁନା ଗାଣିତିକ ପରିପ୍ରକାଶ 79

ଟେବୁଲ୍ 2.9: ଗାଣିତିକ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନର ଚିତ୍ର 80

ଟେବୁଲ୍ 2.10: ଅପରେଟରମାନଙ୍କର ପ୍ରାଥମିକତା ଏବଂ ସହଯୋଗୀତା 84

ଟେବୁଲ୍ 2.11: ସାଧାରଣ ଭାବେ ବ୍ୟବହୃତ କିଛି ଗାଣିତକି ପ୍ରକାର୍ଯ୍ୟ 86

ଅଧ୍ୟାୟ-4: ଆରେ

ଟେବୁଲ୍ 4.1: ବାରମ୍ବାର ବ୍ୟବହୃତ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଫଙ୍ଗସନ୍ ଗୁଡ଼ିକ 173

ଟେବୁଲ୍ 4.2: (strcmp) ଫଙ୍ଗସନ୍ ଦ୍ୱାରା ଫେରସ୍ତ ମୂଲ୍ୟର ବ୍ୟାଖ୍ୟା 175

ଅଧ୍ୟାୟ-5: ମୌଳିକ ଆଲଗୋରିଦମ୍

ଟେବୁଲ୍ 5.1: ଅଗ୍ରାନ୍ତର ସାରଣୀ 220

ଅଧ୍ୟାୟ-6: ପ୍ରକାର୍ଯ୍ୟ

ଟେବୁଲ୍ 6.1: ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ ଏବଂ ରେଫରେନ୍ସଦ୍ୱାରା କଲ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଅଂଶ-1 241

ଟେବୁଲ୍ 6.2: ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ ଏବଂ ରେଫରେନ୍ସଦ୍ୱାରା କଲ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଅଂଶ-2 241

ଅଧ୍ୟାୟ-7: ରିକର୍ସନ୍

ଟେବୁଲ୍ 7.1: ତୁଳନା: ରିକର୍ସନ ବନାମ ଆଇଟରେସନ 275

ଅଧ୍ୟାୟ-9: ପଏଁଟର

ଟେବୁଲ୍ 9.1: ମେମୋରୀ ପରିଚାଳନା ଫଙ୍କ୍ସନଗୁଡ଼ିକ 328

ଅଧ୍ୟାୟ -10: ଫାଇଲ୍ ପରିଚାଳନା

ଟେବୁଲ୍ 10.1: ଟେକ୍ସଟ୍ ଫାଇଲ୍ ବନାମ ବାଇନାରୀ ଫାଇଲ୍ 345

ଟେବୁଲ୍ 10.2: ଫାଇଲ୍ ଖୋଲିବା ମୋଡ୍ 346

ଟେବୁଲ୍ 10.3: (seek) ଫଙ୍କସପାଇଁ wherefrom ର ବିଭିନ୍ନ ମୂଲ୍ୟ 360

ଟେବୁଲ୍ 10.4: (seek) ଫଙ୍କସନ ବ୍ୟବହାରକୁ ବର୍ଣ୍ଣନା କରୁଥିବା କିଛି ଉଦାହରଣ 360

ଶିକ୍ଷକମାନଙ୍କପାଇଁ ନିର୍ଦ୍ଦେଶାବଳୀ

ଫଳାଫଳ ଆଧାରିତ ଶିକ୍ଷା(Outcome Based Education - OBE)ର କାର୍ଯ୍ୟକାରିତା ପାଇଁ ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ଜ୍ଞାନ ଏବଂ ଦକ୍ଷତାର ସ୍ତରକୁ ବୃଦ୍ଧି କରାଯିବା ଉଚିତ୍। ଏହାର ଉପଯୁକ୍ତ କାର୍ଯ୍ୟକାରିତାପାଇଁ ଶିକ୍ଷକବୃନ୍ଦ ପ୍ରମୁଖ ଦାୟିତ୍ୱ ବହନ କରିବା ଆବଶ୍ୟକ। ଏହି OBE ପ୍ରଣାଳୀରେ ଶିକ୍ଷକମାନଙ୍କର ଦାୟିତ୍ୱ ମଧ୍ୟରୁ ନିମ୍ନରେ କିଛି ବର୍ଣ୍ଣନା କରାଯାଇଅଛି:

- ସମସ୍ତ ଛାତ୍ରଛାତ୍ରୀଙ୍କ ସର୍ବୋତ୍ତମ ସୁବିଧାପାଇଁ ସୀମିତ ପ୍ରତିବନ୍ଧକ ମଧ୍ୟରେ ସେମାନେ ସମୟର ଉପଯୋଗ କରିବା ଉଚିତ୍।
- ସେମାନେ ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ଅନ୍ୟ କୌଣସି ସମ୍ଭାବ୍ୟ ଅଯୋଗ୍ୟତାକୁ ବିଚାରକୁ ନେଇ ଭେଦଭାବ ନ କରି କେବଳ ନିର୍ଦ୍ଦିଷ୍ଟ ମାନଦଣ୍ଡ ଉପରେ ଆକଳନ କରିବା ଉଚିତ୍।
- ସେମାନେ ଅନୁଷ୍ଠାନ ଛାଡିବା ପୂର୍ବରୁ ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କ ଶିକ୍ଷଣ ଦକ୍ଷତାକୁ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ସ୍ତରକୁ ବୃଦ୍ଧି କରିବାକୁ ଚେଷ୍ଟା କରିବା ଉଚିତ୍।
- ସମସ୍ତ ଛାତ୍ରଛାତ୍ରୀମାନେ ସେମାନଙ୍କର ଶିକ୍ଷା ସମାପ୍ତ କରିବା ପରେ ଗୁଣାତ୍ମକ ଜ୍ଞାନ ଏବଂ ଦକ୍ଷତା ବୃଦ୍ଧି କରିପାରୁଥିବାର ସେମାନେ ନିଶ୍ଚିତ ହେବା ଆବଶ୍ୟକ।
- ସେମାନେ ସର୍ବଦା ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କୁ ସେମାନଙ୍କର ଚରମ କାର୍ଯ୍ୟଦକ୍ଷତା କ୍ଷମତାକୁ ବିକଶିତ କରିବାରେ ଉତ୍ସାହିତ କରିବା ଉଚିତ୍।
- ନୂତନ ଆଭିମୁଖ୍ୟକୁ ଏକତ୍ର କରିବାପାଇଁ ସେମାନେ ଗୋଷ୍ଠୀ କାର୍ଯ୍ୟ ଏବଂ ଦଳ କାର୍ଯ୍ୟକୁ ସୁଗମ ଏବଂ ଉତ୍ସାହିତ କରିବା ଉଚିତ୍।
- ସେମାନେ ବ୍ଲୁମଙ୍କ ଟ୍ୟାକ୍ସୋନୋମିକୁ ମୂଲ୍ୟାଙ୍କନର ପ୍ରତ୍ୟେକ କ୍ଷେତ୍ରରେ ଅନୁସରଣ କରିବା ଉଚିତ୍।

ମୂଲ୍ୟାଙ୍କନ:

ବ୍ଲୁମଙ୍କ ବର୍ଗୀକରଣ(Bloom's Taxonomy)

ସ୍ତର	ଶିକ୍ଷକ ଯାଞ୍ଚ କରିବା ଉଚିତ	ଛାତ୍ରଛାତ୍ରୀ ସକ୍ଷମ ହେବା ଉଚିତ	ମୂଲ୍ୟାଙ୍କନର ସମ୍ଭାବ୍ୟ ମୋଡ୍
ସୃଷ୍ଟି କରିବା Creating	ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ସୃଷ୍ଟି କରିବାର ଦକ୍ଷତା	ଡିଜାଇନ୍ କିମ୍ବା ସୃଷ୍ଟି କରିବା	କ୍ଷୁଦ୍ର ପ୍ରକଳ୍ପ
ମୂଲ୍ୟାଙ୍କନ କରିବା Evaluating	ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ଯଥାର୍ଥତା ମୂଲ୍ୟାଙ୍କନ କରିବାର ଦକ୍ଷତା	ଯୁକ୍ତି କିମ୍ବା ରକ୍ଷା କରିବା	ନିର୍ଦ୍ଦିଷ୍ଟ କାର୍ଯ୍ୟ
ବିଶ୍ଳେଷଣ କରିବା Analyzing	ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ଭିନ୍ନ କରିବାର ଦକ୍ଷତା	ଭିନ୍ନ କିମ୍ବା ଅଲଗା କରିବା	ପ୍ରକଳ୍ପ / ଲ୍ୟାବ୍ ପଦ୍ଧତି
ଆବେଦନ କରିବା Applying	ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ସୂଚନା ବ୍ୟବହାର କରିବାର ଦକ୍ଷତା	ସଞ୍ଚାଳନ କିମ୍ବା ପ୍ରଦର୍ଶନ କରିବା	ବୈଷୟିକ ଉପସ୍ଥାପନା / ପ୍ରଦର୍ଶନ
ବୁଝାମଣା Understanding	ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ଧାରଣାକୁ ବ୍ୟାଖ୍ୟା କରିବାର ଦକ୍ଷତା	ବ୍ୟାଖ୍ୟା କିମ୍ବା ଶ୍ରେଣୀଭୁକ୍ତ କରିବା	ଉପସ୍ଥାପନା / ସେମିନାର
ସ୍ମରଣ କରିବା Remembering	ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ମନେ ପକାଇବାର ଦକ୍ଷତା	ବ୍ୟାଖ୍ୟା କିମ୍ବା ମନେ ରଖିବା	ପ୍ରଶ୍ନୋତ୍ତରୀ

ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କପାଇଁ ନିର୍ଦ୍ଦେଶାବଳୀ

ଛାତ୍ରଛାତ୍ରୀମାନେ ମଧ୍ଯ OBE କାର୍ଯ୍ୟକାରୀ କରିବାପାଇଁ ସମାନ ଭାବେ ଦାୟିତ୍ୱ ନେବା ଉଚିତ। ସେମାନଙ୍କ ଦାୟିତ୍ୱଗୁଡିକ ମଧ୍ୟରୁ ନିମ୍ନରେ କିଛି ବର୍ଣ୍ଣନା କରାଯାଇଅଛି:

- ପ୍ରତ୍ୟେକ ପାଠ୍ୟକ୍ରମରେ ଏକ ଅଧ୍ୟାୟ ଆରମ୍ଭ ହେବା ପୂର୍ବରୁ ଛାତ୍ରଛାତ୍ରୀମାନେ ପ୍ରତ୍ୟେକ ଅଧ୍ୟାୟ ଫଳାଫଳ (Unit Outcome - UO) ବିଷୟରେ ଭଲ ଭାବରେ ଅବଗତ ହେବା ଉଚିତ୍।
- ପାଠ୍ୟକ୍ରମ ଆରମ୍ଭ ହେବା ପୂର୍ବରୁ ଛାତ୍ରଛାତ୍ରୀମାନେ ପ୍ରତ୍ୟେକ ପାଠ୍ୟକ୍ରମ ଫଳାଫଳ (Course Outcome - CO) ବିଷୟରେ ଭଲ ଭାବରେ ଅବଗତ ହେବା ଉଚିତ୍।
- କାର୍ଯ୍ୟକ୍ରମ ଆରମ୍ଭ ହେବା ପୂର୍ବରୁ ଛାତ୍ରଛାତ୍ରୀମାନେ ପ୍ରତ୍ୟେକ କାର୍ଯ୍ୟକ୍ରମ ଫଳାଫଳ (Programme Outcome - PO) ବିଷୟରେ ଭଲ ଭାବରେ ଅବଗତ ହେବା ଉଚିତ୍।
- ଛାତ୍ରଛାତ୍ରୀମାନେ ଉପଯୁକ୍ତ ପ୍ରତିଫଳନ ଏବଂ କାର୍ଯ୍ୟ ସହିତ ସମାଲୋଚନା ଏବଂ ଯୁକ୍ତିଯୁକ୍ତ ଭାବରେ ଚିନ୍ତା କରିବା ଉଚିତ୍।
- ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କର ଶିକ୍ଷାଗ୍ରହଣ ଦୈନନ୍ଦିନ ଏବଂ ବାସ୍ତବିକ ଜୀବନର ପରିଣାମସହିତ ସଂଯୁକ୍ତ ଏବଂ ଏକୀକୃତ ହେବା ଉଚିତ୍।
- ଛାତ୍ରଛାତ୍ରୀମାନେ OBEର ପ୍ରତ୍ୟେକ ସ୍ତରରେ ସେମାନଙ୍କର ଦକ୍ଷତା ବିଷୟରେ ଭଲ ଭାବରେ ଅବଗତ ହେବା ଉଚିତ୍।

ବିଷୟ ସୂଚୀ

ପ୍ରାକ୍‌କଥନ	iii
ଅନୁବାଦକଙ୍କ କଲମରୁ	v
କୃତଜ୍ଞତା	vii
ମୁଖବନ୍ଧ	ix
ଫଳାଫଳ ଆଧାରିତ ଶିକ୍ଷା	xi
ପାଠ୍ୟକ୍ରମ	xiii
ପାଠ୍ୟକ୍ରମ ଫଳାଫଳ	xv
ସଂକ୍ଷିପ୍ତ ନାମ ଏବଂ ପ୍ରତୀକଗୁଡ଼ିକ	xvii
ଚିତ୍ର ତାଲିକା	xix
ଟେବୁଲ ତାଲିକା	xxii
ଶିକ୍ଷକଙ୍କପାଇଁ ନିର୍ଦ୍ଦେଶାବଳୀ	xxiv
ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କପାଇଁ ନିର୍ଦ୍ଦେଶାବଳୀ	xxvi
ଅଧ୍ୟାୟ 1: ପ୍ରୋଗ୍ରାମିଂ ପରିଚୟ	**1-66**
ଏକକ ବିଶେଷତ୍ୱ	1
ଭୂମିକା	1
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ	1
ଏକକ ଲବ୍ଧ ଜ୍ଞାନ	2
1.1 କମ୍ପ୍ୟୁଟରର ପରିଚୟ (INTRODUCTION TO COMPUTERS)	2
1.1.1 କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ଅଂଗସମୂହ (Components of a Computer System)	3
1.1.2 ହାର୍ଡୱେୟାର୍ (Hardware)	7
1.1.3 ସଫ୍ଟୱେୟାର୍ (Software)	7
1.1.3.1 ଅପରେଟିଂ ସିଷ୍ଟମ୍ (Operating System)	8
1.1.3.2 ଭାଷାନୁବାଦକ (Language Translators)	10
1.1.3.3 ପ୍ରୋଗ୍ରାମିଂ ପରିବେଶ	11
1.1.4 ବୁଟିଂର ଧାରଣା (Concept of Booting)	13
1.2 ଆଲଗୋରିଦମ୍ ର ଧାରଣା (IDEA OF ALGORITHMS)	14
1.2.1 ଫ୍ଲୋଚାର୍ଟ (Flowchart)	15
1.2.2 ସ୍ୟୁଡୋକୋଡ୍ (Pseudocode)	16
ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ	22

1.3 ଆଲଗୋରିଦମ୍ ଠାରୁ ପ୍ରୋଗ୍ରାମ୍ ପର୍ଯ୍ୟନ୍ତ (FROM ALGORITHMS TO PROGRAM) 34
1.3.1 ଏକ C ପ୍ରୋଗ୍ରାମର ସଂରଚନା (Structure of a C Program) 34
1.3.2 C ପ୍ରୋଗ୍ରାମ୍ ର ଉଦାହରଣ (Example C Programs) 36
1.3.3 ଏକ ପ୍ରୋଗ୍ରାମର ତିଆରି, କମ୍ପାଇଲ୍ ଏବଂ ଚାଳନ (Creating, Compiling & Executing a Program) 39
1.3.4 ବିଭିନ୍ନ C କମ୍ପାଇଲର୍ 43
1.4 C ଭାଷା ଆରମ୍ଭ କରିବା (GETTING STARTED WITH C LANGUAGE) 43
1.4.1 C ଭାଷାର ବୈଶିଷ୍ଟ୍ୟଗୁଡିକ 44
1.4.2 C ଭାଷାର ପ୍ରୟୋଗ କ୍ଷେତ୍ର 45
1.4.3 C ଭାଷାର ମୌଳିକ ଗଢ଼ଣ ପିଣ୍ଡ(Basic Building Blocks of C Language) 45
1.4.3.1 ବର୍ଣ୍ଣ ସେଟ୍ 46
1.4.3.2 ଟୋକନ୍ 46
1.4.3.3 ଡାଟା ଟାଇପ୍ ର ଧାରଣା (Concept of Data Type) 49
1.4.3.4 ଧ୍ରୁବକ (Constants) 52
1.4.3.5 ଚଳ (variables) 52
1.4.3.6 ପରିପ୍ରକାଶ (Expressions) 53
1.4.3.7 ବିବୃତ୍ତି (Statements) 54
1.4.3.8 ଇନପୁଟ୍/ଆଉଟପୁଟ୍ ପରିଚାଳନା (Handling Input/Output) 54
ୟୁନିଟ ସାରାଂଶ 59
ଅନୁଶୀଳନୀ 61
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals) 66
ଅଧିକ ଜାଣିବାପାଇଁ 66
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ 66
ଅଧ୍ୟାୟ2: ଗଣିତ ଅଭିବ୍ୟକ୍ତି ଏବଂ ଅଗ୍ରାଧିକାର 67-96
ଏକକ ବିଶେଷତ୍ୱ 67
ଭୂମିକା 67
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ 67
ଏକକ ଲକ୍ଷ୍ଯ ଜ୍ଞାନ 67
2.1 ଉପକ୍ରମ 68
2.2 ଅପରେଟର୍ (operators) 68
2.2.1 ଗଣିତ ଅପରେଟର୍ (Arithmetic Operators) 69
2.2.2 ସମ୍ବନ୍ଧୀୟ (ତୁଳନା) ଅପରେଟର୍ (Relational (Comparison) Operators) 71

2.2.3 ଲଜିକାଲ୍ ଅପରେଟର୍ (Logical Operators) 72
2.2.4 ବିଟ ୱାଇଜ୍ ଅପରେଟର୍ (Bitwise Operators) 72
2.2.5 ସ୍ୱତନ୍ତ୍ର ଅପରେଟର୍ (Special Operators) 74
2.2.5.1 ବୃଦ୍ଧି ଏବଂ ହ୍ରାସ ଅପରେଟର୍ (Increment & Decrement Operators) 74
2.2.5.2 size of ଅପରେଟର୍ (The sizeof Operator) 75
2.2.5.3 ଠିକଣା ଅପରେଟର୍ (The addressof Operator) 76
2.2.5.4 ପରୋକ୍ଷ ଅପରେଟର୍ (Indirection Operator) 76
2.2.5.5 ସର୍ତ୍ତମୂଳକ/ଟର୍ନାରୀ ଅପରେଟର୍ (Conditional/Ternary Operator) 77
2.2.5.6 ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍ (Assignment operators) 77
2.3 ପରିପ୍ରକାଶ (EXPRESSIONS) 78
2.3.1 ଗାଣିତିକ ପରିପ୍ରକାଶ (Arithmetic Expressions) 79
2.3.2 ଗାଣିତିକ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ (Evaluation of Arithmetic Expressions) 80
2.3.3. ପ୍ରକାର ରୂପାନ୍ତରଣ (Type Conversion) 81
2.3.3.1 ଟାଇପର ଅପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ(Implicit Type Conversion) 81
2.3.3.2 ଟାଇପର ପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ(Explicit Type Conversion) 82
2.4 ପ୍ରାଥମିକତା ଏବଂ ସହଯୋଗୀତା (PRECEDENCE AND ASSOCIATIVITY) 83
2.5 ଲାଇବ୍ରେରୀ ଫଙ୍କସନ୍ (LIBRARY FUNCTIONS) 85
ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ 86
ୟୁନିଟ ସାରାଂଶ 89
ଅନୁଶୀଳନୀ 90
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals) 93
ଅଧିକ ଜାଣିବାପାଇଁ 95
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ 96

ଅଧ୍ୟାୟ 3: ସର୍ତ୍ତମୂଳକ ଶାଖା ଏବଂ ଲୁପ୍ 97-145

ଏକକ ବିଶେଷତ୍ୱ 97
ଭୂମିକା 97
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ 97
ଏକକ ଲବ୍ଧ ଜ୍ଞାନ 98
3.1 ଉପକ୍ରମ 98
3.2 ସର୍ତ୍ତମୂଳକ ଶାଖା (CONDITIONAL BRANCHING) 99
3.2.1 if ବିବୃତ୍ତି 99

3.2.2 if – else ବିବୃତ୍ତି	100
3.2.3 if ଏବଂ if-else ବିବୃତ୍ତିର ନୀଡ଼ନ (Nested if and if - else statements)	102
3.2.4 if-elseif ର ସିଢ଼ି (The if-else if Ladder)	104
3.2.5 switch ବିବୃତ୍ତି	105
ସର୍ତ୍ତମୂଳକ ଶାଖାପାଇଁ ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ	108
3.3 ଲୁପିଂ (Looping)	113
3.3.1 for ବିବୃତ୍ତି	115
3.3.2 while ବିବୃତ୍ତି	115
3.3.3 do- while ବିବୃତ୍ତି	117
3.3.4 ନୀଡ଼ନ ହ୍ୱାଇଲ୍, for ଏବଂ do – while ବିବୃତ୍ତି	119
3.4 ଜମ୍ପିଂ ବିବୃତ୍ତି (JUMPING STATEMENTS)	122
3.4.1 break ବିବୃତ୍ତି	122
3.4.2 continue ବିବୃତ୍ତି	123
ଲୁପ୍‌ପାଇଁ ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ	125
ୟୁନିଟ ସାରାଂଶ	131
ଅଭ୍ୟାସ	132
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals)	145
ଅଧିକ ଜାଣିବାପାଇଁ	149
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ	149
ଅଧ୍ୟାୟ 4: ଆରେ	**151-194**
ଏକକ ବିଶେଷତ୍ୱ	151
ଭୂମିକା	151
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ	151
ଏକକ ଲବ୍ଧ ଜ୍ଞାନ	152
4.1 ଉପକ୍ରମ	152
4.2 ଏକ-ପରିସରଯୁକ୍ତ ଆରେ	153
4.2.1 ଘୋଷଣା	154
4.2.2 ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଧିଷ୍ଟକରଣ	155
4.2.3 ଉପାଦାନକୁ ପହଞ୍ଚ (Accessing Elements)	156
4.2.4 ଇନପୁଟ୍	156
4.2.5 ଆଉଟପୁଟ୍	156

1D ଆରେର ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ	157
4.3 ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ	161
4.3.1 ଘୋଷଣାନାମା	162
4.3.2 ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଧିଷ୍ଟକରଣ	162
4.3.3 ଉପାଦାନଗୁଡ଼ିକୁ ପହଞ୍ଚ	163
4.3.4 ଇନପୁଟ୍	163
4.3.5 ଆଉଟପୁଟ୍	163
2D ଆରେର ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ	164
4.4 ଅକ୍ଷର ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍ (CHARACTER ARRAYS AND STRINGS)	169
4.4.1 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଭଣ୍ଡାର	169
4.4.2 ଷ୍ଟ୍ରିଙ୍ଗ୍ ସୀମାଧାର୍ଯ୍ୟ/ଡିଲିମିଟେଟର ଆବଶ୍ୟକତା	170
4.4.3 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ମୂଲ୍ୟ	171
4.4.4 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ	171
4.4.4.1 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳର ଘୋଷଣା	171
4.4.4.2 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଧିଷ୍ଟ	171
4.4.5 ଷ୍ଟ୍ରିଙ୍ଗର ଇନପୁଟ୍/ଆଉଟପୁଟ୍	172
4.4.6 ଷ୍ଟ୍ରିଙ୍ଗ୍ କାର୍ଯ୍ୟ ସାଧନ ଫଙ୍ଗସନ୍	173
4.4.6.1 strlen ଫଙ୍ଗସନ୍	173
4.4.6.2 strcpy ଫଙ୍ଗସନ୍	174
4.4.6.3 strcat ଫଙ୍ଗସନ୍	174
4.4.6.4 strcmp ଫଙ୍ଗସନ୍	175
4.4.6.5 strrev ଫଙ୍ଗସନ୍	176
4.4.6.6 strupr ଫଙ୍ଗସନ୍	177
4.4.6.7 strlwr ଫଙ୍ଗସନ୍	177
4.4.7 ଷ୍ଟ୍ରିଙ୍ଗ ଆରେ	178
ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ	180
ୟୁନିଟ ସାରାଂଶ	183
ଅଭ୍ୟାସ	183
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals)	189
ଅଧିକ ଜାଣିବାପାଇଁ	194
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ	194

ଅଧ୍ୟାୟ 5: ମୌଳିକ ଆଲଗୋରିଦମ୍ **195-226**
ଏକକ ବିଶେଷତ୍ୱ 195
ଭୂମିକା 195
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ 195
ଏକକ ଲବ୍ଧ ଜ୍ଞାନ 196
5.1 ସର୍ଚ୍ଚିଂ(Searching) ଆଲଗୋରିଦମ୍ 196
5.1.1 ଲିନିୟର ସର୍ଚ୍ଚ (Linear Search) 196
5.1.2 ବାଇନାରୀ ସର୍ଚ୍ଚ (Binary Search) 198
5.2 ସର୍ଟିଂ (Sorting) ଆଲଗୋରିଦମ୍ 200
5.2.1 ବବଲ୍ ସର୍ଟ (Bubble sort) 201
5.2.2 ସିଲେକ୍‌ସନ ସର୍ଟ (Selection sort) 206
5.2.3 ଇନ୍‌ସର୍‌ସନ ସର୍ଟ (Insertion Sort) 210
5.3 ସମୀକରଣର ମୂଳ ଅନ୍ୱେଷଣ 213
ୟୁନିଟ ସାରାଂଶ 216
ଅଭ୍ୟାସ 216
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals) 219
ଅଧିକ ଜାଣିବାପାଇଁ 225
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ 225
ଅଧ୍ୟାୟ 6: ପ୍ରକାର୍ଯ୍ୟ **227-259**
ଏକକର ବିଶେଷତ୍ୱ 227
ଭୂମିକା 227
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ 227
ୟୁନିଟ ଲବ୍ଧ ଜ୍ଞାନ 227
6.1 ଉପକ୍ରମ (INTRODUCTION) 228
6.2 ଏକ ପ୍ରକାର୍ଯ୍ୟ କ'ଣ? 228
6.3 ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାରର ଉପଯୋଗୀତା 229
6.4 ପ୍ରକାର୍ଯ୍ୟର ପ୍ରକାରଭେଦ 229
6.5 ସ୍ଥାନୀୟ ଡାଟା ଏବଂ ଗ୍ଲୋବାଲ୍ ଡାଟା ବିଷୟକ ଧାରଣା (Concept of Local Data & Global Data) 230
6.6 ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ପ୍ରକାର୍ଯ୍ୟ (USER-DEFINED FUNCTIONS) 230
6.7 ପ୍ରକାର୍ଯ୍ୟଗୁଡ଼ିକର ଘୋଷଣା ଏବଂ ବର୍ଣ୍ଣନା 231
6.7.1 ଏକ ପ୍ରକାର୍ଯ୍ୟର ଘୋଷଣା 231
6.7.2 ପ୍ରକାର୍ଯ୍ୟର ବର୍ଣ୍ଣନା 232

6.8 ପ୍ରକାର୍ଯ୍ୟ କଲ୍ କରିବା/ଡାକିବା (CALLING A FUNCTION) 234
6.9 ଫେରସ୍ତ ବିବୃତ୍ତି (THE return STATEMENT) 235
6.10 ପ୍ରକାର୍ଯ୍ୟ କୁ ଚର ପ୍ରେରଣ (PASSING ARGUMENTS TO A FUNCTION) 235
6.10.1 ମୂଲ୍ୟଦ୍ୱାରା ଡାକିବା (Call by value) 236
6.10.2 ରେଫରେନ୍ସଦ୍ୱାରା ଡାକିବା (Call by reference) 238
6.10.3 ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ ଏବଂ ରେ ଫରେନ୍ସଦ୍ୱାରା କଲ୍ ମଧ୍ୟରେ ତୁଳନା 241
6.10.4 ଏକ-ପରିସର ଆରେକୁ ଚର ଭାବେ ଗୃହୀତ କରିବା (Passing One-Dimensional Array as Argument) 242
6.10.5 ଦ୍ୱି-ପରିସର ଆରେକୁ ଚର ଭାବେ ଗୃହୀତ କରିବା (Passing Two-Dimensional Array as Argument) 243
6.10.6 ଷ୍ଟ୍ରିଙ୍ଗ ଚର ଭାବେ ଗୃହୀତ (Passing String as Argument) 245
ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ 246
ୟୁନିଟ ସାରାଂଶ 250
ଅନୁଶୀଳନୀ 250
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals) 255
ଅଧିକ ଜାଣିବାପାଇଁ 259
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ 259
ଅଧ୍ୟାୟ 7: ରିକର୍ସନ୍ 261-287
ଏକକ ବିଶେଷତ୍ୱ 261
ଭୂମିକା 261
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ 261
ଏକକ ଲବ୍ଧ ଜ୍ଞାନ 262
7.1 ଉପକ୍ରମ (INTRODUCTION) 262
7.2 ରିକର୍ସିଭ୍ ପ୍ରକାର୍ଯ୍ୟ 263
7.2.1 ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ପ୍ରକାର୍ଯ୍ୟ (Factorial Function) 263
7.2.2 ଫିବୋନାକି (Fibonacci) ସଂଖ୍ୟା ଗୁଡିକ 265
7.2.3 ଏକରମନ୍ (Ackermann) ପ୍ରକାର୍ଯ୍ୟ 266
7.3 କ୍ୱିକ୍ ସର୍ଟ ଆଲଗୋରିଦମ୍ (QUICK SORT ALGORITHM) 267
7.4 ମର୍ଜ ସର୍ଟ ଆଲଗୋରିଦମ୍ (MERGE SORT ALGORITHM) 271
7.5 ରିକର୍ସନ, ଆଇଟରେସନ 275
ୟୁନିଟ ସାରାଂଶ 280
ଅନୁଶୀଳନୀ 280
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals) 287

ଅଧିକ ଜାଣିବାପାଇଁ	287
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ	287
ଅଧ୍ୟାୟ 8: ଷ୍ଟ୍ରକଚର	**289-318**
ଏକକର ବିଶେଷତ୍ୱ	289
ଭୂମିକା	289
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ	289
ଏକକ ଲବ୍ଧ ଜ୍ଞାନ	290
8.1 ଉପକ୍ରମ	290
8.2 ଏକ ଷ୍ଟ୍ରକ୍ଚରର ବ୍ୟାଖ୍ୟା	291
8.3 ଷ୍ଟ୍ରକ୍ଚର ଚଳର ଘୋଷଣା	293
8.4 ଷ୍ଟ୍ରକ୍ଚରର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟକରଣ (INITIALIZING STRUCTURES)	294
8.5 ଷ୍ଟ୍ରକ୍ଚର ଉପାଦାନଗୁଡ଼ିକର ବ୍ୟବହାର (ACCESSING STRUCTURE ELEMENTS)	294
8.6 ଷ୍ଟ୍ରକ୍ଚରର ଆବଣ୍ଟନ (ASSIGNING OF STRUCTURES)	295
8.7 ଷ୍ଟ୍ରକ୍ଚରକୁ ପଢ଼ିବା / ଲେଖିବା (READING/WRITING STRUCTURES)	296
8.8 ଷ୍ଟ୍ରକଚରଗୁଡିକର ଆରେ (ARRAYS OF STRUCTURES)	298
8.8.1 ଷ୍ଟ୍ରକ୍ଚର ଆରେ ର ଉପାଦାନଗୁଡ଼ିକୁ ବ୍ୟବହାର (Accessing Elements of Array of Structures)	298
8.8.2 ଷ୍ଟ୍ରକ୍ଚର ଆରେ ର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ (Initializing Array of Structures)	298
8.9 ଏକ ପ୍ରକାର୍ଯ୍ୟକୁ ଷ୍ଟ୍ରକ୍ଚର ପଠାଇବା (PASSING STRUCTURE TO A FUNCTION)	300
8.10 ପ୍ରକାର୍ଯ୍ୟରୁ ଏକ ଷ୍ଟ୍ରକ୍ଚର ଫେ ରସ୍ତ କରିବା (FUNCTION RETURNING A STRUCTURE)	302
ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ	303
ୟୁନିଟ ସାରାଂଶ	310
ଅନୁଶୀଳନୀ	310
ପ୍ରାକ୍ଟିକାଲ୍	316
ଅଧିକ ଜାଣିବାପାଇଁ	318
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ	318
ଅଧ୍ୟାୟ 9: ପଏଁଟର	**319-342**
ଏକକର ବିଶେଷତ୍ୱ	319
ଭୂମିକା	319
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ	320
ଏକକ ଲବ୍ଧ ଜ୍ଞାନ	320

9.1 ଉପକ୍ରମ (INTRODUCTION) 321
9.2 ପଏଁଟରର ଧାରଣା (IDEA OF POINTERS) 321
9.3 ଚଳର ଠିକଣାରେ ପହଁଞ୍ଚିବା (ACCESSING ADDRESS OF A VARIABLE) 323
9.4 ପଏଁଟରର ଘୋଷଣା (DECLARING A POINTER) 323
9.5 ପଏଁଟରକୁ ଠିକଣା ନ୍ୟସ୍ତ କରିବା (ASSIGNING ADDRESS TO A POINTER) 323
9.6 ପଏଁଟର ବ୍ୟବହାରଦ୍ୱାରା ଚଳକୁ ପହଁଞ୍ଚିବା (ACCESSING VARIABLE USING a POINTER) 324
9.7 ପଏଁଟର ଅଙ୍କଗଣିତ (POINTER ARITHMETIC) 325
9.8 ଫଙ୍କସନ ଚର ଭାବରେ ପଏଁଟର 326
9.9 ପଏଁଟର ଏବଂ ଷ୍ଟ୍ରକ୍‌ଚର 326
9.9.1 ସ୍ୱ-ସୂଚିତ ଷ୍ଟ୍ରକ୍‌ଚର (Self-referential Structures) 327
9.10 ଗତିଶୀଳ ରୂପେ ମେମୋରୀ ଆବଣ୍ଟନ (DYNAMIC MEMORY ALLOCATION) 328
9.10.1 ମେମୋରୀ ଆବଣ୍ଟନ 329
9.10.2 ମେମୋରୀ ଡି-ଆବଣ୍ଟନ/ମୁକ୍ତ କରିବା 330
ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ 331
ୟୁନିଟ ସାରାଂଶ 332
ଅନୁଶୀଳନୀ 333
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals) 337
ଅଧିକ ଜାଣିବାପାଇଁ 342
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ 342
ଅଧ୍ୟାୟ 10: ଫାଇଲ୍ ପରିଚାଳନା 343-372
ଏକକ ବିଶେଷତ୍ୱ 343
ଭୂମିକା 343
ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ 343
ୟୁନିଟ ଲବ୍ଧ ଜ୍ଞାନ 344
10.1 ଉପକ୍ରମ (INTRODUCTION) 344
10.2 ଫାଇଲ ପ୍ରକାର 344
10.3 ଫାଇଲ୍ ପ୍ରକ୍ରିୟାକରଣର ପଦକ୍ଷେପ 345
10.3.1 ଫାଇଲ୍ ଖୋଲିବା 346
10.3.2 ଫାଇଲ୍ ବନ୍ଦ କରିବା 347
10.3.3 ଟେକ୍ସଟ୍ ଫାଇଲ୍ ପଠନ ଏବଂ ଲେଖନ 348
10.3.3.1 ଅକ୍ଷରର ଇନପୁଟ୍ / ଆଉଟପୁଟ୍ ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାର କରି ପଠନ ଏବଂ ଲେଖନ 348

10.3.3.2 ଷ୍ଟ୍ରିଙ୍ଗ୍ I/O ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାରଦ୍ୱାରା ପଠନ ଏବଂ ଲେଖନ 350
10.3.3.3 ସଂରୁପିତ I/O ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାରଦ୍ୱାରା ପଠନ ଏବଂ ଲେଖନ 352
10.3.4 ବାଇନାରୀ ଫାଇଲ୍ ର ପଠନ ଏବଂ ଲେଖନ 354
10.4 ଫାଇଲ୍ ସ୍ଥିତି ପ୍ରକାର୍ଯ୍ୟ (FILE POSITIONING FUNCTIONS) 359
10.4.1 ରିୱାଇଣ୍ଡ ଫାଇଲ୍: (rewind) ପ୍ରକାର୍ଯ୍ୟ (Rewind File: rewind)(Function) 359
10.4.2 ସାମ୍ପ୍ରତିକ ଅବସ୍ଥାନ: ପ୍ରକାର୍ଯ୍ୟ (Current Location: ftell) (Function) 359
10.4.3 ନିର୍ଦ୍ଦିଷ୍ଟ ସ୍ଥାନରେ ରଖିବା: ପ୍ରକାର୍ଯ୍ୟ (Set Position: fseek)(Function) 359
10.5 ଫାଇଲ୍ ଅବସ୍ଥା ପ୍ରକାର୍ଯ୍ୟ (FILE STATUS FUNCTIONS) 360
10.5.1 ଫାଇଲର ଶେଷ ପରୀକ୍ଷା କରିବା: ପ୍ରକାର୍ଯ୍ୟ (Test End of File: feof)(Function) 361
10.5.2 ଫାଇଲର ତ୍ରୁଟି ପରୀକ୍ଷଣ: ପ୍ରକାର୍ଯ୍ୟ (Test Error: ferror), (Function) 361
10.5.3 ତ୍ରୁଟିମୁକ୍ତ କରିବା: (clearerr) ପ୍ରକାର୍ଯ୍ୟ (Clear Error: clearerr), (Function) 361
ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ 362
ୟୁନିଟ ସାରାଂଶ 365
ଅନୁଶୀଳନୀ 365
ପ୍ରାକ୍ଟିକାଲ୍ (Practicals) 368
ଅଧିକ ଜାଣିବାପାଇଁ 372
ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚୀ 372
ଅଧିକ ଶିକ୍ଷାପାଇଁ ସହାୟକ 372
CO ଏବଂ PO ପ୍ରାପ୍ତିପାଇଁ ଟେବୁଲ 373
ସୂଚକାଙ୍କ 374
ପରିଭାଷା 382

1 ପ୍ରୋଗ୍ରାମିଂ ପରିଚୟ

ଏକକ ବିଶେଷତ୍ୱ

ଏହି ଏକକରେ କମ୍ପ୍ୟୁଟର (computer) ଏବଂ ଏହାର ଅଂଗଗୁଡ଼ିକର ପରିଚିତି, ସଫ୍ଟୱେୟାର (software) ଏବଂ ଏହାର ପ୍ରକାରଭେଦ, ପାଠ୍ୟକ୍ରମ ସହିତ ଜଡ଼ିତ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ସଫ୍ଟୱେୟାର ସାଧନସମୂହ, ଆଲଗୋରିଦମର (algorithm) ବିକାଶ, ଆଲଗୋରିଦମ୍‌କୁ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଅନୁବାଦ କରିବା, ଏବଂ C ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବା, ସମ୍ପାଦନା, ସଂକଳନ ଏବଂ ଚାଳନ କରିବା ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ପଦକ୍ଷେପ ବିଷୟରେ ଆଲୋଚନା କରାଯାଇଛି ଏବଂ C ଭାଷାର ମୌଳିକ ତତ୍ତ୍ୱ ଉପାଦାନଗୁଡ଼ିକର ଉପସ୍ଥାପନ କରାଯାଇଅଛି ।

ଭୂମିକା

ଆମେ ସମସ୍ତେ ଏବେ ଡିଜିଟାଲ୍ ଯୁଗରେ ବାସ କରୁଅଛୁ । ତେଣୁ କମ୍ପ୍ୟୁଟର ଏବଂ ଏହାର କାରିଗରି କୌଶଳ ଦକ୍ଷତାର ସହିତ ବ୍ୟବହାର କରିବାପାଇଁ ସମସ୍ତଙ୍କର ଜ୍ଞାନ ଏବଂ ସାମର୍ଥ୍ୟ ସଶକ୍ତ କରିବାର ଆବଶ୍ୟକତା ରହିଛି । ଆମର ନିତିଦିନିଆ ସମସ୍ୟାଗୁଡ଼ିକପାଇଁ ଇଞ୍ଜିନିୟରମାନେ ଏହାର ହାର୍ଡୱେୟାର ଓ ସଫ୍ଟୱେୟାର ସମାଧାନର ସିଦ୍ଧାନ୍ତ, ଡିଜାଇନ୍ ଏବଂ ବିକାଶ କରି ପ୍ରୟୋଗ କରିଥା'ନ୍ତି ।

ଏହି ପ୍ରାଥମିକ ୟୁନିଟ ଶିକ୍ଷାର୍ଥୀମାନଙ୍କୁ ଏକ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମ୍ ଏବଂ ଏହାର ସହଯୋଗୀ ଯନ୍ତ୍ରପାତିଗୁଡ଼ିକର ଅଂଗଗୁଡିକୁ ବୁଝିବାରେ ସାହାଯ୍ୟ କରେ । ସେମାନେ କିପରି ଏକ କାର୍ଯ୍ୟ ହାସଲପାଇଁ ପରସ୍ପର ସହିତ ଯୋଗାଯୋଗ କରେ; ତାର୍କିକ ଏବଂ ସାଂଖ୍ୟକ ସମସ୍ୟାଗଡ଼ିକର ସମାଧାନପାଇଁ ଆଲଗୋରିଦମର ବିକାଶ; ଏବଂ C ଭାଷା ବ୍ୟବହାର କରି ଏକ ଆଲଗୋରିଦମକୁ ଚାଳନ ପ୍ରୋଗ୍ରାମ୍‌ରେ ରୂପାନ୍ତର କରିବାରେ ବିଭିନ୍ନ ପର୍ଯ୍ୟାୟ ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- କମ୍ପ୍ୟୁଟରର କାର୍ଯ୍ୟ
- ମୌଳିକ ଗଣିତ
- ତାର୍କିକ ବିଚାର
- ଯୋଗାଯୋଗ କୌଶଳ

ଏକକ ଲବ୍ଧ ଜ୍ଞାନ

ଏକକ ସମାପ୍ତ ହେବା ପରେ, ବିଦ୍ୟାର୍ଥୀମାନେ ନିମ୍ନଲିଖିତ ବିଷୟଗୁଡ଼ିକରେ ଜ୍ଞାନ ଆହରଣରେ ସମର୍ଥ ହେବେ।

U1-O1 : ସରଳ ଗାଣିତିକ ଏବଂ ତାର୍କିକ ସମସ୍ୟାଗଡ଼ିକପାଇଁ ଆଲଗୋରିଦମ୍ ପ୍ରସ୍ତୁତ କରିବା।

U1-O2 : ଆଲଗୋରିଦମକୁ C ପ୍ରୋଗ୍ରାମ୍‌ରେ ଅନୁବାଦ କରିବା।

U1-O3 : ଦିଆଯାଇଥିବା ପ୍ଲାଟଫର୍ମ ବା ବିକାଶ ପରିବେଶରେ ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ଲେଖିବା, ସଂକଳନ, ଏବଂ ଚାଳନା କରିବା।

U1-O4 : ଏକ ପ୍ରୋଗ୍ରାମ୍‌ର ପରୀକ୍ଷଣ ଓ ତ୍ରୁଟିମୁକ୍ତ କରିବା କୌଶଳର ପ୍ରଦର୍ଶନ।

ୟୁନିଟ-1 ଫଳାଫଳ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ *(1 – ଦୁର୍ବଳ ସମ୍ପର୍କ; 2 – ମଧ୍ୟମ ସମ୍ପର୍କ; 3 – ଦୃଢ ସମ୍ପର୍କ)*							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U1-O1	3	–	–	–	–	–	–	–
U1-O2	–	3	–	–	–	–	–	–
U1-O3	–	–	3	–	–	–	–	–
U1-O4	–	–	3	–	–	–	–	–

1.1 କମ୍ପ୍ୟୁଟରର ପରିଚୟ (INTRODUCTION TO COMPUTERS)

କମ୍ପ୍ୟୁଟର (computer) ହେଉଛି ଏକ ବୈଦ୍ୟୁତିକ ଯନ୍ତ୍ର, ଏହା ପ୍ରୋଗ୍ରାମ କୁହାଯାଉଥିବା ନିର୍ଦ୍ଦେଶାବଳିର ଏକ ସେଟ୍ ଅନୁଯାୟୀ କାର୍ଯ୍ୟ କରିଥାଏ।

1940 ମସିହା ପ୍ରବର୍ତ୍ତିତ ପ୍ରଥମ ସମ୍ପୂର୍ଣ୍ଣ ବୈଦ୍ୟୁତିକ କମ୍ପ୍ୟୁଟରଗୁଡ଼ିକ ବିରାଟକାୟ ଯନ୍ତ୍ର ଥିଲା। ଏଥିରେ କାର୍ଯ୍ୟ କରିବାକୁ କିଛି ଲୋକ ଆବଶ୍ୟକ କରୁଥିଲା। ସେହି ପ୍ରାରମ୍ଭିକ ଯନ୍ତ୍ର ତୁଳନାରେ, ଆଜିର କମ୍ପ୍ୟୁଟରଗୁଡ଼ିକ ଆଶ୍ଚର୍ଯ୍ୟଜନକ - ଏଗୁଡ଼ିକ ହଜାର ଗୁଣ ଦ୍ରୁତ ଏହା ତୁମର ଡେସ୍କରେ, କୋଳରେ, କିମ୍ବା ପକେଟରେ ମଧ୍ୟ ଫିଟ୍ ହୋଇ ପାରିବ।

ସାଧାରଣତଃ ଗୋଟିଏ କମ୍ପ୍ୟୁଟର ନିମ୍ନୋକ୍ତ କାର୍ଯ୍ୟଗୁଡ଼ିକପାଇଁ ସମଥ –

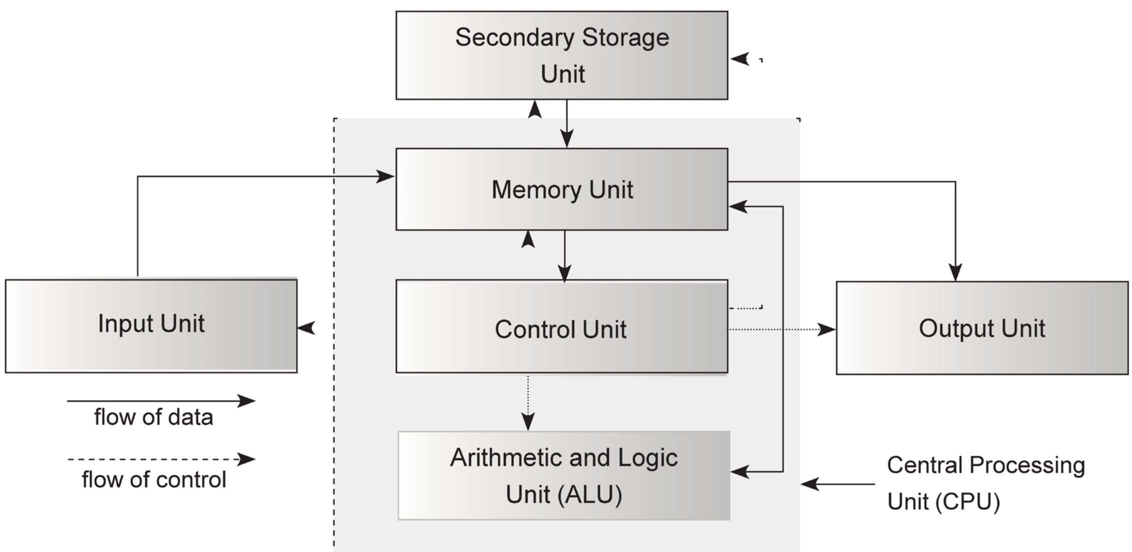
- ବିଭିନ୍ନ ରୂପରେ ଡାଟା (data) ଏବଂ ନିର୍ଦ୍ଦେଶାବଳି ଗ୍ରହଣ କରିବା,
- ଏହାର ମେମୋରି (memory) ରେ ଡାଟା ଏବଂ ନିର୍ଦ୍ଦେଶାବଳି ଗଚ୍ଛିତ କରିବା,
- ମାଧ୍ୟମିକ ଭଣ୍ଡାରରୁ ପୂର୍ବ ସଂଗୃହୀତ ଡାଟା ପୁନଃପ୍ରାପ୍ତ କରିବା,
- ନିର୍ଦ୍ଦିଷ୍ଟ ନିର୍ଦ୍ଦେଶାବଳି ଅନୁଯାୟୀ ଗାଣିତିକ ଏବଂ ତାର୍କିକ କାର୍ଯ୍ୟଗଡ଼ିକ ଅତ୍ୟନ୍ତ ତୀବ୍ର ବେଗରେ ଏବଂ ଅଧିକ ଠିକ୍ ଭାବେ ସମ୍ପାଦନ କରିବା,

- ଫଳାଫଳଗୁଡ଼ିକୁ ଅନେକ ରୂପରେ ପ୍ରସ୍ତୁତ କରିବା, ଏବଂ
- ଅନ୍ୟ କମ୍ପ୍ୟୁଟରଗୁଡ଼ିକ ସହ ଯୋଗାଯୋଗ କରିବା ।

1.1.1 କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ଅଂଗସମୂହ (Components of a Computer System)

ଗୋଟିଏ କମ୍ପ୍ୟୁଟର ହେଉଛି ହାର୍ଡୱେର (hardware) ଏବଂ ସଫ୍ଟୱେୟାରର (software) ସମାବେଶ । ହାର୍ଡୱେୟାର ଏକ କମ୍ପ୍ୟୁଟରର ଶାରୀରିକ ଅଂଗ ଯେପରିକି ମଦରବୋର୍ଡ (motherboard), ମେମୋରି ଯନ୍ତ୍ରାଂଶ (memory devices), ମନିଟର୍ (monitor), କୀବୋର୍ଡ୍ (keyboard) ଇତ୍ୟାଦିର ପ୍ରତିନିଧିତ୍ୱ କରେ । ଏହା ସଫ୍ଟୱେୟାର ଏକ ପ୍ରୋଗ୍ରାମ୍ କିମ୍ବା ନିର୍ଦ୍ଦେଶର ଏକ ସେଟ୍‌କୁ ପ୍ରତିନିଧିତ୍ୱ କରେ । ଉଭୟ ହାର୍ଡୱେୟାର ଏବଂ ସଫ୍ଟୱେୟାର ଏକତ୍ର ହୋଇ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମ୍‌କୁ କାର୍ଯ୍ୟକ୍ଷମ କରିଥାଏ ।

କମ୍ପ୍ୟୁଟରକୁ ଦିଆଯାଇଥିବା ପ୍ରତ୍ୟେକ କାର୍ଯ୍ୟ ଏକ ଇନପୁଟ୍-ପ୍ରୋସେସ-ଆଉଟପୁଟ୍ ଚକ୍ର (IPO cycle) ଅନୁସରଣ କରେ । କମ୍ପ୍ୟୁଟର କେତେକ ଇନପୁଟ୍ ଆବଶ୍ୟକ କରେ, ଇନପୁଟ୍‌ର ପ୍ରୋସେସ କରି ଇପ୍‌ସିତ ଆଉଟପୁଟ୍ ଦେଇଥାଏ । ଇନପୁଟ୍ ୟୁନିଟ ଇନପୁଟ୍ ନେଇଥାଏ, କେନ୍ଦ୍ରୀୟ ପ୍ରକ୍ରିୟାକରଣ ୟୁନିଟ (central processing unit) ଡାଟା ପ୍ରୋସେସ କରେ ଏବଂ ଆଉଟପୁଟ୍ ୟୁନିଟ ଆଉଟପୁଟ୍ ପ୍ରସ୍ତୁତ କରେ । ପ୍ରକ୍ରିୟାକରଣ ସମୟରେ ମେମୋରି ୟୁନିଟ ଡାଟା ଏବଂ ନିର୍ଦ୍ଦେଶାବଳିଗୁଡ଼ିକୁ ଧାରଣ କରେ ।



ଚିତ୍ର 1.1: ଏକ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ଚାଳନ ଅଂଗସମୂହ

ଆସ କମ୍ପ୍ୟୁଟରର ଗୋଟେ-ଗୋଟେ କରି ଚାଳନ ଅଂଗଗୁଡ଼ିକ ଉପରେ ନଜର ପକାଇବା ।

ଇନପୁଟ୍ ୟୁନିଟ (Input Unit)

(a) ସାଧାରଣ କୀବୋର୍ଡ୍ ସହ QWERTY ଲେଆଉଟ୍

(b) ୱେୟାରଲେସ୍ କୀବୋର୍ଡ୍

(c) ମାଉସ୍

(d) ମାଇକ୍ରୋଫୋନ୍

(e) ୱେବ୍ କ୍ୟାମେରା

***ଚିତ୍ର 1.2:** ସାଧାରଣ ଇନପୁଟ୍ ଯନ୍ତ୍ରାଂଶ*

ଇନପୁଟ୍ ୟୁନିଟ ବିଭିନ୍ନ ଇନପୁଟ୍ ଯନ୍ତ୍ରାଂଶକୁ ନେଇ ଗଠିତ ଯାହା ଏକ କମ୍ପ୍ୟୁଟର ସହିତ ସଂଯୋଗ ହୋଇପାରିବ, ଯେପରିକି କୀବୋର୍ଡ, ମାଉସ୍, ଆଲୁଅ କଲମ (light pen), ମାଇକ୍ରୋଫୋନ୍ ଇତ୍ୟାଦି । ଷ୍ଟାଣ୍ଡାର୍ଡ ଇନପୁଟ୍ ଯନ୍ତ୍ରାଂଶ ହେଉଛି କୀବୋର୍ଡ, ଯାହା ପ୍ରତ୍ୟେକ ନୂତନ କମ୍ପ୍ୟୁଟର ସହିତ ଆସିଥାଏ । ଯେକୌଣସି ସମୟରେ, ଏକାଧିକ ଇନପୁଟ୍ ଯନ୍ତ୍ରାଂଶ ଏକ କମ୍ପ୍ୟୁଟରକୁ ସଂଯୋଜିତ ହୋଇପାରିବ ।

ଇନପୁଟ୍ ୟୁନିଟର ଉଦ୍ଦେଶ୍ୟ ହେଉଛି ବାହ୍ୟ ଜଗତରୁ ସୂଚନା ଗ୍ରହଣ କରିବା, ଏହାକୁ ବାଇନାରି (binary) ରୂପରେ ରୂପାନ୍ତର କରିବା, ଏବଂ ତା'ପରେ ଏହାକୁ ମୁଖ୍ୟ ମେମୋରିକୁ ସ୍ଥାନାନ୍ତର କରିବା । ଏହା କିଛି ପ୍ରୋଗ୍ରାମ୍‌ଦ୍ୱାରା ପ୍ରକ୍ରିୟାକରଣ କରିବାକୁ ଡାଟା ଇନ୍‌ପୁଟ କରିବା, ଭାଷଣ ରେକର୍ଡିଂ, ଏକ ଦୃଶ୍ୟ କିମ୍ବା ଫଟୋଗ୍ରାଫ୍ ଉଠାଇବା, କିମ୍ବା ଡିସ୍‌ପ୍ଲେ (display)ରୁ କୌଣସି ବସ୍ତୁ ସିଦ୍ଧାନ୍ତ କରିବା ଅନ୍ତର୍ଭୁକ୍ତ ହୋଇପାରେ ।

ଆଉଟପୁଟ୍ ୟୁନିଟ (Output Unit)

ଆଉଟପୁଟ୍ ୟୁନିଟ ବିଭିନ୍ନ ଆଉଟପୁଟ୍ ଯନ୍ତ୍ରାଂଶକୁ ନେଇ ଗଠିତ । ଏହା ଗୋଟିଏ କମ୍ପ୍ୟୁଟରକୁ ସଂଯୋଗ ହୋଇପାରିବ, ଯେପରିକି ଭିଜୁଆଲ୍ ଡିସ୍‌ପ୍ଲେ ୟୁନିଟ (VDU) କିମ୍ବା ସରଳ ଭାବେ ମନିଟର୍, ପ୍ରିଣ୍ଟର୍, ପ୍ଲଟର, ଅଡିଓ ସିଷ୍ଟମ୍ ଇତ୍ୟାଦି । ଷ୍ଟାଣ୍ଡାର୍ଡ ଆଉଟପୁଟ୍ ଯନ୍ତ୍ରାଂଶ ହେଉଛି ଏକ ମନିଟର୍ । ଏହା ପ୍ରତ୍ୟେକ ନୂତନ କମ୍ପ୍ୟୁଟର ସହିତ ଆସିଥାଏ । ଯେକୌଣସି ସମୟରେ, ଇନପୁଟ୍ ଯନ୍ତ୍ରାଂଶ ପରି, ଏକାଧିକ ଆଉଟପୁଟ୍ ଯନ୍ତ୍ରାଂଶ ଏକ କମ୍ପ୍ୟୁଟରକୁ ସଂଯୋଗ ହୋଇପାରିବ ।

ଆଉଟପୁଟ୍ ୟୁନିଟର ଉଦ୍ଦେଶ୍ୟ ହେଉଛି ମୁଖ୍ୟ ମେମୋରି ୟୁନିଟରୁ ବାଇନାରି ଫର୍ମରେ ସୂଚନା ଗ୍ରହଣ କରିବା । ଏହାକୁ ଜଣେ ଲୋକ ବୁଝିବାଯୋଗ୍ୟ ସୂଚନାରେ ପରିଣତ କରିବା । ତା'ପରେ ଏହାକୁ ଆଉଟପୁଟ୍ କରିବା । ଏହି

ଆଉଟପୁଟ୍ କମ୍ପ୍ୟୁଟରର ପ୍ରୋଗ୍ରାମ୍‌ର ଫଳାଫଳର ପ୍ରଦର୍ଶନ, ପ୍ରିଣ୍ଟିଂ ହୋଇପାରେ ବା ସଙ୍ଗୀତ ବଜାଇବା କିମ୍ବା ଚଳଚ୍ଚିତ୍ର ଦେଖାଇବା ମଧ୍ୟ ହୋଇପାରେ ।

(a) ମନିଟର୍

(b) ପ୍ରିଣ୍ଟର୍

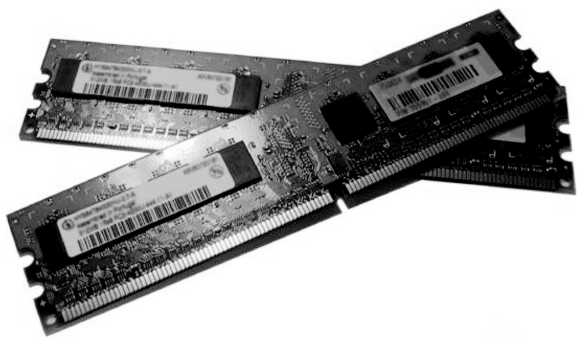
(c) ପ୍ଲଟର

(d) ବକ୍ତା

ଚିତ୍ର 1.3: ସାଧାରଣ ଆଉଟପୁଟ୍ ଯନ୍ତ୍ରାଂଶ

ମେମୋରି ୟୁନିଟ Memory Unit

ମେମୋରି ୟୁନିଟ ଏକ କମ୍ପ୍ୟୁଟରର ଭଣ୍ଡାର ଭାବରେ କାର୍ଯ୍ୟ କରେ । ଏଠାରେ ପ୍ରୋଗ୍ରାମ୍‌ର ଚାଳନ, ଇନପୁଟ୍ ଡାଟା, କମ୍ପ୍ୟୁଟେସନ୍/ଗଣନାର ମଧ୍ୟବର୍ତ୍ତୀ ଏବଂ ଅନ୍ତିମ ଫଳ ଇତ୍ୟାଦି ଗଚ୍ଛିତ ରଖାଯାଏ । ଗୋଟିଏ କମ୍ପ୍ୟୁଟରରେ ସୂଚନା ଭର୍ତ୍ତି କଲାବେଳେ ପ୍ରଥମେ ଏହା ମେମୋରି ୟୁନିଟରେ ଗଚ୍ଛିତ ହୋଇଥାଏ । ସେହିଭଳି, ଡାଟା କିମ୍ବା ଫଳାଫଳ ଗୁଡ଼ିକ ମେମୋରି ୟୁନିଟରୁ ନିଆଯାଇଥାଏ । ମେମୋରି ୟୁନିଟ ଏକ ଉଦ୍‌ବାୟୀ ମେମୋରିର ପ୍ରତିନିଧିତ୍ୱ କରେ । ଏହାର ଅର୍ଥ ହେଉଛି ବିଦ୍ୟୁତ ସ୍ରୋତ ବନ୍ଦ ହେବା ମାତ୍ରେ ଗଚ୍ଛିତ ଥିବା ସୂଚନା ମେମୋରି ୟୁନିଟରୁ ଲିଭିଯାଇଥାଏ ।



ଚିତ୍ର 1.4: ମେମୋରି ଉପକରଣ

ଗଣିତ ଏବଂ ଲଜିକ୍ ୟୁନିଟ (Arithmetic and Logic Unit)

ଗଣିତ ଏବଂ ଲଜିକ୍ ୟୁନିଟ (ALU), ଗାଣିତିକ ଏବଂ ତାର୍କିକ କାର୍ଯ୍ୟ କରେ । ଗାଣିତିକ କାର୍ଯ୍ୟରେ ଯୋଗ, ବିଯୋଗ, ଗୁଣନ, ଏବଂ ବିଭାଜନ ଅନ୍ତର୍ଭୁକ୍ତ । ଦୁଇଟି ଡାଟା ଆଇଟମ୍‌କୁ ତୁଳନା କରିବାକୁ ତାର୍କିକ ସଞ୍ଚାଳନଗୁଡ଼ିକ ବ୍ୟବହୃତ ହୁଏ, ଯେପରିକି, ଠାରୁ କମ୍ (<), ଠାରୁ କମ୍ କିମ୍ବା ସମାନ (≤), ଠାରୁ ଅଧିକ (>),ଠାରୁ ଅଧିକ କିମ୍ବା ସମାନ (≥),ସହିତ ସମାନ (=), ଏବଂ ସହିତ ସମାନ ନୁହେଁ (≠) ।

ନିୟନ୍ତ୍ରଣ ୟୁନିଟ (Control Unit)

ନିୟନ୍ତ୍ରଣ ୟୁନିଟ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ଅନ୍ୟ ସମସ୍ତ ଅଂଶର ସମନ୍ୱୟ ରକ୍ଷା କରେ ଏବଂ ନିୟନ୍ତ୍ରଣ କରେ । ଏକ ପ୍ରୋଗ୍ରାମ୍‌ର ନିର୍ଦ୍ଦେଶରେ, କଣ୍ଟ୍ରୋଲ୍ ୟୁନିଟ ଚାରୋଟି ମୌଳିକ କାର୍ଯ୍ୟ ସମ୍ପାଦନ କରେ:

- ଫେଚ୍ (Fetch) - କମ୍ପ୍ୟୁଟରର ମେମୋରିରୁ ପରବର୍ତ୍ତୀ ପ୍ରୋଗ୍ରାମ୍ ନିର୍ଦ୍ଦେଶାବଳିକୁ ଆଣିଥାଏ ।
- ଡିକୋଡ୍ (Decode)- ପ୍ରୋଗ୍ରାମ୍‌ଟି କମ୍ପ୍ୟୁଟରକୁ କ'ଣ କରିବାକୁ କହୁଛି ତାହା ବୁଝିଥାଏ?
- ଚାଳନ (Execute)- ଅନୁରୋଧ କରାଯାଇଥିବା କାର୍ଯ୍ୟର ସମ୍ପାଦନ କରିଥାଏ, ଯେପରିକି ଧରାଯାଉ ଦୁଇଟି ସଂଖ୍ୟାକୁ ଯୋଡିବା କିମ୍ବା ସେଗୁଡ଼ିକ ମଧ୍ୟରୁ ଗୋଟିଏ ବଡ଼ କି ନାହିଁ ତାହା ସ୍ଥିର କରିବା ।
- ପୁନଃଲିଖନ (Write-Back)- କାର୍ଯ୍ୟ ସମ୍ପାଦନ ପରେ ଫଳକୁ ମେମୋରିରେ ପୁଣିଥରେ ଲେଖିଥାଏ ।

ଏହି ଚାରି-ଷ୍ଟେପ୍ ପ୍ରକ୍ରିୟାକୁ ମେସିନ୍ ସାଇକେଲ୍ ବା ପ୍ରୋସେସିଙ୍ଗ ସାଇକେଲ୍ କୁହାଯାଏ । ଏହା ଦୁଇଟି ପର୍ଯ୍ୟାୟକୁ ନେଇ ଗଠିତ: ନିର୍ଦ୍ଦେଶ ଚକ୍ର (ଫେଚ୍ ଏବଂ ଡିକୋଡ୍) ଏବଂ ଚାଳନ ଚକ୍ର (ଚାଳନ ଏବଂ ପୁନଃଲିଖନ).

ମାଧ୍ୟମିକ ଭଣ୍ଡାରଣ ୟୁନିଟ (Secondary Storage Unit)

ମାଧ୍ୟମିକ ଭଣ୍ଡାରଣ ୟୁନିଟ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଡାଟାର ଭବିଷ୍ୟତ ବ୍ୟବହାରପାଇଁ ଦୀର୍ଘକାଳୀନ ଭଣ୍ଡାର ପ୍ରଦାନ କରିଥାଏ । ଡାଟା, ମେମୋରି ୟୁନିଟରୁ ମାଧ୍ୟମିକ ଭଣ୍ଡାର ୟୁନିଟକୁ ସ୍ଥାନାନ୍ତରିତ ହୋଇଥାଏ । ମାଧ୍ୟମିକ ଭଣ୍ଡାରରେ ରହୁଥିବା ଡାଟାର ପ୍ରକ୍ରିୟାକରଣ କରିବାପାଇଁ, ଏହା ମେମୋରିକୁ ସ୍ଥାନାନ୍ତରିତ ହୋଇଥାଏ । ଦୟାକରି ମନେ ରଖ ଯେ ମାଧ୍ୟମିକ ଭଣ୍ଡାରରେ ଗଚ୍ଛିତ ହୋଇଥିବା ଡାଟାର ସିଧାସଳଖ ପ୍ରକ୍ରିୟାକରଣ ମାଧ୍ୟମିକ ଭଣ୍ଡାରରେ କରାଯାଇପାରିବ ନାହିଁ । କେବଳ ମେମୋରିରେ ଥିବା ଡାଟାର ପ୍ରକ୍ରିୟାକରଣ ହୋଇପାରିବ ।

(a) ହାର୍ଡ ଡିସ୍କ

(b) ସିଡି/ଡିଭିଡି

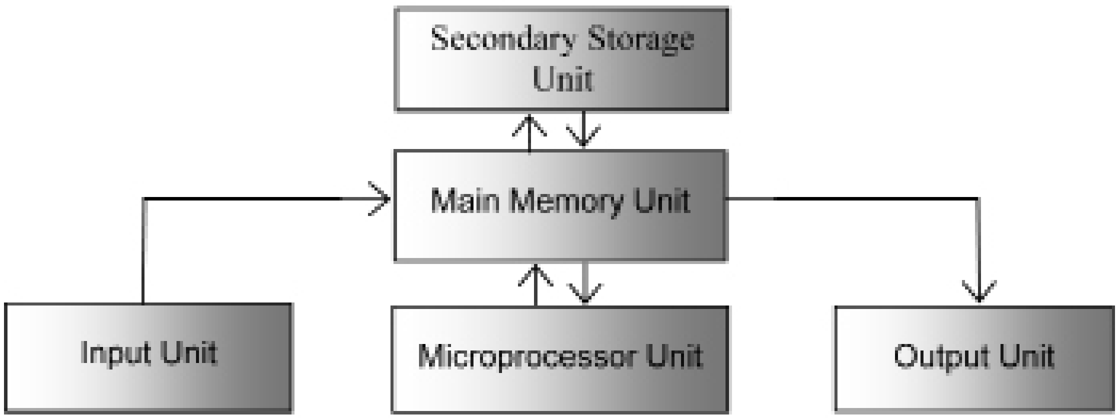
(c) ପେନ୍ ଡ୍ରାଇଭ୍

ଚିତ୍ର 1.5: *ସାଧାରଣ ଭଣ୍ଡାର ଯନ୍ତ୍ରାଂଶସମୂହ*

- ନିୟନ୍ତ୍ରଣ ୟୁନିଟ, ଗଣିତ ଏବଂ ଲଜିକ୍ ୟୁନିଟ, ଏବଂ ମୁଖ୍ୟ ମେମୋରି ୟୁନିଟ, ସାମୂହିକ ଭାବରେ, କେନ୍ଦ୍ରୀୟ ପ୍ରକ୍ରିୟାକରଣ ୟୁନିଟ (Central Processing Unit) କୁହାଯାଏ ।
- ସମସ୍ତ ବାହ୍ୟ ଉପକରଣ, ଯେପରିକି ଇନପୁଟ୍ ଉପକରଣ ଏବଂ ଆଉଟପୁଟ୍ ଉପକରଣ, ଯାହା କମ୍ପ୍ୟୁଟରସହିତ ସଂଯୁକ୍ତ, ସାଧାରଣତଃ ପେରିଫେରାଲ୍ ଉପକରଣ କୁହାଯାଏ ।

ବସ୍ତୁ ଏବଂ ମାଇକ୍ରୋଇଲେକ୍ଟ୍ରୋନିକ୍ସ (microelectronics) କ୍ଷେତ୍ରରେ ଅଗ୍ରଗତି ଫଳରେ, ନିୟନ୍ତ୍ରଣ ୟୁନିଟ, ଗଣିତ ଏବଂ ତାର୍କିକ ୟୁନିଟ, ଏବଂ କିଛି ସୀମିତ ଦ୍ରୁତ ମେମୋରି ଉପକରଣ ରେଜିଷ୍ଟରକୁ (register) ଏକ ଭେରି ଲାର୍ଜ ସ୍କେଲ ଇଣ୍ଟିଗ୍ରେସନ (VLSI) ଚିପ୍ ଭିତରେ ଏମ୍ବେଡ୍ (embed) କରିବା ସମ୍ଭବ ହୋଇଛି ଯାହାକୁ ମାଇକ୍ରୋପ୍ରୋସେସର (microprocessor) କୁହାଯାଏ ।

ଔପଚାରିକ ଭାବରେ, ଏକ ମାଇକ୍ରୋପ୍ରୋସେସର ବା ସାଧାରଣ ଭାବେ ପ୍ରୋସେସର, ସାଧାରଣ-ଉଦ୍ଦେଶ୍ୟରେ ଏକ ପ୍ରୋଗ୍ରାମଯୋଗ୍ୟ ଉପକରଣ, ଯାହା ବାଇନାରି ଫର୍ମରେ ସୂଚନା ଗ୍ରହଣ କରି, ଏହାକୁ ପ୍ରକ୍ରିୟା କରିପାରେ ଏବଂ ଫଳାଫଳ ପଠାଇପାରେ । ନିଜେ ବୁଝିପାରୁଥିବା ନିର୍ଦ୍ଦେଶାବଳିଗୁଡ଼ିକୁ ବ୍ୟବହାର କରି ବିଭିନ୍ନ ପ୍ରକାରର ସମସ୍ୟାର ସମାଧାନପାଇଁ ଏହାକୁ ପ୍ରୋଗ୍ରାମ୍ କରାଯାଇପାରିବ । ଏକ ପ୍ରୋସେସର୍ ବୁଝିପାରୁଥିବା ସମସ୍ତ ନିର୍ଦ୍ଦେଶାବଳିର ସେଟ୍ ହେଉଛି ନିର୍ଦ୍ଦେଶ ସେଟ୍ (instruction set) ।



ଚିତ୍ର 1.6: ଏକ ବ୍ୟକ୍ତିଗତ କମ୍ପ୍ୟୁଟରର ଚାଳନର ଚିତ୍ର

ସାଧାରଣ ଡିଜିଟାଲ୍ କମ୍ପ୍ୟୁଟର କ୍ଷେତ୍ରରେ ଅନ୍ୟ ୟୁନିଟ ଗୁଡ଼ିକର ଭୂମିକା ସମାନ । ଲାପଟପ୍ ସମେତ ବର୍ତ୍ତମାନର ଅଧିକାଂଶ ପର୍ସନାଲ୍ କମ୍ପ୍ୟୁଟର୍ (PC) ଇଣ୍ଟେଲ (Intel) ବା ଏଏମ୍‌ଡି (AMD) ଆଧାରିତ ମାଇକ୍ରୋପ୍ରୋସେସର୍ ଦ୍ୱାରା ନିର୍ମିତ ।

1.1.2 ହାର୍ଡୱେୟାର (Hardware)

ହାର୍ଡୱେୟାର ଏକ କମ୍ପ୍ୟୁଟରର ଅଂଶକୁ ବୁଝାଏ ଯାହାକୁ ତୁମେ ବାକ୍ସ ଓ ଏହା ଭିତରେ ଥିବା ସମସ୍ତ ଜିନିଷ ଦେଖିପାରିବ ଓ ସ୍ପର୍ଶ କରିପାରିବ । କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ବିଭିନ୍ନ ଉପାଦାନ ଯାହା ଏହାର ହାର୍ଡୱେୟାର ଗଠନ କରେ ସେଥିରେ ସିଷ୍ଟମ୍ ୟୁନିଟ (ଯେଉଁଥିରେ ପ୍ରମୁଖ ହାର୍ଡୱେୟାର ଉପାଦାନ ଗୁଡ଼ିକ ଅଛି ଯେପରିକି ମଦରବୋର୍ଡ, ଶକ୍ତି ଯୋଗାଣ ୟୁନିଟ, ହାର୍ଡ ଡିସ୍କ, ଇତ୍ୟାଦି), ଇନପୁଟ୍ /ଆଉଟପୁଟ୍ ଉପକରଣ ସହିତ ଭଣ୍ଡାର ଉପକରଣ ଅନ୍ତର୍ଭୁକ୍ତ ।

1.1.3 ସଫ୍ଟୱେୟାର (Software)

ସାଧାରଣ ଅର୍ଥରେ ସଫ୍ଟୱେୟାର ଏକ ନିର୍ଦ୍ଦେଶାବଳି କିମ୍ବା ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକର ଏକ ସେଟ୍ । ଏହା ହାର୍ଡୱେୟାରକୁ କ'ଣ କରିବାକୁ ହେବ ତାହା ନିର୍ଦ୍ଦେଶ ଦେଇଥାଏ ।

ସଫ୍ଟୱେୟାରକୁ ବର୍ଣ୍ଣନା କରିବା କଷ୍ଟକର ହୋଇପାରେ କାରଣ ଏହା ହେଉଛି ଭର୍ଚୁଆଲ୍ (virtual) ଏବଂ କମ୍ପ୍ୟୁଟର ହାର୍ଡୱେର ପରି ଭୌତିକ ନୁହେଁ । ଏହା ପରିବର୍ତ୍ତେ, ସଫ୍ଟୱେୟାର ହେଉଛି କମ୍ପ୍ୟୁଟର ପ୍ରୋଗ୍ରାମର୍‌ମାନଙ୍କଦ୍ୱାରା ଲିଖିତ

କୋଡ୍‌ର (ନିର୍ଦ୍ଦେଶାବଳି) ଧାଡ଼ି ଗୁଡ଼ିକୁ ନେଇ ଗଠିତ ଯାହାଙ୍କ ଗୋଟିଏ କମ୍ପ୍ୟୁଟର ପ୍ରୋଗ୍ରାମ୍‌ରେ ସଂକଳିତ ହୋଇଥାଏ। ସଫ୍ଟୱେୟାର ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ବାଇନାରି ଡାଟା ଭାବରେ ଗଚ୍ଛିତ ହୋଇଥାଏ। ଏହା ସଂସ୍ଥାପିତ (install) ହେବା ସମୟରେ ଏହା ଏକ କମ୍ପ୍ୟୁଟର୍‌ର ହାର୍ଡ ଡ୍ରାଇଭ୍‌ରେ କପି ହୋଇଥାଏ। ସଫ୍ଟୱେୟାର ଭର୍ଚୁଆଲ୍ ହୋଇଥିବାରୁ ଓ କୌଣସି ଭୌତିକ ସ୍ଥାନ ଗ୍ରହଣ କରୁନଥିବାରୁ କମ୍ପ୍ୟୁଟର ହାର୍ଡୱେୟାର ଅପେକ୍ଷା ଏହାକୁ ଉନ୍ନତତର କରିବା ବହୁତ ସହଜ (ଏବଂ ପ୍ରାୟତଃ ଶସ୍ତା)।

ସଫ୍ଟୱେୟାର୍‌ର ସବୁଠାରୁ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଉଦାହରଣ ହେଉଛି ଏକ ଅପରେଟିଂ ସିଷ୍ଟମ୍ (OS) ଏହା ଏକ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ସାଧନଗୁଡ଼ିକୁ ପରିଚାଳନା କରେ, ଏବଂ ଏକ ଇଣ୍ଟରଫେସ୍ (interface) ପ୍ରଦାନ କରେ। ଏହାର ବ୍ୟବହାର କରି ବ୍ୟବହାରକାରୀ ବିଭିନ୍ନ କାର୍ଯ୍ୟ ସମ୍ପାଦନ କରିବାକୁ କମ୍ପ୍ୟୁଟର ସହିତ ପାରସ୍ପରିକ କ୍ରିୟା (interact) କରିପାରିବେ।

ଏହା ପ୍ରଣିଧାନଯୋଗ୍ୟ ଯେ ହାର୍ଡୱେର ଏବଂ ସଫ୍ଟୱେୟାର ଏକ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ଅବିଚ୍ଛେଦ୍ୟ ଅଙ୍ଗ। ସଫ୍ଟୱେୟାର ବିନା ହାର୍ଡୱେୟାର୍‌ର କୌଣସି ବ୍ୟବହାର ହୋଇପାରିବ ନାହିଁ ଏବଂ ହାର୍ଡୱେୟାର ବିନା ସଫ୍ଟୱେୟାର୍‌ର ବ୍ୟବହାର କରାଯାଇପାରିବ ନାହିଁ। ସଫ୍ଟୱେୟାର ହିଁ ହାର୍ଡୱେୟାର୍‌କୁ ଚଳାଇଥାଏ, ଅର୍ଥାତ, ସଫ୍ଟୱେୟାର ହିଁ ହାର୍ଡୱେୟାର୍‌କୁ ଏହାର ସାମର୍ଥ୍ୟ ଦେଇଥାଏ।

ସଫ୍ଟୱେୟାର୍‌ଗୁଡ଼ିକୁ ବ୍ୟାପକ ଭାବରେ ବର୍ଗୀକୃତ କରାଯାଇ ପାରିବ ଯଥା

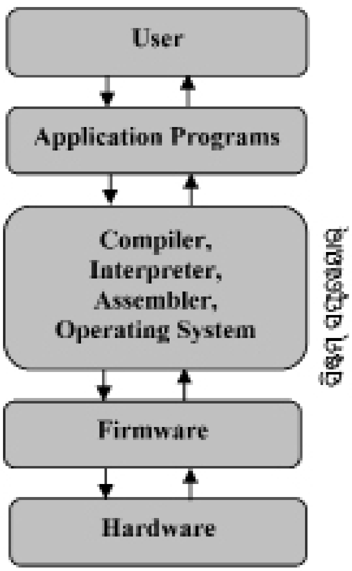
- **ସିଷ୍ଟମ୍ ସଫ୍ଟୱେୟାର (system software) –** ଏହା ଏକ ପ୍ରକାର ସଫ୍ଟୱେୟାର ଯାହା ଏକ କମ୍ପ୍ୟୁଟରର ହାର୍ଡୱେୟାର ଏବଂ ଆପ୍ଲିକେସନ୍ (application) ପ୍ରୋଗ୍ରାମ୍ ଚଳାଇବା ପାଇଁ ଡିଜାଇନ ହୋଇଥାଏ। ଯଦି ଆମେ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମକୁ ଏକ ସ୍ତରିକୃତ (layered) ମଡେଲ୍ ଭାବରେ ଗ୍ରହଣ କରୁ, ତେବେ ସିଷ୍ଟମ୍ ସଫ୍ଟୱେୟାର ହେଉଛି ହାର୍ଡୱେୟାର ଏବଂ ବ୍ୟବହାରକାରୀ ଆପ୍ଲିକେସନ୍ ମଧ୍ୟରେ ଏକ ମିଶ୍ର ପ୍ରତିକ୍ରିୟା ଇଣ୍ଟର୍‌ଫେସ (interface)। ଅପରେଟିଂ ସିଷ୍ଟମ୍ ଏବଂ ଭାଷାନୁବାଦକ (assembler, compiler and interpreter), ଲିଙ୍କର୍ (linker) ଏବଂ ଲୋଡର୍ (loader) ଇତ୍ୟାଦି ସିଷ୍ଟମ୍ ସଫ୍ଟୱେୟାରର ଉଦାହରଣ।
- **ଆପ୍ଲିକେସନ୍ ସଫ୍ଟୱେୟାର (Application Software) –** ଏହା ଏକ ପ୍ରକାର ସଫ୍ଟୱେୟାର ଯାହା ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଉଦ୍ଦେଶ୍ୟପାଇଁ ଡିଜାଇନ୍ ହୋଇଥାଏ ଏବଂ ଶେଷରେ ଏହା ବ୍ୟବହାରକାରୀଙ୍କଦ୍ୱାରା ବ୍ୟବହୃତ ହୁଏ। ୱାର୍ଡ ପ୍ରୋସେସର (Word Processor), ଡାଟାବେସ୍ (Database) ସଫ୍ଟୱେୟାର, ସ୍ପ୍ରେଡସିଟ୍ (Spreadsheet) ସଫ୍ଟୱେୟାର, ମନୋରଞ୍ଜନ ସଫ୍ଟୱେୟାର, ସଂରକ୍ଷଣ (Reservation) ସଫ୍ଟୱେୟାର, ଇନଭେଣ୍ଟୋରି (inventory) ପରିଚାଳନା ସଫ୍ଟୱେୟାର ଇତ୍ୟାଦି ଆପ୍ଲିକେସନ୍ ସଫ୍ଟୱେୟାରର ଉଦାହରଣ।
- **ୟୁଟିଲିଟି ସଫ୍ଟୱେୟାର –** ଏହା ଏକ ପ୍ରକାର ସଫ୍ଟୱେୟାର ଯାହା ଏକ କମ୍ପ୍ୟୁଟରର ବିଶ୍ଳେଷଣ, ବିନ୍ୟାସ, ଅପ୍ଟିମାଇଜ୍ କିମ୍ବା ରକ୍ଷଣାବେକ୍ଷଣ କରିବାରେ ସାହାଯ୍ୟ କରିବାକୁ ଡିଜାଇନ୍ ହୋଇଥାଏ। ଏହା କମ୍ପ୍ୟୁଟରର ଭିତ୍ତିଭୂମିକୁ ସମର୍ଥନ କରିବାପାଇଁ ବ୍ୟବହୃତ ହୋଇଥାଏ। ଆଣ୍ଟିଭାଇରସ୍ (Antivirus) ସଫ୍ଟୱେୟାର, ସଙ୍କୋଚନ ସାଧନ, ଡିସ୍କ ପରିଚାଳନା ସାଧନ ଇତ୍ୟାଦି, ୟୁଟିଲିଟି (utility) ସଫ୍ଟୱେୟାରର ଉଦାହରଣ।

1.1.3.1 ଅପରେଟିଂ ସିଷ୍ଟମ୍ (Operating System)

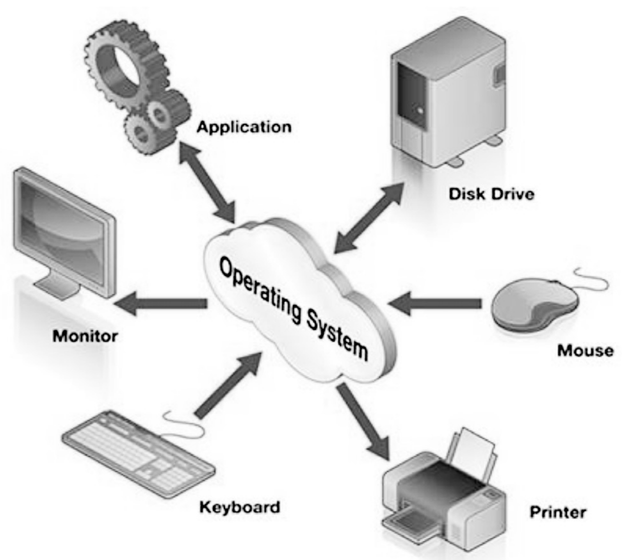
ଯେକୌଣସି କମ୍ପ୍ୟୁଟରରେ ସବୁଠାରୁ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ପ୍ରୋଗ୍ରାମ୍ ହେଉଛି ଅପରେଟିଂ ସିଷ୍ଟମ୍, କହିଲେ ସରଳ ଭାଷାରେ

ଓଏସ୍ (OS) । ଏହା ଅନେକ ଛୋଟ ଛୋଟ ପ୍ରୋଗ୍ରାମ୍‌କୁ ନେଇ ଗଠିତ ଏକ ବୃହତ ପ୍ରୋଗ୍ରାମ୍ । ଏହା କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ସାଧନ ପରିଚାଳକ ଭାବରେ କାର୍ଯ୍ୟକରେ ଆଉ ବ୍ୟବହାରକାରୀ ଏବଂ ହାର୍ଡୱେର ମଧ୍ୟରେ ଏକ ଇଣ୍ଟର୍‌ଫେସ (interface) ପ୍ରଦାନ କରେ ।

ଏକ କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ସାଧନରେ ପ୍ରୋସେସର (processor), ମେମୋରି ଏବଂ ଆଇ/ଓ (I/O) ଉପକରଣ ଆଉ ବ୍ୟବହାରକାରୀ ଇତ୍ୟାଦି ଅନ୍ତର୍ଭୁକ୍ତ। ଏହା ଗୋଟିଏ ପରିଚାଳକ ଭାବରେ ବିଭିନ୍ନ ସାଧନଗୁଡ଼ିକର କାର୍ଯ୍ୟକଳାପର ସମନ୍ୱୟ ରକ୍ଷା କରିଥାଏ ଏହାଦ୍ୱାରା ବ୍ୟବହାରକାରୀକୁ ଆବଶ୍ୟକ ସୂଚନା ପ୍ରଦାନ କରାଯାଇପାରେ ।



ଚିତ୍ର 1.7: କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ସ୍ତରଯୁକ୍ତ ସ୍ଥାପତ୍ୟ



ଚିତ୍ର 1.8: ଅପରେଟିଂ ସିଷ୍ଟମ୍ ପରିଚାଳନା କରୁଥିବା କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ବିଭିନ୍ନ ସାଧନସମୂହ

ସାଧାରଣତଃ ପିସି/ଲାପଟପ୍ପାଇଁ ବ୍ୟବହୃତ ଅପରେଟିଂ ସିଷ୍ଟମ୍‌ଗୁଡ଼ିକ ହେଉଛି

- ୟୁନିକ୍ସ (Unix)
- ଲିନକ୍ସ (Linux)
- ୱିଣ୍ଡୋଜ (Windows)
- ସୋଲାରିସ୍ (Solaris)
- ବସ୍ (ଭାରତ ଅପରେଟିଂ ସିଷ୍ଟମ୍ ସଲ୍ୟୁସନ୍)

ମୋବାଇଲ୍ / ହାତ ଧରା ଉପକରଣପାଇଁ ସାଧାରଣତଃ ବ୍ୟବହୃତ ଅପରେଟିଂ ସିଷ୍ଟମ୍‌ଗୁଡ଼ିକ ହେଉଛି

- ଆଣ୍ଡ୍ରଏଡ୍ (Android)
- ଆଇଓଏସ୍ (iOS)
- ସିମ୍ବିଆନ୍

1.1.3.2 ଭାଷାନୁବାଦକ (Language Translators)

ଭାଷାନୁବାଦକ ହେଉଛି ସଫ୍ଟୱେୟାର ଏକ ଅଙ୍ଗ । ଏହା ଗୋଟିଏ ଭାଷାରେ ଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌କୁ ଅନ୍ୟ ଭାଷାରେ ଅନୁବାଦ କରେ (ଏହାକୁ ମଧ୍ୟ ଭାଷା ପ୍ରୋସେସର କୁହାଯାଏ) ।

ଏଠାରେ ଉଲ୍ଲେଖ କରିବା ପ୍ରାସଙ୍ଗିକ ଯେ ଯାନ୍ତ୍ରିକ (machine) ଭାଷା କୌଣସି ଅନୁବାଦ ଆବଶ୍ୟକ କରେ ନାହିଁ । କିନ୍ତୁ ଉଚ୍ଚ ସ୍ତରୀୟ ଭାଷାରେ ଲେଖାଯାଇଥିବା ପ୍ରୋଗ୍ରାମ୍, ଯେପରିକି C, ଗୋଟିଏ ଇଣ୍ଟରପ୍ରିଟରକୁ C ଭାଷାରେ ଲେଖାଯାଇଥିବା ନିର୍ଦ୍ଦେଶାବଳିକୁ ଯାନ୍ତ୍ରିକ (machine) ଭାଷାରେ ଅନୁବାଦ କରିବାକୁ ଆବଶ୍ୟକ ହୋଇଥାଏ ।

C ଭାଷାରେ ଲିଖିତ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍‌କୁ ଉତ୍ସ କୋଡ୍ (source code) କୁହାଯାଏ । ଉତ୍ସ କୋଡ୍‌ର ଅନୁବାଦିତ ସଂସ୍କରଣ ହେଉଛି ମେସିନ୍ ଭାଷାରେ ଏକ ପ୍ରୋଗ୍ରାମ୍, ଯାହାକୁ ଅବ୍‌ଜେକ୍ଟ କୋଡ୍ (object code) କୁହାଯାଏ ।

ଆସେମ୍ବଲି (assembly) ଭାଷାପାଇଁ ଭାଷାନୁବାଦକକୁ ଏକ ଆସେମ୍ବଲର୍ (assembler) କୁହାଯାଏ, ଏବଂ ଉଚ୍ଚସ୍ତରୀୟ ଭାଷାପାଇଁ ଭାଷାନୁବାଦକଗୁଡ଼ିକୁ କମ୍ପାଇଲର୍ (compilers) ଆଉ ଇଣ୍ଟରପ୍ରିଟର୍ (interpreters) କୁହାଯାଏ ।

ଆସେମ୍ବଲର୍ (Assembler)

ଏକ ଆସେମ୍ବଲର୍ ହେଉଛି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ଆସେମ୍ବଲି ଭାଷାର (ଉତ୍ସ କୋଡ୍) ଏକ ପ୍ରୋଗ୍ରାମ୍‌କୁ ମେସିନ୍ ଭାଷାକୁ (ଅବ୍‌ଜେକ୍ଟ କୋଡ୍‌କୁ) ଅନୁବାଦ କରେ ।

ଯେହେତୁ ମେସିନ୍ ଭାଷା ଏବଂ ଆସେମ୍ବଲି ଭାଷାର ନିର୍ଦ୍ଦେଶାବଳି ମଧ୍ୟରେ ଏକୈକ ଅନୁରୂପତା (one-to-one correspondence) ରହିଛି, ଆସେମ୍ବଲର୍ ଆସେମ୍ବଲି ଭାଷାରେ ପ୍ରତ୍ୟେକ ନିର୍ଦ୍ଦେଶାବଳିକୁ ମେସିନ୍ ଭାଷାରେ ଅନୁରୂପ ନିର୍ଦ୍ଦେଶାବଳିକୁ ଅନୁବାଦ କରେ ।

କମ୍ପାଇଲର୍ (Compiler)

କମ୍ପାଇଲର୍ ହେଉଛି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ଏକ ଉଚ୍ଚସ୍ତରୀୟ ଭାଷାରେ ଲିଖିତ ଏକ ପ୍ରୋଗ୍ରାମ୍‌କୁ (ଉତ୍ସ କୋଡ୍‌କୁ) ମେସିନ୍ ଭାଷାକୁ (ଅବ୍‌ଜେକ୍ଟ କୋଡ୍‌କୁ) ଅନୁବାଦ କରେ ।

ଅନୁବାଦର ଏହି ପ୍ରକ୍ରିୟାକୁ ସଂକଳନ କୁହାଯାଏ । ସଂକଳନ ସମୟରେ କମ୍ପାଇଲର୍ ଉତ୍ସ କୋଡ୍‌କୁ ଧାଡ଼ି ପରେ ଧାଡ଼ି ପଢ଼ି ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟିର ଯାଞ୍ଚ କରେ । ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ ପ୍ରୋଗ୍ରାମିଂ ଭାଷାରେ କୌଣସି ନିୟମର (ଭାଷାର

ବ୍ୟାକରଣ) ଉଲ୍ଲଂଘନକୁ ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ବୋଲି କୁହାଯାଏ।

ଯଦି ସିଣ୍ଟାକ୍ସରେ କୌଣସି ତ୍ରୁଟି ଥାଏ, ତେବେ ଏହା ପ୍ରୋଗ୍ରାମ୍ ଭିତରେ ଥିବା ତ୍ରୁଟିର ଅବସ୍ଥାନ ଦର୍ଶାଉଥିବା ତ୍ରୁଟି ଗୁଡ଼ିକ ପ୍ରଦର୍ଶନ କରେ ଏବଂ ଏକ ତାଲିକା ପ୍ରିଣ୍ଟ କରେ। ଏହା ଅବସ୍ଥାନ ଏବଂ ତ୍ରୁଟିଗୁଡ଼ିକର ସମ୍ଭାବ୍ୟ କାରଣକୁ ଆଲୋକପାତ କରେ, ଆଉ କୌଣସି ଅନୁବାଦ ହୁଏ ନାହିଁ। କିନ୍ତୁ ଯଦି ଉତ୍ସ କୋଡ୍‌ରେ କୌଣସି ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ନାହିଁ, ତେବେ କମ୍ପାଇଲର୍ ଉତ୍ସ କୋଡ୍‌କୁ ଏକ ଅବଜେକ୍ଟ କୋଡ୍‌ରେ ସାଧାରଣତଃ .obj ସମ୍ପ୍ରସାରଣ (extension) ସହିତ (ଅବ୍‌ଜେକ୍ଟ କୋଡ୍‌ପାଇଁ) ଅନୁବାଦ କରି ଅନ୍ୟ ଏକ ଫାଇଲ୍‌ରେ ଲେଖେ।

ଥରେ ଅନୁବାଦ ହେବା ପରେ, ପରବର୍ତ୍ତୀ ପଦକ୍ଷେପରେ କମ୍ପାଇଲରର ଆଉ କୌଣସି ଭୂମିକା ନଥାଏ। କିନ୍ତୁ, ଯଦି ପୁଣି ଉତ୍ସ କୋଡ୍‌ରେ ପରିବର୍ତ୍ତନ କରାଯାଏ, ତେବେ ଅବଜେକ୍ଟ କୋଡ୍‌ରେ ରୂପାନ୍ତରପାଇଁ ଆମକୁ ଏହାକୁ ପୁନଃସଂକଳନ କରିବା ଆବଶ୍ୟକ ହୁଏ।

ଇଣ୍ଟର୍‌ପ୍ରିଟର (Interpreter)

ଇଣ୍ଟର୍‌ପ୍ରିଟର ହେଉଛି ଉଚ୍ଚ ସ୍ତରୀୟ ଭାଷାର ଅନୁବାଦପାଇଁ ବ୍ୟବହୃତ ଅନ୍ୟ ଏକ ଅନୁବାଦକ। କିନ୍ତୁ, ଇଣ୍ଟର୍‌ପ୍ରିଟର ଅବଜେକ୍ଟ କୋଡ୍ ପ୍ରସ୍ତୁତ କରେ ନାହିଁ। ଏହା ବଦଳରେ, ଏହା ଏକ ସମୟରେ ଉତ୍ସ କୋଡ୍‌ର ଗୋଟିଏ ଧାଡ଼ି ଅନୁବାଦ କରେ ଏବଂ ଅନୁବାଦିତ ନିର୍ଦ୍ଦେଶାବଳିକୁ ଚାଳନ କରେ। ସମସ୍ତ ନିର୍ଦ୍ଦେଶାବଳିର ଅନୁବାଦ ଏବଂ ଚାଳନ ନ ହେବା ପର୍ଯ୍ୟନ୍ତ ଏହା ଜାରି ରହିଥାଏ। ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ ଯଦି କୌଣସି ନିର୍ଦ୍ଦେଶକୁ ବାରମ୍ବାର ଚାଳନ କରିବାକୁ ହୁଏ, ତେବେ ଏହାକୁ ପ୍ରତ୍ୟେକ ଥର ଅନୁବାଦ କରାଯାଏ।

ଥରେ ଉତ୍ସ କୋଡ୍ କମ୍ପାଇଲ୍ ଦ୍ୱାରା ଅବଜେକ୍ଟ କୋଡ୍‌ରେ ଅନୁବାଦ ହେବା ପରେ, (ଏହା ଏକ ଏକକାଳୀନ କାର୍ଯ୍ୟ ଅଟେ) ଅବଜେକ୍ଟ କୋଡ୍‌ର ପରବର୍ତ୍ତୀ ପ୍ରକ୍ରିୟାକରଣ (ଲିଙ୍କିଙ୍ଗ୍) ପରେ ସିଧାସଳଖ ଚାଳିତ ହୋଇଥାଏ। ଏହା ଫଳରେ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ଦ୍ରୁତତର ହୋଇଥାଏ। କିନ୍ତୁ, ଇଣ୍ଟର୍‌ପ୍ରିଟର ପ୍ରଥମେ କୋଡ୍ ବୁଝେ ଏବଂ ତା'ପରେ ନିର୍ଦ୍ଦେଶାବଳି ଚାଳନ କରେ। ଏହାଦ୍ୱାରା ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ମନ୍ଥର ହୋଇଥାଏ। ଯେତେ ଥର ଇଣ୍ଟର୍‌ପ୍ରିଟର ବ୍ୟବହାର କରି ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ କରିବାକୁ ହୁଏ ସେତେଥର ଅନୁବାଦର ପୁନରାବୃତ୍ତି ହୋଇଥାଏ।

ଯଦିଓ ଇଣ୍ଟର୍‌ପ୍ରିଟରଗୁଡ଼ିକପାଇଁ ପ୍ରୋଗ୍ରାମ ଅନୁବାଦ ମନ୍ଥର ହୋଇଥାଏ, ତଥାପି, ସେଗୁଡ଼ିକ ଶିକ୍ଷାନବିଶମାନଙ୍କ ପାଇଁ ଭାଷା ଶିଖିବା ଏବଂ ପ୍ରୋଗ୍ରାମଗୁଡିକୁ ଡିବଗ୍ (debug) କରିବାପାଇଁ ସହାୟକ। ଏଥିରେ ଯେହେତୁ ପ୍ରୋଗ୍ରାମ୍‌ଟି ଧାଡ଼ି ପରେ ଧାଡ଼ି ଚାଳନ ହୋଇଥାଏ, ତେଣୁ ପ୍ରୋଗ୍ରାମ୍‌ର ପ୍ରତ୍ୟେକ ଧାଡ଼ି କ'ଣ କରୁଛି ତାହା ଦେଖି ହୁଏ।

ଏଠାରେ ଏହା ଉଲ୍ଲେଖ କରିବା ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଯେ ଇଣ୍ଟର୍‌ପ୍ରିଟର ବ୍ୟବହାର କରି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ କରିବା ପାଇଁ ପ୍ରଥମେ ଆମକୁ ଇଣ୍ଟର୍‌ପ୍ରିଟର ଚଳାଇବା ଆବଶ୍ୟକ। କେବଳ ତା'ପରେ ଆମେ, ବ୍ୟବହାରକାରୀ ପ୍ରୋଗ୍ରାମ୍‌କୁ ଚାଳନ କରିବାକୁ ଇଣ୍ଟର୍‌ପ୍ରିଟରକୁ ନିର୍ଦ୍ଦେଶ ଦେଇପାରିବା। ତେଣୁ, ପ୍ରତ୍ୟେକ ଥର ଯେତେବେଳେ ତୁମେ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଚଳାଅ ତୁମକୁ ଏକ ଇଣ୍ଟର୍‌ପ୍ରିଟର ଚଳାଇବାପାଇଁ ମଧ୍ୟ ଆବଶ୍ୟକ ହୋଇଥାଏ।

1.1.3.3 ପ୍ରୋଗ୍ରାମିଂ ପରିବେଶ

ପ୍ରୋଗ୍ରାମିଂ ପରିବେଶ ଏକ ପରିବେଶ ଯେଉଁଥିରେ ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକର ଲେଖାହୁଏ ଏବଂ ପରୀକ୍ଷା ନିରୀକ୍ଷା ହୁଏ, ଏବଂ ଏହା ପରିଚାଳନା ପରିବେଶ (ଯେଉଥିରେ ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ଚାଳନ ହୁଏ) ଅପେକ୍ଷା ଭାଷାର ଡିଜାଇନ୍ ଉପରେ କମ୍ ପ୍ରଭାବ ପକାଇଥାଏ।

ଏକ ପ୍ରୋଗ୍ରାମିଂ ପରିବେଶ। ସହାୟକ ସାଧନସମୂହ ଏବଂ ଗୁଡ଼ିକର ବ୍ୟବହାରପାଇଁ ନିର୍ଦ୍ଦେଶ-ଭାଷାକୁ (ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକର ଏକ ସେଟ୍) ନେଇ ଗଠିତ। ପ୍ରତ୍ୟେକ ସହାୟକ ସାଧନ ହେଉଛି ଅନ୍ୟ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ପ୍ରୋଗ୍ରାମ୍ ଦ୍ୱାରା ପ୍ରୋଗ୍ରାମ୍ ନିର୍ମାଣର ଏକ କିମ୍ବା ଅଧିକ ପର୍ଯ୍ୟାୟରେ ଏକ ସହାୟକ ଭାବରେ ବ୍ୟବହାର କରାଯାଇପାରେ।

ଏକ ପ୍ରୋଗ୍ରାମିଂ ପରିବେଶରେ ସାଧାରଣ ସାଧନଗୁଡ଼ିକ ଯେପରିକି ଏଡିଟର (editors), ଭାଷାନୁବାଦକ (language translators), ଲିଙ୍କର୍ (linker), ଲୋଡର୍ (loaders), ଇତ୍ୟାଦି ଅନ୍ତର୍ଭୁକ୍ତ ଥାଏ। ଭାଷାନୁବାଦକଗୁଡ଼ିକୁ ପୂର୍ବ ବିଭାଗରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଛି, ଭାଗଶେଷ ଉପକରଣଗୁଡ଼ିକ ଏହିଠାରେ ବର୍ଣ୍ଣନା କରାଯାଇଛି।

- **ଏଡିଟର –** ଏଡିଟର ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ଫାଇଲ୍ ତିଆରି ଏବଂ ସମ୍ପାଦନାରେ ସାହାଯ୍ୟ କରେ। ଆହୁରି ମଧ୍ୟ, କିଛି ଏଡିଟର ଅଛନ୍ତି ଯେଉଁଥିରେ କେବଳ ଟେକ୍ସ୍ଟ ଫାଇଲ୍ (text file) ସହ କାମ କରିହୁଏ, ଏହାକୁ ଟେକ୍ସ୍ଟ ଏଡିଟର କୁହାଯାଏ। ୱିଣ୍ଡୋଜ ଅପରେଟିଂ ସିଷ୍ଟମ୍‌ରେ ଲୋକପ୍ରିୟ ଟେକ୍ସ୍ଟ ଏଡିଟରମାନଙ୍କର ଉଦାହରଣ ହେଉଛି ନୋଟପ୍ୟାଡ୍ (notepad) ଏବଂ ଏଡିଟ (edit), ୟୁନିକ୍ସ ଏବଂ ଲିନକ୍ସ ଅପରେଟିଂ ସିଷ୍ଟମ୍‌ରେ ଭିଆଇ (vi/vim)। ଏପରିକି ଲୋକପ୍ରିୟ ୱାର୍ଡ଼ ପ୍ରୋସେସିଙ୍ଗ ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ଯେପରିକି ଏମଏସ ୱାର୍ଡ, ୱାର୍ଡ ଡକ୍ୟୁମେଣ୍ଟ ସହିତ ଟେକ୍ସ୍ଟ ଫାଇଲ୍ ସୃଷ୍ଟି କରିବାକୁ ମଧ୍ୟ ଅନୁମତି ଦେଇଥାଏ।
- **ଲିଙ୍କର୍ –** ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ କମ୍ପାଇଲ୍ ଦ୍ୱାରା ନିର୍ମିତ ଅବ୍‌ଜେକ୍ଟ କୋଡ୍ କମ୍ପ୍ୟୁଟରଦ୍ୱାରା ଚାଳନ କରାଯାଇପାରିବା ଭଳି ରୂପରେ ନଥାଏ। କମ୍ପ୍ୟୁଟରଦ୍ୱାରା ଚାଳନ ହୋଇପାରୁଥିବା ଏକ ରୂପକୁ ଆଣିବାପାଇଁ ଅବଜେକ୍ଟ କୋଡ୍‌ରେ ଆଉ କିଛି ଅଧିକ ସୂଚନାର ଆବଶ୍ୟକତା ପଡିଥାଏ।

  ଯେତେବେଳେ ତୁମେ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖୁଛ, ତୁମେ ଇନପୁଟ୍ /ଆଉଟପୁଟ୍‌ର ସଞ୍ଚାଳନ ଏବଂ ଗାଣିତିକ ଗଣନାପାଇଁ ବିଭିନ୍ନ ଲାଇବ୍ରେରି ଫଙ୍କ୍ସନଗୁଡ଼ିକ ବ୍ୟବହାର କରିପାରିବ (କିଛି ନିତ୍ୟ ବ୍ୟବହାର୍ଯ୍ୟ ଗାଣିତିକ ଲାଇବ୍ରେରି ଫଙ୍କ୍ସନ୍‌ର ନାମ: sqrt, sin, cos, abs, exp ଇତ୍ୟାଦି)। ଏହି ଲାଇବ୍ରେରି ଫଙ୍କ୍ସନଗୁଡ଼ିକଦ୍ୱାରା ସମ୍ପାଦନ ହେବାକୁ ଥିବା କାର୍ଯ୍ୟର ବ୍ୟାଖ୍ୟା କରୁଥିବା ନିର୍ଦ୍ଦେଶାବଳି ମେସିନ ଭାଷାରେ (binary) ଅଛି ଏବଂ ଏଗୁଡ଼ିକ କମ୍ପାଇଲର୍ ସହିତ ଆସିଥିବା ସିଷ୍ଟମ୍ ଲାଇବ୍ରେରି ଭିତରେ ଥାଏ (ଫାଇଲ୍‌ଗୁଡ଼ିକର ସମ୍ପ୍ରସାରଣ .LIB)। ଏଥିସହ କିଛି ବ୍ୟବହାରକାରୀ-ନିର୍ଦ୍ଧାରିତ ଫଙ୍କ୍ସନ୍ ଯାହା ଏକ ପୃଥକ ପ୍ରୋଗ୍ରାମ୍ ଫାଇଲରେ ଲେଖାଥାଏ। ଯାହାର ଅବଜେକ୍ଟ କୋଡ୍ ବିବେଚିତ ଅବଜେକ୍ଟ କୋଡ୍‌ଠାରୁ ପୃଥକ ଭାବରେ ରହିଥାଏ। ତେଣୁ, ଲାଇବ୍ରେରି ଓ ବ୍ୟବହାରକାରୀ-ନିର୍ଦ୍ଧାରିତ ଫଙ୍କ୍ସନଗୁଡ଼ିକର ଅବଜେକ୍ଟ କୋଡ୍‌କୁ ବିବେଚିତ ଅବଜେକ୍ଟ କୋଡ୍ ସହିତ ଯୋଡ଼ିବାର ଆବଶ୍ୟକତା ଅଛି, ଏବଂ ଏହି କାର୍ଯ୍ୟଟି ସମ୍ପନ୍ନ କରୁଥିବା ପ୍ରୋଗ୍ରାମ୍‌କୁ ଲିଙ୍କର୍ କୁହାଯାଏ।

  ଲିଙ୍କର୍ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ପ୍ରୋଗ୍ରାମ୍‌ର ଅବଜେକ୍ଟ କୋଡ୍‌କୁ ଲାଇବ୍ରେରି ଓ ବ୍ୟବହାରକାରୀ-ନିର୍ଦ୍ଧାରିତ ଫଙ୍କ୍ସନଗୁଡ଼ିକର ଅବଜେକ୍ଟ କୋଡ୍ ସହିତ ଇପ୍‌ସିତ ରୀତିରେ ଯୋଡ଼ିଥାଏ, ଏବଂ ସାଧାରଣତଃ ୱିଣ୍ଡୋଜ୍‌ରେ EXE ସମ୍ପ୍ରସାରଣ ସହିତ ତାହାକୁ ଏକ ପୃଥକ ପ୍ରୋଗ୍ରାମ୍ ଫାଇଲରେ ଲେଖେ। ଲିଙ୍କର୍‌ଦ୍ୱାରା ଗଠିତ ଏହି ଫାଇଲ୍‌କୁ ଚାଳନଯୋଗ୍ୟ ଫାଇଲ୍ କିମ୍ବା ସରଳ ଭାବେ ଚାଳନ ଫାଇଲ୍ କୁହାଯାଏ, ଏବଂ ଏହି ଫାଇଲ୍ ପ୍ରୋଗ୍ରାମ୍‌ର ଚାଳନପାଇଁ ବ୍ୟବହୃତ ହୋଇଥାଏ।
- **ଲୋଡର୍ –** ଲୋଡର୍ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ଫାଇଲ୍‌କୁ ମାଧ୍ୟମିକ ଭଣ୍ଡାରରୁ ମେମୋରିକୁ ସ୍ଥାନାନ୍ତର କରିଥାଏ।

ଚାଳନ ସମୟରେ, ଲୋଡର୍ ଅପରେଟିଂ ସିଷ୍ଟମ୍‌ର ସହାୟତାରେ, ପ୍ରଥମେ ଆବଶ୍ୟକ ପରିମାଣର ଅବ୍ୟବହୃତ ମେମୋରି ଖୋଜିଥାଏ ଏବଂ ଚାଳନ ଫାଇଲ୍‌କୁ ମେମୋରିର ସେହି ଅଂଶକୁ ସ୍ଥାନାନ୍ତର କରେ, ଠିକଣା ଅନୁବାଦ ପରି ଉପଯୁକ୍ତ ପଦକ୍ଷେପ ନେଇଥାଏ, ଏବଂ ତା'ପରେ ଏହାର ଚାଳନ ଆରମ୍ଭ କରିଥାଏ। ସେହି ସମୟରୁ, ଲୋଡରର ଭୂମିକା ସମାପ୍ତ ହୋଇଥାଏ, ଏବଂ ପ୍ରୋଗ୍ରାମ୍‌ଦ୍ୱାରା ନିୟନ୍ତ୍ରଣ ନିଆଯାଇଥାଏ। ଇପ୍‌ସିତ କାର୍ଯ୍ୟ ସମ୍ପାଦନ କରିବାକୁ ପ୍ରୋଗ୍ରାମ୍‌ର ନିର୍ଦ୍ଦେଶାବଳିର ଅପରେଟିଂ ସିଷ୍ଟମର ସାମଗ୍ରିକ ତତ୍ତ୍ୱାବଧାନରେ ଚାଳନ କରାଯାଇଥାଏ। ପ୍ରୋଗ୍ରାମ୍ ଏହାର ଚାଳନ ସମାପ୍ତ କରିବା ମାତ୍ରେ ଏହା ମେମୋରିରୁ ବାହାର କରିନିଆଯାଇଥାଏ।

> ଲୋକପ୍ରିୟ ଇଣ୍ଟିଗ୍ରେଟେଡ୍ ଡେଭେଲପମେଣ୍ଟ ଏନଭିରନମେଣ୍ଟ (ଆଇଡିଇ) ବା ସମନ୍ୱିତ ବିକାଶ ପରିବେଶ, ଯେପରିକି ଟର୍ବୋ C/C++ ଏବଂ ବୋରଲ୍ୟାଣ୍ଡ C++, ଭିଜୁଆଲ୍ ଷ୍ଟୁଡିଓ, ଡେଭ C++, କୋଡ୍ ବ୍ଲକ୍, ନେଟବିନ୍‌ସ, ଏକ୍‌ଲିପ୍‌ସ, ଇତ୍ୟାଦି, ବିଭିନ୍ନ ସାଧନଗୁଡ଼ିକୁ (ଏଡିଟର, କମ୍ପାଇଲର୍, ଲିଙ୍କର୍, ଡିବଗର୍ ଇତ୍ୟାଦି) ଏକ ସ୍ଥାନରେ ଏକୀକୃତ ହୋଇଥାଏ, ଯଦ୍ୱାରା ଜଣେ ପ୍ରୋଗ୍ରାମର୍ ଏହି ସାଧନଗୁଡ଼ିକ ସହ ଅତ୍ୟନ୍ତ ଆରାମରେ ଏବଂ ବର୍ଦ୍ଧିତ ଉତ୍ପାଦକତା ସହିତ କାର୍ଯ୍ୟ କରିପାରିବେ।

1.1.4 ବୁଟିଂର ଧାରଣା (Concept of Booting)

ସରଳ ଶବ୍ଦରେ, ବୁଟିଂ ଏକ ପ୍ରକ୍ରିୟା। ଏଥିରେ ତୁମର କମ୍ପ୍ୟୁଟର ଇନିସିଆଲାଇଜ୍ (initialize) ବା ଚଳନକ୍ଷମ ହୋଇଥାଏ। ଏହି ପ୍ରକ୍ରିୟାରେ କମ୍ପ୍ୟୁଟରର ସମସ୍ତ ହାର୍ଡୱେୟାର ଅଂଶଗୁଡ଼ିକ ଇନିସିଆଲାଇଜ୍/ଚଳନକ୍ଷମ କରିବା ଓ ସେଗୁଡିକୁ ଏକତ୍ର ଭାବେ କାର୍ଯ୍ୟ କରିବା ସହ ଡିଫଲ୍ଟ ଅପରେଟିଂ ସିଷ୍ଟମ୍ ଲୋଡ୍ କରିବା ଅନ୍ତର୍ଭୁକ୍ତ। ଏହା କମ୍ପ୍ୟୁଟରକୁ କାର୍ଯ୍ୟକ୍ଷମ କରିଥାଏ।

ବୁଟିଂ ପ୍ରକ୍ରିୟାର ବୈଷୟିକ ବିବରଣୀ ନିମ୍ନମତେ ସାରାଂଶ କରାଯାଇପାରିବ:

1. ଯେତେବେଳେ କମ୍ପ୍ୟୁଟର୍ ଅନ୍ (on) ହୁଏ, ବୁଟ୍ (boot) ପ୍ରୋଗ୍ରାମ୍‌ର ଏକ କପି ରମ୍‌ରୁ (ROM) ମୁଖ୍ୟ ମେମୋରିକୁ ଅଣା ଯାଇଥାଏ।
2. ତା'ପରେ ସିପିୟୁ (CPU) ଏକ ଜମ୍ପ ନିର୍ଦ୍ଦେଶାବଳିକୁ ଚାଳନ କରେ ଯାହା ବାୟୋସ୍ (BIOS) ବା ମୌଳିକ ଇନପୁଟ୍ ଆଉଟପୁଟ୍ ସିଷ୍ଟମ୍‌କୁ (Basic Input output System) ସ୍ଥାନାନ୍ତର କରେ ଏବଂ ଏହାପରେ ସିପିୟୁର ଚାଳନ ଆରମ୍ଭ ହୁଏ।
3. ବାୟୋସ୍, ସ୍ୱକୀୟ ତ୍ରୁଟି ନିର୍ଣ୍ଣାୟକ ପରୀକ୍ଷଣ ବା ସେଲ୍‌ଫ ଡାଇଗ୍ନୋଷ୍ଟିକ୍ ଟେଷ୍ଟ୍‌ର (self diagnostic tests) ଏକ ଅନୁକ୍ରମକୁ ପରିଚାଳନା କରିଥାଏ। ଏହାକୁ ପୋଷ୍ଟ (POST) ବା ପାୱାର ଅନ୍ ସେଲ୍‌ଫ ଟେଷ୍ଟ (Power On Self Test) କୁହାଯାଏ। ଏହି ପରୀକ୍ଷାଗୁଡ଼ିକ ମଧ୍ୟରେ ମେମୋରି ପରୀକ୍ଷା, ଭିଡିଓ ସର୍କିଟରୀର (video circuitry) କନ୍‌ଫିଗିୟୋର୍ (configure) ଏବଂ ଷ୍ଟାର୍ଟ କରିବା, ସିଷ୍ଟମର ହାର୍ଡୱେୟାର କନ୍‌ଫିଗିୟୋର୍ କରିବା ଏବଂ ଅନ୍ୟ ଯନ୍ତ୍ରାଂଶଗୁଡ଼ିକୁ ଯାଞ୍ଚ କରିବା ଅନ୍ତର୍ଭୁକ୍ତ ଯାହା କମ୍ପ୍ୟୁଟରକୁ ଠିକ୍ ଭାବରେ କାର୍ଯ୍ୟ କରିବାରେ ସାହାଯ୍ୟ କରେ।
4. ଏହାପରେ, ବାୟୋସ୍ ବୁଟ୍ ସେକ୍ଟର ଲୋଡ୍ କରିବାପାଇଁ ଏକ ବୁଟେବଲ୍ ଡ୍ରାଇଭ୍ (bootable drive) ଚିହ୍ନଟ କରେ। ବୁଟ୍‌ଷ୍ଟ୍ରାପ୍ (bootstrap) ଲୋଡର୍ ପ୍ରୋଗ୍ରାମ୍, ଅପରେଟିଂ ସିଷ୍ଟମ୍‌କୁ ଲୋଡ୍ କରେ ଏବଂ ଚାଳନ କରେ।

ବୁଟିଂ ପ୍ରକ୍ରିୟା ଦୁଇ ପ୍ରକାରର:

- **ଶୀତଳ ବୁଟିଂ:** ଯେତେବେଳେ ସିଷ୍ଟମ୍ ପ୍ରାରମ୍ଭିକ ଅବସ୍ଥାରୁ ଆରମ୍ଭ ହୁଏ, ଅର୍ଥାତ, ଏହା ସୁଇଚ୍ ଅନ୍ ହୋଇଥାଏ । ଏହାକୁ ମଧ୍ୟ କଠିନ ବୁଟିଂ କୁହାଯାଏ । ଉପର ବ୍ୟାଖ୍ୟା କରାଯାଇଥିବା ଅନେକ ପର୍ଯ୍ୟାୟ ଦେଇ ସିଷ୍ଟମ୍‌ଟି ଗତି କରିଥାଏ ।
- **ଉଷ୍ମ ବୁଟିଂ:** ରିସେଟ୍ (Reset) ବଟନ୍ ବା Ctrl+Alt+Del ବଟନ୍‌ଗୁଡ଼ିକର ସମାହାର ସିଷ୍ଟମ୍‌କୁ ପୁନାରମ୍ଭ (restart) କରିବାକୁ ବ୍ୟବହାର ହୁଏ । ଏହାକୁ ନରମ ବୁଟିଂ ମଧ୍ୟ କୁହାଯାଏ । ଏଠାରେ ସିଷ୍ଟମ୍ ପ୍ରାରମ୍ଭିକ ଅବସ୍ଥାରୁ ଆରମ୍ଭ ହୁଏ ନାହିଁ ତେଣୁ ଏହି କ୍ଷେତ୍ରରେ ତ୍ରୁଟି ନିର୍ଣ୍ଣାୟକ ପରୀକ୍ଷଣ କରାଯାଏ ନାହିଁ ।

1.2 ଆଲଗୋରିଦମର ଧାରଣା (IDEA OF ALGORITHMS)

ଏକ ଆଲଗୋରିଦମ୍ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ସମସ୍ୟାର ସମାଧାନକୁ ବ୍ୟାଖ୍ୟା କରୁଥିବା ନିର୍ଦ୍ଦେଶାବଳିର ଏକ ସୀମିତ କ୍ରମ ।

ଏକ ଭଲ ଆଲଗୋରିଦମର ବୈଶିଷ୍ଟ୍ୟଗୁଡ଼ିକ:

ଏକ ଆଲଗୋରିଦମର ପାଞ୍ଚଟି ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ବୈଶିଷ୍ଟ୍ୟ ଅଛି ଯାହା ଯେକୌଣସି ସମସ୍ୟାର ସମାଧାନପାଇଁ ଆଲଗୋରିଦମ୍ ଡିଜାଇନ୍ କରିବା ସମୟରେ ବିଚାର କରାଯିବା ଉଚିତ ।

- **ଇନପୁଟ୍:** ଏକ ଆଲଗୋରିଦମରେ ଶୂନ୍ୟ କିମ୍ବା ଅଧିକ କିନ୍ତୁ ସୀମିତ ସଂଖ୍ୟକ ଇନପୁଟ୍ ରହିବା ଆବଶ୍ୟକ, ଯାହା ବାହ୍ୟ ଭାବରେ ଯୋଗାଇ ଦିଆଯାଏ ।

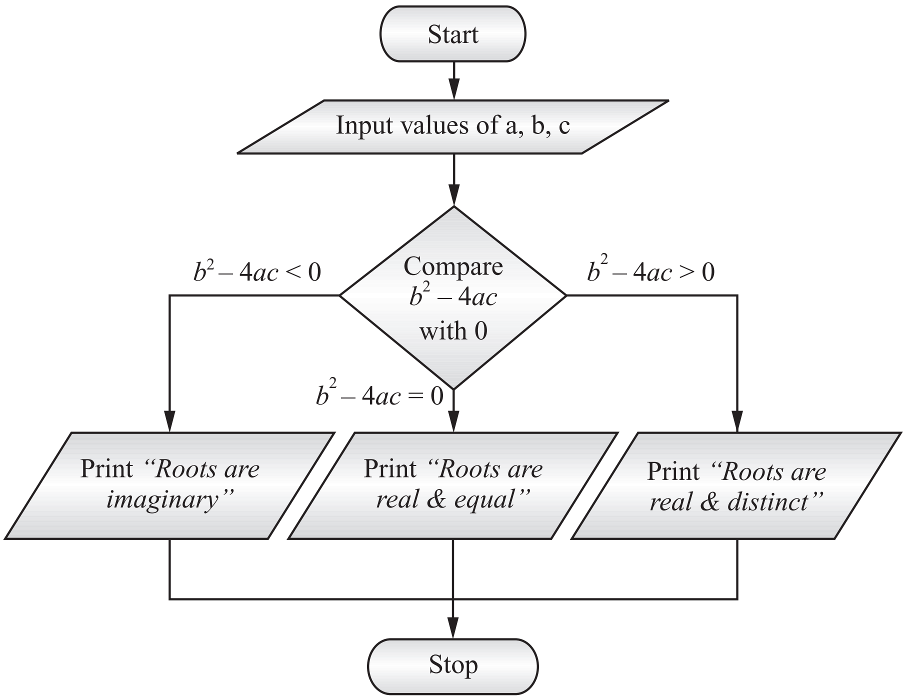
  ଶୂନ୍ୟ ଇନପୁଟ୍ ଆଲଗୋରିଦମର ଏକ ଉଦାହରଣ ହେଉଛି ପ୍ରଥମ 100 ଗଣନ ସଂଖ୍ୟାର ସମଷ୍ଟି ବାହାର କରିବା । ଏଠାରେ, ବ୍ୟବହାରକାରୀ କୌଣସି ବାହ୍ୟ ଇନପୁଟ୍ ଯୋଗାଣ କରିବାର ଆବଶ୍ୟକତା ନାହିଁ କାରଣ ଏହା ପ୍ରଥମ 100 ଗଣନ ସଂଖ୍ୟାର ସମଷ୍ଟି ଖୋଜିବାପାଇଁ ପୂର୍ବରୁ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇଛି ।

  କିନ୍ତୁ, ଯଦି ଉପରୋକ୍ତ ସମସ୍ୟାକୁ ଆଂଶିକ ବଦଳାଇ ପ୍ରଥମ n ଗଣନ ସଂଖ୍ୟାର ସମଷ୍ଟି ବାହାର କରିବାପାଇଁ ପୁନଃ-ଦର୍ଶାଯାଏ, ତେବେ ବର୍ତ୍ତମାନ ବ୍ୟବହାରକାରୀକୁ n ର ମୂଲ୍ୟକୁ ଦର୍ଶାଉଥିବା ଏକ ଇନପୁଟ୍ ଦେବା ଆବଶ୍ୟକ ହେବ ।
- **ଆଉଟପୁଟ୍:** ଏକ ଆଲଗୋରିଦମ୍‌ରେ ଅତି କମ୍‌ରେ ଗୋଟିଏ ବାଞ୍ଛିତ ପରିଣାମ, ଅର୍ଥାତ ଆଉଟପୁଟ୍ ରହିବା ଆବଶ୍ୟକ ।
- **ନିର୍ଦ୍ଦିଷ୍ଟତା (Definiteness) (କୌଣସି ଅସ୍ପଷ୍ଟତା ନ ଥିବା):** ପ୍ରତ୍ୟେକ ପଦକ୍ଷେପ ସ୍ପଷ୍ଟ ଏବଂ ଏକାର୍ଥକ ହେବା ଆବଶ୍ୟକ, ଅର୍ଥାତ, ଗୋଟିଏ ଏବଂ କେବଳ ଗୋଟିଏ ଅର୍ଥ ରହିବା ଦରକାର ।
- **ସୀମିତତା (Finiteness):** ଯଦି ଆମେ ଏକ ଆଲଗୋରିଦମର ପଦକ୍ଷେପଗୁଡ଼ିକର ପର୍ଯ୍ୟବେକ୍ଷଣ କରିବା, ତେବେ ସମସ୍ତ ସ୍ଥଳରେ, ଆଲଗୋରିଦମ୍‌ଟି ଏକ ସୀମିତ ସଂଖ୍ୟକ ପଦକ୍ଷେପ ପରେ ସମାପ୍ତ ହେଉଥିବା ଆବଶ୍ୟକ ।
- **ପ୍ରଭାବଶାଳୀତା (Effectiveness):** ପ୍ରତ୍ୟେକ ପଦକ୍ଷେପ ଯଥେଷ୍ଟ ଭାବେ ଏପରି ମୌଳିକ ହେବା ଆବଶ୍ୟକ ଯେପରି ଏହା ନୀତିଗତ ଭାବେ ଜଣେ ବ୍ୟକ୍ତିଦ୍ୱାରା କେବଳ କାଗଜ ଏବଂ ପେନସିଲ୍ ବ୍ୟବହାର କରି ମଧ୍ୟ ଏହି ଆଲଗୋରିଦମକୁ ଚାଳନ କରାଯାଇପାରିବ । ଏହା ସହିତ, ପ୍ରତ୍ୟେକ ପଦକ୍ଷେପ କେବଳ ନିର୍ଦ୍ଦିଷ୍ଟ ନ ହୋଇ ସମ୍ଭବ ହେଉଥିବା ମଧ୍ୟ ଆବଶ୍ୟକ ।

ଏକ ଫ୍ଲୋ’ଚାର୍ଟ କିମ୍ବା ଏକ ସ୍ୟୁଡୋକୋଡ୍ ବ୍ୟବହାର କରି ଏକ ଆଲଗୋରିଦମ ଦର୍ଶାଯାଇପାରିବ ।

1.2.1 ଫ୍ଲୋ’ଚାର୍ଟ (Flowchart)

ଫ୍ଲୋ’ଚାର୍ଟ ହେଉଛି ଏକ ଆଲଗୋରିଦମର ଏକ ଚିତ୍ରାତ୍ମକ ପ୍ରତିନିଧି । ବିଭିନ୍ନ ପ୍ରକାରର ନିର୍ଦ୍ଦେଶକୁ ସୂଚିତ କରିବାପାଇଁ ଏହା ବିଭିନ୍ନ ଆକୃତି ବ୍ୟବହାର କରେ । ପ୍ରକୃତ ନିର୍ଦ୍ଦେଶାବଳିକୁ ସ୍ପଷ୍ଟ ଏବଂ ସଂକ୍ଷିପ୍ତ ବିବୃତ୍ତି ବ୍ୟବହାର କରି ଆକୃତିଗୁଡ଼ିକ ଭିତରେ ଲେଖାଯାଏ । ନିର୍ଦ୍ଦେଶାବଳି ଚାଳନ ହେବାର କ୍ରମ ସୂଚିତ କରିବାପାଇଁ ଏହି ଆକୃତିଗୁଡ଼ିକ ନିର୍ଦ୍ଦେଶିତ ରେଖାଦ୍ୱାରା ସଂଯୁକ୍ତ ହୋଇଥାଏ ।

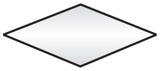
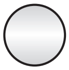
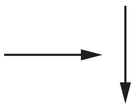


*ଚିତ୍ର **1.9:** ଏକ ଦ୍ୱିଘାତୀ ସମୀକରଣର ମୂଳର ପ୍ରକୃତି ଜାଣିବାପାଇଁ ଫ୍ଲୋ’ଚାର୍ଟ*

ଟେବୁଲ୍ 1.1 ଫ୍ଲୋ’ଚାର୍ଟରେ ବ୍ୟବହୃତ ବିଭିନ୍ନ ପ୍ରତୀକ, ସେଗୁଡ଼ିକ ନାମ ଏବଂ ସଂକ୍ଷିପ୍ତ ବିବରଣୀ ସହିତ ଦର୍ଶାଯାଇଅଛି ।

ଟେବୁଲ୍ 1.1: ବିଭିନ୍ନ ଫ୍ଲୋ’ଚାର୍ଟ ପ୍ରତୀକ ଏବଂ ସେଗୁଡ଼ିକର ସଂକ୍ଷିପ୍ତ ବର୍ଣ୍ଣନା

ପ୍ରତୀକ	ନାମ	ଉଦ୍ଦେଶ୍ୟ
	ବୃତ୍ତାଭାସ କ୍ଷେତ୍ର (Oval)	ଟର୍ମିନାଲ - ପ୍ରୋଗ୍ରାମ୍ ଲଜିକ୍ ପ୍ରବାହର ଆରମ୍ଭ ଏବଂ ଶେଷକୁ ଚିହ୍ନିତ କରିବାପାଇଁ ।
	ସମାନ୍ତରାଳ କ୍ଷେତ୍ର (Parallelogram)	ଇନପୁଟ୍/ଆଉଟପୁଟ୍ - ପ୍ରୋଗ୍ରାମ୍‌କୁ ଇନପୁଟ୍ କିମ୍ବା ପ୍ରୋଗ୍ରାମରୁ ଆଉଟପୁଟ୍ ସୂଚିତ କରିବାକୁ ।

	ଆୟତ କ୍ଷେତ୍ର (Rectangle)	ପ୍ରକ୍ରିୟାକରଣ - ଗଣିତ କାର୍ଯ୍ୟ ଏବଂ ତଥ୍ୟର ଗତିବିଧିକୁ ସୂଚିତ କରିବାପାଇଁ ।
	ହୀରା କ୍ଷେତ୍ର (Diamond)	ନିଷ୍ପତ୍ତି - ଗୋଟିଏ ବିନ୍ଦୁର ସୂଚନା ପାଇଁ ଯେଉଁଠାରେ ଗୋଟିଏ ବିକଳ୍ପକୁ ଶାଖା କରିବାକୁ ନିଷ୍ପତ୍ତି ନେବାକୁ ପଡ଼ିଥାଏ ।
	କ୍ଷୁଦ୍ର ବୃତ୍ତ (Small circle)	ସଂଯୋଜକ - ଗୋଟିଏ ଫ୍ଲୋ'ଚାର୍ଟର ବିଭାଗଗୁଡ଼ିକ ମଧ୍ୟରେ ଏକ ତାର୍କିକ ଲିଙ୍କ୍ ପ୍ରଦାନ କରିବାକୁ ।
	ନିର୍ଦ୍ଦେଶିତ ରେଖା (Directed lines)	ଫ୍ଲୋ ଧାଡ଼ି - ଯେଉଁ କ୍ରମରେ ନିର୍ଦ୍ଦେଶାବଳି ଚାଳନ କରାଯାଏ ତାହା ସୂଚିତ କରିବା ।

1.2.2 ସ୍ୟୁଡୋକୋଡ୍ (Pseudocode)

"ସ୍ୟୁଡୋ (pseudo)" ବା ସ୍ୟୁଡୋ ଶବ୍ଦର ଅର୍ଥ ନକଲି, ଅସତ୍ୟ, ବା ଛଦ୍ମ ଏବଂ "କୋଡ୍" ଶବ୍ଦ ଏକ ପ୍ରୋଗ୍ରାମିଂ ଭାଷାରେ ଲେଖାଯାଇଥିବା ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକୁ ସୂଚିତ କରେ । ତେଣୁ, ସ୍ୟୁଡୋକୋଡ୍ ହେଉଛି ପ୍ରକୃତ କମ୍ପ୍ୟୁଟର ନିର୍ଦ୍ଦେଶର ଏକ ନକଲିକରଣ । ସ୍ୟୁଡୋ ନିର୍ଦ୍ଦେଶାବଳି ହେଉଛି ଇଂରାଜୀରେ ଲେଖାଯାଉଥିବା ବିବୃତ୍ତି ପରି ଦେଖାଯାଉଥିବା ଖଣ୍ଡବାକ୍ୟ । ପ୍ରୋଗ୍ରାମ୍ ତର୍କର ବର୍ଣ୍ଣନା କରିବାକୁ ଫ୍ଲୋ'ଚାର୍ଟ ପରି ପ୍ରତୀକ ବ୍ୟବହାର କରିବା ପରିବର୍ତ୍ତେ, ସ୍ୟୁଡୋକୋଡ୍, କମ୍ପ୍ୟୁଟର ନିର୍ଦ୍ଦେଶାବଳି ସଦୃଶ ଏକ ସଂରଚନା ବ୍ୟବହାର କରେ । କାରଣ, ଏହା ପ୍ରୋଗ୍ରାମ୍‌ର ଡିଜାଇନ୍ ଉପରେ ଗୁରୁତ୍ୱ ଦେଇଥାଏ । ସ୍ୟୁଡୋକୋଡ୍‌କୁ ମଧ୍ୟ ପ୍ରୋଗ୍ରାମ ଡିଜାଇନ୍ ଭାଷା (PDL) କୁହାଯାଏ ।

ସ୍ୟୁଡୋକୋଡ୍/ଛଦ୍ମକୋଡ୍ ନିମ୍ନପ୍ରଦତ୍ତ ମୌଳିକ ଲଜିକ୍ ସଂରଚନାରେ ଗଠିତ ଯାହା ଯେକୌଣସି କମ୍ପ୍ୟୁଟର ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବାପାଇଁ ଯଥେଷ୍ଟ ବୋଲି ପ୍ରମାଣିତ ହୋଇଛି:

1. ସିକ୍ୱେନ୍ସ (Sequence) ବା ଅନୁକ୍ରମ,
2. ସିଲେକ୍ସନ୍ (Selection) ବା ଚୟନ (If . . . Endif, If . . . Else . . . Endif, If . . . Else If . . . Endif),
3. ଆଇଟରେସନ୍ (Iteration) ବା ପୁନରାବୃତ୍ତି (While . . . Endwhile, Do . . . While) ।

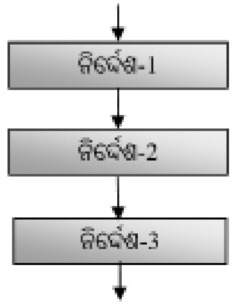
ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକୁ ଗୋଟିଏ ପରେ ଗୋଟିଏ ଅନୁକ୍ରମରେ ସମ୍ପାଦନ କରିବାପାଇଁ ସିକ୍ୱେନ୍ସ ଲଜିକ୍‌ର (logic) ବ୍ୟବହାର ହୁଏ । ତେଣୁ, ସିକ୍ୱେନ୍ସ ଲଜିକ୍ ବା ଲଜିକ୍ ଅନୁକ୍ରମପାଇଁ, ସ୍ୟୁଡୋକୋଡ୍ ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକ ଯେଉଁକ୍ରମରେ ଚାଳନ କରାଯିବ ସେହି କ୍ରମରେ ଲେଖାଯାଇଛି ।

ଲଜିକ୍‌ର ପ୍ରବାହ ଉପରୁ ତଳକୁ ହୁଏ ।

```
:
ନିର୍ଦ୍ଦେଶ-1

ନିର୍ଦ୍ଦେଶ-2

ନିର୍ଦ୍ଦେଶ-3
:
:
```



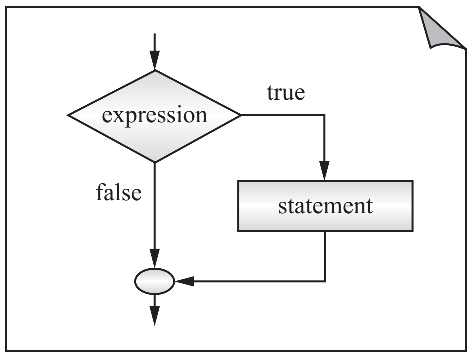
ଚିତ୍ର 1.10: *କ୍ରମ ସଂରଚନାପାଇଁ ସ୍ୟୁଡୋକୋଡ୍ ଏବଂ ଫ୍ଲୋ'ଚାର୍ଟ*

ଚୟନ ତର୍କ (selection logic), ନିଷ୍ପତ୍ତି ତର୍କ (decision logic) ଭାବରେ ମଧ୍ୟ ଜଣାଶୁଣା, ଏହା ନିଷ୍ପତ୍ତି ନେବାପାଇଁ ବ୍ୟବହୃତ ହୁଏ । ପ୍ରୋଗ୍ରାମ୍ ତର୍କରେ ବିକଳ୍ପ ପଥଗୁଡ଼ିକ ମଧ୍ୟରୁ ଗୋଟିଏ ଉପଯୁକ୍ତ ପଥ ଚୟନ କରିବାପାଇଁ ଏହା ବ୍ୟବହୃତ ହୋଇଥାଏ ।

ଚୟନ ତର୍କକୁ ବିଭିନ୍ନ ସଂରଚନାରେ ଲେଖାଯାଇପାରେ, ଯେପରିକି If … Endif or If … Else … Endif ବା If … Else If … Endif structure ଇତ୍ୟାଦି ।

The If … Endif ସଂରଚନା କହେ ଯେ ଯଦି ବିବୃତ୍ତି (expression) ସତ୍ୟ, ତେବେ ବିବୃତ୍ତିକୁ ଚାଳନ କର ଅନ୍ୟଥା (ଯଦି ଅସତ୍ୟ) ବିବୃତ୍ତିଟିକୁ ଚାଳନ ନ କରି ଡେଇଁ ଯାଅ ।

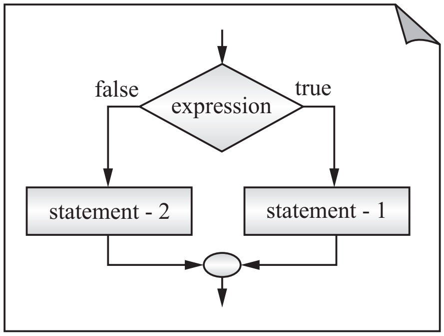
```
    :
    :
If (expression) then
  statement
Endif
    :
    :
```



ଚିତ୍ର 1.11: *If... Endif ପାଇଁ ଚୟନ ସଂରଚନାର ସ୍ୟୁଡୋକୋଡ୍ ଏବଂ ଫ୍ଲୋ'ଚାର୍ଟ*

If … Else … Endif ସଂରଚନା କହେ ଯେ ଯଦି ପରିପ୍ରକାଶ ସତ୍ୟ ତେବେ ବିବୃତ୍ତି-1 ଚାଳନ କରିଥାଏ, ଅନ୍ୟଥା (ଯଦି ପରିପ୍ରକାଶ ଅସତ୍ୟ) ବିବୃତ୍ତି-2 ଚାଳନ କରେ ।

```
     :
     :
If (expression) then
   statement-1
Else
   statement-2
Endif
     :
     :
```



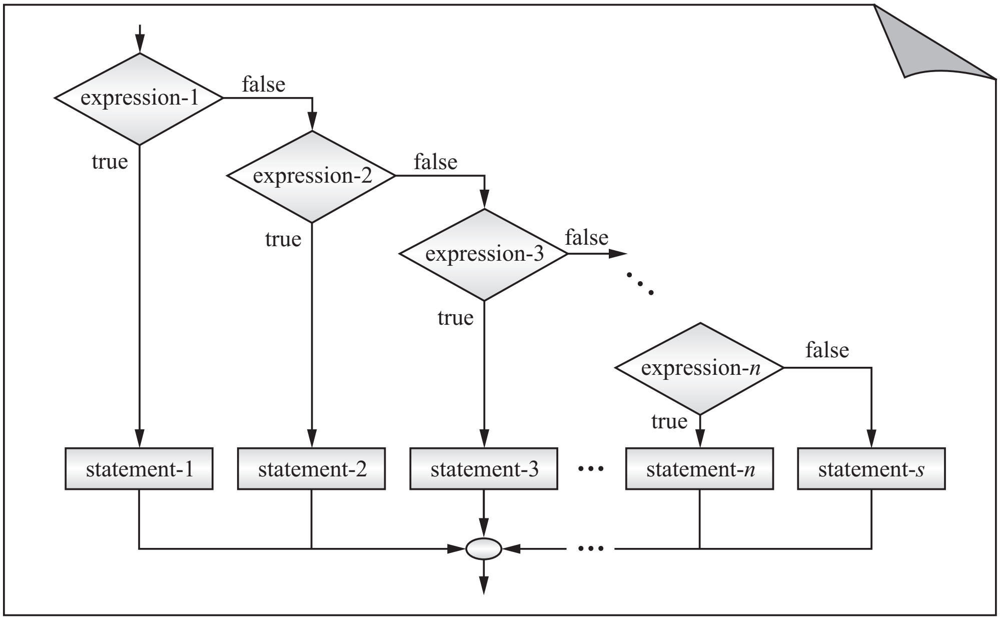
***ଚିତ୍ର 1.12:** If . . . Else . . . Endif ପାଇଁ ଚୟନ ସଂରଚନାର ସ୍ୟୁଡୋକୋଡ୍ ଏବଂ ଫ୍ଲୋ'ଚାର୍ଟ*

ପରୀକ୍ଷଣ ହୋଇଥିବା ପରିପ୍ରକାଶର ପରିଣାମ ଆଧାରରେ, ଯଦି ଏକାଧିକ ବିକଳ୍ପ (ଚାଳନ ପଥ) ଥାଏ, ତେବେ Else If ସିଡ଼ି ଏକ ବହୁତ ସହଜ ନିର୍ମାଣ ।

ବିବୃତ୍ତି କ୍ରମରେ ମୂଲ୍ୟାଙ୍କନ କରାଯାଏ, ଏବଂ ଯଦି କୌଣସି ବିବୃତ୍ତି ସତ୍ୟ ତେବେ ଏହା ସହିତ ଜଡ଼ିତ ବିବୃତ୍ତି ବ୍ଲକ୍‌ଟିର ଚାଳନ ହୁଏ, ଏବଂ ଏହା ସମଗ୍ର ଶୃଙ୍ଖଳାକୁ ସମାପ୍ତ କରେ । ଶେଷ Else ଅଂଶଟି 'ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ' କିମ୍ବା 'ଡିଫଲ୍ଟ ମାମଲା' କୁ ସମ୍ଭାଳିଥାଏ ଯେଉଁଠାରେ ଉପରୋକ୍ତ କୌଣସି ନିର୍ଦ୍ଦିଷ୍ଟ ପରିପ୍ରକାଶ ସନ୍ତୁଷ୍ଟ ନ ଥାଏ ।

```
If (expression-1) then
   statement-1
Else If (expression-2) then
   statement-2
Else If (expression-3) then
   statement-3
      :
Else If (expression-n) then
      (expression-n)
Else
      statement-s
Endif
```

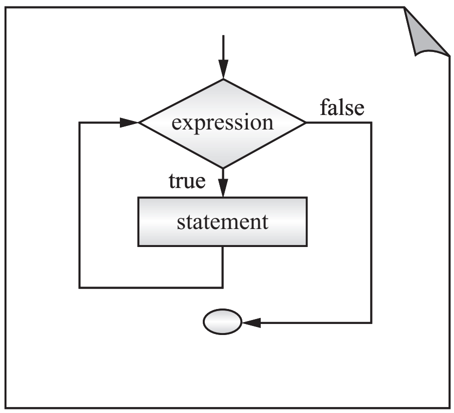
***ଚିତ୍ର 1.13:** Else If ladder ସିଡ଼ିର ସିଣ୍ଟାକ୍ସ*



ଚିତ୍ର 1.14: Else If ladder ସିଡ଼ିର ଲଜିକ୍ ପ୍ରବାହ

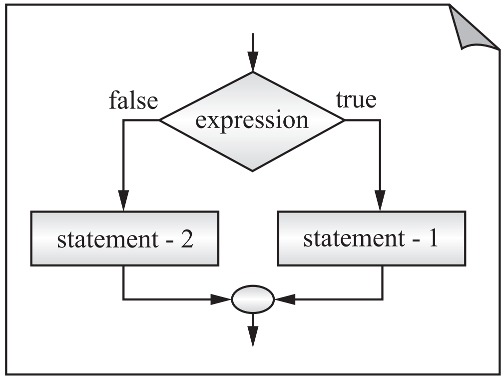
ଯେତେବେଳେ କିଛି ପରିପ୍ରକାଶ ଉପରେ ନିର୍ଭର କରି ଏକ କିମ୍ବା ଏକାଧିକ ନିର୍ଦ୍ଦେଶାବଳିକୁ ଅନେକ ଥର ଚାଳନ କରିବାକୁ ହୁଏ ସେତେବେଳେ ଲୁପ୍ ପ୍ରସ୍ତୁତ କରିବାକୁ ଆଇଟରେଟିଭ୍ ଲଜିକ୍ ବ୍ୟବହୃତ ହୁଏ । ଏହା ଦୁଇଟି ସଂରଚନାର ବ୍ୟବହାର କରେ ଯଥା While ... Endwhile and Do ... While. ।

```
     :
     :
While (expression)
   statement
Endwhile
     :
     :
```



ଚିତ୍ର 1.15: while....Endwhile ସଂରଚନାପାଇଁ ସ୍ୟୁଡୋକୋଡ୍ ଏବଂ ଫ୍ଲୋ’ଚାର୍ଟ

```
    :
    :
If (expression) then
  statement-1
Else
  statement-2
Endif
    :
    :
```



ଚିତ୍ର 1.16: Do ... While ସଂରଚନାପାଇଁ ସ୍ୟୁଡୋକୋଡ୍ ଏବଂ ଫ୍ଲୋ'ଚାର୍ଟ

ସେଗୁଡ଼ିକ ମଧ୍ୟରେ ଏକମାତ୍ର ପାର୍ଥକ୍ୟ ହେଉଛି Do ... While ଲୁପ୍‌ର ଶେଷରେ ପରିପ୍ରକାଶର ପରୀକ୍ଷଣ ହେଉଥିବାରୁ ଲୁପ୍ ସର୍ବଦା ଅତିକମ୍‌ରେ ଥରେ ଚାଳନ ହୁଏ, ଯେତେବେଳେ କି While... Endwhile ଲୁପ୍‌ଟି ଥରୁଟିଏ ମଧ୍ୟ ଚାଳନ ନ ହୋଇ ପାରେ କାରଣ ଏଥିରେ ଆରମ୍ଭରେ ପରିପ୍ରକାଶର ପରୀକ୍ଷଣ ହୁଏ ଏବଂ ପ୍ରଥମ ଥରରେ ହିଁ ବିଫଳ ହୋଇ ଯାଇପାରେ ।

ସ୍ୟୁଡୋକୋଡ୍‌ର ବର୍ଣ୍ଣନା (Pseudocode Description)

ଟିପ୍ପଣୀ (Comments)

ପ୍ରତ୍ୟେକ ନିର୍ଦ୍ଦେଶା ପରେ ଏକ ଟିପ୍ପଣୀ କରାଯାଇପାରେ । ଟିପ୍ପଣୀଗୁଡ଼ିକ ଦୁଇଟି ସ୍ଲାସ୍ ("//") ସହିତ ଆରମ୍ଭ ହୁଏ, ଏବଂ ନିର୍ଦ୍ଦେଶର ଉଦ୍ଦେଶ୍ୟ ବ୍ୟାଖ୍ୟା କରେ, ଯେପରିକି

Read: n   // Enter the value of variable n

ଟିପ୍ପଣୀ ଉପଯୁକ୍ତ ବ୍ୟବହାର ସ୍ୟୁଡୋକୋଡ୍‌ର ପଠନୀୟତାକୁ ବଢାଇଥାଏ, ଯାହା ପରବର୍ତ୍ତୀ ସମୟରେ ସ୍ୟୁଡୋକୋଡ୍ ରକ୍ଷଣାବେକ୍ଷଣରେ ସାହାଯ୍ୟ କରେ ।

ଚଳଗୁଡ଼ିକର ନାମ (Variable Names)

ଚଳଗୁଡ଼ିକର ନାମ ପାଇଁ, ଆମେ ଇଟାଲିକ୍ ହୋଇଥିବା ସାନ (lower case) ରୋମାନ୍ ଅକ୍ଷରଗୁଡ଼ିକ ବ୍ୟବହାର କରି ପାରିବା ଯେପରିକି max, loc, ଇତ୍ୟାଦି । ନିର୍ଦ୍ଧାରିତ ଧ୍ରୁବକ ପାଇଁ, ଆମେ ବଡ (upper case) ରୋମାନ୍ ଅକ୍ଷର ବ୍ୟବହାର କରିପାରିବା ।

ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତି (Assignment Statement)

ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତି ଏହିପରି ସଂକେତ ବ୍ୟବହାର କରିବ

Set max = a

a ର ମୂଲ୍ୟକୁ max ରେ ନ୍ୟସ୍ତ କରିପାରିବା । ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତିର ଡାହାଣ ପାର୍ଶ୍ୱରେ ଏକ ମୂଲ୍ୟ, ଏକ ଚଳ କିମ୍ବା

ଏକ ପରିପ୍ରକାଶ ରହିପାରେ ।

କିନ୍ତୁ, ଯଦି ଅନେକ ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତି ଗୋଟିଏ ଧାଡ଼ି ରେ ଲେଖାଥାଏ,

Set k = 1, loc = 1, max = a_i

ତା'ହେଲେ ସେଗୁଡ଼ିକ ବାମରୁ ଡାହାଣକୁ ଚାଳନ ହୋଇଥାନ୍ତି ।

ଇନପୁଟ୍/ଆଉଟପୁଟ୍ (Input/Output)

ଡାଟାକୁ ନିମ୍ନ ଫର୍ମାଟ୍‌ରେ ଏକ read ବିବୃତ୍ତି ମାଧ୍ୟମରେ ଇନପୁଟ୍ କରାଯାଇପାରେ ଏବଂ ଏକ ଚଳକୁ ନ୍ୟସ୍ତ କରାଯାଇପାରେ ।

Read: *Variable list*

ଯେଉଁଠାରେ Variable list କମାଦ୍ୱାରା (,) ଅଲଗା ହୋଇଥିବା ଏକ କିମ୍ବା ଅଧିକ ଚଳକୁ ନେଇ ଗଠିତ ।

ସେହିଭଳି, ଚଳଦ୍ୱାରା ଧାରଣ ହୋଇଥିବା ଡାଟା ଏବଂ ଡବଲ୍ କୋଟ୍‌ରେ (" ")ଆବଦ୍ଧ ସନ୍ଦେଶ, ଏକ ପ୍ରିଣ୍ଟ ବିବୃତ୍ତି ମାଧ୍ୟମରେ ନିମ୍ନ ଫର୍ମାଟ୍‌ରେ ଆଉଟପୁଟ୍ ହୋଇପାରେ ।

Print: *message and/or Variable list*

ଯେଉଁଠାରେ message ଏବଂ Variable list ର ଚଳ କମାଦ୍ୱାରା ଅଲଗା ହୋଇଥାନ୍ତି ।

ନିର୍ଦ୍ଦେଶାବଳିଗୁଡ଼ିକର ଚାଳନ (Execution of Instructions)

ନିର୍ଦ୍ଦେଶାବଳିଗୁଡ଼ିକ ସାଧାରଣତଃ ସ୍ୟୁଡୋକୋଡ୍‌ର ଲେଖାକ୍ରମରେ ଗୋଟିଏ ପରେ ଗୋଟିଏ କରି ଚାଳନ ହୋଇଥାଏ । କିନ୍ତୁ, ଏପରି କିଛି ନଜିର ଥାଇପାରେ ଯେତେବେଳେ କିଛି ପରିପ୍ରକାଶର ଫଳସ୍ୱରୂପ କିଛି ନିର୍ଦ୍ଦେଶାବଳି ଡେଇଁ ହୋଇଯାଏ କିମ୍ବା କିଛି ନିର୍ଦ୍ଦେଶାବଳିର ପୁନରାବୃତ୍ତି ହୋଇପାରେ ।

ଆଲଗୋରିଦମର ପରିସମାପ୍ତି (Completion of the Algorithm)

ଶେଷ ନିର୍ଦ୍ଦେଶର ଚାଳନ ସହିତ ଏକ ସ୍ୟୁଡୋକୋଡ୍‌ର ପରିସମାପ୍ତି ଘଟିଥାଏ । ତାହା ସତ୍ତ୍ୱେ, exit ନିର୍ଦ୍ଦେଶ ବ୍ୟବହାର କରି ଯେକୌଣସି ମଧ୍ୟବର୍ତ୍ତୀ ଅବସ୍ଥାରୁ ମଧ୍ୟ ଶେଷ କରାଯାଇପାରିବ ।

ଦ୍ୱିଘାତୀ ସମୀକରଣର ମୂଳର ପ୍ରକୃତି ପ୍ରଦର୍ଶନ କରିବାକୁ ସ୍ୟୁଡୋକୋଡ୍

$ax^2 + bx + c = 0$ provided $a \neq 0$.

ସ୍ୟୁଡୋକୋଡ୍ 1.1

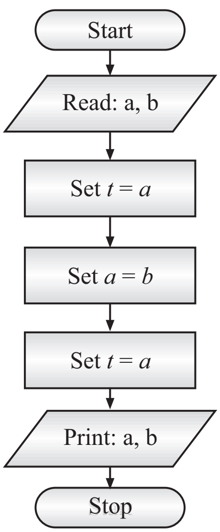
```
Begin
     Read: a, b, c
     Set disc = b^2 - 4ac
     If ( disc = 0 ) then
           Print: "Roots are real and equal"
     Else If  ( disc > 0 ) then
           Print: "Roots are real and distinct"
     Else
           Print: "Roots are imaginary"
     Endif
End.
```

ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 1.1: ଏକ ଫ୍ଲୋ'ଚାର୍ଟ ଅଙ୍କନ କର ଏବଂ ଦୁଇଟି ଚଳ a ଏବଂ b କୁ ଅଦଳବଦଳ କରିବାକୁ ଏକ ସ୍ୟୁଡୋକୋଡ୍ ଲେଖ ।

ସମାଧାନ: ଏକ ପରିଦୃଶ୍ୟ ବିଷୟରେ ଚିନ୍ତା କର – ଆମ ପାଖରେ ଗୋଟିଏ ଗ୍ଲାସରେ ପାଣି ଏବଂ ଅନ୍ୟ ଏକ ଗ୍ଲାସରେ ଜୁସ୍ ଅଛି । ଆମେ ଏବେ ଜୁସ୍ ଥିବା ଗ୍ଲାସରେ ପାଣି, ଏବଂ ପାଣି ଥିବା ଗ୍ଲାସରେ ଜୁସ୍ ରଖିବାକୁ ଚାହୁଁଛୁ । ଏହି କାର୍ଯ୍ୟଟି କିପରି ସମାପନ ହୋଇପାରିବ?

ସମାନ ଉପାୟରେ, ଆମକୁ ଏକ ତୃତୀୟ ଚଳ t, ବ୍ୟବହାର କରିବାକୁ ପଡିବ ଯାହାଦ୍ୱାରା ନିମ୍ନଲିଖିତ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍‌ରେ ପ୍ରଦର୍ଶିତ ଦୁଇଟି ଚଳ ମୂଲ୍ୟର ଅଦଳବଦଳକୁ ସୁଗମ କରିପାରିବା ।



Pseudocode 1.2

Begin
  Read: *a*, *b*
  Set *t* = *a*
  Set *a* = *b*
  Set *b* = *t*
  Print: *a*, *b*
End.

ଚିତ୍ର 1.17: ଦୁଇଟି ଚଳ ଅଦଳବଦଳ କରିବାକୁ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍

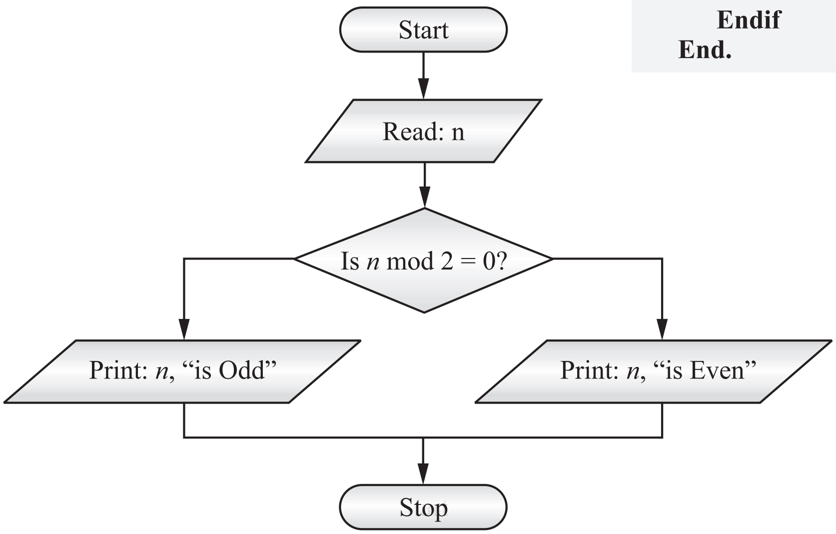
ଉଦାହରଣ 1.2: ଦିଆଯାଇଥିବା ଗଣନ ସଂଖ୍ୟା 'n' ଯୁଗ୍ମ କି ଅଯୁଗ୍ମ ତାହା ପରୀକ୍ଷା କରିବାକୁ ଏକ ଫ୍ଲୋ'ଚାର୍ଟ ଅଙ୍କନ କର ଏବଂ ଏହାର ଏକ ସ୍ୟୁଡୋକୋଡ୍ ଲେଖ ।

ସମାଧାନ: ତୁମେ ସମସ୍ତେ ଜାଣିଥିବ ଯେ କୌଣସି ଗଣନ ସଂଖ୍ୟା ଯୁଗ୍ମ ହୋଇଥାଏ ଯଦି ଏହା 2 ଦ୍ୱାରା ବିଭାଜ୍ୟ, ଅର୍ଥାତ, 2 ଦ୍ୱାରା ଭାଗ କଲେ ଭାଗଶେଷ 0 ଦେଇଥାଏ । ଭାଗଶେଷ ପ୍ରାପ୍ତ କରିବାର କାର୍ଯ୍ୟକୁ ମଡ୍ୟୁଲୋ ଅପରେସନ୍ (modulo) ବା ସଂକ୍ଷିପ୍ତରେ ମଡ୍ (mod) କୁହାଯାଏ ।

ନିମ୍ନପ୍ରଦତ୍ତ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ ଲଜିକ୍ ଦର୍ଶାଉଛି ।

Pseudocode 1.3

```
Begin
    Read: n
    If (n mod 2 = 0) then
        Print n = "is Even"
    Else
        Print n = "is Odd"
    Endif
End.
```



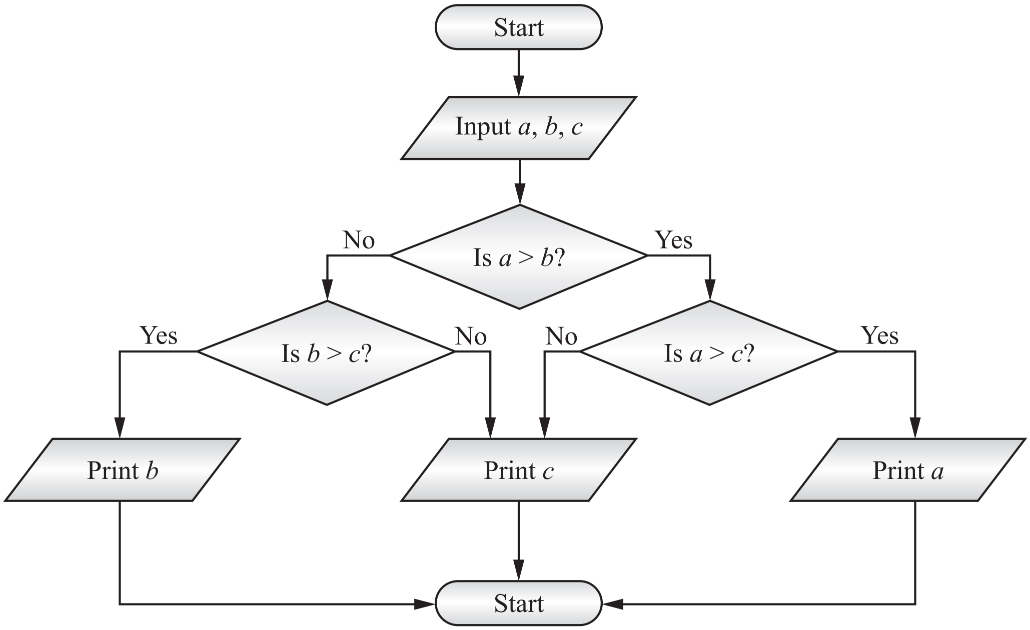
ଚିତ୍ର 1.18: ଦିଆଯାଇଥିବା ସଂଖ୍ୟା ଯୁଗ୍ମ କି ଅଯୁଗ୍ମ ପରୀକ୍ଷା କରିବାକୁ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍

ଉଦାହରଣ 1.3: ତିନୋଟି ସଂଖ୍ୟା a, b, c ମଧ୍ୟରୁ ସର୍ବବୃହତଟିକୁ ପାଇବାପାଇଁ ଏକ ଫ୍ଲୋ'ଚାର୍ଟ ଅଙ୍କନ କର ଏବଂ ଏକ ସ୍ୟୁଡୋକୋଡ୍ ଲେଖ ।

ସମାଧାନ: ଆମେ ପ୍ରଥମେ a ସହ b କୁ ତୁଳନା କରୁ, ଯଦି a, b ଠାରୁ ଅଧିକ ବେବେ ଆମେ a ସହ c କୁ ତୁଳନା କରୁ । ଯଦି a, c ଠାରୁ ଅଧିକ, ତା'ହେଲେ a କୁ ସର୍ବବୃହତ୍ ସଂଖ୍ୟା ଭାବରେ ନିଆଯାଏ ଅନ୍ୟଥା ଆମେ c କୁ ସର୍ବବୃହତ ସଂଖ୍ୟା ଭାବରେ ଗ୍ରହଣ କରୁ ।

କିନ୍ତୁ, ଯଦି a, b ଠାରୁ ଅଧିକ ନୁହେଁ ଆମେ b ସହ c କୁ ତୁଳନା କରୁ । ଯଦି b, c ଠାରୁ ଅଧିକ ତା'ହେଲେ b କୁ ସର୍ବବୃହତ ସଂଖ୍ୟା ଭାବରେ ଆମେ ଗ୍ରହଣ କରୁ ଅନ୍ୟଥା ଆମେ c କୁ ସର୍ବବୃହତ ସଂଖ୍ୟା ଭାବରେ ଗ୍ରହଣ କରୁ ।

ନିମ୍ନପ୍ରଦତ୍ତ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ ଏହି ତର୍କକୁ ଦର୍ଶାଏ ।



ଚିତ୍ର 1.19: ତିନୋଟି ସଂଖ୍ୟା ମଧ୍ୟରୁ ସର୍ବବୃହତ୍‌ଟିକୁ ଖୋଜିବାପାଇଁ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍

ସ୍ୟୁଡୋକୋଡ୍ 1.4

```
Begin
      Read: a, b, c
      If  ( a  > b )
             If  ( a  > c ) then
                     Print: a
             Else
                     Print: c
             Endif
      Else
             If  ( b  > c ) then
                     Print: b
             Else
                     Print: c
             Endif
      Endif
 End.
```

ଉଦାହରଣ 1.4: ଏକ ବିଷୟରେ ମାର୍କର ଶତକଡା (percentage) ଉପରେ ଆଧାର କରି, ନିମ୍ନକ୍ତୋ ପରୀକ୍ଷା ନୀତି ଅନୁଯାୟୀ ଜଣେ ଛାତ୍ରଙ୍କୁ ଅକ୍ଷର ଗ୍ରେଡ୍ ଦିଆଯାଇଛି:

Percentage of Marks	Grade
percentage ≥ 90	A+
90 > percentage ≥ 80	A
80 > percentage ≥ 70	B
70 > percentage ≥ 60	C
60 > percentage ≥ 50	D
percentage < 50	F

ଛାତ୍ରଜଣେର ମାର୍କ ପ୍ରତିଶତକୁ ଅଧାରକରି ଅକ୍ଷର ଗ୍ରେଡ୍ ନ୍ୟସ୍ତ କରିବାକୁ ଏକ ସ୍ୟୁଡୋକୋଡ୍ ଲେଖ ।

Pseudocode 1.5

```
Begin
      Read: percentage
      If ( percentage >= 90 )
             Print: "Grade = A+"
      Else If ( percentage >= 80 )
             Print: "Grade = A"
      Else If ( percentage >= 70 )
             Print: "Grade = B"
      Else If ( percentage >= 60 )
             Print: "Grade = C"
      Else If ( percentage >= 50 )
             Print: "Grade = D"
      Else
             Print: "Grade = F"
      Endif
 End.
```

ଉଦାହରଣ 1.5: ଜଣେ ବିକ୍ରେତାଙ୍କଦ୍ୱାରା ବିକ୍ରୟ ଉପରେ କମିସନ୍ (Commission) ନିମ୍ନପ୍ରଦତ୍ତ ନୀତି ଅନୁସାରେ ହିସାବ କରାଯାଏ:

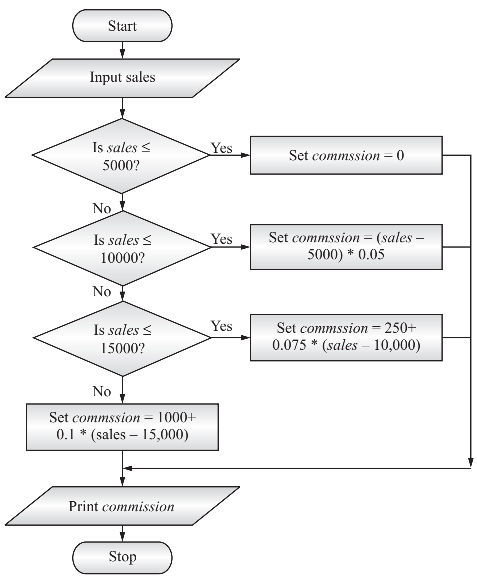
Amount of Sale (in Rs.)	Commision Rate
0 – 5000	Nil
5001 – 10000	5 % EXCESS OF 5000
10001 – 15000	7.5 % EXCESS OF 10000
> 15000	10 % EXCESS OF 15000

ଜଣେ ବିକ୍ରେତାଙ୍କଦ୍ୱାରା ବିକ୍ରି କରାଯାଇଥିବା ପରିମାଣ ଦର୍ଶାଉଥିବା ଏବଂ ଧାର୍ଯ୍ୟ କମିଶନ ପ୍ରଦର୍ଶନ କରୁଥିବା ଏକ ଫ୍ଲୋଚଚାର୍ଟ ଅଙ୍କନ କରି ସ୍ୟୁଡୋକୋଡ ପ୍ରସ୍ତୁତ କର ।

ସମାଧାନ:

ସ୍ୟୁଡୋକୋଡ୍ 1.6

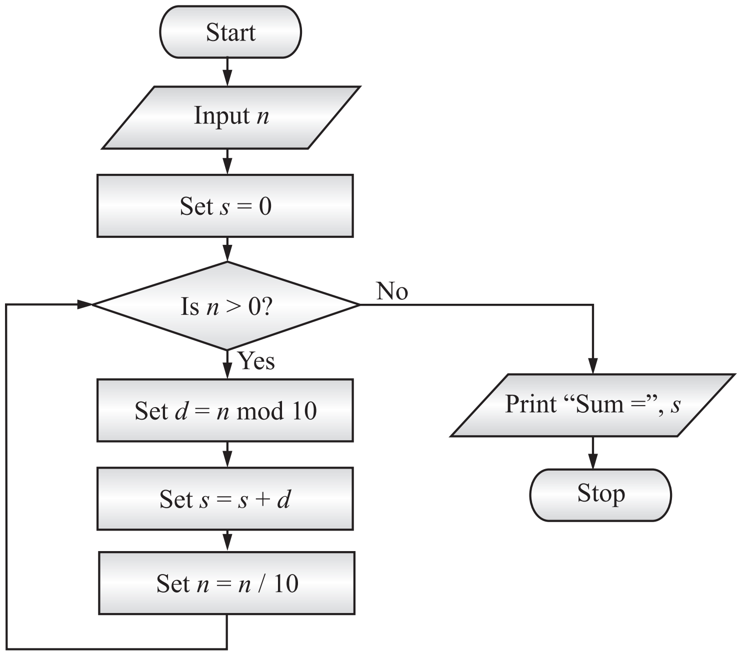
```
Begin
      Read: sales
      If ( sales <= 5000 )
            Set commission = 0
      Else If (sales <= 10000  )
            Set commission = ( sales – 5000 ) * 0.05;
      Else If ( sales <= 15000 )
            Set commission = 250 + ( sales – 10000 ) *
0.075;
      Else
            Set commission = 1000 + ( sales – 15000 ) *
0.1;
      Endif
      Print: "Computed commission = ", commission
 End.
```



ଚିତ୍ର 1.20: କମିଶନ ହିସାବପାଇଁ ଫ୍ଲୋ'ଚାର୍ଟ

ଉଦାହରଣ 1.6: ଏକ ସଂଖ୍ୟା n ର ଅଙ୍କଗୁଡ଼ିକର ସମଷ୍ଟି ପାଇବାପାଇଁ ଏକ ଫ୍ଲୋ'ଚାର୍ଟ ଅଙ୍କନ କର ଏବଂ ଏକ ସ୍ୟୁଡୋକୋଡ୍ ଲେଖ ।

ସମାଧାନ:



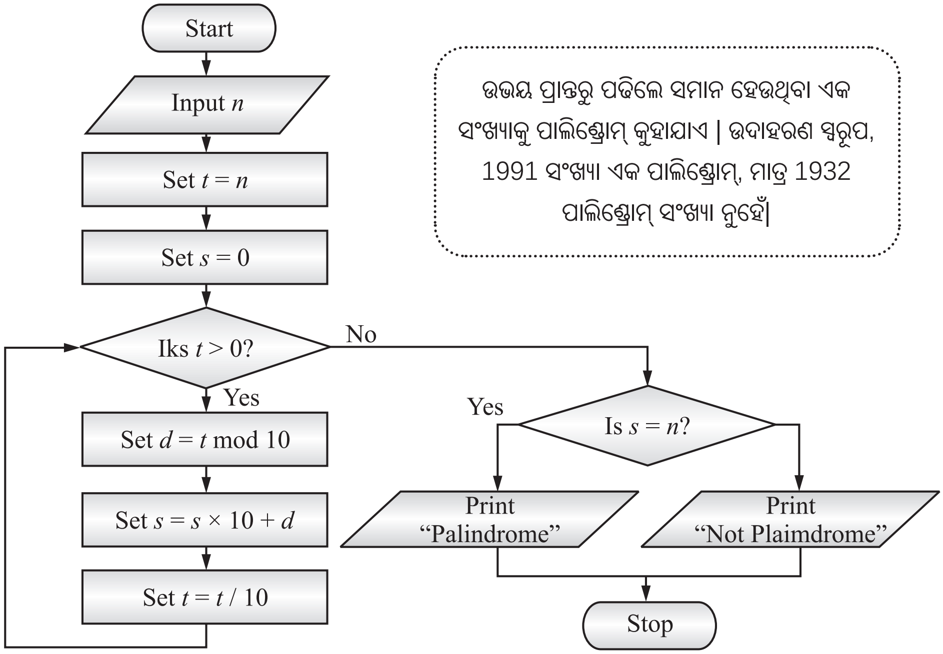
ଚିତ୍ର 1.21: ଏକ ସଂଖ୍ୟାର ଅଙ୍କଗୁଡ଼ିକର ସମଷ୍ଟି ପାଇବାପାଇଁ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍

ସ୍ୟୁଡୋକୋଡ୍ 1.7

```
Begin
      Read: n
      Set s = 0
      While  ( n > 0 ) do
            Set d = n mod 10
            Set s = s + d
            Set n = n / 10
      Endwhile
      Print: "Sum = ", s
 End.
```

ଉଦାହରଣ 1.7: ଦିଆଯାଇଥିବା ସଂଖ୍ୟା n ଟି ପାଲିଣ୍ଡ୍ରୋମ୍ (palindrome) କି ନୁହେଁ ଜାଣିବାପାଇଁ ଏକ ଫ୍ଲୋ'ଚାର୍ଟ ଅଙ୍କନ କର ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ ଲେଖ ।

ସମାଧାନ:



ଉଭୟ ପ୍ରାନ୍ତରୁ ପଢିଲେ ସମାନ ହେଉଥିବା ଏକ ସଂଖ୍ୟାକୁ ପାଲିଣ୍ଡ୍ରୋମ୍ କୁହାଯାଏ | ଉଦାହରଣ ସ୍ୱରୂପ, 1991 ସଂଖ୍ୟା ଏକ ପାଲିଣ୍ଡ୍ରୋମ୍, ମାତ୍ର 1932 ପାଲିଣ୍ଡ୍ରୋମ୍ ସଂଖ୍ୟା ନୁହେଁ|

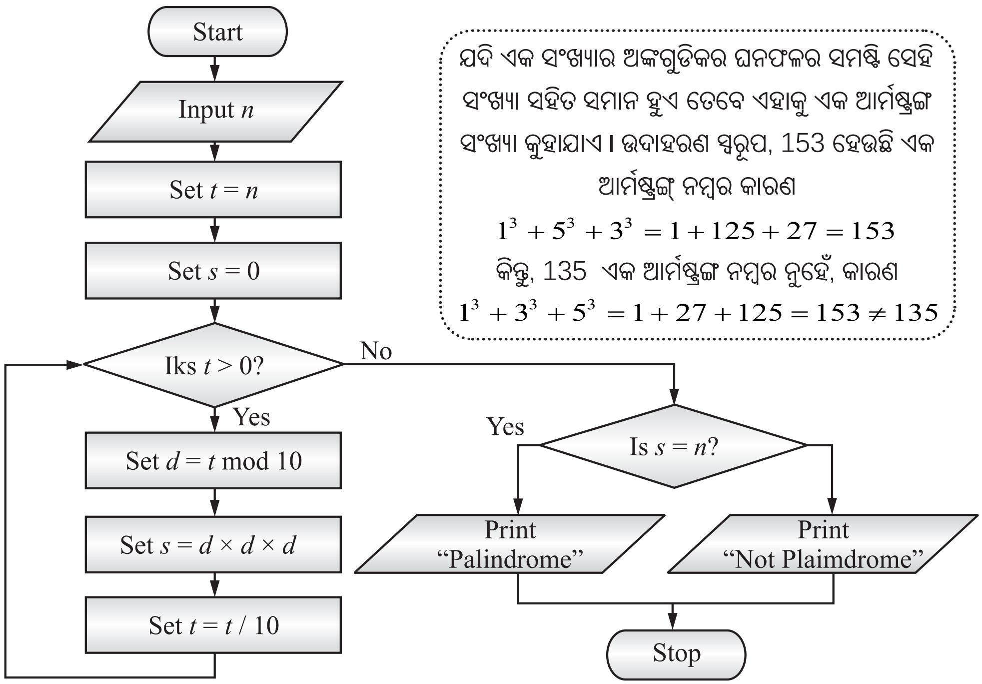
ଚିତ୍ର 1.22: ଦିଆଯାଇଥିବା ସଂଖ୍ୟା n ଟି ପାଲିଣ୍ଡ୍ରୋମ୍ କି ନୁହେଁ ଜାଣିବାକୁ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍

ସ୍ୟୁଡୋକୋଡ୍ 1.8

```
Begin
      Read: n
      Set t = n,  s = 0
      While  ( t > 0 ) do
            Set d = t mod 10
            Set s = s × 10 + d
            Set t = t / 10
      Endwhile
      If  ( s = n ) then
            Print: "Palindrome"
      Else
            Print: "Not a Palindrome"
      Endif
End.
```

ଉଦାହରଣ 1.8: ଦିଆଯାଇଥିବା ସଂଖ୍ୟା n ଟି ଏକ ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ (Armstrong) ନମ୍ବର କିମ୍ବା ନୁହେଁ ଜାଣିବାପାଇଁ ଫ୍ଲୋ'ଚାର୍ଟ ଅଙ୍କନ କର ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ ଲେଖ ।

ସମାଧାନ:



ଚିତ୍ର 1.23: ଦିଆଯାଇଥିବା ସଂଖ୍ୟା n ଟି ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ ସଂଖ୍ୟା କି ନୁହେଁ ଯାଞ୍ଚ କରିବାକୁ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍

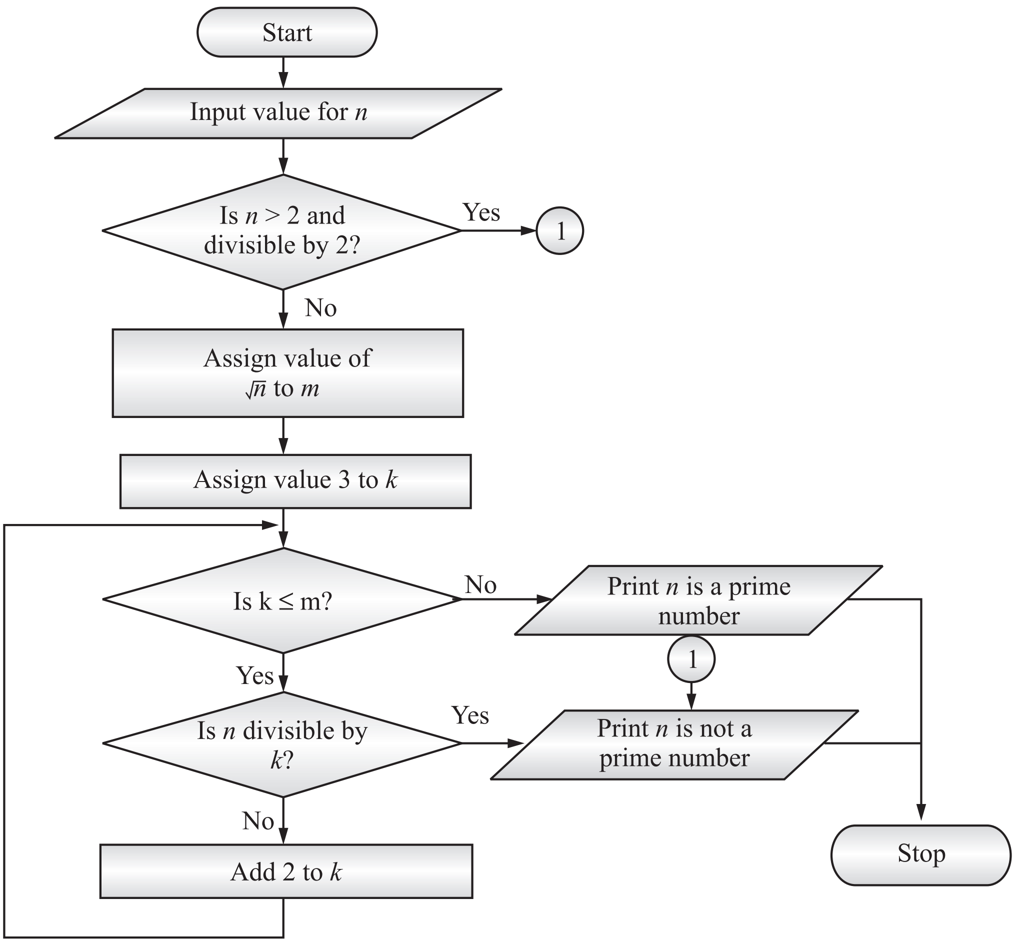
ସ୍ୟୁଡୋକୋଡ୍ 1.9

```
 Begin
     Read: n
     Set t = n,  s = 0
     While  ( t > 0 ) do
           Set d = t mod 10
           Set s = s  + d × d × d
           Set t = t / 10
     Endwhile
     If  ( s = n ) then
            Print: "Armstrong"
     Else
            Print: "Not Armstrong"
     Endif
End.
```

ଉଦାହରଣ 1.9: ଦିଆଯାଇଥିବା ଗଣନ ସଂଖ୍ୟା n ଏକ ମୌଳିକ (prime) ସଂଖ୍ୟା କି ନୁହେଁ ଜାଣିବାପାଇଁ ଏକ ଫ୍ଲୋ'ଚାର୍ଟ ଅଙ୍କନ କର।

ସମାଧାନ: ଏକ ଗଣନ ସଂଖ୍ୟାକୁ ମୌଳିକ ବୋଲି କୁହାଯିବ ଯଦି ଏହା କେବଳ ମାତ୍ର 1 ଓ ନିଜଦ୍ୱାରା ବିଭାଜ୍ୟ ହୁଏ ଏବଂ ଅନ୍ୟ କୌଣସି ସଂଖ୍ୟାଦ୍ୱାରା ବିଭାଜ୍ୟ ହୁଏ ନାହିଁ, ଅର୍ଥାତ ଏହାର ଉତ୍ପାଦକ ନିର୍ଣ୍ଣୟ ହୋଇପାରେ ନାହିଁ। ଏହା ସହିତ 2 ବ୍ୟତୀତ ଅନ୍ୟ କୌଣସି ଯୁଗ୍ମ ସଂଖ୍ୟା, ମୌଳିକ ସଂଖ୍ୟା ନୁହେଁ। ତେଣୁ, ଆମର ପରୀକ୍ଷଣର ସୋପାନଗୁଡ଼ିକ

1. ଯଦି $n > 2$ ଏବଂ ଯୁଗ୍ମ ତେବେ n ଏକ ମୌଳିକ ସଂଖ୍ୟା ନୁହେଁ।
2. ଯଦି ସୋପାନ 1 ବିଫଳ ହୁଏ, ତେବେ ଆମେ n କୁ ଗୁଣନିୟକ k ଦ୍ୱାରା $k = 3, 5, 7, \ldots \sqrt{n}$ ଭାଗ କରିବାକୁ ଚେଷ୍ଟା କରୁ, ଯଦି n, k ର କୌଣସି ମୂଲ୍ୟଦ୍ୱାରା ବିଭାଜ୍ୟ ହୁଏ ତେବେ n ଏକ ମୌଳିକ ସଂଖ୍ୟା ନୁହେଁ।
3. ଯଦି ସୋପାନ 2 ମଧ୍ୟ ବିଫଳ ହୁଏ, ତେବେ n ଏକ ମୌଳିକ ସଂଖ୍ୟା।



ଚିତ୍ର 1.24: ନମ୍ବର୍ n ମୌଳିକ କି ନୁହେଁ ଯାଞ୍ଚ କରିବାକୁ ଫ୍ଲୋ'ଚାର୍ଟ

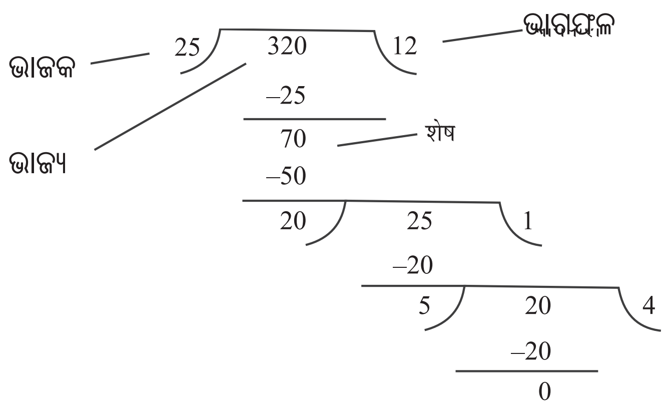
ଦିଆଯାଇଥିବା ଧନାତ୍ମକ ସଂଖ୍ୟା n ଟି ମୌଳିକ କି ନୁହେଁ ଜାଣିବାପାଇଁ ସ୍ୟୁଡୋକୋଡ୍ ନିମ୍ନରେ ଦିଆଯାଇଛି ।

ସ୍ୟୁଡୋକୋଡ୍ 1.10

```
Begin
    Read: n
    If ( n > 2 and n mod 2 == 0 ) then
          Print: n, " is not a prime number"
          Exit
    Else
       Set m = √n
         For k = 3 to m by 2 do
                If ( n mod k == 0 ) then
                       Print: n, " is not a prime number"
                       Exit
                Endif
         Endfor
         Print: n, " is a prime number"
    Endif
End.
```

ଉଦାହରଣ 1.10: ଦୁଇଟି ଗୁଣନ ସଂଖ୍ୟା m ଏବଂ n ର ଗରିଷ୍ଠ ସାଧାରଣ ଗୁଣନୀୟକ (ଗ.ସା.ଗୁ.) ବାହାର କର । ଏହାକୁ ଇଂରାଜିରେ ହାଇଏଷ୍ଟ କମନ୍ ଫ୍ୟାକ୍ଟର୍ (HCF) ବା ଗ୍ରେଟେଷ୍ଟ କମନ ଡିଭାଇଜର (GCD) ବୋଲି କୁହାଯାଏ ।

ସମାଧାନ:



ଚିତ୍ର 1.25: ଗ.ସା.ଗୁ (HCF/GCD) ଗଣନା ପ୍ରକ୍ରିୟାର ଚିତ୍ରଣ

ଚିତ୍ର 1.25 ଦୁଇଟି ଗଣନ ସଂଖ୍ୟାର HCF/GCD ପାଇବାପାଇଁ ନିରନ୍ତର ବିଭାଜନ ପଦ୍ଧତିର ପ୍ରଦର୍ଶନ କରୁଛି । ତୁମେ ନିଶ୍ଚୟ ଦେଖିଥିବ ଯେ କ୍ରମାନ୍ୱୟ ବିଭାଜନରେ, ପୂର୍ବ ବିଭାଜନର ଭାଜକଟି ଭାଜ୍ୟ ହୋଇଯାଏ, ଭାଗଶେଷଟି ଭାଜକ

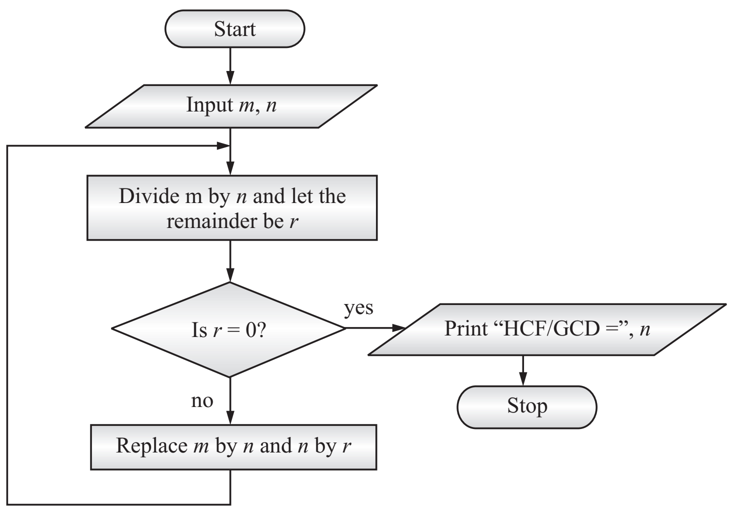
ହୋଇଯାଏ, ଏବଂ ବିଭାଜନ ପୁନର୍ବାର କରାଯାଏ । ଭାଗଶେଷ ଶୂନ ହେବା ପର୍ଯ୍ୟନ୍ତ ଏହି ପ୍ରକ୍ରିୟା ଚାଲୁ ରହେ, ଏବଂ ଚାଲୁ ଭାଜକକୁ ଦତ୍ତ ଗଣନ ସଂଖ୍ୟାର HCF/GCD ଭାବରେ ନିଆଯାଏ ।

ନିମ୍ନଲିଖିତ ଷ୍ଟେପଗୁଡ଼ିକ ବ୍ୟବହାର କରି ଏହି ପ୍ରକ୍ରିୟାକୁ କାର୍ଯ୍ୟକାରୀ କରାଯାଇପାରିବ ।

1. ବିଭାଜନ କର ।
2. ଯଦି ଭାଗଶେଷ ଶୂନ, ତେବେ ଶେଷ କର ଏବଂ ଭାଜକକୁ HCF/GCD ଭାବେ ଗ୍ରହଣ କର ।
3. ଭାଜକ ସହିତ ଭାଜ୍ୟକୁ ବଦଳ କର ।
4. ଭାଗଶେଷ ସହିତ ଭାଜକକୁ ବଦଳ କର ।
5. ଷ୍ଟେପ୍ 1 ରୁ ପୁନରାବୃତ୍ତି କର ।

ସ୍ୟୁଡୋକୋଡ୍ 1.11

```
Begin
     Read: m, n
     Set r = m mod n
     While ( r × 0 ) do
             Set m = n
             Set n = r
             Set r = m mod n
      Endwhile
      Print: "HCF/GCD = ", n
End.
```



ଚିତ୍ର 1.26: ଦୁଇଟି ସଂଖ୍ୟାର ଏଚସିଏଫ୍ ଗଣନା କରିବାକୁ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍

ଉଦାହରଣ 1.11: ଫିବୋନାକି କ୍ରମର ପ୍ରଥମ n ଟି ପଦ ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ଫ୍ଲୋ'ଚାର୍ଟ ଅଙ୍କନ କର ଏବଂ ସ୍ୟୁଡୋକୋଡ୍ ଲେଖ ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଯଦି ଇନପୁଟ୍ n ର ମୂଲ୍ୟ 8, ତେବେ ଆଉଟପୁଟ୍ ନିମ୍ନମତେ ହେବା ଉଚିତ୍

0 1 1 2 3 5 8 13

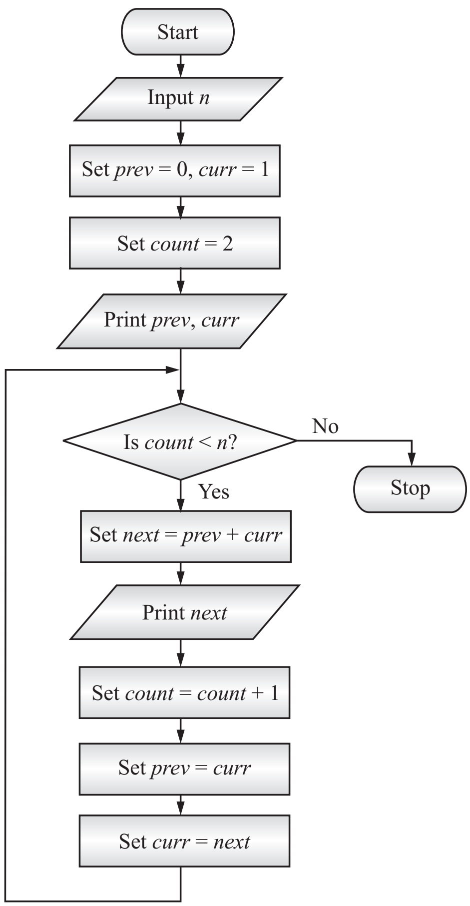
ସମାଧାନ: ନିରୀକ୍ଷଣ କର ଯେ, ପ୍ରଥମ ଦୁଇଟି ପଦକୁ ଛାଡି, ଅନ୍ୟ ପ୍ରତ୍ୟେକ ପଦ ପୂର୍ବବର୍ତ୍ତୀ ଦୁଇଟି ପଦର ସମଷ୍ଟି ଭାବରେ ମିଳିଥାଏ ।

ମନେକର ଆମେ ପୂର୍ବ ପଦପାଇଁ ଚଳ prev, ଚାଲୁ ପଦପାଇଁ curr, ପରବର୍ତ୍ତୀ ପଦପାଇଁ next ବ୍ୟବହାର କରିବା, ଏବଂ ପୂର୍ବ ଏବଂ ଚାଲୁ ପଦର ମୂଲ୍ୟ ଅର୍ଥାତ, କ୍ରମର ପ୍ରଥମ ଦୁଇଟି ପଦର ମୂଲ୍ୟକୁ ଯଥାକ୍ରମେ 0 ଏବଂ 1 ନେବା, ତା'ହେଲେ ନିମ୍ନ ପ୍ରଦତ୍ତ ରେକରେନ୍ସ ସମ୍ବନ୍ଧ ବ୍ୟବହାର କରି ସମଗ୍ର କ୍ରମ ସୃଷ୍ଟି କରାଯାଇପାରିବ ।

next = prev + curr

curr କୁ prev ଦ୍ୱାରା ବଦଳ କର

next କୁ curr ଦ୍ୱାରା ବଦଳ କର



ସ୍ୟୁଡୋକୋଡ୍ 1.12

```
Begin
    Read: n
    Set prev = 0, curr = 1
    Set count = 2
    Print: prev, curr
    While ( count < n ) do
    Set next = prev + curr
    Print: next
    Set count = count + 1
    Set prev = curr
    Set curr = next
    Endwhile
End.
```

ଚିତ୍ର 1.27: *ଫିବୋନାକି କ୍ରମର ପ୍ରଥମ n ଟି ପଦ ପ୍ରିଣ୍ଟ କରିବାକୁ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୟୁଡୋକୋଡ୍*

1.3 ଆଲଗୋରିଦମ୍‌ରୁ ପ୍ରୋଗ୍ରାମ୍ ପର୍ଯ୍ୟନ୍ତ (FROM ALGORITHMS TO PROGRAM)

ବର୍ତ୍ତମାନ ସୁଦ୍ଧା, ତୁମେ ଶିଖିଛ ଯେ କମ୍ପ୍ୟୁଟର ବ୍ୟବହାର କରି ଦିଆଯାଇଥିବା ସମସ୍ୟାର ସମାଧାନପାଇଁ ଆଲଗୋରିଦମ୍ ହେଉଛି ଏକ ଉପାୟ। ଆଲଗୋରିଦମ୍ ହେଉଛି ଦିଆଯାଇଥିବା ସମସ୍ୟାର ସମାଧାନପାଇଁ ଏକ ଲଜିକ୍। ଏହି ଲଜିକ୍‌କୁ ଏକ ପ୍ରୋଗ୍ରାମିଂ ଭାଷା ବ୍ୟବହାର କରି ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ରୂପାନ୍ତରଣ କରିବା ଆବଶ୍ୟକ। ଏହି ପାଠ୍ୟକ୍ରମରେ C ଭାଷା ହେଉଛି ଆମର ପ୍ରୋଗ୍ରାମିଂ ଭାଷା।

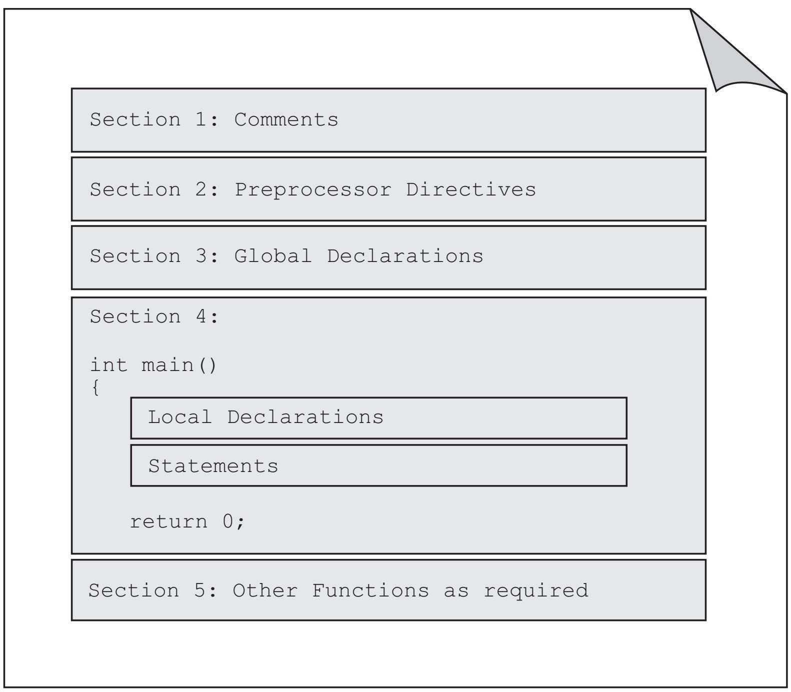
ଏହି ବିଭାଗରେ, ଆମେ C ଭାଷାର ମୌଳିକ ଦିଗଗୁଡ଼ିକୁ ଶିଖିବୁ ଯାହା ଆମକୁ ଆଲଗୋରିଦମକୁ ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ରେ ପରିଣତ କରିବାରେ ସମର୍ଥ କରିବ।

1.3.1 ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ର ସଂରଚନା (Structure of a C Program)

ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ରେ ଲେଖାଯାଉଥିବା ନିର୍ଦ୍ଦେଶାବଳି/ବିବୃତ୍ତିର ସ୍ୱାଭାବିକ କ୍ରମ ଚିତ୍ର 1.28 ରେ ଦର୍ଶାଯାଇଛି।

ଏକ C ପ୍ରୋଗ୍ରାମ୍‌କୁ ସ୍ୱାଧୀନ ଫର୍ମାଟ୍‌ରେ ଲେଖାଯାଏ। ସ୍ୱାଧୀନ ଫର୍ମାଟ୍‌ର ଅର୍ଥ ହେଉଛି ଏକ ବିବୃତ୍ତି ଗୋଟିଏ ଧାଡ଼ି ର ଯେକୌଣସି ସ୍ଥାନରେ ଆରମ୍ଭ ଏବଂ ଶେଷ ହୋଇପାରେ। ଏପରିକି ଏକ ବିବୃତ୍ତିକୁ ଏକାଧିକ ଧାଡ଼ିରେ ଲେଖା ଯାଇପାରେ। C ଭାଷାରେ ଇଣ୍ଡେଣ୍ଟେସନ୍‌ର (indentation) କୌଣସି ଗୁରୁତ୍ୱ ନାହିଁ, ଏହା କେବଳ ପ୍ରୋଗ୍ରାମ୍‌ର ପଠନୀୟତା ବୃଦ୍ଧିପାଇଁ ବ୍ୟବହୃତ ହୋଇଥାଏ।

ଆହୁରି ମଧ୍ୟ, C ହେଉଛି କେସ୍ ସମ୍ବେଦନଶୀଳ, ଅର୍ଥାତ, ଏହା ସାନ (lower case) ଏବଂ ବଡ଼ (uppercase) ଅକ୍ଷର ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ କରେ। ପ୍ରତ୍ୟେକ C ପ୍ରୋଗ୍ରାମ୍ କେବଳ ଲୋୟରକେସ୍ ଅକ୍ଷରରେ ଲେଖାଯାଏ।



ଚିତ୍ର 1.28: ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ର ସାଧାରଣ ଗଠନ

ବର୍ତ୍ତମାନ, ଆସ ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ରେ ପ୍ରତ୍ୟେକ ବିଭାଗର ଭୂମିକାକୁ ଗୋଟିଏ ପରେ ଗୋଟିଏ ବୁଝିବା –

- **ବିଭାଗ** 1 ଇଚ୍ଛାଧୀନ ଅଟେ । ଯଦି ଅଛି ତେବେ ଏଥିରେ ପ୍ରୋଗ୍ରାମ୍ ବିଷୟରେ ବର୍ଣ୍ଣନା ରହିଥାଏ । ଏହି ବିବରଣୀ ସାଧାରଣତଃ ପ୍ରୋଗ୍ରାମ୍‌ଦ୍ୱାରା ହେଉଥିବା କାର୍ଯ୍ୟ ବିଷୟରେ ସୂଚନା ଦିଏ ।

  ଉଦାହରଣ ବଶତଃ

```
/*
*       Program to compute the factorial of a given
*       natural number (positive integer number).
                   *
*       Filename : prime.c
*/
```

  ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ "/*" ଓ "*/"ଯୋଡ଼ିରେ ଆବଦ୍ଧ ହୋଇଥିବା ଯେକୌଣସି ଲେଖାକୁ କମ୍ପାଇଲର୍‌ଦ୍ୱାରା ଉପେକ୍ଷା କରାଯାଇଥାଏ, ଅର୍ଥାତ, ଏହାକୁ ମେସିନ୍ କୋଡ୍‌କୁ ଅନୁବାଦ କରାଯାଏ ନାହିଁ ।

- **ବିଭାଗ 2** ପ୍ରିପ୍ରୋସେସର୍ *(preprocessor)* ନିର୍ଦ୍ଦେଶାଗୁଡ଼ିକୁ *(directives)* ଧାରଣ କରେ । ନିତ୍ୟବ୍ୟବହୃତ ପ୍ରିପ୍ରୋସେସର୍ ନିର୍ଦ୍ଦେଶାବଳିଗୁଡ଼ିକ ହେଉଛି #include ଏବଂ #define । ଏହି ନିର୍ଦ୍ଦେଶାବଳି ପ୍ରିପ୍ରୋସେସର୍‌କୁ କମ୍ପାଇଲ୍‌ପାଇଁ ପ୍ରୋଗ୍ରାମ୍‌କୁ କିପରି ପ୍ରସ୍ତୁତ କରାଯିବ ତାହା ଜଣାଇଥାଏ ।

- #include ପ୍ରୋଗ୍ରାମ୍‌ରେ କେଉଁ ହେଡର୍ ଫାଇଲ୍ ଅନ୍ତର୍ଭୁକ୍ତ କରାଯିବ ତାହା ଜଣାଇଥାଏ #define ସାଧାରଣତଃ ଏକ ଅଭିଜ୍ଞାପକକୁ *(identifier)* ଏକ ଧ୍ରୁବକ ମୂଲ୍ୟ *(literal)* ସହିତ ଜଡ଼ିତ କରିବାପାଇଁ ବ୍ୟବହୃତ ହୁଏ ଯାହା ପ୍ରୋଗ୍ରାମ୍‌ର ଅନେକ ସ୍ଥାନରେ ବ୍ୟବହୃତ ହେବାକୁ ଥିବ ।

  ଉଦାହରଣ

```
#include<stdio.h>
```

- stdio.h ନାମକ ହେଡର୍ ଫାଇଲ୍‌ର ଅନ୍ତର୍ନିହିତ ବିଷୟବସ୍ତୁକୁ ପ୍ରୋଗ୍ରାମ୍ ଫାଇଲ୍‌ର ଏହି ସ୍ଥାନରେ ଭର୍ତ୍ତି କରିବାକୁ ପ୍ରିପ୍ରୋସେସର୍ କହିଥାଏ ।

  ସେହିଭଳି,

```
#define N 200
```

  ପ୍ରତ୍ୟେକ ଅଭିଜ୍ଞାପକ N ର ମୂଲ୍ୟକୁ 200 ଦ୍ୱାରା ବଦଳାଇବାପାଇଁ ପ୍ରିପ୍ରୋସେସରକୁ କହିଥାଏ ।

- **ବିଭାଗ** 3 ଇଚ୍ଛାଧୀନ ଅଟେ । ଯଦି ଅଛି, ତେବେ ଏଥିରେ ଗ୍ଲୋବାଲ(global) ଘୋଷଣାନାମା(declarations) ରହିଛି । ଏଥିରେ ସାଧାରଣତଃ ଡାଟା ଆଇଟମ୍‌ର (ଚଳ/variables) ଘୋଷଣା ଗୁଡ଼ିକ ଥାଏ ଯାହା ପ୍ରୋଗ୍ରାମ୍‌ର ଏକାଧିକ ଫଙ୍କ୍‌ସନଗୁଡ଼ିକ ମଧ୍ୟରେ ଭାଗ ନେଇଥାଏ । ଏହା ବ୍ୟତୀତ, ଏହି ଘୋଷଣାନାମାରେ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଉପଯୋଗ ହେବାକୁ ଥିବା ଅନ୍ୟ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକର ଘୋଷଣା (ପ୍ରୋଟୋଟାଇପ୍/prototype) ମଧ୍ୟ ଅନ୍ତର୍ଭୁକ୍ତ ହୋଇପାରେ ।

- **ବିଭାଗ** 4 main() ଫଙ୍କ୍‌ସନ୍ ଧାରଣ କରେ । ପ୍ରୋଗ୍ରାମ୍‌ର ଚାଳନ ସର୍ବଦା main() ଫଙ୍କ୍‌ସନ୍‌ରୁ ଆରମ୍ଭ ହୁଏ । ଏହା ଅନ୍ୟ ଯେକୌଣସି ଫଙ୍କ୍‌ସନକୁ ଡାକିପାରେ(call), ଏବଂ ସେହି ଡକା ହୋଇଥିବା(called) ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ମଧ୍ୟ ତା'ପରେ ଅନ୍ୟ ଫଙ୍କ୍‌ସନକୁ ଡାକି ପାରିବେ ।

main (), ଏବଂ ଅନ୍ୟ ଫଙ୍କ୍ସନଗୁଡ଼ିକର ପ୍ରଥମ ବିଭାଗ, ସ୍ଥାନୀୟ (local) ଘୋଷଣାଗୁଡ଼ିକ ଧାରଣ କରେ । ଏହି ଘୋଷଣାଗୁଡ଼ିକ ଏହି ଅର୍ଥରେ ସ୍ଥାନୀୟ ଯେ ସେଗୁଡ଼ିକ କେବଳ ସେହି ଫଙ୍କ୍ସନ୍‌ର ଆବଶ୍ୟକତା ସହିତ ଜଡ଼ିତ । ଦ୍ୱିତୀୟ ବିଭାଗରେ ନିର୍ଦ୍ଦେଶାବଳି (ବିବୃତ୍ତି) ଥାଏ ଯାହା ଫଙ୍କ୍ସନ୍‌ଦ୍ୱାରା ସମ୍ପାଦନ ହେବାକୁ ଥିବା କାର୍ଯ୍ୟର ବ୍ୟାଖ୍ୟା କରେ ।

- **ବିଭାଗ 5** ଏହା ମଧ୍ୟ ଇଚ୍ଛାଧୀନ ଅଟେ । ଯଦି ଅଛି, ଏଥିରେ ଅନ୍ୟ ଫଙ୍କ୍ସନ୍‌ଗୁଡ଼ିକର ସଂଜ୍ଞାସମୂହ ରହିଥାଏ ।

  ଯଦି ସମାଧାନ ହେବାକୁ ଥିବା ସମସ୍ୟାଟି ସରଳ ଏବଂ ଆକାରରେ ଛୋଟ, ତେବେ କେବଳ main () ଫଙ୍କ୍ସନ୍ ହିଁ କାର୍ଯ୍ୟ ସମ୍ପାଦନପାଇଁ ଯଥେଷ୍ଟ ।

  ଯାହାହେଉ, ଯଦି ସମସ୍ୟାଟି ଜଟିଳ ଏବଂ ଆକାରରେ ବଡ଼, ତେବେ ଏହାକୁ ଛୋଟ-ଛୋଟ ସ୍ୱାଧୀନ ଉପ-ସମସ୍ୟାଗୁଡ଼ିକରେ ଭାଗ କରାଯାଏ, ଏବଂ ତା'ପରେ ଆମେ ପ୍ରତ୍ୟେକ ଉପ-ସମସ୍ୟାପାଇଁ ପୃଥକ ଫଙ୍କ୍ସନ୍ ଲେଖିଥାଉ ।

  main () ଫଙ୍କ୍ସନ୍ ହିଁ ଏହି ଫଙ୍କ୍ସନଗୁଡ଼ିକର ସଙ୍ଗତ କଲ୍‌ଦ୍ୱାରା ଫଙ୍କ୍ସନଗୁଡ଼ିକର ଚାଳନର ସମନ୍ୱୟ ରକ୍ଷାକରେ, ଏବଂ ଏହି ଫଙ୍କ୍ସନଗୁଡ଼ିକରୁ ପ୍ରାପ୍ତ ସମାଧାନଗୁଡ଼ିକୁ ସଂଶ୍ଳେଷଣ କରି ପ୍ରକୃତ ସମସ୍ୟାର ସମାଧାନ ପ୍ରସ୍ତୁତ କରେ ।

1.3.2 C ପ୍ରୋଗ୍ରାମ୍‌ର ଉଦାହରଣ (Example C Programs)

ଏକ C ପ୍ରୋଗ୍ରାମ୍ ଗଠନର ଅନୁଭବ ପାଇବା ପାଇଁ, ଆସ କିଛି ସରଳ C ପ୍ରୋଗ୍ରାମ୍‌ର ଉଦାହରଣ ବିଚାରକୁ ନେବା ।

ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ଟି ହେଉଛି ପ୍ରଥମ ମାନକ C ପ୍ରୋଗ୍ରାମ ଯାହା ପ୍ରତ୍ୟେକ C ପୁସ୍ତକରେ ଦିଆଯାଇଥାଏ । ଯେତେବେଳେ ଏହାର ଚାଳନ ହୁଏ, ଏହା କେବଳ "Hello World" ନାମକ ସନ୍ଦେଶ (message) ପ୍ରଦର୍ଶନ କରେ ।

Listing 1.1

```
/*
*  Program to greet the user with the message "Hello World".
*  File name: hello.c
*/
#include<stdio.h>
int main ()
{
   printf ("Hello, World\n");
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Hello, World
```

ପ୍ରୋଗ୍ରାମ୍ ବ୍ୟବଚ୍ଛେଦନ (Dissection)

ଏଠାରେ ସହଜ ସମ୍ବନ୍ଧ ସ୍ଥାପନନିମିତ୍ତ ଧାଡ଼ି ସଂଖ୍ୟାଗୁଡ଼ିକ ଯୋଡାଯାଇଛି।

ଧାଡ଼ି 1-4 ଟିପ୍ପଣୀଗୁଡ଼ିକ ଅଛି।

ଧାଡ଼ି 5 ପ୍ରିପ୍ରୋସେସର୍ #include ନିର୍ଦ୍ଦେଶନାମା ଅଛି।

ଧାଡ଼ି 6 main() ନାମକ ଏକ ଫଙ୍କସନକୁ ଉଲ୍ଲେଖ କରୁଛି। ଏହା ଏକ ବିଶେଷ ନାମ ଯାହା ସିଷ୍ଟମଦ୍ୱାରା ସ୍ୱୀକୃତିପ୍ରାପ୍ତ। ଏହା ପ୍ରୋଗ୍ରାମ୍‌ସ୍ଥ ନିର୍ଦ୍ଦିଷ୍ଟ ସ୍ଥାନକୁ ସୂଚିତ କରେ ଯେଉଁଠାରୁ ପ୍ରୋଗ୍ରାମ୍‌ର ଚାଳନ ଆରମ୍ଭ ହୁଏ। ପ୍ରତ୍ୟେକ C ପ୍ରୋଗ୍ରାମ୍‌ରେ ଏକ main() ଫଙ୍କସନ୍ ରହିବା ଆବଶ୍ୟକ।

ପ୍ରତ୍ୟେକ C ଫଙ୍କସନ୍ ସହିତ ଏକ ଫେରସ୍ତ ପ୍ରକାର(return type) ଜଡ଼ିତ ରହିଥାଏ। ଯେଉଁଠି ଫେରସ୍ତ ପ୍ରକାର ହେଉଛି ଡାଟା ଟାଇପ୍‌ଗୁଡ଼ିକ ମଧ୍ୟରୁ ଗୋଟିଏ ଯାହା ଫଙ୍କସନଦ୍ୱାରା ଫେରସ୍ତ ହେଉଥିବା ମୂଲ୍ୟର ପ୍ରକାର ନିର୍ଦ୍ଧାରଣ କରେ। ଯଦି ଏକ ଫଙ୍କସନ୍ କୌଣସି ନିର୍ଦ୍ଦିଷ୍ଟ ମୂଲ୍ୟ ଫେରସ୍ତ କରିବା ଦରକାର ନାହିଁ, ତେବେ ଏହାର ଫେରସ୍ତ ପ୍ରକାରକୁ ଭଏଡ(void) ଲେଖାଯାଏ।

ଯେହେତୁ main ଫଙ୍କସନ୍ ହେଉଛି ଏକ ଫଙ୍କସନ୍ ଯେଉଁଠାରୁ ପ୍ରୋଗ୍ରାମ୍‌ର ଚାଳନ ଆରମ୍ଭ ହୁଏ, ତେଣୁ ଏହାକୁ ସାଧାରଣତଃ ଇଣ୍ଟ(int) ପ୍ରକାରର ଘୋଷଣା କରାଯାଏ। main ଫଙ୍କସନ୍‌କୁ ଏକ ଚନ୍ଦ୍ର ବନ୍ଧନୀ ଯୋଡି ଅନୁସରଣ କରିଥାନ୍ତି।

ଧାଡ଼ି 7 '{' ଚିହ୍ନ ଅଛି, ଯାହାକୁ ଲେଫ୍ଟ ବ୍ରାସ୍ କିମ୍ବା ବାମ କୁଟୀଳ ବନ୍ଧନୀ କୁହାଯାଏ। ଧାଡ଼ି 10 ରେ ଧାଡ଼ି 7 ବନ୍ଧନୀର ମେଳ ରାଇଟ୍ ବ୍ରାସ୍ '}' ଦିଆଯାଇଛି, ଯେଉଁଠାରେ main ଫଙ୍କସନ୍‌ର ଶେଷ ହୋଇଥାଏ। ଏହି ବ୍ରାସ୍ ମେଳି, ଫଙ୍କସନ୍‌ର କାୟାକୁ ଆବଦ୍ଧ କରିଥାଏ।

ଧାଡ଼ି 8 ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ printf() ବ୍ୟବହାର କରି କମ୍ପ୍ୟୁଟର ପରଦାରେ"Hello, World"ପ୍ରଦର୍ଶନ କରେ।

ଧାଡ଼ି 9 ଫେରସ୍ତ ବିବୃତ୍ତି ବ୍ୟବହାର କରିଥାଏ, ଯାହାର ଚାଳନ ହୋଇ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନର ପରିସମାପ୍ତି ହୁଏ ଏବଂ ପ୍ରୋଗ୍ରାମ୍ ନିୟନ୍ତ୍ରଣ ଅପରେଟିଂ ସିଷ୍ଟମକୁ ଫେରସ୍ତ ହୁଏ। ଏହା ସହିତ, ଏହା ଅପରେଟିଂ ସିଷ୍ଟମକୁ 0 ମୂଲ୍ୟ ଫେରସ୍ତ ହୁଏ ଯାହା ସୂଚିତ କରେ ଯେ ପ୍ରୋଗ୍ରାମ୍ ସଫଳତାର ସହ ସମାପ୍ତ ହୋଇଛି।

ଧାଡ଼ି 10 ଏହା main ଫଙ୍କସନ୍‌ର ଶେଷ ସହିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଶେଷକୁ ମଧ୍ୟ ଚିହ୍ନିତ କରୁଛି ଯେହେତୁ ଏହା ହେଉଛି ପ୍ରୋଗ୍ରାମ୍‌ର ଏକମାତ୍ର ଫଙ୍କସନ୍।

ଆସ ଅନ୍ୟ ଏକ ଉଦାହରଣ ପ୍ରୋଗ୍ରାମ୍ ନେବା, ଯେଉଁଠାରେ ବ୍ୟବହାରକାରୀଙ୍କଦ୍ୱାରା କିଛି ଇନପୁଟ୍ ପ୍ରଦାନ କରାଯାଏ, କମ୍ପ୍ୟୁଟେସନ୍ କରାଯାଏ, ଏବଂ କମ୍ପ୍ୟୁଟେସନ୍‌ର ପରିଣାମ ପ୍ରଦର୍ଶିତ ହୁଏ।

ଏହି ଉଦାହରଣରେ ପ୍ରୋଗ୍ରାମ୍, ଇନପୁଟ୍ ଭାବରେ ସେଲସିୟସ୍ ସ୍କେଲରେ ତାପମାତ୍ରା ନେଇଥାଏ, ଏହାର ସମତୁଲ୍ୟ ତାପମାତ୍ରା ଫାରେନହାଇଟ୍ ସ୍କେଲରେ ଗଣନା କରେ, ଏବଂ ତାହା କମ୍ପ୍ୟୁଟର ପରଦାରେ ପ୍ରଦର୍ଶନ କରେ।

ସେଲସିୟସ୍ ଏବଂ ଫାରେନହାଇଟ୍ ତାପମାତ୍ରା ମଧ୍ୟରେ ସମ୍ବନ୍ଧ ହେଉଛି

$$\frac{C}{100} = \frac{F-32}{180}$$

ଯାହା ସରଳୀକରଣ ପରେ ଦେଇଥାଏ

$$F = 1.8 \times C + 32$$

ଏହି ଗାଣିତିକ ସମ୍ବନ୍ଧ ଦିଆଯାଇଥିବା C ର ମୂଲ୍ୟପାଇଁ F କୁ ହିସାବ କରିବାପାଇଁ ବ୍ୟବହୃତ ହୁଏ ।

Listing 1.2

```
 1: /*
 2: * Program to convert temperature from Centigrade scale
 3: * to Fahrenheit scale
 4: */
 5: #include<stdio.h>
 6: int main()
 7: {
 8:    float f, c;
 9:    printf("\nEnter temperature in Celsius scale: ");
10:    scanf("%f", &c);
11:    f = 1.8 * centigrade + 32;
12:    printf("\nEquivalent temperature in Fahrenheit scale");
13:    printf( " = %.2f\n",f);
14:    return 0;
15: }
```

ଟେଷ୍ଟ ରନ୍

```
Enter temperature in Celsius scale: 30
Equivalent temperature in Fahrenheit scale = 86.00
```

ପ୍ରୋଗ୍ରାମ୍ ବ୍ୟବଛେଦ

ଧାଡ଼ି 1-4 ଟିପ୍ପଣୀ ।

ଧାଡ଼ି 5 ପ୍ରିପ୍ରୋସେସର୍ ନିର୍ଦ୍ଦେଶନାମା ।

ଧାଡ଼ି 6-15 ବନ୍ଧନୀ {} ଯୋଡିରେ ଆବଦ୍ଧ ହୋଇଥିବା main ଫଙ୍କସନ୍‌ର ବ୍ୟାଖ୍ୟା ।

ଧାଡ଼ି 8 ଦୁଇଟି ଫ୍ଲୋଟ (float) ପ୍ରକାରର ଚଳ f ଏବଂ c କୁ ଘୋଷଣା କରେ, ଯାହା କମ୍ପ୍ୟୁଟର ମେମୋରିରେ ଦୁଇଟି ବାସ୍ତବ ସଂଖ୍ୟା (ଫ୍ଲୋଟ୍) ପ୍ରତିନିଧିତ୍ୱ ଏବଂ ଗଚ୍ଛିତ କରିପାରିବ । ଚଳ c ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ସମୟରେ ବ୍ୟବହାରକାରୀଙ୍କଦ୍ୱାରା ଦିଆହେବାକୁ ଥିବା ସେଲସିୟସ୍ ସ୍କେଲର ତାପମାତ୍ରାର ମୂଲ୍ୟକୁ ଗଚ୍ଛିତ ରଖିବାପାଇଁ ବ୍ୟବହୃତ ହୁଏ । ଦିଆଯାଇଥିବା ସେଲସିୟସ୍ ସ୍କେଲର ତାପମାତ୍ରାର ସମତୁଲ୍ୟ ତାପମାତ୍ରା ଫାରେନହାଇଟ୍ ସ୍କେଲରେ ଗଣନା କଲାପରେ ଏହି ମୂଲ୍ୟ ଗଚ୍ଛିତ ରଖିବାପାଇଁ ଚଳ f ବ୍ୟବହୃତ ହୁଏ ।

ଧାଡ଼ି 9 ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ printf() ବ୍ୟବହାର କରି କମ୍ପ୍ୟୁଟର ପରଦାରେ"Enter temperature in Celsius scale: "ପ୍ରଦର୍ଶନ କରେ । ଆମେ ଏହି ପ୍ରକାରର ବାର୍ତ୍ତାକୁ ୟୁଜର ପ୍ରମ୍ପ୍ଟ (user prompt) କହିଥାଉ

କାରଣ ଏହା ବ୍ୟବହାରକାରୀ ଚାହୁଁଥିବା ଇନପୁଟ୍ ଦେବାରେ ମାର୍ଗଦର୍ଶନ କରେ। ଏହି କ୍ଷେତ୍ରରେ, ମୂଲ୍ୟ ହେଉଛି ସେଲସିୟସ୍ ସ୍କେଲରେ ତାପମାତ୍ରା।

ତୁମେ ନିଶ୍ଚୟ ଦେଖିଥିବ ଯେ ଡବଲକୋଟ୍‌ରେ ଆବଦ୍ଧ ଲେଖାଗୁଡ଼ିକର ସମଗ୍ର କ୍ରମ ମୁଦ୍ରିତ ହୋଇନାହିଁ। ତା'ପରେ, ଆରମ୍ଭରେ ବର୍ଣ୍ଣ '\n' ର ଭୂମିକା କ'ଣ? ଏହି ଦୁଇଟି ବର୍ଣ୍ଣ ମିଳିତ ଭାବରେ ନୂତନ ଧାଡ଼ିକୁ ଇଙ୍ଗିତ କରେ। ଯଦିଓ ନୂତନ ଧାଡ଼ିପାଇଁ ବର୍ଣ୍ଣ ହେଉଛି ଦୁଇଟି ବର୍ଣ୍ଣର ମିଶ୍ରଣ, ଅର୍ଥାତ, '\' ଏବଂ 'n', ସେଗୁଡ଼ିକୁ C କମ୍ପାଇଲର୍‌ଦ୍ୱାରା ଗୋଟିଏ ବର୍ଣ୍ଣରେ ଅନୁବାଦ କରାଯାଏ। ନୂତନ ଧାଡ଼ି ବର୍ଣ୍ଣ ପରବର୍ତ୍ତୀ ସୂଚନା ମୁଦ୍ରଣ କରିବା ପୂର୍ବରୁ ପରବର୍ତ୍ତୀ ଧାଡ଼ିକୁ ଅଗ୍ରଗତି କରିବାକୁ କମ୍ପାଇଲର୍‌କୁ ନିର୍ଦ୍ଦେଶ ଦେଇଥାଏ।

ଧାଡ଼ି 10 ଅନ୍ୟ ଏକ ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ scanf() ଫଙ୍କସନ୍ ବ୍ୟବହାର କରି ବ୍ୟବହାରକାରୀ-ଇନପୁଟ୍ ଗ୍ରହଣ କରେ ଏବଂ ଏହାକୁ ଚଳ c ରେ ଗଚ୍ଛିତ କରେ (ବାସ୍ତବରେ, ଏହା c ପାଇଁ ସଂରକ୍ଷିତ ଏକ ମେମୋରି ସ୍ଥାନ)। ପ୍ରଥମ ଚର ହେଉଛି ଏକ ଫର୍ମାଟ୍ ଷ୍ଟ୍ରିଙ୍ଗ୍ (format string) ତାହା ଡବଲକୋଟ୍‌ରେ ଆବଦ୍ଧ। ଇନ୍‌ପୁଟ୍ ହୋଇଥିବା ମୂଲ୍ୟକୁ କିପରି ବ୍ୟାଖ୍ୟା କରାଯିବ ସେ ବିଷୟରେ ସିଷ୍ଟମକୁ ନିର୍ଦ୍ଦେଶ ଦେଇଥାଏ। ଫର୍ମାଟ୍ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଅନୁସରଣ କରି, ଏକ କିମ୍ବା ଅଧିକ ଚଳର ଏକ ତାଲିକା ରହିଥାଏ। ପ୍ରତ୍ୟେକ ଚଳ, "&" ସହିତ ଉପସର୍ଗ ହୋଇଥାଏ, ଯାହାକୁ ଠିକଣା (address) ଅପରେଟର କୁହାଯାଏ।

ଧାଡ଼ି 11 ଏକ ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତି ଯାହା ସେଲସିୟସ୍ ସ୍କେଲରେ ଦିଆଯାଇଥିବା ତାପମାତ୍ରା ସହିତ ସମତୁଲ୍ୟ ଫାରେନହାଇଟ୍ ସ୍କେଲରେ ତାପମାତ୍ରା ଗଣନା କରେ, ଏବଂ ଚଳ f କୁ ନ୍ୟସ୍ତ କରେ। ଅର୍ଥାତ, ଚଳ f ରେ ଗଚ୍ଛିତ ହୁଏ।

ଧାଡ଼ି 12-13 ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ printf() ବ୍ୟବହାର କରି ଚଳ f ରେ ଗଚ୍ଛିତ ହୋଇଥିବା ମୂଲ୍ୟକୁ ବାର୍ତ୍ତା "Equivalent Temperature in fahrenheit = " ସହ ଆଉଟପୁଟ୍ କରେ।ଏହି ପ୍ରକାରର ବାର୍ତ୍ତାଗୁଡ଼ିକର ବ୍ୟବହାର ବାଧ୍ୟତାମୂଳକ ନୁହେଁ, କିନ୍ତୁ ଏହା ଅତ୍ୟନ୍ତ ଉପଯୋଗୀ କାରଣ ସେମାନେ ଆଉଟପୁଟ୍ ବ୍ୟାଖ୍ୟାକୁ ସାବଲୀଳ କରିଥାଏ।

ଧାଡ଼ି 14 ଫେରସ୍ତ ବିବୃତ୍ତି, ଯାହା ପ୍ରୋଗ୍ରାମ୍‌କୁ ସମାପ୍ତ କରେ ଏବଂ ଅପରେଟିଂ ସିଷ୍ଟମକୁ ମୂଲ୍ୟ 0 ଫେରସ୍ତ କରେ।

1.3.3 ଏକ ପ୍ରୋଗ୍ରାମ୍‌ର ତିଆରି, କମ୍ପାଇଲ୍ ଏବଂ ଚାଳନ (Creating, Compiling & Executing a Program)

ଥରେ ଆଲଗୋରିଦମ୍ ପ୍ରସ୍ତୁତ ହେବାପରେ, ପରବର୍ତ୍ତୀ ପଦକ୍ଷେପ ହେଉଛି ଆଲଗୋରିଦମ୍‌କୁ ଏକ କମ୍ପ୍ୟୁଟର ପ୍ରୋଗ୍ରାମ୍‌ରେ ପରିଣତ କରିବା। ଏହି ରୂପାନ୍ତରଣ ସମୟରେ ଆଲଗୋରିଦମର ପ୍ରତ୍ୟେକ ପଦକ୍ଷେପ ଏକ କିମ୍ବା ଅଧିକ C ଭାଷା ନିର୍ଦ୍ଦେଶାବଳି ଭାବରେ କୋଡ୍ ହୋଇଥାଏ। ଏହା ସୁପାରିଶ କରାଯାଇଛି ଯେ ଶିକ୍ଷାର୍ଥୀ କମ୍ପ୍ୟୁଟରରେ ଟାଇପ୍ କରିବା ପୂର୍ବରୁ ପ୍ରଥମେ ଏକ ଖାତାରେ ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବା ଉଚିତ।

ବିଭିନ୍ନ ପଦକ୍ଷେପ ପ୍ରଦର୍ଶନ କରିବାକୁ, ଆମେ ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍ ବିଷୟରେ ବିଚାର କରିବା, ଯାହା ଶିକ୍ଷାନବିଶମାନଙ୍କପାଇଁ ବ୍ୟବହାର କରିବା ସହଜ ଅଟେ।

ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ତିଆରି ଏବଂ ସମ୍ପାଦନ କରିବା (Creating and Editing Programs)

ଥରେ ପ୍ରୋଗ୍ରାମ୍ ଖାତାରେ ପ୍ରସ୍ତୁତ ହେବା ପରେ, ଆମେ ଏକ ଟେକ୍ସ୍ଟ ଏଡିଟର ବ୍ୟବହାର କରି ଏହାକୁ କମ୍ପ୍ୟୁଟର ମେମୋରିରେ ଟାଇପ୍ କରୁ। ଏକ ଟେକ୍ସ୍ଟ ଏଡିଟର ଆମକୁ କମ୍ପ୍ୟୁଟର ମେମୋରିରେ ବର୍ଣ୍ଣ ସମ୍ବଳିତ ଡାଟାର ଇନ୍‌ପୁଟରେ ସାହାଯ୍ୟ କରେ, କମ୍ପ୍ୟୁଟର ମେମୋରିରେ ଡାଟା ସମ୍ପାଦନ (ପରିବର୍ତ୍ତନ) କରିବାକୁ ଅନୁମତି ଦିଏ, ଏବଂ ସମ୍ପ୍ରସାରଣ ".C"

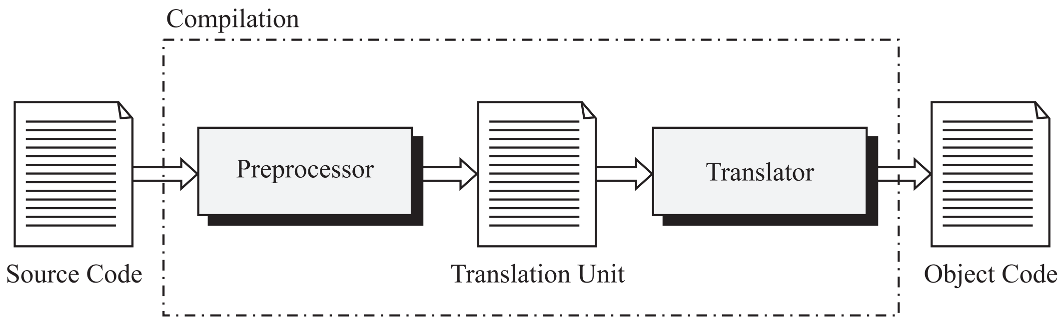
ସହିତ ଏକ ଡିସ୍କ ଫାଇଲରେ ମେମୋରିରୁ ମାଧ୍ୟମିକ ମେମୋରିକୁ ଡାଟା ସଞ୍ଚୟ (save) କରେ।

ଏହି ଗଚ୍ଛିତ ଥିବା ଫାଇଲ୍‌କୁ ଉତ୍ସ ଫାଇଲ୍ (source file) କୁହାଯାଏ, ଏବଂ ଏହାର ବିଷୟବସ୍ତୁକୁ ଉତ୍ସ କୋଡ୍ କୁହାଯାଏ. ଏହି ଉତ୍ସ ଫାଇଲ୍ କମ୍ପାଇଲ୍‌ପାଇଁ ଇନପୁଟ୍ ହୋଇଥାଏ।

ପ୍ରୋଗ୍ରାମର୍ ନିଶ୍ଚିତ ଭାବରେ C ଭାଷା ନିୟମକୁ ଯତ୍ନର ସହ ଅନୁସରଣ କରିବେ। ଭାଷା ନିୟମର ଉଲ୍ଲଂଘନର ପରିଣାମ ବ୍ୟାକରଣଗତ ତ୍ରୁଟି, ବା ଅଧିକ ଠିକ୍ ଭାବରେ ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି (syntax errors) ବୋଲି କୁହାଯାଏ। କମ୍ପାଇଲର୍‌ଟି ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟିଗୁଡ଼ିକପାଇଁ ଯାଞ୍ଚ କରିବ। ଆଗକୁ ବଢ଼ିବା ପୂର୍ବରୁ ଏହି ତ୍ରୁଟିଗୁଡ଼ିକ ଦୂର ହେବା ଆବଶ୍ୟକ।

ପ୍ରୋଗ୍ରାମ୍‌ର କମ୍ପାଇଲ୍ (Compiling Programs)

ଡିସ୍କରେ ଗଚ୍ଛିତ ଥିବା ଉତ୍ସ ଫାଇଲରେ ଉତ୍ସ କୋଡ୍ ମେସିନ୍ ଭାଷାରେ ଅନୁବାଦ ହେବା ଆବଶ୍ୟକ। ଏହି କାର୍ଯ୍ୟ କମ୍ପାଇଲର୍‌ଦ୍ୱାରା କରାଯାଏ. C କମ୍ପାଇଲର୍ ପ୍ରକୃତରେ ଦୁଇଟି ପୃଥକ ପ୍ରୋଗ୍ରାମ୍‌ର ସମାବେଶ - ପ୍ରିପ୍ରୋସେସର ଏବଂ ଟ୍ରାନ୍‌ସଲେଟର।

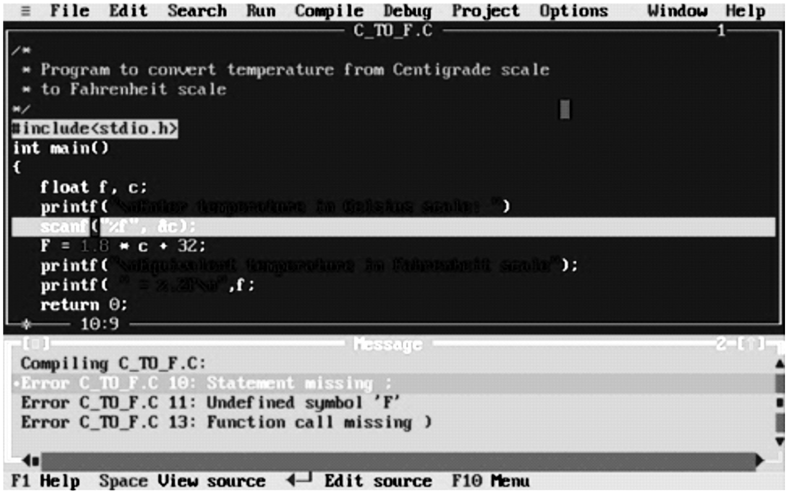


ଚିତ୍ର 1.29: ସଂକଳନ ପ୍ରକ୍ରିୟା

ଉତ୍ସ କୋଡ୍‌କୁ ପ୍ରଥମେ ପ୍ରିପ୍ରୋସେସର୍ ପଢେ ଏବଂ ଏହାକୁ ଅନୁବାଦପାଇଁ ପ୍ରସ୍ତୁତ କରେ। ଉତ୍ସ କୋଡ୍ ପଢିବାବେଳେ, ଏହା ପ୍ରିପ୍ରୋସେସର୍ ନିର୍ଦ୍ଦେଶାବଳି କୋଡ୍ ଦେଖିଲେ ସେହି ଅନୁଯାୟୀ ପ୍ରକ୍ରିୟା କରେ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଯେତେବେଳେ ଏହା #include ନିର୍ଦ୍ଦେଶନାମାର ସାମ୍ନା କରେ, ଏହା ନିର୍ଦ୍ଦିଷ୍ଟ ହେଡର୍ ଫାଇଲ୍‌ର ବିଷୟବସ୍ତୁକୁ ସେହି ନିର୍ଦ୍ଦେଶନାମା ସ୍ଥାନରେ ପ୍ରତିସ୍ଥାପନ କରେ (ଯେପରିକି stdio.h)। ଯେତେବେଳେ ଏହା #define ନିର୍ଦ୍ଦେଶନାମାର ସାମ୍ନା କରେ, ଏହା ନିର୍ଦ୍ଦିଷ୍ଟ ଧ୍ରୁବକ ମୂଲ୍ୟ ଅଭିଜ୍ଞାପକ ସ୍ଥାନରେ ପ୍ରତିସ୍ଥାପନ କରେ। ପ୍ରିପ୍ରୋସେସରର ଆଉଟପୁଟ୍ ହେଉଛି ଏକ ମଧ୍ୟବର୍ତ୍ତୀ ଫାଇଲ୍, ଯାହାକୁ ଅନୁବାଦ ୟୁନିଟ କୁହାଯାଏ।

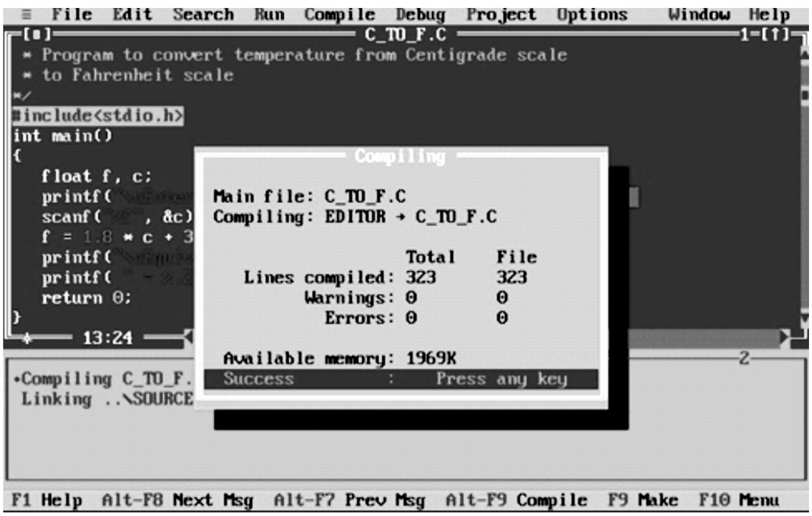
ଟ୍ରାନ୍‌ସଲେଟରଟି ଅନୁବାଦ ଏକକର ନିର୍ଦ୍ଦେଶ-ପରେ-ନିର୍ଦ୍ଦେଶକୁ ପଢେ ଏବଂ ସେମାନଙ୍କର ବ୍ୟାକରଣଗତ ସଠିକତା ଯାଞ୍ଚ କରେ। ଯଦି କୌଣସି ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ଥାଏ, ତେବେ ଏହା ପରଦାରେ ଏକ ତ୍ରୁଟି ସନ୍ଦେଶ ଦେଖାଏ – ଯାହାକୁ ଡାଇଗ୍ନୋଷ୍ଟିକ୍ ସନ୍ଦେଶ କୁହାଯାଏ। ଏହି ଡାଇଗ୍ନୋଷ୍ଟିକ୍ସ ସନ୍ଦେଶଗୁଡ଼ିକ ପ୍ରୋଗ୍ରାମ୍‌ରକୁ ଏହି ତ୍ରୁଟିଗୁଡ଼ିକର କାରଣ ଏବଂ ଏହାର ଉପସ୍ଥିତି ଚିହ୍ନଟ କରିବାରେ ସାହାଯ୍ୟ କରେ।



ଚିତ୍ର 1.30: ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ସୂଚିତ କରୁଥିବା ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍ ସ୍କ୍ରିନ୍‌ସଟ୍

ତେଣୁ, ଯଦି ଗୋଟିଏ ମଧ୍ୟ ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ଥାଏ, ଅନୁବାଦ ପ୍ରକ୍ରିୟା (ବା ସଂକଳନ), ଶେଷ ହୋଇଯାଏ । ଏହି କ୍ଷେତ୍ରରେ, ଟେକ୍ସ୍ଟ ଏଡିଟର ବ୍ୟବହାର କରି ପୁଣି ଉତ୍ସ ଫାଇଲ୍ ଖୋଲି ଏବଂ ଆବଶ୍ୟକ ସଂଶୋଧନ କରି ସଂକଳନର ପୁନରାବୃତ୍ତି କରେ ।

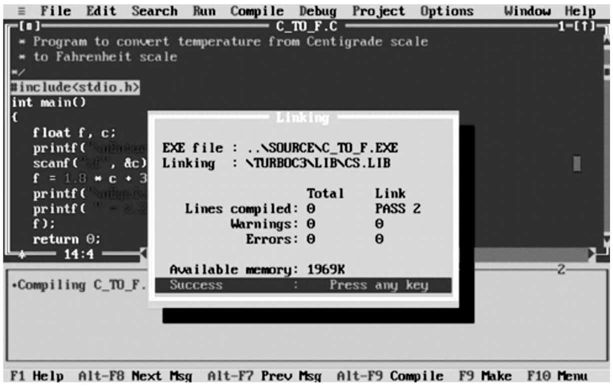
କିନ୍ତୁ, ଯଦି ଅନୁବାଦ ଏକକରେ କୌଣସି ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ନଥାଏ, ଟ୍ରାନ୍ସଲେଟର ପୁଣି ଆରମ୍ଭରୁ ନିର୍ଦ୍ଦେଶକୁ ପଢ଼େ, ସେଗୁଡ଼ିକୁ ମେସିନ୍ ଭାଷାରେ ଅନୁବାଦ କରି ତାକୁ ଏକ ଡିସ୍କ ଫାଇଲରେ ଲେଖେ । ଉତ୍ସ କୋଡ୍‌ର ଅନୁଦିତ ସଂସ୍କରଣକୁ ଅବ୍‌ଜେକ୍ଟ କୋଡ୍ କୁହାଯାଏ ଏବଂ ଏହା ସମ୍ପ୍ରସାରଣ "*.obj" ସହିତ ଡିସ୍କ ଫାଇଲରେ ଗଚ୍ଛିତ ହୁଏ ।



ଚିତ୍ର 1.31: ସଂକଳନ ପ୍ରକ୍ରିୟାର ସଫଳତା ସୂଚିତ କରୁଥିବା ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍ ସ୍କ୍ରିନ୍‌ସଟ୍

ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ଲିଙ୍କ୍ କରିବା (Linking Programs)

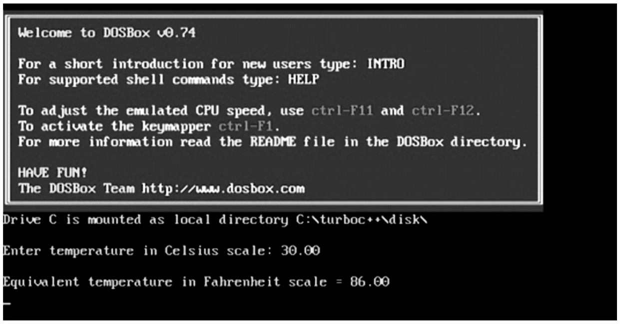
ଥରେ ଉତ୍ସ କୋଡ୍ ଅବଜେକ୍ଟ କୋଡ୍‌ରେ ଅନୁବିତ ହେବା ପରେ, ଯଦିଓ ଏହା ମେସିନ୍ ଭାଷାରେ ଅଛି, କିନ୍ତୁ ଏହା ଚାଳନ ହେଲା ଭଳି ଅବସ୍ଥାରେ ନ ଥାଏ । ଏହାର କାରଣ ହେଉଛି ଏଥିରେ ଲାଇବ୍ରେରି ଫଙ୍କସନଗୁଡ଼ିକର ଉଲ୍ଲେଖ ଥାଇପାରେ । ତେଣୁ, ଏକ ପୂର୍ଣ୍ଣ ମେସିନ୍ କୋଡ୍ ପାଇବାପାଇଁ ଅବଜେକ୍ଟ କୋଡ୍‌ରେ ଏହି ସମସ୍ତ ଫଙ୍କସନଗୁଡ଼ିକୁ ମଧ୍ୟ ଅନ୍ତର୍ଭୁକ୍ତ କରାଯିବା ଆବଶ୍ୟକ, ଯାହା ଚାଳନଯୋଗ୍ୟ ଅବସ୍ଥାରେ ଥାଏ, ଏହାକୁ ଚାଳନଯୋଗ୍ୟ କୋଡ୍ କୁହାଯାଏ ଏବଂ ତାହା ସମ୍ପ୍ରସାରଣ "*.exe" ସହିତ ଡିସ୍କ ଫାଇଲରେ ଗଚ୍ଛିତ ଥାଏ । ଏହି ଚାଳନଯୋଗ୍ୟ କୋଡ୍ ହେଉଛି ପ୍ରୋଗ୍ରାମ୍‌ର ଅନ୍ତିମ ରୂପ ଏବେ ଏହା ଚାଳନପାଇଁ ପ୍ରସ୍ତୁତ ।



ଚିତ୍ର 1.32: ସଂକଳନ ପ୍ରକ୍ରିୟାର ସଫଳତା ସୂଚିତ କରୁଥିବା ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍ ସ୍କ୍ରିନ୍‌ସଟ୍

ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ (Executing Programs)

ଥରେ ପ୍ରୋଗ୍ରାମ୍‌ଟି ଲିଙ୍କ୍ ହେବା ପରେ, ଏହା ଚାଳନପାଇଁ ପ୍ରସ୍ତୁତ ହୋଇଥାଏ । ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ କରିବାକୁ ଆମେ ଏକ ଅପରେଟିଂ ସିଷ୍ଟମ୍ କମାଣ୍ଡ ଦେଇଥାଉ । ଯାହାଫଳରେ ରନ୍ (run), ପ୍ରୋଗ୍ରାମ୍‌କୁ କମ୍ପ୍ୟୁଟର ମେମୋରିରେ ଲୋଡ୍ କରି ଏହାକୁ ଚାଳନ କରିପାରେ । ପ୍ରୋଗ୍ରାମ୍‌କୁ ମେମୋରିରେ ଭର୍ତ୍ତି କରିବା ହେଉଛି ଏକ ଅପରେଟିଂ ସିଷ୍ଟମ୍ ପ୍ରୋଗ୍ରାମ୍‌ର କାର୍ଯ୍ୟ ଏହାକୁ ଲୋଡର୍ କୁହାଯାଏ । ଲୋଡର୍ ମାଧ୍ୟମିକ ମେମୋରିରେ ଚାଳନ ହେବାକୁଥିବା ପ୍ରୋଗ୍ରାମ୍‌ର ଅବସ୍ଥିତି ନିରୂପଣ କରେ, ଏହାକୁ ପଢ଼େ ଏବଂ ମେମୋରିକୁ ଆଣିଥାଏ । ଥରେ ପ୍ରୋଗ୍ରାମ୍ ଲୋଡ୍ ହେବା ପରେ, ଅପରେଟିଂ ସିଷ୍ଟମ୍ ପ୍ରୋଗ୍ରାମ୍‌ର ନିୟନ୍ତ୍ରଣ ସ୍ଥାନାନ୍ତର କରେ ଏବଂ ପ୍ରୋଗ୍ରାମ୍ ଏହାର ଚାଳନ ଆରମ୍ଭ କରେ ।



ଚିତ୍ର 1.33: ପ୍ରୋଗ୍ରାମ୍ ଆଉଟପୁଟ୍ ଦର୍ଶାଉଥିବା ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍‌ର ବ୍ୟବହାରକାରୀ-ସ୍କ୍ରିନ୍

ପ୍ରୋଗ୍ରାମ୍ ପରୀକ୍ଷଣ (Testing the Program)

ପ୍ରୋଗ୍ରାମ୍‌ଟିର ଚାଳନ ହେଲେ ମଧ୍ୟ, ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ ଠିକ୍ ହୋଇ ନ ପାରେ ।

ପ୍ରୋଗ୍ରାମ୍‌ରେ ତାର୍କିକ ତ୍ରୁଟି ହେତୁ ଏପରି ହୋଇପାରେ । ଗୋଟିଏ ତାର୍କିକ ତ୍ରୁଟି ସମସ୍ୟାର ସମାଧାନ ଡିଜାଇନ୍ କରିବା ସମୟରେ ପ୍ରୋଗ୍ରାମର୍ କରିଥିବା ଗୋଟିଏ ଭୁଲ୍, ଉଦାହରଣ ସ୍ୱରୂପ, ଜଣେ ଯଦି ପ୍ରୋଗ୍ରାମର୍ କମ୍ପ୍ୟୁଟରକୁ କାଟଗୁଡ଼ିକୁ (deductions) ବିୟୋଗ କରିବା ପରିବର୍ତ୍ତେ ମୋଟ ଦରମାରେ ଯୋଗ କରି ନେଟ୍ ଦରମା ଗଣନା କରିବାକୁ କୁହନ୍ତି, ତେବେ ଏକ କମ୍ପାଇଲର୍ ଏହି ତ୍ରୁଟିଗୁଡ଼ିକ ଚିହ୍ନଟ କରିପାରିବ ନାହିଁ ।

ତେଣୁ, ପ୍ରୋଗ୍ରାମର୍‌କୁ ପରିଣାମ ଜଣାଥିବା ଡାଟାର ଏକ ସେଟ୍‌ପାଇଁ ପ୍ରୋଗ୍ରାମ୍ ଆଉଟପୁଟ୍‌କୁ ସଯତ୍ନ ପରୀକ୍ଷଣଦ୍ୱାରା ତାର୍କିକ ତ୍ରୁଟିଗୁଡ଼ିକୁ ଖୋଜି ଏହାକୁ ତ୍ରୁଟିଶୁନ୍ୟ କରିବା ଆବଶ୍ୟକ । ଏହି ପ୍ରକାରର ଡାଟାକୁ ଟେଷ୍ଟ ଡାଟା କୁହାଯାଏ ।

> ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ଏବଂ ତାର୍କିକ ତ୍ରୁଟିଗୁଡ଼ିକୁ ସାମୂହିକ ଭାବରେ ବଗ୍ (bug) କୁହାଯାଏ । ଏହି ତ୍ରୁଟିଗୁଡ଼ିକର ଚିହ୍ନଟ ଏବଂ ଦୂର କରିବାର ପ୍ରକ୍ରିୟାକୁ ଡିବଗିଂ କୁହାଯାଏ ।

1.3.4 ବିଭିନ୍ନ C କମ୍ପାଇଲର୍

ନିମ୍ନପ୍ରଦତ୍ତ ଦୁଇଟି C କମ୍ପାଇଲର୍ ମୁଖ୍ୟତଃ ୱିଣ୍ଡୋ ଆଧାରିତ ସିଷ୍ଟମରେ ବ୍ୟବହୃତ ହୁଏ ।

- ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍
- ଡେଭ୍ C++ କମ୍ପାଇଲର୍

ଅନଲାଇନ୍ C କମ୍ପାଇଲର୍‌ଗୁଡ଼ିକପାଇଁ କିଛି ଲିଙ୍କ୍ ନିମ୍ନରେ ଦିଆଯାଇଛି:

- https://www.codechef.com/ide
- https://ide.geeksforgeeks.org
- https://www.onlinegdb.com/online_c_compiler
- https://www.programiz.com/c-programming/online-compiler/
- https://www.tutorialspoint.com/compile_c_online.php
- https://www.jdoodle.com/c-online-compiler/
- https://techiedelight.com/compiler/

1.4 C ଭାଷା ଆରମ୍ଭ କରିବା (GETTING STARTED WITH C LANGUAGE)

C ପ୍ରୋଗ୍ରାମିଂ ଭାଷା 1972 ରେ ଏଟି ଆଣ୍ଡ ଟି ବେଲ୍ ଲାବୋରେଟୋରିରେ (AT & T Bell laboratory) ଡେନିସ୍ ରିଚିଙ୍କଦ୍ୱାରା (Dennis Ritchie) ବିକଶିତ ହୋଇଥିଲା, ଏପର୍ଯ୍ୟନ୍ତ ଏହା ସବୁଠାରୁ ଲୋକପ୍ରିୟ ଭାଷା ମଧ୍ୟରୁ ଗୋଟିଏ ।

କମ୍ପ୍ୟୁଟର ହାର୍ଡୱେୟାର ସହିତ ପ୍ରୋଗ୍ରାମିଂ ଭାଷାର ପାରସ୍ପରିକ ସମ୍ପର୍କ ଉପରେ ନିର୍ଭର କରି ସେଗୁଡ଼ିକୁ ଦୁଇଟି ବ୍ୟାପକ ବର୍ଗରେ ବିଭକ୍ତ କରାଯାଇପାରିବ:

- **ନିମ୍ନସ୍ତରୀୟ (Low-level) ଭାଷା -** ଏହି ବର୍ଗ ଅଧୀନରେ, ମେସିନ୍ ଭାଷା ଏବଂ ଆସେମ୍ବ୍ଲି ଭାଷା ଅଛି । ଏହି ଭାଷାଗୁଡ଼ିକ କମ୍ପ୍ୟୁଟରର ସୁଦକ୍ଷ ବ୍ୟବହାର ପାଇଁ ଅନୁମତି ଦିଏ ।

କିନ୍ତୁ ଏହି ଭାଷା ସହିତ ସମସ୍ୟାଗୁଡ଼ିକ ହେଉଛି:

» ଏଗୁଡ଼ିକ ହାର୍ଡୱେୟାର ଉପରେ ନିର୍ଭରଶୀ। ଏହି ଭାଷା ବ୍ୟବହାର କରି ଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ପୋର୍ଟେବଲ୍ (portable) ହୋଇ ନଥିବାରୁ ଅନ୍ୟ କମ୍ପ୍ୟୁଟରରେ ବ୍ୟବହାର କରାଯାଇପାରିବ ନାହିଁ।

» ଏହି ଭାଷାଗୁଡ଼ିକ ବ୍ୟବହାର କରି ପ୍ରୋଗ୍ରାମିଂ କରିବା ଏକ ସହଜ କାମ ନୁହେଁ। ଏହା କରିବାପାଇଁ ଜଣଙ୍କ ପାଖରେ କମ୍ପ୍ୟୁଟରର ସ୍ଥାପତ୍ୟ (architecture) ବିଷୟରେ ନିଶ୍ଚିତ ଭାବରେ ପୁଙ୍ଖାନୁପୁଙ୍ଖ ଜ୍ଞାନ ରହିବା ଆବଶ୍ୟକ।

- **ଉଚ୍ଚସ୍ତରୀୟ (High-level) ଭାଷା-** ଏହି ବର୍ଗ ଅଧୀନରେ, ଫୋର୍ଟ୍ରାନ୍ (FORTRAN), କୋବୋଲ୍ (COBOL), ପାସ୍କାଲ୍ (PASCAL) ଇତ୍ୟାଦିଠାରୁ ଆରମ୍ଭ କରି ବିସ୍ତୃତ ଭାଷାର ସଂଗ୍ରହ ଅଛି। ବର୍ତ୍ତମାନ ଚାହିଦାରେ ଥିବା ପ୍ରମୁଖ ଉଚ୍ଚସ୍ତରୀୟ ଭାଷାଗୁଡ଼ିକ ହେଉଛି C, C++, ଜାଭା (JAVA), ଓ ପାଇଥନ୍ (Python) ଇତ୍ୟାଦି। ଏହି ଭାଷାଗୁଡ଼ିକ ଉନ୍ନତ ପ୍ରୋଗ୍ରାମିଂ ଦକ୍ଷତାପାଇଁ ଡିଜାଇନ୍ କରାଯାଇଛି। ଅର୍ଥାତ, ଦ୍ରୁତ ପ୍ରୋଗ୍ରାମ୍ ବିକାଶରେ ସହାୟକ ହୋଇଛି।

ଏହି ଭାଷାଗୁଡ଼ିକର ନିମ୍ନଲିଖିତ ସୁବିଧା ଥାଏ:

» ପ୍ରୋଗ୍ରାମ୍ ସିଣ୍ଟାକ୍ସ ଲେଖିବାପାଇଁ ଇଂରାଜୀ ବିବୃତ୍ତି ଭଳି ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକ ବ୍ୟବହାର କରାଯାଏ। ଏହା ପାଠକମାନଙ୍କୁ ଶୀଘ୍ର ଉଚ୍ଚସ୍ତରୀୟ ଭାଷା (HLLs) ଶିଖିବାକୁ ସମର୍ଥ କରେ। ଏହା ସହିତ, ଏଚଏଲଏଲ (HLL) ରେ ଲେଖାଯାଇଥିବା ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ସହଜରେ ବୁଝାଯାଇପାରିବ, ଫଳରେ ଏହାର ରକ୍ଷଣାବେକ୍ଷଣକୁ ସୁଗମ ହୋଇପାରିବ।

» ଏଚଏଲଏଲ (HLL) ରେ ଲେଖାଯାଇଥିବା ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ହାର୍ଡୱେୟାର ନିର୍ଭରଶୀଳ ନୁହେଁ। ଏହାର ଅର୍ଥ ହେଉଛି ଗୋଟିଏ ମେସିନ୍‌ପାଇଁ ଲେଖାଯାଇଥିବା ପ୍ରୋଗ୍ରାମ୍ ଅତ୍ୟଳ୍ପ ବା ବିନା ପରିବର୍ତ୍ତନ ସହିତ ଅନ୍ୟ ମେସିନ୍‌କୁ ସ୍ଥାନାନ୍ତରିତ ହୋଇପାରିବ। ଅର୍ଥାତ, ଏହି ଭାଷାଗୁଡ଼ିକୁ ପୋର୍ଟେବଲ୍ ବୋଲି କହିପାରିବା।

C ଭାଷା ଏହି ଦୁଇଟି ବର୍ଗ ମଧ୍ୟରେ ଥାଏ। ଏହା ଉଭୟ ଭାଷାର ସର୍ବୋତ୍ତମ ଡିଜାଇନ୍ ଭାବେ ପ୍ରସ୍ତୁତ କରାଯାଇଥାଏ, ସେଥିପାଇଁ ଏହାକୁ ପ୍ରାୟତଃ ମଧ୍ୟମ ସ୍ତରର ଭାଷା କୁହାଯାଏ। ଅର୍ଥାତ, ଭଲ ପ୍ରୋଗ୍ରାମିଂ ଦକ୍ଷତା ସହିତ ଭଲ ମେସିନ୍ ଦକ୍ଷତା।

- ପ୍ରୋଗ୍ରାମିଂ ଦକ୍ଷତା ହାସଲପାଇଁ, C ଭାଷାରେ ଅନ୍ୟ ଯେକୌଣସି ଆଧୁନିକ ଉଚ୍ଚସ୍ତରୀୟ ଭାଷାର ସମସ୍ତ ଉପାଦାନ ମହଜୁଦ୍ ଅଛି।
- ମେସିନ୍ ଦକ୍ଷତା ହାସଲ କରିବାକୁ, C ଭାଷାରେ ସିଷ୍ଟମର ଯେକୌଣସି ହାର୍ଡୱେୟାରକୁ ପହଞ୍ଚପାଇଁ (access), ରେଜିଷ୍ଟର (register) ସ୍ତରର ବା ଦ୍ରୁତଗାମୀ ଆସେମ୍ବ୍ଲି ଭାଷାର ରୁଟିନ୍ ସହିତ ଇଣ୍ଟର୍‌ଫେସ (interface) କରିବାପାଇଁ ଆବଶ୍ୟକ ହେଉଥିବା ସମସ୍ତ ବୈଶିଷ୍ଟ୍ୟ ଏଥିରେ ଉପଲବ୍ଧ।

1.4.1 C ଭାଷାର ବୈଶିଷ୍ଟ୍ୟଗୁଡ଼ିକ

C ଭାଷା ସବୁଠାରୁ ଲୋକପ୍ରିୟ ପ୍ରୋଗ୍ରାମିଂ ଭାଷା ଭାବରେ ଥିଲା ଏବଂ ରହିବ।

ଏହାର ଲୋକପ୍ରିୟତାକୁ ବଢ଼ାଉଥିବା କିଛି ପ୍ରମୁଖ ବୈଶିଷ୍ଟ୍ୟ ହେଉଛି:

- ଏହା ଏକ ସାଧାରଣ ଉପଯୋଗୀ ପ୍ରୋଗ୍ରାମିଂ ଭାଷା। ଏହାକୁ ବିଭିନ୍ନ ସମସ୍ୟାର ସମାଧାନପାଇଁ ବ୍ୟବହୃତ ହୋଇପାରିବ।

- ଏହା ସଂରଚନା ପ୍ରୋଗ୍ରାମିଂକୁ ସମର୍ଥନ କରେ । ତେଣୁ, ଏଥିରେ ବିକଶିତ ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ ସହଜରେ ବୁଝିହେବ ।
- ଏହା ସ୍ୱାଧୀନ ଫର୍ମ, ଅର୍ଥାତ, ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବାପାଇଁ କୌଣସି କଠୋର ଫର୍ମାଟ୍ ନାହିଁ । ଯେକୌଣସି ନିର୍ଦ୍ଦେଶ ଯେକୌଣସି ସ୍ଥାନରୁ ଆରମ୍ଭ ହୋଇ ଯେକୌଣସି ସ୍ଥାନରେ ଶେଷ ହୋଇପାରେ । ଏହା ସହିତ, ଗୋଟିଏ ନିର୍ଦ୍ଦେଶ ଏକାଧିକ ଧାଡ଼ିକୁ ଲମ୍ବିପାରେ (ଲେଖା ହୋଇପାରେ) ।
- ଏହା କେସ୍ ସମ୍ବେଦନଶୀଳ । ଏହା ଲୋୟରକେସ୍ (ଛୋଟ) ଏବଂ ଅପରକେସ୍ (ବଡ) ବର୍ଣ୍ଣମାଳା ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଦର୍ଶାଇପାରେ ।
- ଅପରେଟରମାନଙ୍କର ଏକ ସମୃଦ୍ଧ ସେଟ୍ ପ୍ରଦାନ କରୁଥିବା ଏହା ପ୍ରଥମ ଭାଷା ଥିଲା ।
- ଏହା ତୁମକୁ ପଏଣ୍ଟର୍ ମାଧ୍ୟମରେ, ଯେକୌଣସି ଭଣ୍ଡାରଣର ଅବସ୍ଥାନ ଏବଂ ପ୍ରୋଗ୍ରାମ୍ ମାଧ୍ୟମରେ କମ୍ପ୍ୟୁଟରରେ ସଂଯୋଜିତ ଉପକରଣର ପହଞ୍ଚପାଇଁ ଅନୁମତି ଦେଇଥାଏ ।
- ଏହା ତୁମକୁ ଆଭ୍ୟନ୍ତରୀଣ ପ୍ରୋସେସର୍ ରେଜିଷ୍ଟରଗୁଡ଼ିକରେ ବଦଳାବଦଳି କରିବାକୁ ଅନୁମତି ଦିଏ, ଏହିପରି ଏହା ନିମ୍ନସ୍ତରୀୟ ପ୍ରୋଗ୍ରାମିଂପାଇଁ ବହୁତ ଉପଯୋଗୀ ଅଟେ ।
- ଏହା ପୋର୍ଟେବଲ୍, ଅର୍ଥାତ ଏଥିରେ ଲେଖାଯାଇଥିବା ଯେକୌଣସି ପ୍ରୋଗ୍ରାମ୍ ବିନା କିମ୍ବା ଅଳ୍ପ ପରିବର୍ତ୍ତନଦ୍ୱାରା ଯେକୌଣସି କମ୍ପ୍ୟୁଟରରେ ବ୍ୟବହୃତ ହୋଇପାରିବ ।
- ଏହା ତୁମକୁ ତୁମର ନିଜସ୍ୱ ଲାଇବ୍ରେରିର ଫଙ୍କସନଗୁଡ଼ିକ ବିକଶିତ କରିବାକୁ ଅନୁମତି ଦିଏ ଯାହା ମାନକ ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ ପରି ଯେକୌଣସି ପ୍ରୋଗ୍ରାମ୍ ସହିତ ଲିଙ୍କ୍ ହୋଇପାରିବ ।

1.4.2 C ଭାଷାର ପ୍ରୟୋଗ କ୍ଷେତ୍ର

C ଭାଷାର ନିମ୍ନ ସାମର୍ଥ୍ୟଗୁଡ଼ିକ ଏହାକୁ ପସନ୍ଦ କରିବାର ସ୍ୱାଭାବିକ କାରଣ ।

- ଭାଷାନୁବାଦକ, ଯନ୍ତ୍ରାଂଶର ଡ୍ରାଇଭର, ଏଡିଟର, ଲିଙ୍କର୍, ଲୋଡର୍ ଇତ୍ୟାଦିର ସଫ୍ଟୱେୟାର ତିଆରିପାଇଁ ବ୍ୟବହୃତ ସିଷ୍ଟମ୍ ପ୍ରୋଗ୍ରାମିଂ ।
- ବିଭିନ୍ନ ଯୋଗାଯୋଗ ପ୍ରୋଟୋକଲ୍ ତିଆରିପାଇଁ ନେଟୱାର୍କ୍ ସଫ୍ଟୱେୟାର ।
- ଗ୍ରାଫିକାଲ୍ ୟୁଜର ଇଣ୍ଟରଫେସ୍ (GUIs), ବିଜ୍ଞାନ ସମ୍ବନ୍ଧୀୟ ପରିଦୃଶ୍ୟତା, ଏବଂ ଉପସ୍ଥାପନା ଗ୍ରାଫିକ୍ସ ଇତ୍ୟାଦିର ସଫ୍ଟୱେୟାର ଲେଖିବାପାଇଁ ଗ୍ରାଫିକ୍ସ ପ୍ରୋଗ୍ରାମିଂ ।
- ଏମ୍ବେଡେଡ୍ (embddded) ସିଷ୍ଟମ୍ ଯେଉଁଠାରେ C ରୁଟିନ୍‌ଗୁଡ଼ିକ ଦ୍ରୁତଗାମୀ ଆସେମ୍ବଲି ଭାଷା ରୁଟିନ୍ ସହିତ ଯୋଡ଼ାଯାଏ ଏବଂ ପରିଣାମି କୋଡ୍ ରମ୍ (Rom) ଚିପ୍‌ରେ ଗଚ୍ଛିତ ରଖାଯାଏ ।

1.4.3 C ଭାଷାର ମୌଳିକ ଗଢ଼ଣ ପିଣ୍ଡ (Basic Building Blocks of C Language)

କୌଣସି ପ୍ରୋଗ୍ରାମିଂ ଭାଷା ଶିଖିବାର ଏକ କଠିନ ଦିଗ ହେଉଛି ପ୍ରାୟ ପ୍ରତ୍ୟେକ ଏବଂ ସବୁକିଛି ପରସ୍ପର ସହ ଜଡ଼ିତ । ତେଣୁ, ତୁମେ ସମସ୍ତ ବିଷୟରେ ଅଳ୍ପଅଳ୍ପ ନ ଜାଣିଲେ କିଛି ବୁଝିବା ପ୍ରାୟ ଅସମ୍ଭବ ମନେହୁଏ ।

ଏହି ବିଭାଗରେ, ଆମେ ବିଭିନ୍ନ ବ୍ୟାବହାରିକ ଉପାଦାନ ବିଷୟରେ ଶିଖିବୁ, ଯାହାକୁ C ଭାଷାର ଗଢ଼ଣ ପିଣ୍ଡ (building blocks) ମଧ୍ୟ କୁହାଯାଏ ।

1.4.3.1 ବର୍ଣ୍ଣ ସେଟ୍

ଅତ୍ୟନ୍ତ ମୌଳିକ ଅର୍ଥରେ, ଏକ C ପ୍ରୋଗ୍ରାମ୍ ହେଉଛି ବର୍ଣ୍ଣର ଏକ କ୍ରମ । ଯେତେବେଳେ ଏହି ବର୍ଣ୍ଣଗୁଡ଼ିକ କମ୍ପାଇଲର୍‌କୁ ଦିଆଯାଏ, ସେଗୁଡ଼ିକ ପ୍ରସଙ୍ଗ ଅନୁଯାୟୀ ବୁଝାଯାଏ । ଯେପରିକି ବର୍ଣ୍ଣ, ଅଭିଜ୍ଞାପକ, ଧ୍ରୁବକ, ଏବଂ ବିବୃତ୍ତି ଇତ୍ୟାଦି । ଏକ C ଉତ୍ସ ପ୍ରୋଗ୍ରାମ୍‌ରେ ବ୍ୟବହୃତ ବର୍ଣ୍ଣ ଗୁଡ଼ିକ ଆମେରିକାନ୍ ଷ୍ଟାଣ୍ଡାର୍ଡ କୋଡ ଫର ଇନଫରମେସନ ଇଣ୍ଟର୍‌ଚେଞ୍ଜ (ASCII ଉଚ୍ଚାରଣରେ ଆସ୍‌କି) ସେଟ୍‌ର ଅନ୍ତର୍ଭୁକ୍ତ ।

C ରେ ବର୍ଣ୍ଣଗୁଡ଼ିକ ନିମ୍ନଲିଖିତ ବର୍ଗଗୁଡ଼ିକରେ ଗୋଷ୍ଠୀଭୁକ୍ତ ହୋଇଛି:

- ଅକ୍ଷର (Letters) (A-Z, a-z)
- ଅଙ୍କ (Digits) (0-9)
- ବିଶେଷ ବର୍ଣ୍ଣ (Special characters) (, : ; ~ > <# % ' ^ + - * . = ! | ( ) { } [ ] / &)
- ଶୂନ୍ୟସ୍ଥାନ (White space) ବର୍ଣ୍ଣ (ଫାଙ୍କା ସ୍ଥାନ (Blank space), ଭୂସମାନ୍ତର ଟ୍ୟାବ୍ (horizontal tab), ଇତ୍ୟାଦି)

> ସମସ୍ତ C କମ୍ପାଇଲର୍ ଶୂନ୍ୟସ୍ଥାନ ବର୍ଣ୍ଣଗୁଡ଼ିକୁ ଅଣଦେଖା କରେ । ଏହି ବର୍ଣ୍ଣଗୁଡ଼ିକ ମୂଳତଃ ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ର ପଠନୀୟତା ଏବଂ ବୋଧଗମ୍ୟତା ନିମିତ୍ତ ବ୍ୟବହୃତ ହୋଇଥାଏ ।

1.4.3.2 ଟୋକନ୍

ଟୋକନ୍ ହେଉଛି ଏକ କ୍ଷୁଦ୍ରତମ ଉପାଦାନ ଯାହାର ନିଜସ୍ୱ ଅର୍ଥ ଅଛି । C ପ୍ରୋଗ୍ରାମ୍ ହେଉଛି ଟୋକନ୍‌ମାନଙ୍କର ଏକ ଅନୁକ୍ରମ । C ଭାଷାର ଟୋକନ୍‌ଗୁଡ଼ିକ ନିମ୍ନଲିଖିତ ବର୍ଗରେ ଶ୍ରେଣୀଭୁକ୍ତ କରାଯାଇପାରିବ:

- କୀୱାର୍ଡ (keywords)
- ଅଭିଜ୍ଞାପକ (identifiers)
- ଧ୍ରୁବକ ମୂଲ୍ୟ (literals)
- ବିରାମ ଚିହ୍ନ (punctuators)
- ଅପରେଟର (operators)

କୀୱାର୍ଡ (Keywords)

C ରେ ପ୍ରତ୍ୟେକ ଶବ୍ଦକୁ ଏକ କୀୱାର୍ଡ କିମ୍ବା ଏକ ଅଭିଜ୍ଞାପକ ରୂପେ ଶ୍ରେଣୀଭୁକ୍ତ କରାଯାଏ ।

କୀୱାର୍ଡଗୁଡ଼ିକ ମୂଳତଃ ସେହି ଶବ୍ଦଗୁଡ଼ିକ ଯାହା ପୂର୍ବନିର୍ଦ୍ଧାରିତ ସ୍ଥାୟୀ ଅର୍ଥରେ ଅର୍ଥାତ ଏହାର ଅର୍ଥଗୁଡ଼ିକ ପରିବର୍ତ୍ତନ କରାଯାଇପାରିବ ନାହିଁ । ଏହି କୀୱାର୍ଡଗୁଡ଼ିକ ପ୍ରୋଗ୍ରାମ୍ ନିର୍ଦ୍ଦେଶାବଳି (ବିବୃତ୍ତି) ଗଠନପାଇଁ ମୌଳିକ ଗଢ଼ଣ ପିଣ୍ଡ ଭାବରେ କାର୍ଯ୍ୟ କରେ । ସମସ୍ତ କୀୱାର୍ଡ ସାନ ଅକ୍ଷରରେ ଲେଖାଯାଇଥାଏ । ସେମାନଙ୍କର ଭୁଲ୍ ବ୍ୟବହାର ଏକ ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟିରେ ପରିଣତ ହୁଏ ।

ଟେବୁଲ୍ 1.2: C ର କୀୱାର୍ଡଗୁଡ଼ିକ

asm	default	for	short	union
auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
Const	extern	register	switch	
continue	float	return	typedef	

ଅଭିଜ୍ଞାପକ (Identifiers)

ଅଭିଜ୍ଞାପକ ମୂଳତଃ ପ୍ରୋଗ୍ରାମ୍‌ରେ ବ୍ୟବହୃତ ଏକ ନାମ । ଅଭିଜ୍ଞାପକଗୁଡ଼ିକ ଚଳ, ଆରେ (array)ଏବଂ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକୁ ସୂଚିତ କରିବାକୁ ବ୍ୟବହାର କରାଯାଏ । ଏହି ବ୍ୟବହାରକାରୀ-ନିର୍ଦ୍ଧାରିତ ନାମଗୁଡ଼ିକ ବର୍ଣ୍ଣଗୁଡ଼ିକର କ୍ରମକୁ ନେଇ ଗଠିତ, ଯେଉଁଠାରେ ପ୍ରତ୍ୟେକ ବର୍ଣ୍ଣ ଏକ ଅକ୍ଷର, ଏକ ଅଙ୍କ ବା ଏକ ଅଣ୍ଡରସ୍କୋର୍ '_' ହୋଇପାରେ, ଏବଂ ଏହାର ପ୍ରାରମ୍ଭିକ ସଂକେତଟି ଏକ ଅଙ୍କ ହେଇପାରିବ ନାହିଁ । ଉଭୟ ବଡ଼ ଏବଂ ସାନ ଅକ୍ଷର ଅନୁମୋଦିତ, କିନ୍ତୁ, ସାଧାରଣତଃ ସାନ ଅକ୍ଷର ବ୍ୟବହୃତ ହୁଏ । ଅଣ୍ଡରସ୍କୋର୍ ସାଧାରଣତଃ ଦୁଇ କିମ୍ବା ଅଧିକ ଶବ୍ଦକୁ ଲିଙ୍କ୍ କରି ଅର୍ଥପୂର୍ଣ୍ଣ ନାମ ତିଆରିବାକୁ ବ୍ୟବହୃତ ହୁଏ ।

ଯେହେତୁ C ଭାଷା କେସ୍ ସମ୍ବେଦନଶୀଳ (ଏହା ସାନ ଓ ବଡ଼ ଅକ୍ଷର ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ କରେ), ତେଣୁ, price ଏବଂ Price ଦୁଇଟି ଭିନ୍ନ ଅଭିଜ୍ଞାପକ ଭାବରେ ବିବେଚନା କରାଯିବ ।

ଅଭିଜ୍ଞାପକପାଇଁ ନିୟମ

1. ପ୍ରଥମ ବର୍ଣ୍ଣଟି ନିଶ୍ଚିତ ଭାବରେ ଏକ ଅକ୍ଷର କିମ୍ବା ଏକ ଅଣ୍ଡରସ୍କୋର୍ ହେବା ଆବଶ୍ୟକ ।
2. କେବଳ ଅକ୍ଷର, ଅଙ୍କ କିମ୍ବା ଅଣ୍ଡରସ୍କୋର୍‌କୁ ନେଇ ଗଠିତ ହେବା ଆବଶ୍ୟକ ।
3. ଏକ କୀୱାର୍ଡ୍ ବ୍ୟବହାର କରାଯାଇ ପାରିବ ନାହିଁ ।

ବୈଧ ଅଭିଜ୍ଞାପକଗୁଡ଼ିକର ଉଦାହରଣ:

myFile	roll_no	date_of_birth	_chk
file10	N1TO10	area	AbCdE

ଧ୍ରୁବକ ମୂଲ୍ୟ (Literals)

ଧ୍ରୁବକ ମୂଲ୍ୟ (ଧ୍ରୁବକ) ସ୍ଥିର ମୂଲ୍ୟକୁ ସୂଚିତ କରେ ଯାହାର ମୂଲ୍ୟ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ସମୟରେ ପରିବର୍ତ୍ତନ ହୁଏ ନାହିଁ ।

- **ଇଣ୍ଟିଜର ଧ୍ରୁବକ ମୂଲ୍ୟ** – ଦଶମିକ ବିନ୍ଦୁ ବିନା ଏକ ସଂଖ୍ୟା । ଅଙ୍କଗୁଡ଼ିକର କ୍ରମ ଅନୁସାରେ ଏକ ପୂର୍ଣ୍ଣସଂଖ୍ୟା ଯାହା ଏକ ଇଚ୍ଛାଧୀନ ସଂକେତ ଯୁକ୍ତ (+) କିମ୍ବା ବିଯୁକ୍ତ (–) ଦ୍ୱାରା ଉପସର୍ଗ ହେଇଥାଏ ।
- **ବାସ୍ତବ ଧ୍ରୁବକ ମୂଲ୍ୟ** – ଭଗ୍ନାଂଶ ସଂଖ୍ୟା ।
- **ଏକ ଅକ୍ଷର ଧ୍ରୁବକ ମୂଲ୍ୟ** – ସିଙ୍ଗଲ୍‌କୋଟ୍‌ରେ (') ଆବଦ୍ଧ ଗୋଟିଏ ଅକ୍ଷରକୁ ନେଇ ଗଠିତ; ଯେପରିକି

'A'। କମ୍ପ୍ୟୁଟର୍ ଭିତରେ ପ୍ରତ୍ୟେକ ବର୍ଣ୍ଣ ଆସ୍କି (ASCII) କୋଡ୍ ଅନ୍ତସ୍ଥ ଏକ ପୂର୍ଣ୍ଣସଂଖ୍ୟାଦ୍ୱାରା ପ୍ରତିନିଧିତ୍ୱ ହୋଇଥାଏ। ଉଦାହରଣ ସ୍ୱରୂପ, ବର୍ଣ୍ଣ 'A' ର ପୂର୍ଣ୍ଣସଂଖ୍ୟା ମୂଲ୍ୟ 65, ବର୍ଣ୍ଣ 'B' କୁ 66 ଇତ୍ୟାଦିଦ୍ୱାରା ପ୍ରତିନିଧିତ୍ୱ କରାଯାଇପାରିବ।

C ଭାଷା କିଛି ବର୍ଣ୍ଣକୁ ଅନୁମତି ଦେଇଥାଏ ଯାହା କୀବୋର୍ଡରୁ ସିଧାସଳଖ ଭର୍ତ୍ତି/ଟାଇପ୍ ହୋଇପାରିବ ନାହିଁ ଯେପରିକି ବ୍ୟାକସ୍ପେସ୍, ଟ୍ୟାବ୍, କ୍ୟାରେଜ-ରିଟର୍ଣ୍ଣ, ନ୍ୟୁ-ଲାଇନ୍, (backspace, tab, carriage return, newline) ଇତ୍ୟାଦି। ଏହି ଅକ୍ଷରସମୂହ ଏକ ଅନୁକ୍ରମଦ୍ୱାରା ପ୍ରତିନିଧିତ୍ୱ ହୋଇଥାଏ ଯାହା ହେଉଛି ବର୍ଣ୍ଣ \ (ବ୍ୟାକ୍ ସ୍ଲାଶ୍)ରୁ ଆରମ୍ଭ ହୁଏ, ଏବଂ ଏହାକୁ ଏସ୍କେପ ସିକ୍ୱେନ୍ସ (escape sequences) ବୋଲି କୁହାଯାଏ।

ଟେବୁଲ୍ 1.3: ସାଧାରଣ ଏସ୍କେପ ସିକ୍ୱେନ୍ସ

ଏସ୍କେପ ସିକ୍ୱେନ୍ସ	ପ୍ରତିନିଧି	ପ୍ରଭାବ
\n	ନୂତନ ଧାଡ଼ି (Newline)	ପରବର୍ତ୍ତୀ ଆଉଟପୁଟ୍ ନୂତନ ଧାଡ଼ିରୁ ଆରମ୍ଭ ହୁଏ।
\t	ଭୂସମାନ୍ତର ଟ୍ୟାବ୍ (Horizontal Tab)	ପରବର୍ତ୍ତୀ ଆଠ-ଅଙ୍କ ବ୍ଯବଧାନ କ୍ଷେତ୍ରକୁ ଗତି କରେ।
\r	କ୍ୟାରେଜ-ରିଟର୍ଣ୍ଣ (Carriage Return)	କ୍ୟାରେଜ-ଫେରସ୍ତ।
\0	ନଲ୍ ବର୍ଣ୍ଣ (Null character)	ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ର ସମାପ୍ତି।

- ଷ୍ଟ୍ରିଙ୍ଗ୍ (ବହୁ-ବର୍ଣ୍ଣ) ଧ୍ରୁବକ ମୂଲ୍ୟ – ଏଥିରେ ଡବଲ୍‌କୋଟ୍‌ରେ ଆବଦ୍ଧ ଅକ୍ଷରସମୂହର ଏକ ଅନୁକ୍ରମ ଥାଏ। ଏଥିରେ ବର୍ଣ୍ଣମାଳାର ଯେକୌଣସି ବର୍ଣ୍ଣକୁ ନିଆଯାଇପାରିବ।

ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ ଏକ ଅକ୍ଷର ଧ୍ରୁବକ ମୂଲ୍ୟ ଯେପରିକି 'A', ଏକ ଅକ୍ଷର ବିଶିଷ୍ଟ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ମୂଲ୍ୟ "A" ସହିତ ସମାନ ନୁହେଁ।

ବିରାମ ଚିହ୍ନ (Punctuators)

ବିରାମ ଚିହ୍ନକୁ ମଧ୍ୟ ବିଭାଜକ କୁହାଯାଏ। C ରେ ବ୍ୟବହୃତ ବିଭିନ୍ନ ବିରାମ ଚିହ୍ନ ନିମ୍ନ ଟେବୁଲରେ ବର୍ଣ୍ଣିତ।

ଟେବୁଲ୍ 1.4: କିଛି ବିରାମ ଚିହ୍ନ ଏବଂ ସେମାନଙ୍କର ବର୍ଣ୍ଣନା

ବିରାମ ଚିହ୍ନ	ବର୍ଣ୍ଣନା
ବର୍ଗ ବନ୍ଧନୀ [ ]	ଆରେ ସବସ୍କ୍ରିପ୍ଟ ଗୁଡ଼ିକ ଆବଦ୍ଧ କରିବାକୁ ବ୍ୟବହୃତ ହୁଏ।
ଚନ୍ଦ୍ର ବନ୍ଧନୀ ( )	ଫଙ୍କସନ୍ ଘୋଷଣାନାମାରେ ଚରଗୁଡ଼ିକ ଆବଦ୍ଧ କରିବା ସହିତ ଫଙ୍କସନ୍ ସଂଜ୍ଞା, ଫଙ୍କସନ୍ କଲ୍ ଏବଂ ପରିପ୍ରକାଶରେ ପାରାମିଟର୍ ଆବଦ୍ଧ କରିବାପାଇଁ ବ୍ୟବହୃତ ହୁଏ।
କୁଟିଳ ବନ୍ଧନୀ { }	ବିବୃତ୍ତିର ଏକ ବ୍ଲକ୍‌କୁ ଆବଦ୍ଧ କରିବାପାଇଁ ବ୍ୟବହୃତ ହୋଇଥାଏ। ଆରେର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଧିଷ୍ଟକରଣ ସମୟରେ ଉପାଦାନଗୁଡ଼ିକର ତାଲିକାକୁ ଆବଦ୍ଧ କରିବାପାଇଁ ମଧ୍ୟ ବ୍ୟବହୃତ ହୁଏ।
କମା ,	ଫଙ୍କସନ୍ ସଂଜ୍ଞାରେ ଚରଗୁଡ଼ିକ ପୃଥକ କରିବାକୁ ଏବଂ ଫଙ୍କସନ୍ କଲ୍‌ରେ ପାରାମିଟର୍ ଗୁଡ଼ିକ ପୃଥକ କରିବାକୁ ବ୍ୟବହୃତ ହୁଏ।
ସେମିକୋଲୋନ୍ ;	ଏକ ବିବୃତ୍ତିର ସମାପ୍ତିପାଇଁ ବ୍ୟବହୃତ ହୋଇଥାଏ।

ଅପରେଟର (Operators)

ଅପରେଟର ହେଉଛି C ଭାଷାର କ୍ରିୟା ଯାହା ବ୍ୟବହାରକାରୀକୁ ମୂଲ୍ୟ ଉପରେ କାର୍ଯ୍ୟ କରିବାକୁ ଦେଇଥାଏ । C ଭାଷାର ଅପରେଟରମାନଙ୍କର ଏକ ସମୃଦ୍ଧ ସେଟ୍ ହେଉଛି ଏହାର ଏକ ବିଶିଷ୍ଟ ବୈଶିଷ୍ଟ୍ୟ ।

ଏହି ଅପରେଟର୍‌ଗୁଡ଼ିକ ନିମ୍ନଲିଖିତ ବର୍ଗଗୁଡ଼ିକରେ ଗଣାଯାଇପାରିବ:

- ଗାଣିତିକ (Arithmetic) ଅପରେଟର୍ (+, -, *, /, %)
- ସମ୍ବନ୍ଧୀୟ (Relational) ଅପରେଟର୍ (<, <=, >, >=, =, ==)
- ଲଜିକାଲ୍ (Logical) ଅପରେଟର୍ (!, &&, ||)
- ବିଟ୍-ୱାଇଜ୍ (Bit-wise) ଅପରେଟର୍ (~, >>, <<, &, |)
- ବିଶେଷ (Special) ଅପରେଟର
- ବୃଦ୍ଧି (Increment) ଏବଂ ହ୍ରାସ (Decrement) ଅପରେଟର୍ (++ , - -)
- sizeof ଅପରେଟର
- ଠିକଣା (addressof) ଅପରେଟର (&)
- ଇନ୍‌ଡାଇରେକ୍‌ସନ୍ (Indirection)/ଡି-ରେଫରେନ୍‌ସ (de-reference) ଅପରେଟର (*)
- ଟେର୍ନାରୀ (Ternary)/ସର୍ତ୍ତମୂଳକ (Conditional) ଅପରେଟର (? :)

1.4.3.3 ଡାଟା ଟାଇପ୍‌ର ଧାରଣା (Concept of Data Type)

ଏକ ଡାଟା ଟାଇପ୍ (type) ବିଟ୍‌ଗୁଡିକର ଏକ ଷ୍ଟ୍ରିଙ୍ଗରେ ପ୍ରୟୋଗ ହୋଇଥିବା ଏକ ବ୍ୟାଖ୍ୟା । ଔପଚାରିକ ଭାବେ, ଡାଟା ଟାଇପ୍‌କୁ ମୂଲ୍ୟଗୁଡ଼ିକର ଏକ ସସୀମ ସେଟ୍ ଓ ଏହା ସହିତ ଏହି ମୂଲ୍ୟଗୁଡ଼ିକ ଉପରେ ପ୍ରୟୋଗ ହୋଇପାରୁଥିବା ଉତ୍ତମରୂପେ ବର୍ଣ୍ଣିତ ନିୟମଗୁଡ଼ିକର ଏକ ସେଟ୍ ଭାବରେ ବର୍ଣ୍ଣନା କରାଯାଏ ।

C ରେ ଡାଟା ଟାଇପ୍‌ଗୁଡ଼ିକ ଦୁଇ ପ୍ରକାରର:

- **ପୂର୍ବ-ପ୍ରସ୍ତୁତ/ବିଲ୍ଟ-ଇନ୍ ଡାଟା ଟାଇପ୍ (Built-in Data Types)**
  - » ବିଲ୍ଟ-ଇନ୍ ଡାଟା ଟାଇପ୍‌ଗୁଡ଼ିକ ହେଉଛି C ର ସବୁଠାରୁ ମୌଳିକ ଡାଟା ଟାଇପ୍ ।
  - » ବିଲ୍ଟ-ଇନ୍‌ର ଅର୍ଥ ହେଉଛି ସେଗୁଡ଼ିକ C ଭାଷାରେ ପୂର୍ବରୁ ବ୍ୟାଖ୍ୟା କରାଯାଇଛି ଏବଂ ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ସିଧାସଳଖ ବ୍ୟବହୃତ ହୋଇପାରିବ ।
  - » ଉଦାହରଣଗୁଡ଼ିକ ହେଉଛି କ୍ୟାର୍ (char), ଇଣ୍ଟ୍ (int), ଫ୍ଲୋଟ୍ (float), ଏବଂ ଡବଲ୍ (Double) ।
  - » ଏଗୁଡ଼ିକ ବ୍ୟତୀତ, ଆମର ମଧ୍ୟ ଭଏଡ୍ (void)ଡାଟା ଟାଇପ୍ ଅଛି ।
- **ବ୍ୟୁତ୍ପନ୍ନ/ଡିରାଇଭ୍‌ଡ ଡାଟା ଟାଇପ୍ (Derived Data Types)**
  - » ଏଗୁଡ଼ିକ ବିଦ୍ୟମାନ ଡାଟା ଟାଇପ୍‌ଗୁଡିକରୁ (ବିଲ୍ଟ-ଇନ୍ କିମ୍ବା ବ୍ୟବହାରକାରୀ-ବ୍ୟାଖ୍ୟା) ବ୍ୟୁତ୍ପନ୍ନ ହୋଇଥାଏ ।
  - » ଉଦାହରଣଗୁଡ଼ିକ ହେଉଛି ଆରେ (array) ଏବଂ ପଏଣ୍ଟର୍ (pointer) ।

- **ବ୍ୟବହାରକାରୀ- ବର୍ଣ୍ଣିତ ଡାଟା ଟାଇପ୍ (User-defined Data Types)**
  - » ଏଗୁଡ଼ିକ ବ୍ୟବହାରକାରୀଙ୍କଦ୍ୱାରା ସେମାନଙ୍କର ଆବଶ୍ୟକତା ପୂରଣ କରିବାପାଇଁ ସୃଷ୍ଟି କରାଯାଏ ।
  - » ଉଦାହରଣଗୁଡ଼ିକ ହେଉଛି ଷ୍ଟ୍ରକ୍‌ଚର (structure), ୟୁନିଅନ (union), ଏବଂ ଏନୁମେରେସନ (enumeration) ।

ଏହି ବିଭାଗରେ, ଆମେ କେବଳ ବିଲ୍ଟ-ଇନ୍ ଡାଟା ଟାଇପ୍ ବିଷୟରେ ଶିଖିବା । ଅବଶିଷ୍ଟ ଡାଟା ଟାଇପ୍‌ଗୁଡ଼ିକ, ପାଠ୍ୟକ୍ରମର ଆବଶ୍ୟକତା ଅନୁଯାୟୀ ଉପଯୁକ୍ତ ସ୍ଥାନରେ ଆଲୋଚନା କରାଯିବ ।

C ଦ୍ୱାରା ସମର୍ଥିତ ବିଭିନ୍ନ ବିଲ୍ଟ-ଇନ୍ ଡାଟା ଟାଇପ୍, ମୌଳିକ ଡାଟା ଟାଇପ୍ କିମ୍ବା ମୂଳ ଡାଟା ଟାଇପ୍ ଭାବରେ ମଧ୍ୟ ଜଣା, ଟେବୁଲ୍ 1.5 ରେ ପ୍ରଦର୍ଶିତ ହୋଇଛି ।

ଟେବୁଲ୍ 1.5: ବିଲ୍ଟ-ଇନ୍ ଡାଟା ଟାଇପ୍

ନାମ	ବର୍ଣ୍ଣନା
ଇଣ୍ଟିଜର Integer Int	ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା
ରିୟଲ/ଫ୍ଲୋଟିଂ-ପଏଣ୍ଟ Real/Floating-point float double	ଏକକ ସଠିକତା ଫ୍ଲୋଟିଂ-ପଏଣ୍ଟ ନମ୍ବରଗୁଡ଼ିକ (ସଠିକତା ହେଉଛି ଦଶମିକ 6 ସ୍ଥାନ ପର୍ଯ୍ୟନ୍ତ) ଡବଲ୍ ସଠିକତା ଫ୍ଲୋଟିଂ-ପଏଣ୍ଟ ନମ୍ବରଗୁଡ଼ିକ (ସଠିକତା ହେଉଛି ଦଶମିକ 12 ସ୍ଥାନ ପର୍ଯ୍ୟନ୍ତ)
କ୍ୟାରେକ୍ଟର Charater char	ଗୋଟିଏ ଅକ୍ଷର/ସଂକେତ

int ଡାଟା ଟାଇପ୍ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାପାଇଁ ବ୍ୟବହୃତ ହୁଏ, ଏବଂ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାର ଏକ ଉପ-ସେଟ୍‌କୁ ନେଇ ଗଠିତ । ଟେବୁଲ୍ 1.6 ବିଭିନ୍ନ ପ୍ରକାରର ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା, ସେମାନଙ୍କର ମେମୋରି ଆବଶ୍ୟକତା, ଏବଂ ପ୍ରତ୍ୟେକ ଡାଟା ଟାଇପ୍‌ପାଇଁ ମୂଲ୍ୟର ପରିସର ଦର୍ଶାଏ ।

ଟେବୁଲ୍ 1.6: ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ନମ୍ବରର ପ୍ରକାର

ପ୍ରକାର	ମେମୋରି ଆବଶ୍ୟକତା (ବାଇଟ୍ ରେ)		ମୂଲ୍ୟର ପରିସର
	16-ବିଟ୍ କମ୍ପାଇଲର୍ (ଟର୍ବୋ ସି/ସି++)	32-ବିଟ୍ କମ୍ପାଇଲର୍ (ଡେଭ୍ ସି/ସି++)	
short	2	2	2-ବାଇଟ୍ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ପାଇଁ – 32768 ରୁ +32767
int	2	4	
long	4	4	4-ବାଇଟ୍ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ପାଇଁ – 2147483647 ରୁ +2147483648

float ଏବଂ double ଡାଟା ଟାଇପ୍ ଗୁଡ଼ିକ ସଂଖ୍ୟାପାଇଁ ବ୍ୟବହୃତ ହୁଏ ଯାହାର ଏକ ଦଶମିକ ବିନ୍ଦୁ ଥାଏ । float ଏବଂ double ଡାଟା ଟାଇପ୍ ମଧ୍ୟରେ ଏକମାତ୍ର ପାର୍ଥକ୍ୟ ହେଉଛି ପରିସର (range) ଏବଂ ସଠିକତା (precision), ଫ୍ଲୋଟ୍ ଡାଟା ଟାଇପ୍ ତୁଳନାରେ ଡବଲ୍ ଡାଟା ଟାଇପ୍ର ଅଧିକ ସଠିକତା ଅଛି ।

ଏକ ବାସ୍ତବ ମୂଲ୍ୟ ହେଉଛି ଏକ ବାସ୍ତବ ସଂଖ୍ୟାର ନିର୍ଦ୍ଦିଷ୍ଟ ଦଶମିକ ସ୍ଥିତି ପର୍ଯ୍ୟନ୍ତ ଠିକ୍ ଆସନ୍ନ ମାନ । float ପ୍ରକାରର ଏକ ବାସ୍ତବ ମୂଲ୍ୟ ଦଶମିକ ବିନ୍ଦୁ ପରେ ଛଅଟି ସ୍ଥାନ ପର୍ଯ୍ୟନ୍ତ ଠିକ୍; ସେଯାଏ double ପ୍ରକାରର ଏକ ବାସ୍ତବ ମୂଲ୍ୟ ଦଶମିକ ବିନ୍ଦୁ ପରେ ବାରଟି ସ୍ଥାନ ପର୍ଯ୍ୟନ୍ତ ହୋଇପାରେ ।

ଟେବୁଲ୍ 1.7 ବିଭିନ୍ନ ପ୍ରକାରର ବାସ୍ତବ ସଂଖ୍ୟା, ସେମାନଙ୍କର ମେମୋରି ଆବଶ୍ୟକତା, ଏବଂ ପ୍ରତ୍ୟେକ ଡାଟା ଟାଇପ୍ପାଇଁ ମୂଲ୍ୟର ପରିସର ଦେଖାଏ ।

ଟେବୁଲ୍ 1.7: ବାସ୍ତବ ସଂଖ୍ୟାର ଡାଟା ଟାଇପ୍

ଟାଇପ୍	ବାଇଟ୍	ପରିସର
float	4	3.4×10^{-38} to $3.4 \times 10^{+38}$
double	8	1.7×10^{-308} to $1.7 \times 10^{+308}$

char ଡାଟା ଟାଇପ୍ ବର୍ଣ୍ଣଗୁଡ଼ିକପାଇଁ ବ୍ୟବହୃତ ହୁଏ, ଏବଂ ଏହା କେବଳ 1-ବାଇଟ୍ ମେମୋରି ଆବଶ୍ୟକ କରେ । ଏଠାରେ char ଡାଟା ଟାଇପ୍ ଏକ ବିଶେଷତା ଅଛି - ବର୍ଣ୍ଣଗୁଡ଼ିକ ସେମାନଙ୍କର ଆସ୍କି (ascii) କୋଡ୍ ବ୍ୟବହାର କରି ମେମୋରିରେ ଗଚ୍ଛିତ ହୁଏ ଯାହା ସଂଖ୍ୟାତ୍ମକ, ଅର୍ଥାତ, ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ।

ଭଏଡ଼ (void) ଶବ୍ଦ ଅର୍ଥ ଖାଲି, ତେଣୁ, ଆମେ କହିପାରିବା void ଡାଟା ଟାଇପ୍ର କୌଣସି ମୂଲ୍ୟ ନାହିଁ ।

ଏହା ଅନେକ ପରିସ୍ଥିତିରେ ଉପଯୋଗୀ । ଏହିପରି ପରିସ୍ଥିତି ମଧ୍ୟରୁ ଗୋଟିଏ - ଏହା ଏକ ମୂଲ୍ୟ ଫେରସ୍ତ କରୁ ନ ଥିବା ଫଙ୍କସନପାଇଁ ଫେରସ୍ତ ଟାଇପ୍ ଭାବରେ ବ୍ୟବହୃତ ହୁଏ ।

1.4.3.4 ଧ୍ରୁବକ (Constants)

ଏହା ଏକ ନିରନ୍ତର ସ୍ଥିର ମୂଲ୍ୟକୁ ସୂଚିତ କରେ । ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ କରିବା ସମୟରେ ଏହାର ପରିବର୍ତ୍ତନ ହୁଏ ନାହିଁ । ନିମ୍ନଲିଖିତ ଉପାୟଗୁଡ଼ିକ ବ୍ୟବହାର କରି ଧ୍ରୁବକଗୁଡ଼ିକୁ ପରିଚାଳନା କରାଯାଇପାରିବ:

- **ଧ୍ରୁବକ ମୂଲ୍ୟର ବ୍ୟବହାର** (Using literal) – ସିଧାସଳଖ ଏକ ଗାଣିତିକ ପରିପ୍ରକାଶର ମୂଲ୍ୟକୁ ଏନକୋଡିଂ (encoding) କରିବା ।
- **ସାଙ୍କେତିକ ଧ୍ରୁବକର ବ୍ୟବହାର** (Using symbolic constants) – ଏକ ସାଙ୍କେତିକ ଧ୍ରୁବକ ହେଉଛି ଏକ ନାମ ଯାହା ଏକ ଧ୍ରୁବକ ମୂଲ୍ୟ ପ୍ରତିସ୍ଥାପିତ କରେ ।

ଉଦାହରଣ ବଶତଃ

```
#define PI 3.142
```

ଏଠାରେ, #define ପ୍ରି-ପ୍ରୋସେସର୍ ନିର୍ଦ୍ଦେଶନାମାଟି ଧ୍ରୁବକ ମୂଲ୍ୟ 3.142 ସହିତ PI ନାମକୁ ସଂପୃକ୍ତ କରେ ।

କମ୍ପାଇଲ୍ ସମୟରେ, ପ୍ରିପ୍ରୋସେସର୍ ନାମ PI ର ପ୍ରତ୍ୟେକ ଉପସ୍ଥିତିକୁ ଧ୍ରୁବକ ମୂଲ୍ୟ 3.142 ସହିତ ବଦଳାଇଥାଏ ।

- ଧ୍ରୁବକ ଭାବରେ ଘୋଷିତ ଏକ ଚଳର ବ୍ୟବହାର (Using a variable declared as constant) – ଏକ ଚଳ, ଯାହାକି ପରବର୍ତ୍ତୀ ସମୟରେ ଆଲୋଚନା କରାଯାଇଛି, ଏପରି ଏକ ଜିନିଷ ଯାହାର ମୂଲ୍ୟ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ସମୟରେ ପରିବର୍ତ୍ତନ ହୋଇପାରେ । କିନ୍ତୁ, କୀୱାର୍ଡ୍ const ବ୍ୟବହାର କରି ଚଳଟିଏ ଧ୍ରୁବକ ଭାବରେ ଚିହ୍ନିତ ହୋଇପାରିବ ।

ଉଦାହରଣ ବଶତଃ

```
const float pie = 3.142;
```

ଏଠାରେ, pie ନାମକ ଚଳ, ଏହାର ଡାଟା ଟାଇପ୍ float ଅଟେ, ପ୍ରାଥମିକ ମୂଲ୍ୟ 3.141 ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇଛି, ଏବଂ ଧ୍ରୁବକ ଭାବରେ ଚିହ୍ନିତ ହୋଇଛି । ଏହା ପରେ, ଆମେ ଆମର ପ୍ରୋଗ୍ରାମ୍‌ରେ ଚଳର ମୂଲ୍ୟ ବ୍ୟବହାର କରିପାରିବା, କିନ୍ତୁ ଏହାର ମୂଲ୍ୟ ପରିବର୍ତ୍ତନ କରିପାରିବା ନାହିଁ । ଅର୍ଥାତ ଚଳ pie, ଏକ ଧ୍ରୁବକ ଭାବରେ ଆଚରଣ କରିପାରିବ ।

1.4.3.5 ଚଳ (variables)

ଏକ ଚଳ ଆମକୁ ଏକ ନାମିତ ଭଣ୍ଡାରଣ ପ୍ରଦାନ କରେ ଯେଉଁଥିରେ ଆମେ ପ୍ରୋଗ୍ରାମ୍ ସମୟରେ ଲେଖିପାରିବା, ପୁନଃପ୍ରାପ୍ତ, ଏବଂ ଅଦଳବଦଳ କରିପାରିବା ।

ଅନ୍ୟ ଭାବରେ କହିଲେ, ଚଳ ହେଉଛି କମ୍ପ୍ୟୁଟରର ମେମୋରିର ଏକ ମେମୋରି ଅବସ୍ଥାନ । ଚଳର ବିଷୟବସ୍ତୁ ବିଭିନ୍ନ ସମୟରେ ଭିନ୍ନ ହୋଇପାରେ ଡାଟାର ଧାରଣ କରିପାରୁଥିବା – ତେଣୁ ଚଳ ବୋଲି ନାମ ଦିଆଯାଇଛି । ଚଳଗୁଡ଼ିକ ବିଭିନ୍ନ ପ୍ରକାରର ଡାଟା ଧାରଣ କରିପାରେ ଏବଂ ସମାନ ଚଳ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ସମୟରେ ବିଭିନ୍ନ ମୂଲ୍ୟ ଧାରଣ କରିପାରେ ।

C ଭାଷାରେ ପ୍ରତ୍ୟେକ ଚଳ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଡାଟା ପ୍ରକାର ସହିତ ଜଡ଼ିତ । ଏହାର ସଂଚିତ ମେମୋରିର ଆକାର ଉକ୍ତ ମେମୋରି ମଧ୍ୟରେ ଗଚ୍ଛିତ ମୂଲ୍ୟର ପରିସର, ଲେଆଉଟ୍ ଏବଂ ଏହା ଉପରେ ପ୍ରୟୋଗ ହୋଇପାରୁଥିବା ଏକ ଅପରେସନ୍ ସେଟ୍ ଇତ୍ୟାଦି ନିର୍ଦ୍ଧାରଣ କରିପାରେ ।

ଚଳର ଘୋଷଣା (Declaring Variables)

C ଭାଷାରେ, ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ପ୍ରତ୍ୟେକ ଚଳକୁ ଘୋଷଣା କରାଯିବା ଆବଶ୍ୟକ। ଏକ ଚଳର ଘୋଷଣା ଏବଂ ବ୍ୟାଖ୍ୟା କରିବାପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
type v1, v2, . . ., vn;
```

ଯେଉଁଠାରେ v1, v2,..., vn କମାଦ୍ୱାରା ଅଲଗା ହୋଇଥିବା ଚଳର ନାମ।

ଚଳ ଘୋଷଣା ଏବଂ ସଂଜ୍ଞାର କିଛି ଉଦାହରଣ ନିମ୍ନରେ ଦିଆଯାଇଛି:

```
int count, m, n;
float value, sum;
char ch;
double deviation;
```

ପ୍ରଥମ ବିବୃତ୍ତିଟି ଚଳ count, m ଓ n କୁ int ପ୍ରକାରର ଘୋଷଣା କରୁଛି। ଉକ୍ତ ପ୍ରତ୍ୟେକ ଚଳଗୁଡ଼ିକପାଇଁ କମ୍ପାଇଲର୍ 2 କିମ୍ବା 4 ବାଇଟ୍ (କମ୍ପାଇଲର୍ ଉପରେ ନିର୍ଭର କରି) ମେମୋରି ସଂରକ୍ଷଣ କରିଥାଏ।

ଦ୍ୱିତୀୟ ବିବୃତ୍ତିଟି ଚଳ value ଏବଂ sum କୁ float ଟାଇପ୍‌ର ଘୋଷଣା କରେ। ଏହି ପ୍ରତ୍ୟେକ ଚଳଗୁଡ଼ିକପାଇଁ କମ୍ପାଇଲର୍ 4 ବାଇଟ୍ ମେମୋରି ସଂରକ୍ଷଣ କରିଥାଏ।

ତୃତୀୟ ବିବୃତ୍ତିଟି ଚଳ ch କୁ char ଟାଇପ୍‌ର ଘୋଷଣା କରେ। କମ୍ପାଇଲର୍ ଏଥିପାଇଁ 1ବାଇଟ୍ ମେମୋରି ସଂରକ୍ଷଣ କରିଥାଏ।

ଚତୁର୍ଥ ବିବୃତ୍ତିଟି ଚଳ deviation କୁ double ଟାଇପ୍‌ର ଘୋଷଣା କରେ। କମ୍ପାଇଲର୍ ଏଥିପାଇଁ ୫ ବାଇଟ୍ ମେମୋରି ସଂରକ୍ଷଣ କରିଥାଏ।

ଚଳଗୁଡ଼ିକର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟକରଣ (Initializing Variables)

ଚଳ ଘୋଷଣା କରିବା ସହିତ, ଏହାକୁ ଏକ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ମଧ୍ୟ ପ୍ରଦାନ କରାଯାଇପାରିବ।

ଏହା କରିବା ପାଇଁ, ଏକ ଚଳ ପରେ ସଙ୍କେତ '=' ଏବଂ ତା'ପରେ ଚଳକୁ ଦିଆଯିବାକୁ ଥିବା ମୂଲ୍ୟକୁ ଲେଖାଯାଏ। ନିମ୍ନଲିଖିତ ବିବୃତ୍ତିଟିକୁ ବିଚାର କର –

```
int sum = 0;
```

ଏହା ଚଳ sum କୁ ମୂଲ୍ୟ 0 ସହିତ ଆରମ୍ଭ କରେ।

1.4.3.6 **ପରିପ୍ରକାଶ** (Expressions)

ଗୋଟିଏ ପରିପ୍ରକାଶ ଗୋଟିଏ ମୂଲ୍ୟ ଗଣନାର ଏକ ସୂତ୍ର। ଏହା ଅପରେଟର ଏବଂ ଅପେରାଣ୍ଡଙ୍କ ଏକ ଅନୁକ୍ରମକୁ ନେଇ ଗଠିତ। ଅପେରାଣ୍ଡରେ ଫଙ୍କସନ୍ ସନ୍ଦର୍ଭ, ଚଳ, ଏବଂ ଧ୍ରୁବକ ଇତ୍ୟାଦି ରହିପାରନ୍ତି। ଅପରେଟର୍‌ମାନେ ଅପେରାଣ୍ଡଗୁଡ଼ିକ ଉପରେ କରାଯାଉଥିବା କାର୍ଯ୍ୟକୁ ନିର୍ଦ୍ଦିଷ୍ଟ କରେ।

ନିମ୍ନଲିଖିତ ପରିପ୍ରକାଶରେ

```
a + b
```

ଯୁକ୍ତ (+) ଏକ ଅପରେଟର୍ ଏବଂ a, b ହେଉଛନ୍ତି ଅପେରାଣ୍ଡ୍।

C ଭାଷା ନିମ୍ନଲିଖିତ ପ୍ରକାରର ପରିପ୍ରକାଶଗୁଡ଼ିକୁ ସମର୍ଥନ କରେ:

- ଗାଣିତିକ ପରିପ୍ରକାଶ (Arithmetic expressions)
- ସମ୍ବନ୍ଧିତ ପରିପ୍ରକାଶ (Relational expressions)
- ତାର୍କିକ ପରିପ୍ରକାଶ (Logical expressions)

ପ୍ରତ୍ୟେକ ପ୍ରକାରର ପରିପ୍ରକାଶ ନିର୍ଦ୍ଦିଷ୍ଟ ପ୍ରକାରର ଅପେରାଣ୍ଡ୍‌କୁ ନେଇଥାଏ ଏବଂ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ସେଟ୍ ଅପରେଟର୍‌ର ବ୍ୟବହାର କରେ। ପ୍ରତ୍ୟେକ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ ନିର୍ଦ୍ଦିଷ୍ଟ ପ୍ରକାରର ମୂଲ୍ୟ ପ୍ରସ୍ତୁତ କରେ। ମନେରଖିବା ଉଚିତ ଯେ ପରିପ୍ରକାଶ ବିବୃତ୍ତି ନୁହେଁ, ବରଂ ବିବୃତ୍ତିର ଅଂଶ ହୋଇପାରେ।

ଉଦାହରଣ ସ୍ୱରୂପ, ନିମ୍ନଲିଖିତ ଟେକ୍ସ୍ଟ (text) ଧାଡ଼ିଟି ବିଚାର କର

```
x = 2.0/3.0 + a * b;
```

ସମଗ୍ର ଧାଡ଼ିଟି ଏକ ବିବୃତ୍ତି, କିନ୍ତୁ ସମାନ "=" ସଙ୍କେତ ପର ଅଂଶଟି ଏକ ପରିପ୍ରକାଶ। ବିଶେଷ କରି, ପାଠ୍ୟର ଏହି ଧାଡ଼ି ଏକ ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତିକୁ ପ୍ରତିନିଧିତ୍ୱ କରେ ଯାହା ପରିପ୍ରକାଶର ମୂଲ୍ୟକୁ ଚଳ x କୁ ନ୍ୟସ୍ତ କରେ।

1.4.3.7 ବିବୃତ୍ତି (Statements)

ବିବୃତ୍ତିଗୁଡ଼ିକ ଅନେକ କାର୍ଯ୍ୟ ସମ୍ପାଦନ କରେ। ଯେପରିକି ଗଣନା କରିବା, ଗଣନାର ଫଳାଫଳକୁ ଗଚ୍ଛିତ କରିବା, ନିୟନ୍ତ୍ରଣର ପ୍ରବାହ ପରିବର୍ତ୍ତନ କରିବା, ଯନ୍ତ୍ରାଂଶ /ଫାଇଲ୍‌ରୁ ପଢିବା ଏବଂ ଲେଖିବା, ଏବଂ କମ୍ପାଇଲ୍‌ପାଇଁ ସୂଚନା ପ୍ରଦାନ କରିବା।

1.4.3.8 ଇନପୁଟ୍/ଆଉଟପୁଟ୍‌ର ପରିଚାଳନା (Handling Input/Output)

ପ୍ରାୟ ପ୍ରତ୍ୟେକ ପ୍ରୋଗ୍ରାମ୍‌ରେ, ବ୍ୟବହାରକାରୀଙ୍କୁ କିଛି ଡାଟା ଇନପୁଟ୍ କରିବାକୁ ପଡିଥାଏ। ଏହା ମେମୋରିରେ ଗଚ୍ଛିତ ରହେ। ତା'ପରେ ପରବର୍ତ୍ତୀ ସମୟରେ ପ୍ରକ୍ରିୟାକରଣ ହୋଇଥାଏ। କମ୍ପ୍ୟୁଟେସନ୍‌ର ମଧ୍ୟବର୍ତ୍ତୀ ଏବଂ ଅନ୍ତିମ ଫଳାଫଳ ମଧ୍ୟ ମେମୋରିରେ ଗଚ୍ଛିତ ହୁଏ, ଏବଂ ଶେଷରେ, କମ୍ପ୍ୟୁଟେସନ୍‌ର ଫଳାଫଳକୁ ଆଉଟପୁଟ୍ କରିବାର ଆବଶ୍ୟକତା ଅଛି, ଫଳରେ ବ୍ୟବହାରକାରୀ ସେଗୁଡ଼ିକୁ ସେମାନଙ୍କର ଦୈନନ୍ଦିନ କାର୍ଯ୍ୟରେ ବ୍ୟବହାର କରିପାରିବେ।

ସାଧାରଣ ଇନପୁଟ୍ ଉପକରଣ ହେଉଛି କୀବୋର୍ଡ୍ ଏବଂ ଆଉଟପୁଟ୍ ଉପକରଣ ହେଉଛି ଏକ କମ୍ପ୍ୟୁଟରର ପରଦା, ଏହାକୁ ମନିଟର୍ (monitor) ମଧ୍ୟ କୁହାଯାଏ। କୀବୋର୍ଡ୍ ଏବଂ ମନିଟର୍‌କୁ ନେଇ ଏହି ଉପ-ସିଷ୍ଟମ୍‌କୁ ମିଳିତ ଭାବରେ କନସୋଲ୍ (console) କୁହାଯାଏ।

I/O ଫଙ୍କ୍ସନଗୁଡ଼ିକର ସମ୍ପୂର୍ଣ୍ଣ ପରିସରକୁ ବ୍ୟବହାର କରିବାକୁ, ଆମକୁ stdio.h ନାମକ ହେଡର୍ ଫାଇଲ୍‌କୁ ଆମର ସମସ୍ତ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଅନ୍ତର୍ଭୁକ୍ତ କରିବାକୁ ହେବ।

ଟେବୁଲ୍ 1.୫: I/O ଫଙ୍କ୍ସନଗୁଡ଼ିକ

ଫଙ୍କ୍ସନ୍‌ର ବର୍ଗ	ଫଙ୍କ୍ସନ୍ ନାମ	ବର୍ଣ୍ଣନା
ଅଣ ଫର୍ମାଟେଡ୍ (Unformatted)	getchar ( )	ବର୍ତ୍ତମାନ ଟଇପ୍ ହୋଇଥିବା ଏକ ବର୍ଣ୍ଣ ଫେରସ୍ତ କରେ। ଟାଇପ୍ ହୋଇଥିବା ବର୍ଣ୍ଣ କମ୍ପ୍ୟୁଟର ସ୍କ୍ରିନ୍‌ରେ ପ୍ରତିଧ୍ୱନିତ ହୁଏ। ଉପଯୁକ୍ତ ବର୍ଣ୍ଣ ଟାଇପ୍ କରିବା ପରେ, ବ୍ୟବହାରକାରୀ Enter କୀ ଦବାଇବାକୁ ପଡ଼େ।
	getche ( )	ବର୍ତ୍ତମାନ ଟଇପ୍ ହୋଇଥିବା ଏକ ବର୍ଣ୍ଣ ଫେରସ୍ତ କରେ। ଟାଇପ୍ ହୋଇଥିବା ବର୍ଣ୍ଣ ମଧ୍ୟ କମ୍ପ୍ୟୁଟର ସ୍କ୍ରିନ୍‌ରେ ପ୍ରତିଧ୍ୱନିତ ହୁଏ। କିନ୍ତୁ ବ୍ୟବହାରକାରୀକୁ Enter କୀ ଦବାଇବାର ଆବଶ୍ୟକ ନାହିଁ।
	getch ( )	ବର୍ତ୍ତମାନ ଟଇପ୍ ଏକ ବର୍ଣ୍ଣ ଫେରସ୍ତ କରେ। କିନ୍ତୁ, ବ୍ୟବହାରକାରୀକୁ Enter କୀ ଦବାଇବାକୁ ଆବଶ୍ୟକ ନାହିଁ କିମ୍ବା ଟାଇପ୍ ହୋଇଥିବା ବର୍ଣ୍ଣ କମ୍ପ୍ୟୁଟର ସ୍କ୍ରିନ୍‌ରେ ପ୍ରତିଧ୍ୱନିତ ହୁଏ ନାହିଁ।
	putchar ( )	ସ୍କ୍ରିନ୍‌ରେ ଏକ ବର୍ଣ୍ଣ ପ୍ରଦର୍ଶନ କରେ।
ଫର୍ମାଟେଡ୍ (Formatted)	ghets ( )	କୀବୋର୍ଡ୍‌ରୁ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଗ୍ରହଣ କରେ।
	puts ( )	ସ୍କ୍ରିନ୍‌ରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପ୍ରଦର୍ଶନ କରେ।
	scanf ( )	କୀ'ବୋର୍ଡ୍‌ରୁ ଫର୍ମାଟେଡ୍ ଡାଟା ଗ୍ରହଣ କରେ।
	printf ( )	ଫର୍ମାଟେଡ୍ ଡାଟା ସ୍କ୍ରିନ୍‌ରେ ପ୍ରଦର୍ଶନ କରେ।

ଏହି ଫଙ୍କ୍ସନଗୁଡ଼ିକ ମଧ୍ୟରେ ଏକମାତ୍ର ପାର୍ଥକ୍ୟ ହେଉଛି ଯେ ଫର୍ମାଟେଡ୍ ଫଙ୍କ୍ସନଗୁଡ଼ିକଦ୍ୱାରା ଆବଶ୍ୟକତା ଅନୁଯାୟୀ କୀ'ବୋର୍ଡ୍‌ରୁ ନେବାକୁ ଥିବା ଇନପୁଟ୍ କିମ୍ବା ସ୍କ୍ରିନ୍‌କୁ ପଠାଇବାକୁ ଥିବା ଆଉଟପୁଟ୍‌ର ଫର୍ମାଟ୍ କରିବାକୁ ଅନୁମତି ଦିଏ।

ଉଦାହରଣ ସ୍ୱରୂପ, ବିଭିନ୍ନ ମୂଲ୍ୟ ଯଦି ପ୍ରଦର୍ଶିତ କରିବାକୁ ଥାଏ, ଯେପରିକି ସ୍କ୍ରିନ୍‌ରେ କେତୋଟି ସ୍ତମ୍ଭ ବ୍ୟବହାର କରାଯିବ, ଏବଂ ଦୁଇଟି ମୂଲ୍ୟ ମଧ୍ୟରେ କେତେ ଫାଙ୍କାସ୍ଥାନ ଦିଆଯିବ। ଯଦି ପ୍ରଦର୍ଶିତ ହେବାକୁ ଥିବା ମୂଲ୍ୟ ବାସ୍ତବ ଟାଇପ୍‌ର, ତେବେ ଆଉଟପୁଟ୍‌ପାଇଁ କେତେ ଦଶମିକ ପରେ କେତୋଟି ସ୍ଥାନ ଦିଆଯିବ ଇତ୍ୟାଦି।

Listing 1.3

```
/*
    Program to demonstrate working of character I/O func-
tions
 */
 #include <stdio.h>
 int main()
 {
    char ch;
    printf( "\nEnter any character: " );
    ch = getchar();
    printf( "\nCharacter you typed is " );
    putchar(ch);
    printf( "\nEnter any character: " );
    ch = getche();
    printf( "\nCharacter you typed is " );
    putchar(ch);
    printf( "\nEnter any character: " );
    ch = getch();
    printf( "\nCharacter you typed is " );
    putchar(ch);
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter any character: A
Character you typed is A
Enter any other character: x
Character you typed is x
Enter any other character:
Character you typed is H
```

Listing 1.4

```
/*
    Program to illustrate working of string I/O functions
 */
 #include <stdio.h>
 int main()
 {
    char str[31];
    printf( "\nEnter string of length <= 30: " );
    gets( str );
    printf( "\nString you typed is " );
    puts( str );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter string of length <= 30: Wel Come
String you typed is Wel Come
```

Scanf () ଫଙ୍କସନ୍ ଆମକୁ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଫର୍ମାଟରେ ଡାଟା ଇନପୁଟ୍ କରିବାକୁ ଅନୁମତି ଦିଏ । ଏହାର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
scanf( "format string", list of addresses of variables );
```

ଯେଉଁଠାରେ କି ଫର୍ମାଟ୍ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଫର୍ମାଟ୍, ସଙ୍କେତ '%' ସହିତ ଆରମ୍ଭ ହେଉଥିବା ସ୍ପେସିଫାୟର୍‌ଗୁଡ଼ିକୁ ଧାରଣ କରେ ଏବଂ ଏଗୁଡ଼ିକୁ ସ୍ପେସ୍ କିମ୍ବା କମାଦ୍ୱାରା ଅଲଗା କରାଯାଇଥାଏ ।

ଚଳଗୁଡ଼ିକର ଠିକଣାର ଏକ ତାଲିକା ବ୍ୟବହାର କରାଯାଏ ଯାହାଦ୍ୱାରା scanf() ଫଙ୍କସନ୍ କୀବୋର୍ଡ଼ରୁ ପ୍ରାପ୍ତ ହୋଇଥିବା ଡାଟା ରଖିପାରେ । ଠିକଣା ଅପରେଟର୍ (ସଙ୍କେତ &) ବ୍ୟବହାର କରି ଏକ ଚଳର ଠିକଣା ପ୍ରାପ୍ତ କରାଯାଏ, ଏବଂ ଏହା addressof(ଆଡ୍ରେସଅଫ୍) ଅପରେଟର କୁହାଯାଏ ।

ଟେବୁଲ୍ 1.9: ସାଧାରଣ ଭାବେ ବ୍ୟବହୃତ ଫର୍ମାଟ୍ ସ୍ପେସିଫାୟର୍‌ଗୁଡ଼ିକର ତାଲିକା

ଫର୍ମାଟ୍ ସ୍ପେସିଫାୟର୍	ପାଇଁ ବ୍ୟବହୃତ
%d	ସଙ୍କେତଯୁକ୍ତ ଡେସିମାଲ୍ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା
%f	ସିଙ୍ଗଲ୍ ପ୍ରିସିସନ ବାସ୍ତବ ଫ୍ଲୋଟ ସଂଖ୍ୟା
%If	ଡବଲ ପ୍ରିସିସନ ବାସ୍ତବ ଫ୍ଲୋଟ ସଂଖ୍ୟା
%c	ଗୋଟିଏ ମାତ୍ର ବର୍ଣ୍ଣ
%c	ଏକ-ଶବ୍ଦ ବିଷିଷ୍ଟ ଷ୍ଟ୍ରିଙ୍ଗ୍

ଚାଳନ ବେଳେ, ନିର୍ଦ୍ଦିଷ୍ଟ ଫର୍ମାଟ୍ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଅନୁଯାୟୀ ଇନପୁଟ୍ ଡାଟା ବିଧିବଦ୍ଧ ଭାବରେ ଭର୍ତ୍ତି କରାଯିବା ଆବଶ୍ୟକ ଅନ୍ୟଥା ଫଳାଫଳ ଅତ୍ୟନ୍ତ ଅଦ୍ଭୁତ ମିଳିପାରେ ।

printf*()* ଫଙ୍କ୍ସନ୍ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଫର୍ମାଟରେ ଡାଟା ଆଉଟପୁଟ୍ କରିବାକୁ ଅନୁମତି ଦିଏ । ଏହାର ସିଣ୍ଟାକ୍ସ

```
printf( "format string", list of variables );
```

ଯେଉଁଠାରେ ଫର୍ମାଟ୍ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ଟି ଫର୍ମାଟ୍ ସ୍ପେସିଫାୟର୍‌ଗଡ଼ିକୁ ଧାରଣ କରେ, ଯେପରିକି ଏସକେପ ସିକ୍ୱେନ୍ସ (escape sequences), ଏବଂ ଆଉଟପୁଟ୍ ହେବାକୁ ଥିବା ଡାଟା ସହିତ ଲେଖା ।

scanf*()* ଫଙ୍କ୍ସନପାଇଁ ବ୍ୟବହୃତ ସମସ୍ତ ଫର୍ମାଟ୍ ସ୍ପେସିଫାୟର୍ printf *()* ଫଙ୍କ୍ସନପାଇଁ ମଧ୍ୟ ବୈଧ ଅଟେ ।

Listing 1.5

```
/*
   Program to demonstrate working of formatted I/O func-
tions
*/
#include <stdio.h>
int main(void)
{
   int a;
   printf( "\nEnter values for a : " );
   scanf ( "%d", &a );
   printf( "\nValue of a = %d\n", a );
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter values for a : 150
Value of a = 150
```

ୟୁନିଟ ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଯାହା ଶିଖିଛୁ

- କମ୍ପ୍ୟୁଟର ହେଉଛି ଏକ ଇଲେକ୍ଟ୍ରୋନିକ୍ ମେସିନ୍ ଏହା ନିର୍ଦ୍ଦେଶାବଳିର ଏକ ସେଟ୍ ଅନୁଯାୟୀ କାର୍ଯ୍ୟ କିମ୍ବା ଗଣନା କରେ ଯାହାକୁ ପ୍ରୋଗ୍ରାମ୍ କୁହାଯାଏ ।
- କମ୍ପ୍ୟୁଟର ବିଭିନ୍ନ ପ୍ରକାରର ଫର୍ମରେ ଇନପୁଟ୍ ଗ୍ରହଣ କରିପାରେ । ନିର୍ଦ୍ଦେଶ ଅନୁଯାୟୀ ଏହି ଇନପୁଟ୍‌ଗୁଡ଼ିକୁ ପ୍ରକ୍ରିୟା କରିପାରେ, ଏବଂ ବିଭିନ୍ନ ପ୍ରକାରର ଫର୍ମରେ ଫଳାଫଳ ଉପସ୍ଥାପନ କରିପାରେ ।
- କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ଅଁଶଗୁଡ଼ିକ ମଧ୍ୟରେ ଇନପୁଟ୍ ୟୁନିଟ, ଆଉଟପୁଟ୍ ୟୁନିଟ, ମେମୋରି ୟୁନିଟ, କଣ୍ଟ୍ରୋଲ୍ ୟୁନିଟ, ଗଣିତ ଏବଂ ଲଜିକ୍ ୟୁନିଟ, ଏବଂ ମାଧ୍ୟମିକ ଭଣ୍ଡାରଣ ୟୁନିଟ ଅନ୍ତର୍ଭୁକ୍ତ ଥାଏ ।
- କଣ୍ଟ୍ରୋଲ୍ ୟୁନିଟ , ଗଣିତ ଏବଂ ଲଜିକ୍ ୟୁନିଟ, ଏବଂ ମୁଖ୍ୟ ମେମୋରି ୟୁନିଟ , ସାମୂହିକ ଭାବରେ, ସେନ୍‌ଟ୍ରାଲ୍ ପ୍ରୋସେସିଂଗ ୟୁନିଟ (CPU) କୁହାଯାଏ ।
- କମ୍ପ୍ୟୁଟରକୁ ସଂଯୋଜିତ ଇନପୁଟ୍ ଉପକରଣ ଏବଂ ଆଉଟପୁଟ୍ ଉପକରଣ ପରି ସମସ୍ତ ବାହ୍ୟ ଉପକରଣଗୁଡ଼ିକୁ ସାଧାରଣ ଭାବରେ ପେରିଫେରାଲ୍ (peripheral) ବୋଲି କୁହାଯାଏ ।
- ହାର୍ଡୱେୟାର କମ୍ପ୍ୟୁଟରର ଏକ ଅଂଶକୁ ବୁଝାଏ କମ୍ପ୍ୟୁଟର ତୁମେ ଏହାକୁ ଦେଖିପାରିବ ଏବଂ ସ୍ପର୍ଶ କରିପାରିବ । ଏହାର ଖୋଳ ଏବଂ ତା' ଭିତରେ ଥିବା ସମସ୍ତ ଜିନିଷ କମ୍ପ୍ୟୁଟର ଅନ୍ତର୍ଭୁକ୍ତ ।
- ସଫ୍ଟୱେୟାର, ସବୁଠାରୁ ସାଧାରଣ ଅର୍ଥରେ ନିର୍ଦ୍ଦେଶାବଳି ବୁଝାଏ କିମ୍ବା ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକର ଏକ ସେଟ୍ ହାର୍ଡୱେୟାରକୁ କ'ଣ କରାଯିବ ତାହା କହିଥାଏ ତା'ର ନିର୍ଦ୍ଦେଶ ଦେଇଥାଏ ।
- ଅପେରେଟିଂ ସିଷ୍ଟମ୍ (OS) କମ୍ପ୍ୟୁଟର ସିଷ୍ଟମର ସାଧନଗୁଡ଼ିକର ପରିଚାଳନା କରେ; ଏକ ଇଣ୍ଟରଫେସ୍ ପ୍ରଦାନ କରେ ଯାହାର ବ୍ୟବହାର କରି ବ୍ୟବହାରକାରୀ ବିଭିନ୍ନ କାର୍ଯ୍ୟ ସମ୍ପାଦନ କରିବାକୁ କମ୍ପ୍ୟୁଟର ସହିତ ଇଣ୍ଟରାକ୍ଟ କରିପାରିବେ ।
- ସଫ୍ଟୱେୟାରଗୁଡ଼ିକୁ ସିଷ୍ଟମ୍ ସଫ୍ଟୱେୟାର, ଆପ୍ଲିକେସନ୍ ସଫ୍ଟୱେୟାର, ଏବଂ ୟୁଟିଲିଟି ସଫ୍ଟୱେୟାର ଭାବରେ ବର୍ଗୀକୃତ କରା ଯାଇପାରିବ ।
- ବୁଟିଂ ଏକ ପ୍ରକ୍ରିୟା ଯାହା ସିଷ୍ଟମକୁ ପ୍ରାରମ୍ଭିକ ହେବାକୁ ସୂଚିତ କରେ ।
- ଏକ ଆଲଗୋରିଦମ୍ ହେଉଛି ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ସମସ୍ୟାର ସମାଧାନକୁ ବ୍ୟାଖ୍ୟା କରୁଥିବା ନିର୍ଦ୍ଦେଶାବଳିର ଏକ ସସୀମ ଅନୁକ୍ରମ ।
- ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ ସ୍ୱିଡୋକୋଡ୍ ବ୍ୟବହାର କରି ଏକ ଆଲଗୋରିଦମ୍‌କୁ ଉପସ୍ଥାପନ କରାଯାଇପାରିବ ।
- ଫ୍ଲୋ'ଚାର୍ଟ ହେଉଛି ଏକ ଆଲଗୋରିଦମର ଏକ ଚିତ୍ରିତ ଉପସ୍ଥାପନ, ଯେଉଁଠାରେ ବିଭିନ୍ନ ପ୍ରକାରର ଅପରେସନ୍‌ର ପ୍ରତିନିଧିତ୍ୱ କରିବାପାଇଁ ବିଭିନ୍ନ ଜ୍ୟାମିତିକ ଆକୃତିଗୁଡ଼ିକ ବ୍ୟବହୃତ ହୁଏ ।
- ସ୍ୟୁଡୋକୋଡ୍ ହେଉଛି ସରଳ ଭାଷାରେ ବର୍ଣ୍ଣିତ ଏକ ଆଲଗୋରିଦମ୍‌ର ସୋପାନଗୁଡ଼ିକ ।
- ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟିଗୁଡ଼ିକ ହେଉଛି ଭାଷା ନିୟମର ଭୁଲ୍ ବ୍ୟବହାରର ପରିଣାମ ।
- ତାର୍କିକ ତ୍ରୁଟି ହେଉଛି ସମସ୍ୟା କିମ୍ବା ଏହାର ସମାଧାନ ବୁଝିବାର ଅଭାବର ପରିଣାମ ।

- C ଭାଷା ଡେନିସ୍ ରିଚିଙ୍କଦ୍ୱାରା ବିକଶିତ।
- C ଭାଷା ହେଉଛି ଏକ ମଧ୍ୟମ-ସ୍ତରର ଭାଷା ଏବଂ ପୋର୍ଟେବଲ୍।
- C ଭାଷା ହେଉଛି କେସ୍ ସମ୍ବେଦନଶୀଳ।
- ଗୋଟିଏ C ପ୍ରୋଗ୍ରାମ୍‌ର ସାଧାରଣ ସଂରଚନା ପାଞ୍ଚଟି ବିଭାଗକୁ ନେଇ ଗଠିତ - ଟିପ୍ପଣୀ, ପ୍ରିପ୍ରୋସେସର୍ ନିର୍ଦ୍ଦେଶାବଳି, ଗ୍ଲୋବାଲ ଘୋଷଣା, main () ଫଙ୍କସନ, ଏବଂ ଆବଶ୍ୟକତା ଅନୁଯାୟୀ ଅନ୍ୟ ଫଙ୍କସନଗୁଡ଼ିକ।
- C ଭାଷାର ମୁଖ୍ୟ ପ୍ରୟୋଗ କ୍ଷେତ୍ରଗୁଡ଼ିକ ମଧ୍ୟରେ ସିଷ୍ଟମ୍ ପ୍ରୋଗ୍ରାମିଂ, ନେଟୱାର୍କ ପ୍ରୋଗ୍ରାମିଂ, GUI, ବୈଜ୍ଞାନିକ ଭିଜୁଆଲାଇଜେସନ୍, ଏମ୍ବେଡେଡ୍ ସିଷ୍ଟମ୍ ଇତ୍ୟାଦି ଅନ୍ତର୍ଭୁକ୍ତ।
- ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ନିର୍ମାଣରେ ବିଭିନ୍ନ ପଦକ୍ଷେପଗୁଡ଼ିକ ହେଉଛି - ଲେଖିବା, କମ୍ପାଇଲ୍, ଲିଙ୍କ୍, ଏବଂ ଚାଳନ କରିବା।
- ଟେକ୍ସଟ ଏଡିଟର ହେଉଛି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ଆମକୁ କମ୍ପ୍ୟୁଟର ମେମୋରିରେ ପ୍ରୋଗ୍ରାମ୍‌କୁ ଭର୍ତ୍ତି ଏବଂ ପରିବର୍ତ୍ତନ କରିବାରେ ସାହାଯ୍ୟ କରେ, ଏବଂ ଶେଷରେ ଏକ ଡିସ୍କ ଫାଇଲରେ ସଞ୍ଚୟ କରେ।
- ଉତ୍ସ କୋଡ୍ ହେଉଛି ଏକ ଉଚ୍ଚ-ସ୍ତରୀୟ ଭାଷାରେ ଲିଖିତ ପ୍ରୋଗ୍ରାମ୍ ଯେପରିକି C ଏବଂ ଏହା ଏକ ଡିସ୍କ ଫାଇଲରେ ଗଚ୍ଛିତ କରାହୁଏ ଯାହାକୁ ଉତ୍ସ ଫାଇଲ୍ କୁହାଯାଏ।
- କମ୍ପାଇଲର୍ ହେଉଛି ଏକ ପ୍ରୋଗ୍ରାମ ଯାହା C ପ୍ରୋଗ୍ରାମ୍‌କୁ ମେସିନ୍ ଭାଷାରେ ଅନୁବାଦ କରେ। ଏହାର ଦୁଇଟି ଅଂଶ ଅଛି - ପ୍ରିପ୍ରୋସେସର୍ ଏବଂ ଟ୍ରାନ୍ସଲେଟର। ପ୍ରିପ୍ରୋସେସର୍‌ଟି ପ୍ରୋଗ୍ରାମ୍‌ର ପ୍ରକ୍ରିୟା କରି ଏହାକୁ ଅନୁବାଦପାଇଁ ପ୍ରସ୍ତୁତ କରେ। ଟ୍ରାନ୍ସଲେଟର ଏହାର ଅନ୍ତିମ ଅନୁବାଦ କରେ।
- ଉତ୍ସ ପ୍ରୋଗ୍ରାମ୍‌କୁ ଅବଜେକ୍ଟ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଅନୁବାଦ କରିବାର ପ୍ରକ୍ରିୟାକୁ ସଂକଳନ କୁହାଯାଏ।
- ଅବଜେକ୍ଟ କୋଡ୍ ହେଉଛି ମେସିନ୍ ଭାଷାରେ କମ୍ପାଇଲର୍‌ଦ୍ୱାରା ପ୍ରସ୍ତୁତ ପ୍ରୋଗ୍ରାମ୍ ଏବଂ ଏହାକୁ ଅବଜେକ୍ଟ ଫାଇଲ୍ କୁହାଯାଏ ଯାହା ଏକ ଫାଇଲରେ ଗଚ୍ଛିତ ହୋଇଥାଏ।
- ଲିଙ୍କର୍ ହେଉଛି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଫଙ୍କସନଗୁଡ଼ିକର ଏବଂ ସିଷ୍ଟମ୍ ଲାଇବ୍ରେରିରୁ ଲାଇବ୍ରେରି ଫଙ୍କସନଗୁଡ଼ିକର ମେସିନ୍ ଭାଷାକୁ ଯୋଡିଥାଏ, ଏବଂ ଅନ୍ତିମ ଚାଳନଯୋଗ୍ୟ ପ୍ରୋଗ୍ରାମ୍ ପ୍ରସ୍ତୁତ କରିଥାଏ।
- ଚାଳନ କୋଡ୍ ହେଉଛି ମେସିନ୍ ଭାଷାରେ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ଲିଙ୍କର୍‌ଦ୍ୱାରା ପ୍ରସ୍ତୁତ ଏବଂ ଏହା ରନ୍ ଫାଇଲ୍ ଭାବରେ ଏକ ଫାଇଲ୍‌ରେ ଗଚ୍ଛିତ ହୋଇଥାଏ, ଯାହା ଚାଳନପାଇଁ ପ୍ରସ୍ତୁତ ଥାଏ।
- ଲୋଡର୍ ହେଉଛି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଯାହା ଏକ ଚାଳନ ପ୍ରୋଗ୍ରାମ୍‌କୁ ମାଧ୍ୟମିକ ଭଣ୍ଡାରଣରେ ଥିବା ଡିସ୍କ ଫାଇଲ୍‌ରୁ କମ୍ପ୍ୟୁଟର ମେମୋରିକୁ ସ୍ଥାନାନ୍ତର କରେ ଏବଂ ଏହାକୁ ଚାଳନପାଇଁ ପ୍ରସ୍ତୁତ କରେ।
- ପରୀକ୍ଷଣ ହେଉଛି ଏକ ପ୍ରକ୍ରିୟା ଯାହା ନିଶ୍ଚିତ କରେ ଯେ ସମସ୍ତ ସମ୍ଭାବ୍ୟ ଇନପୁଟ୍‌ଗୁଡ଼ିକପାଇଁ ପ୍ରୋଗ୍ରାମ୍‌ଟି ଠିକ୍ ଭାବରେ କାର୍ଯ୍ୟ କରିବ।
- ଡିବଗିଂ ହେଉଛି ତ୍ରୁଟିର ଚିହ୍ନଟ, ସ୍ଥାନ ନିରୂପଣ ଏବଂ ଏହାକୁ ଠିକ୍ କରିବାର ପ୍ରକ୍ରିୟା। ପ୍ରୋଗ୍ରାମ ବିକାଶ ପ୍ରକ୍ରିୟାରେ ଏହା ଏକ ସ୍ୱତନ୍ତ୍ର କାର୍ଯ୍ୟକଳାପ ନୁହେଁ; ବରଂ ସର୍ବଦା ଏହା ପରୀକ୍ଷଣ ସହ ଜଡ଼ିତ।
- ସିଧାସଳଖ କୋଡ୍ ହୋଇଥିବା ବା ଏକ ଅଭିଜ୍ଞାପକ ମାଧ୍ୟମରେ ପ୍ରତିନିଧିତ୍ୱ ହୋଇଥିବା ଏକ ମୂଲ୍ୟ ଯାହା

ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ସମୟରେ ପରିବର୍ତ୍ତନ ହୁଏ ନାହିଁ, ତାହାକୁ ଏକ ଧ୍ରୁବକ କୁହାଯାଏ, ଯେଉଁଠାରେ ମୂଲ୍ୟଗୁଡ଼ିକ ପରିବର୍ତ୍ତନ ହୋଇପାରେ ତାକୁ ଏକ ଚଳ କୁହାଯାଏ ।

- ଡାଟା ଟାଇପ୍ ହେଉଛି ମୂଲ୍ୟର ଏକ ସେଟ୍ ଏବଂ ଏହା ଉପରେ ଅନୁମୋଦିତ ଅପରେସନଗୁଡ଼ିକର ଏକ ସେଟ୍ ।
- C ଭାଷାଦ୍ୱାରା ସମର୍ଥିତ ବିଭିନ୍ନ ଡାଟା ଟାଇପ୍ ଗୁଡ଼ିକ ବିଲ୍ଟ-ଇନ୍ ଡାଟା ଟାଇପ୍, ବ୍ୟୁତ୍ପନ୍ନ ଡାଟା ଟାଇପ୍, ଏବଂ ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଡାଟା ଟାଇପ୍ ଭାବରେ ନାମିତ ହୋଇଛନ୍ତି ।
- ପ୍ରୋଗ୍ରାମ୍‌ଦ୍ୱାରା ପ୍ରକ୍ରିୟାକରଣ ହେବାକୁ ଥିବା ପ୍ରତ୍ୟେକ ଡାଟା ଆଇଟମ୍ ଘୋଷଣା କରାଯିବା ଆବଶ୍ୟକ ।
- ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ରେ ଇନପୁଟ୍/ଆଉଟପୁଟ୍ ଫର୍ମାଟେଡ୍ କିମ୍ବା ଅଣଫର୍ମାଟେଡ୍ ହୋଇପାରିବ ।

ଅନୁଶୀଳନୀ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. କମ୍ପ୍ୟୁଟର କ'ଣ?
2. ଏକ ବ୍ଲକ୍ ଚିତ୍ର ସାହାଯ୍ୟରେ କମ୍ପ୍ୟୁଟରର କାର୍ଯ୍ୟକ୍ଷମ ଅଂଶଗୁଡ଼ିକର କାର୍ଯ୍ୟକୁ ବ୍ୟାଖ୍ୟା କର ।
3. ଯେତେବେଳେ ଆମେ କହିଥାଉ ମେମୋରି ଉଦ୍‌ବାୟୀ (volatile), ଏହାର ଅର୍ଥ କ'ଣ?
4. ହାର୍ଡୱେୟାର ଏବଂ ସଫ୍ଟୱେୟାର ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଦର୍ଶାଅ ।
5. ଅପରେଟିଂ ସିଷ୍ଟମ୍ କଣ ?
6. ବୁଟିଂ କହିଲେ ତୁମେ କ'ଣ ବୁଝ?
7. ସଫ୍ଟୱେୟାରର ପ୍ରକାରଗୁଡ଼ିକ କ'ଣ?
8. ଏକ ଆଲଗୋରିଦମ୍ କ'ଣ ? ଏକ ଭଲ ଆଲଗୋରିଦମର ବାଞ୍ଛିତ ବୈଶିଷ୍ଟ୍ୟଗୁଡ଼ିକ କ'ଣ?
9. ଏକ ଫ୍ଲୋ'ଚାର୍ଟ କ'ଣ? ଫ୍ଲୋ'ଚାର୍ଟର ବିଭିନ୍ନ ପ୍ରତୀକଗୁଡ଼ିକ ବିଷୟରେ ଆଲୋଚନା କର ।
10. ଏକ ସ୍ୟୁଡୋକୋଡ୍ କ'ଣ?
11. ତୁମକୁ ଗୋଟିଏ ବିକଳ୍ପ ବାଛିବାକୁ ହେଲେ, ତୁମେ ଫ୍ଲୋ'ଚାର୍ଟ କିମ୍ବା ସ୍ୟୁଡୋକୋଡ୍ ମଧ୍ୟରୁ କେଉଁଟିକୁ ବ୍ୟବହାର କରି ଏକ ଆଲଗୋରିଦମ୍ ଉପସ୍ଥାପନ କରିବାକୁ ଚାହିଁବ ?
12. ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ର ସାଧାରଣ ସଂରଚନା ବର୍ଣ୍ଣନା କର ।
13. ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ର ବିକାଶରେ ବ୍ୟବହୃତ ବିଭିନ୍ନ ପଦକ୍ଷେପଗୁଡ଼ିକୁ ବର୍ଣ୍ଣନା କର ।
14. C ଭାଷାକୁ ମଧ୍ୟମସ୍ତରୀୟ ଭାଷା କାହିଁକି କୁହାଯାଏ?
15. ଏକ ଟୋକନ୍ କ'ଣ? C ର ବିଭିନ୍ନ ଟୋକନ୍‌ଗୁଡ଼ିକ ବର୍ଣ୍ଣନା କର ।
16. ଏକ ଡାଟା ଟାଇପ୍ କ'ଣ? C ଭାଷାଦ୍ୱାରା ସମର୍ଥିତ ବିଭିନ୍ନ ଡାଟା ଟାଇପ୍‌ଗୁଡ଼ିକର ନାମ ଲେଖ ।
17. ଧ୍ରୁବକ ଏବଂ ଚଳ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଦର୍ଶାଅ ।
18. ଚଳର ଘୋଷଣା ଏବଂ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରିବାପାଇଁ ସିଣ୍ଟାକ୍ସ ଲେଖ ।
19. C ରେ ଇନପୁଟ୍ /ଆଉଟପୁଟ୍ ପରିଚାଳନାପାଇଁ ବିଭିନ୍ନ ଫଙ୍କ୍‌ସନ୍‌ର ନାମ ଲେଖ ।

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. ପ୍ରଦତ୍ତ ପ୍ଲାଟଫର୍ମରେ କୌଣସି ଏକ ଭାଷାରେ ଲେଖାଯାଇଥିବା ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍‌କୁ କୌଣସି ପରିବର୍ତ୍ତନ ବିନା କିମ୍ବା ସର୍ବନିମ୍ନ ପରିବର୍ତ୍ତନ ସହିତ ଅନ୍ୟ ପ୍ଲାଟଫର୍ମକୁ ପରିବହନ ହୋଇପାରିବ, ତେବେ ଏହାକୁ _______ କୁହାଯିବ ।

   (a) ବୁଝିବା ଯୋଗ୍ୟ (b) ପଠନଯୋଗ୍ୟ

   (c) ପୋର୍ଟେବଲ୍ (d) ରକ୍ଷଣାବେକ୍ଷଣଯୋଗ୍ୟ

2. C ଭାଷାକୁ କିଏ ବିକଶିତ କରିଛି?

   (a) ଭନ୍ ନ୍ୟୁମ୍ୟାନ୍ (b) ଡେନିସ୍ ରିଚି

   (c) ପିଟର ନzର୍ଟନ୍ (d) କେନ୍ ଥମ୍ପସନ୍

3. ନିମ୍ନରେ ପ୍ରଦତ୍ତ କେଉଁ ବର୍ଣ୍ଣଟି ଏକ C ପ୍ରୋଗ୍ରାମ୍‌ରେ ନିର୍ଦ୍ଦେଶର ସମାପ୍ତିପାଇଁ ବ୍ୟବହୃତ ହୁଏ?

   (a) , (କମା) (b) : (କୋଲୋନ୍)

   (c) ; (ସେମିକୋଲୋନ୍) (d) . (ଅବଧି)

4. ନିମ୍ନରେ ପ୍ରଦତ୍ତ କେଉଁଟି ଏକ ଅଭିଜ୍ଞାପକର ପ୍ରଥମ ବର୍ଣ୍ଣ ହୋଇପାରିବ ନାହିଁ?

   (a) ଅଙ୍କ (b) ଅଣ୍ଡରସ୍କୋର୍

   (c) ବର୍ଣ୍ଣମାଳା (d) ବଡ଼ ଅକ୍ଷର

5. ନିମ୍ନରେ ପ୍ରଦତ୍ତ କେଉଁଟି ଏକ ଫଙ୍କ୍‌ସନ୍‌ର କାୟାକୁ ଆବଦ୍ଧ କରିବାପାଇଁ ବ୍ୟବହୃତ ହୁଏ?

   (a) [ ] (b) { }

   (c) ( ) (d) < >

6. ନିମ୍ନରେ ପ୍ରଦତ୍ତ କେଉଁ ପ୍ଲାଟଫର୍ମରେ C ଭାଷା ବ୍ୟବହାର କରାଯାଇପାରିବ?

   (a) ୟୁନିକ୍ସ (b) ୱିଣ୍ଡୋଜ୍

   (c) ଲିନକ୍ସ (d) ଉପରୋକ୍ତ ସମସ୍ତ

7. C ଭାଷା ହେଉଛି ଏକ ____ ସ୍ତରର ଭାଷା

   (a) ଉଚ୍ଚ (b) ନିମ୍ନ

   (c) ମଧ୍ୟମ (d) ସାଙ୍କେତିକ

8. ଆକ୍ରୋନିମ୍ ANSI (ଆନ୍‌ସି) ର ଅର୍ଥ ହେଉଛି ____

   (a) American National Standards International

   (b) American National Software Incorporation

   (c) American National Standards Institute

   (d)American National Standards Instructions

9. ଅଲଗା ପଦଟି ବାହାର କର

   (a) ଉତ୍ସ କୋଡ୍ (b) ଅବଜେକ୍ଟ କୋଡ୍

   (c) ଚାଳନ କୋଡ୍ (d) ଆସ୍କି କୋଡ୍

10. ପ୍ରୋଗ୍ରାମ୍‌ର ପରୀକ୍ଷଣ କରାଯାଏ ଏହା ସୁନିଶ୍ଚିତ କରିବାପାଇଁ ଯେ

    (a) ଏଥିରେ କୌଣସି ସିଣ୍ଟାକ୍‌ସ ତ୍ରୁଟି ରହିବା ଉଚିତ୍ ନୁହେଁ

    (b) ଏହା ଏହାର ଉଦ୍ଦିଷ୍ଟ କାର୍ଯ୍ୟ ସମ୍ପାଦନ କରେ

    (c) ଏହା ସଫଳତାର ସହ ସଂକଳନ କରେ

    (d) ଏହା ଅସୀମ ଭାବରେ ଚାଲିବା ଉଚିତ୍ ନୁହେଁ

11. ପଦ ବଗ୍ _____ କୁ ବୁଝାଏ

    (a) ସିଣ୍ଟାକ୍‌ସ ତ୍ରୁଟି (b) ତାର୍କିକ ତ୍ରୁଟି

    (c) ରନ୍ ଟାଇମ୍ ତ୍ରୁଟି (d) ଉପରୋକ୍ତ ସମସ୍ତ

12. ଆକ୍ରୋନିମ୍ ASCII (ଆସ୍କି) ର ଅର୍ଥ ହେଉଛି ____

    (a) American Standard Code for International Information

    (b) American System for Compiler Information International

    (c) American System for Code Information International

    (d) American Standard Code for Information Interchange

13. ନିମ୍ନରେ ପ୍ରଦତ୍ତ କେଉଁ ଉକ୍ତିଟି C ଭାଷା ବିଷୟରେ ସତ୍ୟ ନୁହେଁ?

    (a) ପ୍ରତ୍ୟେକ ନିର୍ଦ୍ଦେଶ ସେମିକୋଲୋନ୍‌ଦ୍ୱାରା ସମାପ୍ତ ହୁଏ

    (b) ଏହା କେସ୍ ସମ୍ବେଦନହୀନ

    (c) ପ୍ରୋଗ୍ରାମ୍ କୋଡ୍‌ର ଯେକୌଣସି ସ୍ଥାନରେ ଟିପ୍ପଣୀଗୁଡ଼ିକ ରଖାଯାଇପାରିବ

    (d) ଏଥିରେ ଉଭୟ ଉଚ୍ଚସ୍ତରୀୟ ଏବଂ ନିମ୍ନସ୍ତରୀୟ ଭାଷାର ବୈଶିଷ୍ଟ୍ୟ ରହିଛି

14. ପଦ 'ଡିବଗିଂ' __________ ପ୍ରକ୍ରିୟାକୁ ବୁଝାଏ

    (a) ଅବଜେକ୍ଟ କୋଡ୍‌କୁ ଉତ୍ସ କୋଡ୍‌ରେ ଅନୁବାଦ କରିବା।

    (b) ସିଷ୍ଟମ୍ ଲାଇବ୍ରେରିଗୁଡ଼ିକ ସହ ଅବଜେକ୍ଟ କୋଡ୍ ଲିଙ୍କ୍ କରିବା।

    (c) ବଗ୍‌ଗୁଡ଼ିକ ଚିହ୍ନଟ କରିବା ଏବଂ ସେଗୁଡ଼ିକ ଠିକ୍ କରିବା।

    (d) ଉପରୋକ୍ତ କୌଣସିଟି ନୁହେଁ।

15. ଶବ୍ଦ 'ପୋର୍ଟେବିଲିଟି'ର ଅର୍ଥ ହେଉଛି _______

    (a) ପ୍ରୋଗ୍ରାମ୍ କୋଡ୍ ବୁଝିବା ଯୋଗ୍ୟ

    (b) ପ୍ରୋଗ୍ରାମ୍ କୋଡ୍ ରକ୍ଷଣାବେକ୍ଷଣଯୋଗ୍ୟ

(c) ପ୍ରୋଗ୍ରାମ୍ କୋଡ୍ ବଗ୍ ମୁକ୍ତ ଅଟେ

(d) ଅନ୍ୟ କମ୍ପ୍ୟୁଟରରେ ସର୍ବନିମ୍ନ ପରିବର୍ତ୍ତନ ବିନା/ସହ ପ୍ରୋଗ୍ରାମ୍ କୋଡ୍‌କୁ ପରିବହନ କରାଯାଇପାରିବ

16. C ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକ_____ ବ୍ୟବହାର କରି ମେସିନ୍ ଭାଷାରେ ରୂପାନ୍ତରିତ ହୁଏ ।

(a) ଆସେମ୍ବଲର୍ (b) ଇଣ୍ଟର୍‌ପ୍ରିଟର (c) କମ୍ପାଇଲର୍ (d) ଅପରେଟିଂ ସିଷ୍ଟମ୍

17. ଟର୍ବୋ C/C++ କମ୍ପାଇଲର୍‌ଦ୍ୱାରା ଉତ୍ପାଦିତ ମେସିନ୍ କୋଡ୍ ଲେଖାଥିବା ଫାଇଲ୍‌ର ସମ୍ପ୍ରସାରଣ ହେଉଛି _______

(a) *.com (b) *.exe (c) *.c (d) *.obj

18. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଏକ କୀୱାର୍ଡ ନୁହେଁ?

(a) for (b) main (c) break (d) else

19. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି C ଭାଷାର ଏକ ବୈଧ ଡାଟା ଟାଇପ ନୁହେଁ?

(a) char (b) float (c) long (d) double

20. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି C ରେ ଏକ ମୌଳିକ ଡାଟା ମୂଲ୍ୟକୁ ସୂଚିତ କରେ ନାହିଁ?

(a) "a" (b) 'k' (c) 35.25 (d) 14

ଉତ୍ତର									
1.	**(c)**	**2.**	**(b)**	**3.**	**(c)**	**4.**	**(a)**	**5.**	**(b)**
6.	**(d)**	**7.**	**(c)**	**8.**	**(c)**	**9.**	**(d)**	**10.**	**(b)**
11.	**(d)**	**12.**	**(d)**	**13.**	**(b)**	**14.**	**(a)**	**15.**	**(a)**
16.	**(c)**	**17.**	**(d)**	**18.**	**(b)**	**19.**	**(a)**	**20.**	**(a)**

ଗଣନାମୂଳକ ସମସ୍ୟା (Computational Problems)

ଆଲଗୋରିଦମ୍‌ର ବିକାଶ କର ଏବଂ ଫ୍ଲୋ'ଚାର୍ଟ ଏବଂ/କିମ୍ବା ସୁଡୋକୋଡ୍ ବ୍ୟବହାର କରି ଏହାକୁ ଉପସ୍ଥାପନ କର:

1. ଦିଆଯାଇଥିବା ବର୍ଷଟି ଅଧିବର୍ଷ କି ନୁହେଁ ତାହା ପରୀକ୍ଷା କରିବାପାଇଁ।
2. ଦିଆଯାଇଥିବା ତାରିଖ ବୈଧ କି ନୁହେଁ ପରୀକ୍ଷା କରିବାକୁ।
3. ଏକ ଗଣନ ସଂଖ୍ୟା 'n' ରେ ସ୍ଥିତ ବୃହତ୍ତମ ଅଙ୍କ ଖୋଜିବାକୁ।
4. ଏକ ଗଣନ ସଂଖ୍ୟା 'n' ରେ ସ୍ଥିତ କ୍ଷୁଦ୍ରତମ ଅଙ୍କ ଖୋଜିବାକୁ।
5. n ତମ ମୌଳିକ ସଂଖ୍ୟା ଖୋଜିବାପାଇଁ।
6. 'a ', ' b ' ଏବଂ 'c ' ସେମି ଦୈର୍ଘ୍ୟ ବିଶିଷ୍ଟ ତିନୋଟି ରେଖା ସହିତ ଏକ ତ୍ରିଭୂଜ ଗଠନ କରାଯାଇପାରିବ କି ନାହିଁ ତାହା ପରୀକ୍ଷା କରିବାକୁ।
7. 'a ', ' b ' ଏବଂ 'c ' ଡିଗ୍ରୀରେ ତିନୋଟି କୋଣ ସହିତ ଏକ ତ୍ରିଭୂଜ ଗଠନ କରାଯାଇପାରିବ କି ନାହିଁ ତାହା ପରୀକ୍ଷା କରିବାକୁ।
8. ପ୍ରଥମ ରେଖାର ଦୁଇଟି ପଏଣ୍ଟ A (xa, ya) ଏବଂ B (xb, yb) ଏବଂ ଦ୍ୱିତୀୟ ରେଖାର C (xc, yx) ଏବଂ D (xd, yd) ଦିଆଯାଇଛି। ପରୀକ୍ଷା କରିବା ଯେ ରେଖା ଦୁଇଟି ପରସ୍ପରକୁ ଛେଦ କରନ୍ତି କି ନାହିଁ।
9. ଜଣେ ବ୍ୟକ୍ତିଙ୍କ ବଡି ମାସ୍ ଇଣ୍ଡେକ୍ସ (BMI) ଗଣନା ଏହିପରି କରାଯାଏ, ଯେଉଁଠାରେ ଓଜନ W କିଲୋଗ୍ରାମରେ ଏବଂ ଉଚ୍ଚତା H ମିଟରରେ ଅଛି।

ବିଏମ୍‌ଆଇ	ବର୍ଗୀକରଣ
< 18.5	ଓଜନ ତଳେ
18.5–24.9	ସାଧାରଣ ଓଜନ
25.0–29.9	ଅଧିକ ଓଜନ
30.0–34.9	ପ୍ରଥମ ଶ୍ରେଣୀ ମେଦବହୁଳତା
35.0–39.9	ଦ୍ୱିତୀୟ ଶ୍ରେଣୀମେଦବହୁଳତା

ଦିଆଯାଇଥିବା ଓଜନ ଏବଂ ଉଚ୍ଚତା ମୂଲ୍ୟରୁ ବ୍ୟକ୍ତିଙ୍କ ମେଦବହୁଳତାକୁ ଉପରୋକ୍ତ ତାଲିକାନୁଯାୟୀ ଶ୍ରେଣୀଭୁକ୍ତ କରିବାକୁ।

10. ମାସିକ ଟେଲିଫୋନ୍ ବିଲ୍ ନିମ୍ନମତେ ଗଣନା କରାଯିବ:

    100 ଟି କଲ୍‌ପାଇଁ ସର୍ବନିମ୍ନ 200 ଟଙ୍କା

    ଏଥିସହ ପରବର୍ତ୍ତୀ 50 ଟି କଲ୍‌ପାଇଁ କଲ୍ ପିଛା 0.60 ଟଙ୍କା

    ଏଥିସହ ପରବର୍ତ୍ତୀ 50 ଟି କଲ୍‌ପାଇଁ କଲ୍ ପିଛା 0.50 ଟଙ୍କା

    ଏଥିସହ 200 କଲ୍ ବାହାରେ ଯେକୌଣସି କଲ୍‌ପାଇଁ କଲ୍ ପିଛା 0.40 ଟଙ୍କା।

ଦିଆଯାଇଥିବା କଲ୍ ସଂଖ୍ୟାପାଇଁ ମାସିକ ବିଲ୍ ଗଣନା କରିବାକୁ।

ପ୍ରାକ୍ଟିକାଲ୍ (Practicals)

1. **ପିସି/ଲାପଟପ୍‌ର ଅପରେସନ୍ (ସଞ୍ଚାଳନ) ସହ ପରିଚିତି:** ବୁଟିଂ (booting), କନଫିଗିଓରିଙ୍ଗ୍ ଡେସ୍କଟପ୍ (configuring desktop), ଫାଇଲ୍ (file) ଏବଂ ଫୋଲ୍ଡର୍ (folder) ସହିତ କାର୍ଯ୍ୟ କରିବା, ସଫ୍ଟୱେୟାର ସଂସ୍ଥାପନା (install) କରିବା ଏବଂ ବିଭିନ୍ନ ପ୍ରୟୋଗଗୁଡ଼ିକର (applications) ଚାଳନ କରିବାର ଧାରଣା।
2. **ପ୍ରୋଗ୍ରାମିଂ ପରିବେଶ ସହିତ ପରିଚିତି:** ଏକ C ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବା, ସଂକଳନ କରିବା, ଏବଂ ଚାଳନ କରିବା।

ଅଧିକ ଜାଣିବାପାଇଁ

ସମସ୍ୟା ସମାଧାନ ହେଉଛି ଏକ କୌଶଳ, ଏବଂ କୌଶଳ ରାତାରାତି ଶିଖିହୁଏ ନାହିଁ। ତେଣୁ, ଶିକ୍ଷାର୍ଥୀମାନଙ୍କ ମଧ୍ୟରେ ଆବଶ୍ୟକ ଦକ୍ଷତା ବିକାଶପାଇଁ, ଶିକ୍ଷକମାନେ ଉପଯୁକ୍ତ ଉଦାହରଣ ନେଇ ସମାଧାନ ବିକାଶରେ ଶିକ୍ଷାର୍ଥୀମାନଙ୍କୁ ଜଡ଼ିତ କରି ସମସ୍ୟା ସମାଧାନ ପଦ୍ଧତିକୁ ପ୍ରଦର୍ଶନ କରିବା ଉଚିତ।

ଶେଷରେ, ଶିକ୍ଷକ ପର୍ଯ୍ୟାପ୍ତ ସମ୍ବନ୍ଧିତ ସମସ୍ୟା ଦେବା ଉଚିତ ଯାହାଦ୍ୱାରା ବିଦ୍ୟାର୍ଥୀମାନେ ପୂର୍ବରୁ ଦେଖି ନ ଥିବା କିମ୍ବା ସମାଧାନ କରି ନ ଥିବା ସମସ୍ୟାର ସମାଧାନପାଇଁ ଶିଖିଥିବା ଧାରଣାଗୁଡ଼ିକୁ ପ୍ରୟୋଗ କରିପାରିବେ।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R.G. Dromey, How to solve it by Computer, Pearson Education.
2. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co (P) Ltd., New Delhi.
3. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
4. V. Anton Spraul, Think Like a Programmer, No Starch Press.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

2 ଗାଣିତିକ ପରିପ୍ରକାଶ ଏବଂ ଅପରେଟର୍ ପ୍ରାଥମିକତା

ଏକକ ବିଶେଷତ୍ୱ

ଏହି ୟୁନିଟରେ ବିଭିନ୍ନ ପ୍ରକାରର ଅପରେଟରଗୁଡ଼ିକ ସହ ଜଡିତ ବିଷୟ, ସେଗୁଡ଼ିକର ପ୍ରାଥମିକତା ଏବଂ ସହଯୋଗିତା, ବିଭିନ୍ନ ପ୍ରକାରର ପରିପ୍ରକାଶ, ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନକୁ ନିୟନ୍ତ୍ରଣ କରୁଥିବା ନିୟମ ଏବଂ ଲାଇବ୍ରେରି (library) ପ୍ରକାର୍ଯ୍ୟ ବିଷୟରେ ଆଲୋଚନା କରାଯାଇଛି ।

ଭୂମିକା

ବିଜ୍ଞାନ ଏବଂ ଇଞ୍ଜିନିୟରିଙ୍ଗ୍ ସମ୍ବନ୍ଧୀୟ ସମସ୍ୟାରେ, ପ୍ରୋଗ୍ରାମରଙ୍କୁ ବିଭିନ୍ନ ପ୍ରକାରର ବୀଜଗାଣିତିକ ପରିପ୍ରକାଶର ସମ୍ମୁଖୀନ ହେବାକୁ ପଡ଼େ । ଏଥିରେ ବିଭିନ୍ନ ଗାଣିତିକ ଅପରେସନ୍ ଏବଂ ମାନକ ଗାଣିତିକ ଏବଂ ତ୍ରିକୋଣମିତିକ ଫଙ୍କସନ୍ ଅନ୍ତର୍ଭୁକ୍ତ ହୋଇପାରେ । ସେହି ବୀଜଗାଣିତିକ ପରିପ୍ରକାଶକୁ C ରେ ସେମାନଙ୍କର ସମକକ୍ଷ ପରିପ୍ରକାଶରେ ପରିଣତ କରିବାକୁ, ଏବଂ ସେଗୁଡ଼ିକର ମୂଲ୍ୟ ନିର୍ଦ୍ଧାରଣକ୍ରମକୁ ନିୟନ୍ତ୍ରଣ କରିବା ପାଇଁ, ପରିପ୍ରକାଶଗୁଡ଼ିକ କିପରି ମୂଲ୍ୟାଙ୍କନ କରାଯାଏ, ଏବଂ ଅନ୍ୟାନ୍ୟ ସମ୍ବନ୍ଧୀୟ ସମସ୍ୟାଗୁଡ଼ିକ ପାଇଁ, ପ୍ରୋଗ୍ରାମର୍ ନିକଟରେ C ଦ୍ୱାରା ସମର୍ଥିତ ବିଭିନ୍ନ ଅପରେଟରଗୁଡ଼ିକ ବିଷୟରେ ଜ୍ଞାନ ଥିବା ଦରକାର ।

ଏହି ଏକକ ବିଦ୍ୟାର୍ଥୀସଂଖ୍ୟାକୁ ଅପରେଟର, ପରିପ୍ରକାଶ ଏବଂ ସେମାନଙ୍କର ମୂଲ୍ୟାଙ୍କନ ସହିତ ଜଡିତ ବିଭିନ୍ନ ଦିଗଗୁଡ଼ିକୁ ବୁଝିବାରେ ସାହାଯ୍ୟ କରିବ ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- ରୈଖିକ ବୀଜଗଣିତ
- ମାନକ ଗାଣିତିକ ଫଙ୍କସନ୍‌ସମୂହ

ଏକକଲବ୍ଧ ଜ୍ଞାନ

ଏକକ ସମାପ୍ତି ପରେ, ବିଦ୍ୟାର୍ଥୀମାନେ ନିମ୍ନ ବିଷୟଗୁଡ଼ିକରେ ସମର୍ଥ ହେବେ

U2-O1: ବିଭିନ୍ନ ଅପରେଟର୍ ଏବଂ ସେମାନଙ୍କ ବ୍ୟବହାର ବ୍ୟାଖ୍ୟା କରିବାରେ

U2-O2: ବୀଜଗାଣିତିକ ପରିପ୍ରକାଶକୁ C ର ପରିପ୍ରକାଶରେ ପରିଣତ କରିବାରେ

U2-O3: ପରିପ୍ରକାଶଗୁଡିକର ମୂଲ୍ୟାଙ୍କନ କରିବାର ଉପାୟ ବ୍ୟାଖ୍ୟା କରିବାରେ

U2-O4: ଅପରେଟରମାନଙ୍କ ମଧ୍ୟରେ ପ୍ରାଥମିକତା ଏବଂ ସହଯୋଗିତାର ଧାରଣା ବ୍ୟାଖ୍ୟା କରିବାରେ

U2-O5: ଆସାଇନମେଣ୍ଟ ଓ ପରିପ୍ରକାଶରେ ଡାଟା ଟାଇପ୍‌ର ରୂପାନ୍ତରଣ ବ୍ୟାଖ୍ୟା କରିବାରେ

ୟୁନିଟ୍ -2 ୟୁନିଟ ଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ମ୍ୟାପିଂ (1 – ଦୁର୍ବଳ ସହବନ୍ଧନ; 2 – ମଧ୍ୟମ ସହବନ୍ଧନ; 3 – ଦୃଢ ସହବନ୍ଧନ)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U2-01	1	–	–	–	–	–	–	–
U2-02	–	1	–	–	–	–	–	–
U2-03	–	–	2	–	–	–	–	–
U2-04	–	–	2	–	–	–	–	–
U2-05	–	–	1	–	–	–	–	–

2.1 ଉପକ୍ରମ

ଅପରେଟର ହେଉଛି ଭାଷାର ଏକ କ୍ରିୟା । ଏହା ଜଣେ ବ୍ୟବହାରକାରୀକୁ ମୂଲ୍ୟ ଗଣନା କରିବାକୁ ଦେଇଥାଏ । ତୁମେ ଅପରେଟର୍‌ଗୁଡ଼ିକୁ କ୍ରିୟା ଏବଂ ଅପେରାଣ୍ଡ୍‌କୁ ସେହି କ୍ରିୟାଗୁଡ଼ିକର କର୍ତ୍ତା ଏବଂ କର୍ମପଦ ଭାବରେ ଭାବିପାରିବ । C ଭାଷାରେ ଥିବା ଅପରେଟର୍‌ଗୁଡ଼ିକର ସମୃଦ୍ଧ ସେଟ୍ ହେଉଛି ଏହାର ଏକ ବିଶିଷ୍ଟ ବିଶେଷତ୍ୱ ।

ଏକ ପରିପ୍ରକାଶ ହେଉଛି ମୂଲ୍ୟ ଗଣନାପାଇଁ ଏକତ୍ର ଯୋଡ଼ାଥିବା ଅପେରାଣ୍ଡ୍ ଏବଂ ଅପରେଟରଗୁଡ଼ିକୁ ନେଇ ଗଠିତ ଏକ ସୂତ୍ର । ଭବିଷ୍ୟତ ବ୍ୟବହାରପାଇଁ ଗଣନା ହେଉଥିବା ମୂଲ୍ୟକୁ ଏକ ଚଳରେ ରଖାଯିବା ଉଚିତ । ଏହା ଏକ ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତି ବ୍ୟବହାର କରି କରାଯାଏ ।

ଏକ କମ୍ପ୍ୟୁଟର୍ ପ୍ରୋଗ୍ରାମ୍‌ରେ, ହାତରେ ଥିବା ସମସ୍ୟାର ଆବଶ୍ୟକତା ଅନୁଯାୟୀ ବିଭିନ୍ନ କମ୍ପ୍ୟୁଟେସନ୍ କରାଯାଇପାରେ । ତଥାପି, କିଛି ନିତ୍ୟନୈମିତ୍ତିକ ଗଣନା ଅଛି ଯାହା ପ୍ରାୟ ପ୍ରତ୍ୟେକ ପ୍ରୋଗ୍ରାମ୍‌ର ଅଂଶ ହୋଇପାରେ । ତେଣୁ ଉପଯୋଗିତା ଦୃଷ୍ଟିରୁ, ପ୍ରତ୍ୟେକ ଆଧୁନିକ ପ୍ରୋଗ୍ରାମିଂ ଭାଷା ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍ ଆକାରରେ ଏହି ନିତ୍ୟନୈମିତ୍ତିକ ଗଣନାପାଇଁ ଏକ ସୁବିଧା ପ୍ରଦାନ କରେ ।

ଏହି ୟୁନିଟ୍‌ରେ C ଭାଷାର ଏହି ସେମାଣ୍ଟିକ୍ ୟୁନିଟ୍‌ଗୁଡ଼ିକୁ ବର୍ଣ୍ଣନା କରିବାକୁ ଚେଷ୍ଟା କରାଯାଇଛି ।

2.2 ଅପରେଟର (operators)

ଅପରେଟର୍‌ଗୁଡ଼ିକ ଯେକୌଣସି ପ୍ରୋଗ୍ରାମିଂ ଭାଷାର ହେଉଛି ମୂଳଦୁଆ। ଯେକୌଣସି ପ୍ରୋଗ୍ରାମିଂ ଭାଷାର କାର୍ଯ୍ୟକାରିତା ଅପରେଟର୍‌ର ବ୍ୟବହାର ବିନା ଅସମ୍ପୂର୍ଣ୍ଣ ଅଟେ ।

ଅପରେଟର୍‌ଗୁଡ଼ିକ, ସାଧାରଣତଃ, ଦୁଇ ପ୍ରକାରର:

- **ୟୁନାରି (Unary) ଅପରେଟର୍ –** ଏହି ଅପରେଟର୍ ଗୋଟିଏ ମାତ୍ର ଅପେରାଣ୍ଡ ଉପରେ କାର୍ଯ୍ୟ କରେ । ୟୁନାରି ଅପରେଟର୍‌ଗୁଡ଼ିକ ସେମାନଙ୍କର ଅପେରାଣ୍ଡ ସହିତ ଉପସର୍ଗ ହୋଇଥାଏ (ପୂର୍ବରୁ ରହିଥାନ୍ତି) ।

  ଉଦାହରଣ ସ୍ୱରୂପ, (-x), ଏଠାରେ ୟୁନାରି ବିୟୁକ୍ତ '-' ଅପରେଟର୍ x ର ପୂର୍ବରୁ ରହିଛି (x ଏକ ସଂଖ୍ୟାତ୍ମକ ମୂଲ୍ୟ ଧାରଣ କରିବା ଦରକାର), ଏବଂ ଏହାର ମୂଲ୍ୟକୁ ବିୟୁକ୍ତାତ୍ମକ କରେ, ଅର୍ଥାତ, ଏହାର ସଙ୍କେତ ପରିବର୍ତ୍ତନ କରେ ।

- **ବାଇନାରି (Binary) ଅପରେଟର୍ –** ଏହି ଅପରେଟର୍ ଦୁଇଟି ଅପେରାଣ୍ଡ ସହିତ କାର୍ଯ୍ୟ କରେ । ବାଇନାରି ଅପରେଟର୍‌ଗୁଡ଼ିକ ସେଗୁଡ଼ିର ଅପେରାଣ୍ଡର ମଧ୍ୟବର୍ତ୍ତୀ ସ୍ଥାନରେ ଅବସ୍ଥାନ କରିଥାନ୍ତି ।

  ଉଦାହରଣ ସ୍ୱରୂପ, a+b, ଏଠାରେ ବାଇନାରି ଯୁକ୍ତ '+' ଅପରେଟର୍ ଏହାର ଅପେରାଣ୍ଡ a ଏବଂ b ମଧ୍ୟରେ ଅବସ୍ଥାନ କରିଛି (ଉଭୟ a ଓ b ସଂଖ୍ୟାତ୍ମକ ମୂଲ୍ୟ ଧାରଣ କରିବା ଉଚିତ), ଏବଂ ଚଳ b ର ମୂଲ୍ୟକୁ a ସହିତ ଯୋଗ କରିଥାଏ ।

C ଭାଷାର ଅପରେଟର୍‌ଗୁଡ଼ିକର ସମୃଦ୍ଧ ସେଟ୍ ହେଉଛି ଏହାର ଏକ ବିଶିଷ୍ଟ ବୈଶିଷ୍ଟ୍ୟ । ଏହି ଅପରେଟର୍‌ଗୁଡିକୁ ନିମ୍ନଲିଖିତ ବର୍ଗଗୁଡିକରେ ଗଣା ଯାଇପାରିବ:

- ଗାଣିତିକ ଅପରେଟର୍ (+, –, *, /, %)
- ସମ୍ବନ୍ଧୀୟ ଅପରେଟର୍ (<, <=, >, >=, =, ==)
- ଲଜିକାଲ୍ ଅପରେଟର୍ (!, & &, | |)
- ବିଟ୍-ୱାଇଜ୍ (bit-wise) ଅପରେଟର୍ (~, >>, <<, ଏବଂ | , ^)
- ବିଶେଷ (Special) ଅପରେଟର୍
  - » ବୃଦ୍ଧି ଏବଂ ହ୍ରାସ (Increment & Decrement) ଅପରେଟର୍ (++ , - -)
  - » ସାଇଜ-ଅଫ୍ (sizeof) ଅପରେଟର୍
  - » ଠିକଣା ଅପରେଟର୍ (&)
  - » ଇନ୍‌ଡାଇରେକ୍‌ସନ୍ (indirection)/ଡି-ରେଫରେନ୍ସ(de-reference) ଅପରେଟର (*)
  - » ଟର୍‌ନାରୀ (Ternary)/ସର୍ତ୍ତମୂଳକ ଅପରେଟର (? :)

ଏହି ୟୁନିଟ୍‌ରେ, ଆମେ ମୁଖ୍ୟତଃ ଗାଣିତିକ ଗଣନାରେ ବ୍ୟବହୃତ ଅପରେଟର୍‌ଗୁଡ଼ିକ ଉପରେ ଧ୍ୟାନ ଦେବୁ, କେବଳ ଆଲୋଚନାର ସମ୍ପୂର୍ଣ୍ଣତା ପାଇଁ ଅନ୍ୟ ଅପରେଟର୍‌ଗୁଡ଼ିକର ଉପସ୍ଥାପନ କରାଯିବ ।

2.2.1 ଗାଣିତିକ ଅପରେଟର୍ (Arithmetic Operators)

ଗାଣିତିକ ଅପରେଟର୍‌ଗୁଡ଼ିକ ଯୋଗ, ବିଯୋଗ, ଗୁଣନ ଇତ୍ୟାଦି ଗାଣିତିକ କାର୍ଯ୍ୟ କରିବା ପାଇଁ ବ୍ୟବହୃତ ହୁଅନ୍ତି ।

ଟେବୁଲ୍ 2.1: ବାଇନାରି ଗାଣିତିକ ଅପରେଟର୍

ଅପରେଟର	ପ୍ରତୀକ	ଫର୍ମ (form)	ଅପରେସନ୍
ଯୋଗ	+	$x + y$	X ମୂଲ୍ୟକୁ Y ମୂଲ୍ୟ ସହ ଯୋଗ କରେ
ବିୟୋଗ	–	$x - y$	X ମୂଲ୍ୟରୁ Y ମୂଲ୍ୟ ବିୟୋଗ କରେ
ଗୁଣନ	*	$x * y$	X ମୂଲ୍ୟକୁ Y ମୂଲ୍ୟଦ୍ୱାରା ଗୁଣନ କରେ
ବିଭାଜନ	/	x/y	X ମୂଲ୍ୟକୁ Y ମୂଲ୍ୟଦ୍ୱାରା ବିଭାଜିତ କରେ
ମୋଡୁଲସ୍ (Modulus)	%	$x \% y$	X ମୂଲ୍ୟକୁ Y ମୂଲ୍ୟଦ୍ୱାରା ଭାଗ କରି ଭାଗଶେଷ ମୂଲ୍ୟ ଦେଇଥାଏ ।

ଇଣ୍ଟିଜର/ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ଗଣିତ (Integer Arithmetic)

ଯେତେବେଳେ ଉଭୟ ଅପେରାଣ୍ଡ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ହୋଇଥାଏ ସେତେବେଳେ ଅପରେସନ୍‌କୁ ଇଣ୍ଟିଜର ଗଣିତ କୁହାଯାଇଥାଏ ଏବଂ ସର୍ବଦା ଏହା ଏକ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ମୂଲ୍ୟ ପ୍ରଦାନ କରିଥାଏ ।

ନିମ୍ନଲିଖିତ ଉକ୍ତିଗୁଡିକ ବିଚାର କର

```
int x = 13, y = 5;
```

ତା'ପରେ ଆମର ନିମ୍ନଲିଖିତ ଫଳାଫଳ ଅଛି:

```
x – y = 8
x + y = 18
x * y = 65
x / y = 2 (ଦଶମିକ ଅଂଶ କଟା ଯାଇଅଛି )
x % y = 3 (ବିଭାଜନ ର ଭାଗଶେଷ)
```

ବାସ୍ତବ ସଂଖ୍ୟା ଗଣିତ (Real Arithmetic)

ଯେତେବେଳେ ଉଭୟ ଅପେରାଣ୍ଡ ବାସ୍ତବ ସଂଖ୍ୟା ହୋଇଥାଏ ସେତେବେଳେ, ଅପରେସନ୍‌କୁ ବାସ୍ତବ ଗଣିତ କୁହାଯାଇଥାଏ ଏବଂ ଏହା ସର୍ବଦା ଏକ ବାସ୍ତବ ମୂଲ୍ୟ ପ୍ରଦାନ କରିଥାଏ । ଯେହେତୁ ବାସ୍ତବ ସଂଖ୍ୟା (ଫ୍ଲୋଟିଂ-ପଏଣ୍ଟ ମୂଲ୍ୟ) ଅନୁମୋଦିତ ସଂଖ୍ୟକ ଅର୍ଥପୂର୍ଣ୍ଣ ଅଙ୍କର ନିକଟ ହୋଇଥାଏ, ତେଣୁ ଗାଣିତିକ ଅପରେସନ୍‌ର ଫଳାଫଳ ଠିକ୍ ଫଳାଫଳର ଏକ ଆସନ୍ନ ମାନ ହୋଇଥାଏ ।

ଯାହାକି

```
6.0  / 7.0  = 0.857143
-1.0 / 3.0  = -0.333333
3.0  / -2.0 = -1.500000
```

ମୋଡୁଲସ୍ ଅପରେସନ୍ ବାସ୍ତବ ଅପେରାଣ୍ଡ ସହିତ ବ୍ୟବହୃତ ହୋଇପାରିବ ନାହିଁ ।

ମିଶ୍ରିତ-ମୋଡ୍ ଗଣିତ (Mixed-mode Arithmetic)

ଯେତେବେଳେ ଗୋଟିଏ ଅପେରାଣ୍ଡ ବାସ୍ତବ ଏବଂ ଅନ୍ୟଟି ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ହୋଇଥାଏ ସେତେବେଳେ, ଅପରେସନ୍‌କୁ ମିଶ୍ରିତ-ମୋଡ୍ ଗଣିତ କୁହାଯାଇଥାଏ ଏବଂ ଏହା ସର୍ବଦା ଏକ ବାସ୍ତବ ମୂଲ୍ୟ ପ୍ରଦାନ କରିଥାଏ । ଏଠାରେ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାର ମୂଲ୍ୟ ବାସ୍ତବ ମୂଲ୍ୟରେ ପରିଣତ ହୁଏ ଏବଂ ତା'ପରେ ବାସ୍ତବ ଗଣନା କରାଯାଏ, ଯାହାର ଫଳାଫଳ ମୂଲ୍ୟ ବାସ୍ତବ ହୋଇଥାଏ ।

ଯାହାକି

$6 / 7.0 \rightarrow 6.0 / 7.0 = 0.857143$

$-1.0 / 3 \rightarrow -1.0 / 3.0 = -0.333333$

ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ

$15 / 10.0 = 1.5$

$15.0 / 10 = 1.5$

ଯେତେବେଳେ କି

$15 / 10 = 1$

2.2.2 ସମ୍ବନ୍ଧୀୟ (ତୁଳନା) ଅପରେଟର୍ (Relational (Comparison) Operators)

ନିଷ୍ପତ୍ତି ସୁଗମ କରିବା ପାଇଁ ଏହି ଅପରେଟର୍‌ଗୁଡ଼ିକ ଅପେରାଣ୍ଡର ମୂଲ୍ୟ ତୁଳନା କରିବାରେ ବ୍ୟବହୃତ ହୁଏ, ଏହା ସୁସଙ୍ଗତ ହେବା ଆବଶ୍ୟକ । ତୁଳନା ସଫଳ ହେଲେ, ସେଗୁଡ଼ିକର ମୂଲ୍ୟ 1 ଫେରସ୍ତ କରୁଥିବା (ବୁଲିଆନ୍ ମୂଲ୍ୟ True ସହ ସମାନ) ଏବଂ ଯଦି ତୁଳନାରୁ ବିଫଳ ହୁଏ ତେବେ ସେଗୁଡ଼ିକ ମୂଲ୍ୟ 0 ଫେରସ୍ତ କରନ୍ତି (ବୁଲିଆନ୍ ମୂଲ୍ୟ False ସହ ସମାନ) ।

ଟେବୁଲ୍ 2.2: ସମ୍ବନ୍ଧୀୟ (ତୁଳନା) ଅପରେଟର୍

ଅପରେଟର	ପ୍ରତୀକ	ଫର୍ମ	ଅର୍ଥ
ଠାରୁ ଅଧିକ (Greater than)	>	x > y	ଯଦି x ହେଉଛି y ଠାରୁ ଅଧିକ ତେବେ ଏହା True ଫେରସ୍ତ କରିଥାଏ ।
ଠାରୁ ଅଧିକ କିମ୍ବା ସମାନ (Greater than or equal to)	> =	x > = y	ଯଦି x ହେଉଛି y ଠାରୁ ଅଧିକ କିମ୍ବା ସମାନ ତେବେ ଏହା True ଫେରସ୍ତ କରିଥାଏ ।
ଠାରୁ କମ୍ (Less than)	<	x < y	ଯଦି x ହେଉଛି y ଠାରୁ କମ୍ ତେବେ ଏହା True ଫେରସ୍ତ କରିଥାଏ ।
ଠାରୁ କମ୍ କିମ୍ବା ସମାନ (Less than or equal to)	< =	x < = y	ଯଦି x ହେଉଛି y ଠାରୁ କମ୍ କିମ୍ବା ସମାନ ତେବେ ଏହା True ଫେରସ୍ତ କରିଥାଏ ।
ସହିତ ସମାନ (Equal to)	==	x == y	ଯଦି x ହେଉଛି y ସହିତ ସମାନ ତେବେ ଏହା True ଫେରସ୍ତ କରିଥାଏ ।
ସହିତ ସମାନ ନୁହେଁ (Not equal to)	!=	x! = y	ଯଦି x ର ମୂଲ୍ୟ ସହିତ y ର ମୂଲ୍ୟ ସମାନ ନ ଥାଏ ତେବେ ଏହା True ଫେରସ୍ତ କରିଥାଏ ।

ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ C ଭାଷାରେ, ଅଣ-ଶୂନ୍ୟ ମୂଲ୍ୟକୁ ବୁଲିଆନ୍ True ଏବଂ ଶୂନ୍ୟ ମୂଲ୍ୟକୁ ବୁଲିଆନ୍ False ସହିତ ସମାନ ଭାବରେ ବିବେଚନା କରାଯାଏ । ଆହୁରି ମଧ, C ଭାଷାର ପୁରୁଣା ସଂସ୍କରଣରେ, C ଭାଷାରେ ବୁଲିଆନ୍ ତଥ୍ୟ ପାଇଁ କୌଣସି ସମର୍ଥନ ନାହିଁ । C ଭାଷାର ନୂତନ ସଂସ୍କରଣ, C99 ମାନକ ଭାବରେ ଜଣା ଏବଂ ଏହା ବୁଲିଆନ୍ ପ୍ରକାରକୁ ସମର୍ଥନ କରିଥାଏ ।

2.2.3 ଲଜିକାଲ୍ ଅପରେଟର୍ (Logical Operators)

ସମ୍ବନ୍ଧୀୟ ଅପରେଟର୍‌ଦ୍ୱାରା ଗଠିତ ଦୁଇ କିମ୍ବା ଅଧିକ ସରଳ ସର୍ତ୍ତକୁ ଯୋଡ଼ି ଏକ ଯୌଗିକ ସର୍ତ୍ତର ଗଠନପାଇଁ ଏହି ଅପରେଟର୍‌ଗୁଡ଼ିକ ବ୍ୟବହୃତ ହୁଏ ।

ଟେବୁଲ୍ 2.3: ଲଜିକାଲ୍ ଅପରେଟର

ଅପରେଟର	izrhd	:i	ଅର୍ଥ
ଲଜିକାଲ୍ AND	&&	x&&y	ଯଦି ଉଭୟ ଅପେରାଣ୍ଡ True ଅଛି ତେବେ ଏହା True ଫେରସ୍ତ କରେ ।
ଲଜିକାଲ୍ OR	||	x||y	ଯଦି କୌଣସି ଏକ ଅପେରାଣ୍ଡ True ଅଛି ତେବେ ଏହା True ଫେରସ୍ତ କରେ ।
ଲଜିକାଲ୍ NOT	!	!x	ଏହା True ପ୍ରଦାନକରେ ଯଦି ଅପେରାଣ୍ଡଟି False ଅଛି, ଏବଂ False ପ୍ରଦାନକରେ ଯଦି ଅପେରାଣ୍ଡଟି True ଅଛି (ଏହା ଅପେରାଣ୍ଡର ଅନୁପୂରକ କରେ)

2.2.4 ବିଟ୍‌ୱାଇଜ୍ ଅପରେଟର୍ (Bitwise Operators)

ବିଟ୍‌ୱାଇଜ୍ ଅପରେଟର୍‌ଗୁଡ଼ିକ ଅପେରାଣ୍ଡ ଉପରେ କାର୍ଯ୍ୟରେ ଯେମିତି କି ଅପେରାଣ୍ଡଟି ବାଇନାରି ଅଙ୍କଗୁଡିକର ଷ୍ଟ୍ରିଙ୍ଗ୍ । ଏହା ବାଇନାରି ଅଙ୍କଗୁଡିକ ଉପରେ କାର୍ଯ୍ୟ କରିଥାଏ, ତେଣୁ ଏହାର ନାମ ବିଟ୍‌ୱାଇଜ୍ । ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ ଏହି ଅପରେଟର୍‌ଗୁଡ଼ିକ କେବଳ ପୂର୍ଣ୍ଣାଙ୍କ ଅପେରାଣ୍ଡ ସହିତ ବ୍ୟବହୃତ ହୋଇଥାଏ ।

ବିଟ୍‌ୱାଇଜ୍ ଅପରେଟରର କାର୍ଯ୍ୟକୁ ପ୍ରଦର୍ଶନ କରିବାକୁ, ଧରାଯାଉ x = 10 (0000 1010 ବାଇନାରିରେ) ଏବଂ y = 4 (0000 0100 ବାଇନାରିରେ), ଏବଂ ଏକ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାକୁ ପ୍ରତିନିଧିତ୍ୱ କରିବା ପାଇଁ 8 ବିଟ୍ ବ୍ୟବହୃତ ହୋଇଛି ।

ଟେବୁଲ୍ 2.4: ବିଟ୍‌ୱାଇଜ୍ ଅପରେଟର୍

ଅପରେଟର	ପ୍ରତୀକ	ଫର୍ମ	ଅର୍ଥ	ଉଦାହରଣ
ବିଟ୍‌ୱାଇଜ୍ AND	&	x & y	ଯଦି ଅନୁରୂପ ବିଟ୍ ଗୁଡିକ 1 ହୁଅନ୍ତି ତେବେ ଏହା 1 ଫେରସ୍ତ କରେ ଅନ୍ୟଥା 0 ।	x = 0000 1010 10 y = 0000 0100 (4) x & y = 0000 0000 (0)
ବିଟ୍‌ୱାଇଜ୍ OR	|	x | y	ଯଦି କୌଣସି ଗୋଟିଏ କିମ୍ବା ଉଭୟ ଅନୁରୂପ ବିଟ୍‌ଗୁଡିକ 1 ହୁଅନ୍ତି ତେବେ ଏହା 1 ଫେରସ୍ତ କରେ ଅନ୍ୟଥା 0 ।	x = 0000 1011 (11) y = 0000 0101 (5) x | y = 0000 1111 (15)
ବିଟ୍‌ୱାଇଜ୍ XOR	^	x ^ y	ଯଦି ଅନୁରୂପ ବିଟ୍ ସମାନ ନୁହେଁ ତେବେ ଏହା 1 ଫେରସ୍ତ କରେ ।	x = 0000 1011 (11) y = 0000 0101 (5) x ^ y = 0000 1110 (14)
ବିଟ୍‌ୱାଇଜ୍ NOT	~	~x	1 କୁ 0 ଏବଂ 0 କୁ 1 କୁ ଓଲଟାଇଥାଏ ।	x = 0000 1011 (11) ~x = 1111 0100 (–12)
ବିଟ୍‌ୱାଇଜ୍ right shift	>>	x>>y	x ର ବିଟ୍ ଗୁଡିକୁ ଡାହାଣପଟକୁ y ସଂଖ୍ୟକ ସ୍ଥାନ ଲେଖାଏଁ ଘୁଞ୍ଚାଯାଏ । ପ୍ରତ୍ୟେକ ସ୍ଥାନାନ୍ତରଣରେ ଶେଷ ଡାହାଣ ବିଟ୍‌ଟି ସଂଖ୍ୟାର ବାହାରକୁ ବାହାରିଯାଏ, କିନ୍ତୁ ଶେଷ ବାମ ବିଟ୍‌ଟି ସଂଖ୍ୟା ଭିତରେ ରହେ । ଟିପ୍ପଣୀ: ପ୍ରତ୍ୟେକ 1-ବିଟ୍ ଡାହାଣ ସିଫ୍ଟ ହେଉଛି ମୂଳ ସଂଖ୍ୟାକୁ 2 ଦ୍ୱାରା ପୂର୍ଣ୍ଣାଙ୍କ ବିଭାଜନ ସହିତ ସମାନ ।	x = 0000 1011 (11) x >> 1 = 0000 0101 (5)
ବିଟ୍‌ୱାଇଜ୍ left shift	<<	x << y	x ର ବିଟ୍ ଗୁଡିକୁ ବାମପଟକୁ y ସଂଖ୍ୟକ ସ୍ଥାନ ଲେଖାଏଁ ଘୁଞ୍ଚାଯାଏ । ପ୍ରତ୍ୟେକ ସ୍ଥାନାନ୍ତରଣରେ ଶେଷ ବାମ ବିଟ୍‌ଟି ସଂଖ୍ୟାର ବାହାରକୁ ବାହାରିଯାଏ, କିନ୍ତୁ ଶେଷ ଡାହାଣ ବିଟ୍‌ଟି ସଂଖ୍ୟା ଭିତରେ ରହେ । ଟିପ୍ପଣୀ: ପ୍ରତ୍ୟେକ 1-ବିଟ୍ ବାମ ସିଫ୍ଟ ହେଉଛି ମୂଳ ସଂଖ୍ୟାକୁ 2 ଦ୍ୱାରା ଗୁଣନ ସହିତ ସମାନ ।	x = 0000 1011 (11) x << 1 = 0001 0110 (22)

2.2.5 ସ୍ୱତନ୍ତ୍ର ଅପରେଟର୍ (Special Operators)

ଆସ ସେଗୁଡ଼ିକ ସଂକ୍ଷେପରେ ଆଲୋଚନା କରିବା। ତୁମେ ପରବର୍ତ୍ତୀ ବିଭାଗ ଏବଂ ଅଧ୍ୟାୟରେ ଯଥୋଚିତ ରୂପେ ସେଗୁଡ଼ିର ଉପଯୋଗିତା ଉପଲବ୍ଧି କରିବାକୁ ସକ୍ଷମ ହେବ।

2.2.5.1 ବୃଦ୍ଧି ଏବଂ ହ୍ରାସ ଅପରେଟର୍ (Increment & Decrement Operators)

C ଭାଷା ଅତ୍ୟନ୍ତ ଉପଯୋଗୀ ଦୁଇଟି ଅପରେଟରଙ୍କୁ ସମର୍ଥନ କରେ - ବୃଦ୍ଧି ଅପରେଟର (++) ଏବଂ ହ୍ରାସ ଅପରେଟର (--). ଏହି ଅପରେଟରଗୁଡିକ ସମସ୍ତ ମୌଳିକ ଡାଟା ପ୍ରକାର ସହିତ ବ୍ୟବହୃତ ହୋଇପାରେ। ବୃଦ୍ଧି ଅପରେଟର ଅପେରାଣ୍ଡରେ 1 ଯୋଗ କରୁଥିବାବେଳେ ହ୍ରାସ ଅପରେଟର ଅପେରାଣ୍ଡରୁ 1 ବିୟୋଗ କରେ।

ଉଭୟ ୟୁନାରି (unary) ଅପରେଟର୍ ଏବଂ ନିମ୍ନରେ ଦର୍ଶାଯାଇଥିବା ପରି ପୂର୍ବଲଗ୍ନ (prefix) ବା ଅନୁଲଗ୍ନ (postfix) ସଂକେତନ ଭାବରେ ବ୍ୟବହୃତ ହୋଇପାରିବ:

```
++k;                 or        k++;
--k;                 or        k--;
```

ଏଠାରେ, ++k (ଓ k++) ହେଉଛି k = k + 1 ବା k+=1 ସହ ସମାନ, ସେହିପରି --k (ଓ k--) ହେଉଛି k =k -1 ବା k-=1 ସହ ସମାନ।

ଏହି ଅପରେଟର୍‌ଗୁଡ଼ିକ while ଏବଂ for ଲୁପ୍‌ରେ ନିୟନ୍ତ୍ରଣ ଚଳ (control variables) ଭାବରେ ପ୍ରାୟତଃ ବ୍ୟବହୃତ ହୁଅନ୍ତି।

ଗୁଡ଼ିକ ସ୍ୱାଧୀନ ଭାବରେ ବିବୃତ୍ତି ଗଠନ କରିଲେ ++K କିମ୍ବା K++ ଏହାର ଅର୍ଥ ସମାନ ହୋଇଥାଏ, ଗୁଡ଼ିକ ଅନ୍ୟ ପରିପ୍ରକାଶର ଏକ ଅଂଶ ଭାବରେ ବ୍ୟବହୃତ ହୁଏ ସେତେବେଳେ ସେଗୁଡ଼ିକ ଭିନ୍ନ ଆଚରଣ କରେ।

ନିମ୍ନଲିଖିତ ଉକ୍ତିଗୁଡିକ ବିଚାର କର

```
int y, k = 5;
y = ++k;
```

ଏହି କ୍ଷେତ୍ରରେ, ପ୍ରଥମେ k ର ମୂଲ୍ୟ ବୃଦ୍ଧି ହୋଇଛି ଏବଂ ତା'ପରେ k ର ମୂଲ୍ୟ y କୁ ନ୍ୟସ୍ତ ହୋଇଛି ତେଣୁ y ଏବଂ k ର ମୂଲ୍ୟ ସମାନ ଏବଂ 6 ରହିବ। ସେହିଭଳି ଉପରୋକ୍ତ ବିବୃତ୍ତିଗୁଡିକ ନିମ୍ନଲିଖିତ ବିବୃତ୍ତି ସହିତ ସମାନ।

```
int y, k = 5;
++k;
y = k;
```

ତଥାପି, ଯଦି ଆମେ ଉପରୋକ୍ତ ବିବୃତ୍ତିଗୁଡିକ ଲେଖିବା –

```
int y, k = 5;
 y = k++;
```

ଏହି କ୍ଷେତ୍ରରେ, ପ୍ରଥମେ k ର ସାମ୍ପ୍ରତିକ ମୂଲ୍ୟ (ମୂଲ୍ୟ 5) y କୁ ନ୍ୟସ୍ତ ହୋଇଛି ଏବଂ ତା'ପରେ k ର ମୂଲ୍ୟ ବୃଦ୍ଧି ହୋଇଛି ଏବଂ ତେଣୁ y ମୂଲ୍ୟ 5 ଥିବା ବେଳେ k ମୂଲ୍ୟ 6 ଭାବରେ ରହିବ।

ସେହିଭଳି, ଉପରୋକ୍ତ ବିବୃତ୍ତିଗୁଡିକ ନିମ୍ନଲିଖିତ ବିବୃତ୍ତିସହିତ ସମାନ–

```
int y, k = 5;                          int y, k = 5;
 y = k;                 or                   y = k;
 ++k;                                        k++;
```

2.2.5.2 sizeof ଅପରେଟର୍ (The sizeof Operator)

sizeof ଅପରେଟର୍ ଦିଆଯାଇଥିବା ଅପେରାଣ୍ଡର ଆକାରକୁ ଏଥିରେ ଥିବା ବାଇଟ ସଂଖ୍ୟାରେ ଫେରସ୍ତ କରିଥାଏ। sizeof ଅପରେଟର୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
sizeof( exp )
```

ଯେଉଁଠାରେ exp ଏକ ଡାଟା ଟାଇପ୍ (ବିଲ୍ଟ-ଇନ୍ କିମ୍ବା ବ୍ୟବହାରକାରୀ-ପରିଭାଷିତ), ଏକ ଧ୍ରୁବକ ମୂଲ୍ୟ / ଧ୍ରୁବକ କିମ୍ବା ଏକ ଚଳର ପ୍ରତିନିଧିତ୍ୱ କରେ। ଉଦାହରଣ ସ୍ୱରୂପ

```
sizeof( float )
```

ଟେବୁଲ୍ 2.5: sizeof ଅପରେଟର୍‌ର ବ୍ୟବହାର

sizeof ଅପରେଟର୍ ନିମ୍ନମତେ ବ୍ୟବହାର ହୁଏ	ଫେରସ୍ତ	ଯଥାର୍ଥତା
sizeof (1)	2	ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ଧ୍ରୁବକଟିଏ ଡାଟା ଟାଇପ୍ int ସହ ସମ୍ପୃକ୍ତ ଅଟେ, ଏବଂ ଟର୍ବୋ C/C++ ରେ ଏହାର ଆକାର 2 ଏବଂ dev C ++ ରେ 4 ଅଟେ।
sizeof ('a)	1	ବର୍ଣ୍ଣ ଧ୍ରୁବକର ଆକାର ହେଉଛି 1।
sizeof (int)	2	ଡାଟା ଟାଇପ୍ int ର ଆକାର ଟର୍ବୋ C/C++ ରେ 2 ଏବଂ dev C ++ ରେ 4 ଅଟେ।
sizeof (125.25)	8	ବାସ୍ତବ ସଂଖ୍ୟା ଧ୍ରୁବକର ଡାଟା ଟାଇପ୍ double ଅଟେ।
sizeof (125.25F)	4	ବାସ୍ତବ ସଂଖ୍ୟା ଧ୍ରୁବକ ଡାଟାର ଡାଟା ଟାଇପ୍ float ଅଟେ, ଏବଂ ଏହାର ଆକାର 4।
double x; sizeof (x);	8	ଚଳ x ର ଟାଇପ୍ double ଅଟେ, ଏବଂ ଏହାର ଆକାର 8।

2.2.5.3 ଠିକଣା ଅପରେଟର୍ (The addressof Operator)

ସଂକେତ '&' ଏକ ଚଳରେ ଉପସର୍ଗ ହୋଇଥିଲେ ତାହା ମେମୋରି ଅବସ୍ଥାନର ଠିକଣା ଫେରସ୍ତ କରେ । ତେଣୁ ଏହାର ନାମ ଠିକଣା ଅପରେଟର । ଏହା scanf() ଫଙ୍କ୍ସନ୍‌ରେ ଏବଂ ପଏଁଟର୍‌ର (pointer) ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟପାଇଁ ହୁଏ । ଠିକଣା (&) ଅପରେଟର୍‌ର ବ୍ୟବହାର ଟେବୁଲ୍ 5.11 ରେ ପ୍ରଦର୍ଶିତ ହୋଇଛି ।

2.2.5.4 ପରୋକ୍ଷ ଅପରେଟର୍ (Indirection Operator)

ଯେତେବେଳେ ସଂକେତ '*' ଏକ ପଏଁଟର୍ ଚଳ ସହିତ ଉପସର୍ଗ ହୋଇଥାଏ ଏହା ସେହି ମେମୋରି ସ୍ଥାନରେ ଗଚ୍ଛିତ ହୋଇଥିବା ମୂଲ୍ୟକୁ ଫେରସ୍ତ କରେ ଯାହାର ଠିକଣା ପଏଁଟର୍ ଚଳରେ ଥାଏ । ଅର୍ଥାତ, ପରୋକ୍ଷରେ ପଏଁଟର୍ ଚଳ ମାଧ୍ୟମରେ ମୂଲ୍ୟ ଗ୍ରହଣ କରାଯାଏ, ତେଣୁ ଏହାର ନାମ ପରୋକ୍ଷ ଅପରେଟର୍ ।

ନିମ୍ନଲିଖିତ ଉକ୍ତିଗୁଡିକୁ ବିଚାର କର

```
int y = 10, x;
 int *py;
 py = &y;
 x = *py;
```

ପରିପ୍ରକାଶ "*py", ମେମୋରି ଠିକଣାରେ ଥିବା ମୂଲ୍ୟ ଫେରସ୍ତ କରେ ଯେଉଁ ଠିକଣା ପଏଁଟର୍ py ଚଳରେ ଥାଏ, ଅର୍ଥାତ, ଚଳ y ର ମୂଲ୍ୟ ଯାହା 10, ଚଳ x କୁ ନ୍ୟସ୍ତ ହୁଏ ।

ଟେବୁଲ୍ 2.6: ଠିକଣା (&) ଅପରେଟର୍‌ର ବ୍ୟବହାର

ଠିକଣା (&) ଅପରେଟର୍ ନିମ୍ନମତେ ବ୍ୟବହାର ହୁଏ	ହେଉଥିବା କାର୍ଯ୍ୟ

int y = 10;
inn *py;
py = &y;
	ପ୍ରଥମ ବିବୃତ୍ତି ଚଳ y କୁ int ପ୍ରକାରର ଘୋଷଣା ଏବଂ ବ୍ୟାଖ୍ୟା କରେ । ଏହା ମଧ୍ୟ ଏହାର ପ୍ରାଥମିକ ମୂଲ୍ୟ 10 ନ୍ୟସ୍ତ କରେ ।
	ଦ୍ୱିତୀୟ ବିବୃତ୍ତି ଏକ ପଏଁଟର୍ ଚଳ py ଘୋଷଣା ଏବଂ ବ୍ୟାଖ୍ୟା କରେ ଯାହା ଏହା int ପ୍ରକାରର ମୂଲ୍ୟ ସଂରକ୍ଷଣ ପାଇଁ ସଂରକ୍ଷିତ ଏକ ମେମୋରି ଅବସ୍ଥାନର ଠିକଣା ରଖିପାରେ ।
	ତୃତୀୟ ବିବୃତ୍ତି ଚଳ y ର ଠିକଣାକୁ ପଏଁଟର୍ ଚଳ py କୁ ନ୍ୟସ୍ତ କରେ ।

int x, y; scanf (" % d % d", &x, &y);	ପ୍ରଥମ ବିବୃତ୍ତି ଦୁଇଟି ଚଳ x ଏବଂ y କୁ int ପ୍ରକାରର ଘୋଷଣା ଏବଂ ବ୍ୟାଖ୍ୟା କରେ । ଦ୍ୱିତୀୟ ବିବୃତ୍ତି କୀବୋର୍ଡ୍‌ରୁ ଇନପୁଟ୍ ଭାବରେ ଦୁଇଟି ପୂର୍ଣସଂଖ୍ୟା ମୂଲ୍ୟ ନେଇଥାଏ ଏବଂ ଚଳ x ଏବଂ y ଠିକଣାରେ ଯଥାକ୍ରମେ ସେହି ମୂଲ୍ୟଗୁଡ଼ିକୁ ଗଚ୍ଛିତ କରେ ।

2.2.5.5 ସର୍ତ୍ତମୂଳକ/ଟର୍ନାରୀ ଅପରେଟର୍ (Conditional/Ternary Operator)

ନିମ୍ନ ଖଣ୍ଡରେ ଲେଖାଯାଇଥିବା ଛଦ୍ମକୋଡ୍/ସ୍ୟୁଡୋକୋଡ୍କୁ ବିଚାର କର

```
if ( a > b ) then
    set big = a
 else
    set big = b
 endif
```

ଏହି କୋଡ୍ ଖଣ୍ଡରେ a ଏବଂ b ମଧ୍ୟରୁ ସର୍ବାଧିକ ମୂଲ୍ୟ big କୁ ନ୍ୟସ୍ତ କରେ। ଏହା ପ୍ରତ୍ୟକ୍ଷ ଯେ ଚଳ big କୁ ଦିଆଯାଇଥିବା ମୂଲ୍ୟ "a > b" ସର୍ତ୍ତ ପରୀକ୍ଷଣର ଫଳାଫଳ ଉପରେ ନିର୍ଭର କରିବ।

ଏହିପରି ପରିପ୍ରକାଶଗୁଡିକ ସର୍ତ୍ତମୂଳକ ପରିପ୍ରକାଶ ଭାବରେ ଜଣାଶୁଣା, ଏବଂ ଏହା ସର୍ତ୍ତମୂଳକ ଅପରେଟର୍ ବ୍ୟବହାର କରି ଲେଖାଯାଇପାରିବ। ଟର୍ନାରି ଅପରେଟର ବ୍ୟବହାର କରିବାର ସିଣ୍ଟାକ୍ସ ହେଉଛି –

```
exp1 ? exp2 : exp3;
```

ଯେଉଁଠାରେ exp1, exp2, exp3 ଗୁଡ଼ିକ ହେଉଛନ୍ତି ପରିପ୍ରକାଶ।

ପରିପ୍ରକାଶ exp1 ପ୍ରଥମେ ମୂଲ୍ୟାଙ୍କନ କରାଯାଇଛି। ଯଦି ଏହା ଅଣ-ଶୂନ୍ୟ (True), ତେବେ ପରିପ୍ରକାଶ exp2 ମୂଲ୍ୟାଙ୍କନ କରାଯାଏ, ଏବଂ ତାହା ହେଉଛି ସର୍ତ୍ତମୂଳକ ପରିପ୍ରକାଶର ମୂଲ୍ୟ; ଅନ୍ୟଥା ପରିପ୍ରକାଶ exp3 ମୂଲ୍ୟାଙ୍କନ କରାଯାଏ, ଏବଂ ତାହା ହେଉଛି ସର୍ତ୍ତମୂଳକ ପରିପ୍ରକାଶର ମୂଲ୍ୟ। ଧ୍ୟାନ ଦିଅନ୍ତୁ ଯେ exp2 ଏବଂ exp3 ମଧ୍ୟରୁ କେବଳ ଗୋଟିଏ ପରିପ୍ରକାଶକୁ ମୂଲ୍ୟାଙ୍କନ କରାଯାଇଛି।

ଏହିପରି, ସର୍ତ୍ତମୂଳକ ଅପରେଟର୍ ବ୍ୟବହାର କରି ଉପରୋକ୍ତ ଛଦ୍ମକୋଡ୍ ଖଣ୍ଡ ଲେଖା ଯାଇପାରିବ

```
big = ( a > b ) ? a : b;
```

ସର୍ତ୍ତମୂଳକ ପରିପ୍ରକାଶର ପ୍ରଥମ ପରିପ୍ରକାଶରେ ଚନ୍ଦ୍ର ବନ୍ଧନୀର ଆବଶ୍ୟକତା ନାହିଁ କାରଣ '? :' ର ପ୍ରାଥମିକତା ବହୁତ କମ୍ ଏବଂ ଆସାଇନମେଣ୍ଟ ଅପରେଟରଠାରୁ ଠିକ୍ ଉପରେ। ତଥାପି, ଚନ୍ଦ୍ର ବନ୍ଧନୀର ବ୍ୟବହାରକୁ ସୁପାରିଶ କରାଯାଇଛି କାରଣ ସେଗୁଡ଼ିକ ସର୍ତ୍ତମୂଳକ ପରିପ୍ରକାଶର ସର୍ତ୍ତକୁ ବୁଝିବାପାଇଁ ସହଜ କରିଥାଆନ୍ତି।

2.2.5.6 ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍ (Assignment operators)

ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍ଗୁଡ଼ିକ ଚଳକୁ ମୂଲ୍ୟ ନ୍ୟସ୍ତ କରିବାକୁ ବ୍ୟବହୃତ ହୋଇଥାଆନ୍ତି। ଉଦାହରଣ ସ୍ୱରୂପ

```
a = 5
```

ଏଠାରେ '=' ହେଉଛି ଏକ ସରଳ ଆସାଇନମେଣ୍ଟ ଅପରେଟର ଯାହା ମୂଲ୍ୟ 5 ଚଳ a କୁ ନ୍ୟସ୍ତ କରେ। ଏହା ଅନ୍ୟ ଚଳ କିମ୍ବା ପରିପ୍ରକାଶର ମୂଲ୍ୟ ନ୍ୟସ୍ତ କରିବାକୁ ମଧ୍ୟ ବ୍ୟବହୃତ ହୋଇପାରେ।

C ଭାଷାରେ ବିଭିନ୍ନ ଯୌଗିକ ଅପରେଟର ଅଛନ୍ତି ଯେପରିକି

```
a += 5
```

ଯାହା ଚଳ ସହିତ ମୂଲ୍ୟକୁ ମିଶାଇଥାଏ ଏବଂ ପରେ ସେହି ମୂଲ୍ୟ ଚଳକୁ ନ୍ୟସ୍ତ କରେ। ଏହା ନିମ୍ନ ଲିଖିତ ବିବୃତ୍ତି ସହିତ ସମାନ।

```
a = a + 5
```

ଏହି '+=' ପ୍ରକାରର ଅପରେଟରଗୁଡ଼ିକୁ ମଧ୍ୟ ସଂକ୍ଷିପ୍ତ ଆସାଇନମେଣ୍ଟ ଅପରେଟର କୁହାଯାଏ ।

ଟେବୁଲ୍ 2.7: ଆସାଇନମେଣ୍ଟ ଅପରେଟର

ଅପରେଟର	ଉଦାହରଣ	ସମକକ୍ଷ ଅଟେ
=	x = 5	x = 5
+ =	x + = 5	x = x + 5
− =	x − = 5	x = x − 5
*=	x * = 5	x = x * 5
/=	x / = 5	x = x / 5
%=	x % = 5	x = x % 5

ଅପରେଟର	ଉଦାହରଣ	ସମକକ୍ଷ ଅଟେ
& =	x & = 5	x = x & 5
| =	x | = 5	x = x | 5
^ =	x ^ = 5	x = x ^ 5
>>=	x >> = 5	x = x >> 5
<<=	x << = 5	x = x << 5

2.3 ପରିପ୍ରକାଶ (EXPRESSIONS)

ପରିପ୍ରକାଶ ଏକ ସୂତ୍ର ଯାହା ଏକ କିମ୍ବା ଅଧିକ ଅପେରାଣ୍ଡ୍ ଏବଂ ଶୂନ କିମ୍ବା ଅଧିକ ଅପରେଟରଙ୍କୁ ନେଇ ମୂଲ୍ୟ ଗଣନା କରିବା ପାଇଁ ଗଠିତ ହୋଇଥାଏ । ଗୋଟିଏ ଅପେରାଣ୍ଡ୍ ଏକ ଫଙ୍କ୍ସନ୍ ରେଫରେନ୍ସ, ଏକ ଚଳ , ଏକ ଆରେ ଉପାଦାନ କିମ୍ବା ଏକ ଧ୍ରୁବକ ହୋଇପାରେ ।

ଉଦାହରଣ ସ୍ୱରୂପ, ନିମ୍ନ ଲିଖିତ ପରିପ୍ରକାଶ

```
a + b
```

ରେ ଯୁକ୍ତ ଚିହ୍ନ '+' ହେଉଛି ଏକ ଅପରେଟର୍ ଏବଂ a ଏବଂ b ଗୁଡ଼ିକ ଅପେରାଣ୍ଡ୍ ।

C ରେ ଚାରି ପ୍ରକାରର ପରିପ୍ରକାଶ ଅଛନ୍ତି । ଏଗୁଡ଼ିକ ହେଉଛି

1. ଗାଣିତିକ ପରିପ୍ରକାଶ
2. ସମ୍ବନ୍ଧୀୟ ପରିପ୍ରକାଶ
3. ତାର୍କିକ/ଲଜିକାଲ୍ ପରିପ୍ରକାଶ
4. ସର୍ତ୍ତମୂଳକ ପରିପ୍ରକାଶ

ପ୍ରତ୍ୟେକ ପ୍ରକାରର ପରିପ୍ରକାଶ ନିର୍ଦ୍ଦିଷ୍ଟ ପ୍ରକାରର ଅପରେଟର୍ ନେଇଥାଏ ଏବଂ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଅପରେଟର୍ ସେଟ୍ ବ୍ୟବହାର କରେ । ପ୍ରତ୍ୟେକ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ ନିର୍ଦ୍ଦିଷ୍ଟ ପ୍ରକାରର ମୂଲ୍ୟ ଉତ୍ପନ୍ନ କରେ । ପରିପ୍ରକାଶଗୁଡିକ ବିବୃତ୍ତି ନୁହେଁ, କିନ୍ତୁ ବିବୃତ୍ତିର ଅଂଶ ହୋଇପାରେ ।

ଉଦାହରଣ ସ୍ୱରୂପ, ନିମ୍ନ ଲିଖିତ ଧାଡିକୁ ବିଚାର କର

```
x = 2.0/3.0 + a * b;
```

ସମଗ୍ର ଧାଡିଟି ଏକ ବିବୃତ୍ତି, କିନ୍ତୁ ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍‌ର ପର ଅଂଶଟି ଏକ ପରିପ୍ରକାଶ ଅଟେ ।

ଏହି ବିଭାଗରେ, ଆମର ଆଲୋଚନା ଗାଣିତିକ ପରିପ୍ରକାଶରେ ସୀମିତ ରହିବ ।

2.3.1 ଗାଣିତିକ ପରିପ୍ରକାଶ (Arithmetic Expressions)

ଏକ ଗାଣିତିକ ପରିପ୍ରକାଶ ଅପେରାଣ୍ଡ ଏବଂ ଗାଣିତିକ ଅପରେଟର୍‌ଗୁଡ଼ିକୁ ନେଇ ଗଠିତ । ଏହା ଏକ ମୂଲ୍ୟ ଉତ୍ପନ୍ନ କରେ ଯାହା *int, float* ବା *double* ପ୍ରକାରର ହୋଇଥାଏ । ଯେତେବେଳେ କୌଣସି ପରିପ୍ରକାଶ କେବଳ ପୂର୍ଣ୍ଣାଙ୍କ ଅପେରାଣ୍ଡ ସହିତ ଜଡିତ, ତାହାକୁ ଏକ ଶୁଦ୍ଧ ପୂର୍ଣ୍ଣାଙ୍କ ପରିପ୍ରକାଶ କୁହାଯାଏ । ଯେତେବେଳେ ଏଥିରେ କେବଳ ବାସ୍ତବ ସଂଖ୍ୟା ଅପେରାଣ୍ଡ ଅନ୍ତର୍ଭୁକ୍ତ ହୁଏ ତାହାକୁ ଏକ ଶୁଦ୍ଧ ବାସ୍ତବ ସଂଖ୍ୟା ପରିପ୍ରକାଶ କୁହାଯାଏ । ଯେତେବେଳେ ଏଥିରେ ଉଭୟ ପୂର୍ଣ୍ଣାଙ୍କ ଏବଂ ବାସ୍ତବ ସଂଖ୍ୟା ଅପେରାଣ୍ଡ ଅନ୍ତର୍ଭୁକ୍ତ ହୁଏ ତାହାକୁ ଏକ ମିଶ୍ରିତ ମୋଡ୍ ପରିପ୍ରକାଶ କୁହାଯାଏ ।

ଟେବୁଲ୍ 2.8: ନମୁନା ଗାଣିତିକ ପରିପ୍ରକାଶ

ଗାଣିତିକ/ ବୀଜଗାଣିତିକ ପରିପ୍ରକାଶ	C ଗାଣିତିକ ପରିପ୍ରକାଶ
$\frac{a+b}{2}$	`(a + b)/2`
$a+\frac{b}{c}+d$	`a+b/c+d`
$\frac{ab}{c-d^2}+d$	`(a*b)/(c-d*d)+d`
$1+\frac{a}{b+\frac{1}{c}}$	`1+a/(b+1/c)`
$\frac{a}{\frac{c+b}{d}}-e$	`a/((c+b)/d)-e`
ax^2+bx+c	`a*x*x+b*x+c`
$ut+\frac{1}{2}at^2$	`u*t+0.5*a*t*t`
$m\left(a\times h+\frac{v^2}{2}\right)$	`m*(a*h+(v*v)/2)`

ଗାଣିତିକ/ବୀଜଗାଣିତିକ ପରିପ୍ରକାଶକୁ C ଭାଷାର ଗାଣିତିକ ପରିପ୍ରକାଶରେ ପରିଣତ କରିବା ସମୟରେ, ଚନ୍ଦ୍ର ବନ୍ଧନୀ ଗୁଡ଼ିକୁ ସହଯୋଗିତା ଏବଂ ଅପରେଟରଗୁଡ଼ିକୁ ମୂଲ୍ୟାଙ୍କନ କରାଯାଉଥିବା କ୍ରମର ନିୟନ୍ତ୍ରଣ କରିବାପାଇଁ ବ୍ୟବହାର କରାଯାଇପାରିବ । ଯଦି ଚନ୍ଦ୍ର ବନ୍ଧନୀଗୁଡ଼ିକ ଗୋଟିଏ କ୍ଷେତ୍ରରେ ଉପସ୍ଥିତ ଥାଆନ୍ତି, ତେବେ ଚନ୍ଦ୍ର ବନ୍ଧନୀ ମଧ୍ୟରେ ଥିବା ପରିପ୍ରକାଶକୁ ପ୍ରଥମେ ମୂଲ୍ୟାଙ୍କନ କରାଯାଏ ଏବଂ ଚନ୍ଦ୍ର ବନ୍ଧନୀ ମଧ୍ୟରେ ଅପ୍ରତ୍ୟକ୍ଷ ପ୍ରାଥମିକତା ଦେଖାଯାଏ । ଯଦି ଚନ୍ଦ୍ର ବନ୍ଧନୀର ଅବସ୍ଥାନ (nesting) ଥାଏ (ଚନ୍ଦ୍ର ବନ୍ଧନୀ ଭିତରେ ଚନ୍ଦ୍ର ବନ୍ଧନୀ), ତେବେ ଆଭ୍ୟନ୍ତରୀଣ ଚନ୍ଦ୍ର ବନ୍ଧନୀ ମଧ୍ୟରେ ଥିବା ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ ପ୍ରଥମେ କରାଯାଏ ।

2.3.2 ଗାଣିତିକ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ (Evaluation of Arithmetic Expressions)

ତୁମେ ଜାଣିଛ ଯେ ଏକ ସମୟରେ ଗୋଟିଏ ଅପରେସନ୍ ମାଧ୍ୟମରେ ପରିପ୍ରକାଶଗୁଡିକର ମୂଲ୍ୟାଙ୍କନ କରାଯାଏ। ସେହିପରି ଭାବରେ ଗୋଟିଏ କମ୍ପ୍ୟୁଟରଦ୍ୱାରା ମଧ୍ୟ ପରିପ୍ରକାଶଗୁଡିକର ମୂଲ୍ୟାଙ୍କନ କରାଯାଇଥାଏ। ଏକକ ଅପରେସନ୍‌ର ମୂଲ୍ୟାଙ୍କନର କ୍ରମ ଅପରେଟରଗୁଡ଼ିକର ପ୍ରାଥମିକତା ଏବଂ ସହଯୋଗିତା ଦ୍ୱାରା ପରିଚାଳିତ ହୁଏ?

ତଥାପି, ଯେତେବେଳେ ଏକକ ଅପରେସନ୍ କରାଯାଏ, ନିମ୍ନଲିଖିତ ମାମଲାଗୁଡିକ ହୋଇପାରେ –

- ଉଭୟ ଅପେରାଣ୍ଡ ଯେତେବେଳେ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ହୋଇଥାନ୍ତି, ସେତେବେଳେ ପୂର୍ଣ୍ଣାଙ୍କ ଗଣିତ କରାଯିବ ଏବଂ ଅପରେସନ୍‌ର ଫଳାଫଳ ଏକ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ମୂଲ୍ୟ ହେବ।

  ଉଦାହରଣ ସ୍ୱରୂପ, 5/2 ର ଫଳାଫଳ 2.5 ନୁହେଁ ବରଂ 2 ହେବ କାରଣ ଏଠାରେ ଭଗ୍ନାଂଶ ଅଂଶକୁ ଅଣଦେଖା କରାଯାଏ।

- ଉଭୟ ଅପେରାଣ୍ଡ ଯେତେବେଳେ ବାସ୍ତବ ସଂଖ୍ୟା ପ୍ରକାରର, ତେବେ ବାସ୍ତବ ସଂଖ୍ୟା ଗଣିତ କରାଯିବ ଏବଂ ଅପରେସନର ଫଳାଫଳ ବାସ୍ତବ ମୂଲ୍ୟ ହେବ।

  ଉଦାହରଣ ସ୍ୱରୂପ, 5.0/2.0 ର ଫଳାଫଳ 2.0 ନୁହେଁ ବରଂ 2.5 ହେବ।

- ଗୋଟିଏ ଅପେରାଣ୍ଡ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ପ୍ରକାରର ଏବଂ ଦ୍ୱିତୀୟଟି ବାସ୍ତବ ସଂଖ୍ୟା ପ୍ରକାରର, ତେବେ ଏଠାରେ ମିଶ୍ରିତ ମୋଡ୍ ଗଣିତ କରାଯିବ। ଏହି କ୍ଷେତ୍ରରେ, ପ୍ରଥମ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ଅପେରାଣ୍ଡ ବାସ୍ତବ ଅପେରାଣ୍ଡରେ ପରିଣତ ହେବ, ଏବଂ ତା'ପରେ ବାସ୍ତବ ସଂଖ୍ୟା ଗଣିତ କରାଯିବ ଏବଂ ଅପରେସନ୍‌ର ଫଳାଫଳ ଏକ ବାସ୍ତବ ମୂଲ୍ୟ ହେବ। ଉଦାହରଣ ସ୍ୱରୂପ, 5/2.0 କିମ୍ବା 5.0/2 ର ଫଳାଫଳ 2.0 ନୁହେଁ ବରଂ 2.5 ହେବ, କାରଣ ପ୍ରଥମ କ୍ଷେତ୍ରରେ, 5 ର ମୂଲ୍ୟ 5.0 କୁ ରୂପାନ୍ତରିତ ହେବ ଏବଂ ତା'ପରେ ହରଣ କରାଯିବ ସେହିପରି ଦ୍ୱିତୀୟ କ୍ଷେତ୍ରରେ 2 ର ମୂଲ୍ୟ 2.0 କୁ ରୂପାନ୍ତରିତ ହେବ ଏବଂ ତା'ପରେ ହରଣ କରାଯିବ।

  ଗାଣିତିକ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ କରିବାର ଉପାୟ ବର୍ଣ୍ଣନା କରିବାକୁ, ନିମ୍ନଲିଖିତ ପରିପ୍ରକାଶକୁ ବିଚାର କର

  5 * 4 / (1 + 5 * 2 / 3 + 6) + 8 * (7 / 4)

ଏହି ପରିପ୍ରକାଶଟି ନିମ୍ନ ଟେବୁଲରେ ପ୍ରଦର୍ଶିତ ଉପାୟରେ ମୂଲ୍ୟାଙ୍କନ କରାଯାଏ:

ଟେବୁଲ୍ 2.9: ଗାଣିତିକ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନର ଚିତ୍ର

ଅପରେସନ୍ ଦ୍ୱାରା ଅପରେସନ୍ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ	ପ୍ରତ୍ୟେକ ଅପରେସନର ବର୍ଣ୍ଣନା
5 * 4 / (1 + 5 * 2 / 3 + 6) + 8 * (7 / 4)	ଦିଆଯାଇଥିବା ପରିପ୍ରକାଶ
5 * 4 / (1 + 10 / 3 + 6) + 8 * (7 / 4)	5 ଓ 2 ର ଗୁଣନ ହୁଏ, 10 ଦିଏ
5 * 4 / (1 + 3 + 6) + 8 * (7 / 4)	10 କୁ 3 ଦ୍ୱାରା ପୂର୍ଣ୍ଣାଙ୍କ ବିଭାଜନ କରି , 3 ଦିଏ
5 * 4 / 10 + 8 * (7 / 4)	3 କୁ 1 ରେ ମିଶାଯାଏ, 4 ଦିଏ
5 * 4 / 10 + 8 * 1	6 ଓ 4 ମିଶାଯାଏ, 10 ଦିଏ

ଅପରେସନ୍ ଦ୍ୱାରା ଅପରେସନ୍ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ	ପ୍ରତ୍ୟେକ ଅପରେସନର ବର୍ଣ୍ଣନା
`20 / 10 + 8 * 1`	7 କୁ 4 ଦ୍ୱାରା ପୂର୍ଣ୍ଣାଙ୍କ ବିଭାଜନ କରି, 1 ଦିଏ
`2 + 8 * 1`	5, 4 ଦ୍ୱାରା ଗୁଣନ ହୋଇ, 20 ଦିଏ
`2 + 8`	20 କୁ 10 ଦ୍ୱାରା ପୂର୍ଣ୍ଣାଙ୍କ ବିଭାଜନ ହୋଇ, 2 ଦିଏ
`10`	8, 1 ଦ୍ୱାରା ଗୁଣନ ହୋଇ, 8 ଦିଏ
	8 କୁ 2 ରେ ମିଶାଯାଏ, 10 ଦିଏ, ଯାହା ପରିପ୍ରକାଶର ଅନ୍ତିମ ମୂଲ୍ୟ ଅଟେ ।

2.3.3 ପ୍ରକାର ରୂପାନ୍ତରଣ (Type Conversion)

ଯେତେବେଳେ ବିଭିନ୍ନ ପ୍ରକାରର ମୂଲ୍ୟ ଗୋଟିଏ ପରିପ୍ରକାଶରେ ମିଳିତ ହୁଏ, ସେତେବେଳେ ସେଗୁଡିକ କୌଣସି ଅପରେସନ୍ କରିବା ପୂର୍ବରୁ ସମାନ ପ୍ରକାରରେ ରୂପାନ୍ତରିତ ହୁଏ । ମୂଲ୍ୟକୁ ଗୋଟିଏ ପ୍ରକାରରୁ ଅନ୍ୟ ପ୍ରକାରକୁ ରୂପାନ୍ତର କରିବାର ଏହି ପ୍ରକ୍ରିୟାକୁ ପ୍ରକାର ରୂପାନ୍ତରଣ କୁହାଯାଏ ।

C ଭାଷା ନିମ୍ନଲିଖିତ ଦୁଇଟି ରୂପରେ ପ୍ରକାର ରୂପାନ୍ତରଣର ସୁବିଧା ପ୍ରଦାନ କରିଥାଏ:

- ଟାଇପ୍‌ର ଅପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ
- ଟାଇପ୍‌ର ପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ

2.3.3.1 ଟାଇପ୍‌ର ଅପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ (Implicit Type Conversion)

ପ୍ରୋଗ୍ରାମରଙ୍କ ହସ୍ତକ୍ଷେପ ବିନା କମ୍ପାଇଲର୍ ଦ୍ୱାରା ଏକ ଅପ୍ରତ୍ୟକ୍ଷ ପ୍ରକାର ରୂପାନ୍ତରଣ ସ୍ୱତଃସ୍ଫୂର୍ତ୍ତ ଭାବେ କରାଯାଏ । ପରିପ୍ରକାଶ ମିଶ୍ରିତ ମୋଡରେ ଥିବା ବେଳେ ଏହା କରାଯାଏ, ଯାହା ଦ୍ୱାରା ସୂଚନା ନଷ୍ଟ ହେବ ନାହିଁ ।

ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ ବେଳେ ରୂପାନ୍ତର (Conversion in Evaluation of Expressions)

ଯେତେବେଳେ ଅଲଗା ଟାଇପ୍‌ର ଅପେରାଣ୍ଡ ସହିତ ଗୋଟିଏ ଅପରେସନ୍ କରିବାକୁ ପଡିଥାଏ, ସେଠାରେ ଛୋଟ ଆକାରର ଅପେରାଣ୍ଡ୍‌କୁ ଅନ୍ୟ ପ୍ରକାରକୁ ରୂପାନ୍ତର କରାଯାଏ ଯାହା ବଡ଼ ଆକାରର ଅପେରାଣ୍ଡ ସହିତ ମେଳ ଖାଏ । ଥରେ ରାଙ୍କ୍ (ranks) ଗୁଡିକର ରୂପାନ୍ତରଣ ସରିଲା ପରେ, ଅପରେସନ୍ କରାଯାଏ ଏବଂ ଅପରେସନ୍‌ର ଫଳାଫଳ ବଡ଼ ଟାଇପ୍‌ର ହୋଇଥାଏ ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଧରାଯାଉ ନିମ୍ନ ଲିଖିତ ପରିପ୍ରକାଶରେ ଚଳ x ଏବଂ y ର ଟାଇପ୍ float ଏବଂ ଚଳ i ଏବଂ j ର ଟାଇପ୍ *int:*

```
x / i + y * j
```

ପରିପ୍ରକାଶ "x/i" ର ମୂଲ୍ୟାଙ୍କନ କରିବା ସମୟରେ ଇଣ୍ଟିଜର୍ ଅପେରାଣ୍ଡ i ଟି float ଟାଇପ୍‌କୁ ରୂପାନ୍ତରିତ ହେବ ଏବଂ ତା'ପରେ i, ଅପେରାଣ୍ଡ x ଦ୍ୱାରା ବିଭାଜିତ ହେବ ଏବଂ ପରିପ୍ରକାଶର ଫଳାଫଳ float ଟାଇପ୍‌ର ମୂଲ୍ୟ ହେବ ।

ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତି ମଧ୍ୟରେ ରୂପାନ୍ତରଣ (Conversions in Assignment Statements)

ଏକ ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତିରେ ଗୋଟିଏ ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍ (=) ଏବଂ ଦୁଇଟି ଅପେରାଣ୍ଡ ଅନ୍ତର୍ଭୁକ୍ତ ଥାଏ। ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍‌ର ବାମ ପାର୍ଶ୍ୱରେ ଥିବା ଅପେରାଣ୍ଡ୍ ଏକ ଚଳ ହୋଇଥିବାବେଳେ ଡାହାଣ ପାର୍ଶ୍ୱରେ ଥିବା ଅପେରାଣ୍ଡ୍ ଏକ ଧ୍ରୁବକ ମୂଲ୍ୟ/ ଧ୍ରୁବକ, ଚଳ କିମ୍ବା ପରିପ୍ରକାଶ ହୋଇପାରେ।

ଅପେରାଣ୍ଡ୍‌ର ଆକାରର ପାର୍ଥକ୍ୟ ଅନୁଯାୟୀ, C ସିଷ୍ଟମ୍ ଡାହାଣ ଅପେରାଣ୍ଡ୍‌କୁ ଉନ୍ନତ କିମ୍ବା ଅବନତ କରିବାକୁ ଚେଷ୍ଟା କରେ ଯାହାଦ୍ୱାରା ଏହାର ଆକାର ବାମ ଅପେରାଣ୍ଡ୍ (ଚଳ) ସହିତ ମେଳ ଖାଏ। ଯଦି ଡାହାଣ ଅପେରାଣ୍ଡ୍‌ର ଆକାର ଛୋଟ ଥାଏ ତେବେ ଉନ୍ନତ ହୋଇଥାଏ; ସେହିପରି ଯଦି ଡାହାଣ ଅପେରାଣ୍ଡ୍‌ର ଆକାର ଅଧିକ ଥାଏ ତେବେ ଅବନତ ହୁଏ।

2.3.3.2 ଟାଇପ୍‌ର ପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ (Explicit Type Conversion)

ଏକ ଟାଇପ୍‌ର ପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ ବ୍ୟବହାରକାରୀଦ୍ୱାରା କରାଯାଇଥାଏ ଯାହା ଏକ ଅପେରାଣ୍ଡ୍ କିମ୍ବା ପରିପ୍ରକାଶକୁ ନିର୍ଦ୍ଦିଷ୍ଟ ଟାଇପ୍‌ର ହେବାକୁ ବାଧ୍ୟ କରେ। ଟାଇପ୍‌ର ପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ ପ୍ରକ୍ରିୟାକୁ ଟାଇପ୍ କାଷ୍ଟିଂ(type casting) ମଧ୍ୟ କୁହାଯାଏ।

ଟାଇପ୍‌ର ପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ କିମ୍ବା ଟାଇପ୍ କାଷ୍ଟିଂ ପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
(type)operand
```

ଏଠାରେ ଚନ୍ଦ୍ର ବନ୍ଧନୀ ମଧ୍ୟରେ ଥିବା ଟାଇପ୍‌କୁ କାଷ୍ଟ ଅପରେଟର କୁହାଯାଏ। ଏହାର ପ୍ରାଥମିକତା 2। ସିଣ୍ଟାକ୍ସରେ ଦର୍ଶାଯାଇଥିବା ଭଳି, ଗୋଟିଏ ପ୍ରକାରରୁ ଅନ୍ୟ ପ୍ରକାରକୁ ଡାଟା କାଷ୍ଟ କରିବାକୁ, ଆମେ ରୂପାନ୍ତର କରିବାକୁ ଚାହୁଁଥିବା ମୂଲ୍ୟ ପୂର୍ବରୁ ଏକ ନୂତନ ଟାଇପ୍ ଚନ୍ଦ୍ର ବନ୍ଧନୀ ମଧ୍ୟରେ ଦର୍ଶାଇ ଥାଉ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଚଳ x କୁ int ପ୍ରକାରରୁ float ପ୍ରକାରରେ ରୂପାନ୍ତରିତ କରିବାକୁ ଆମେ ଏହି ପରିପ୍ରକାଶକୁ ଲେଖିବା।

```
(float)x
```

ଏଠାରେ ଧ୍ୟାନ ଦେବା ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଯେ ଅନ୍ୟ ୟୁନାରି ଅପରେସନ୍ ପରି ଏହି ଅପରେସନରେ ମଧ୍ୟ, x ରେ ଗଚ୍ଛିତ ଥିବା ମୂଲ୍ୟ ଏପର୍ଯ୍ୟନ୍ତ int ପ୍ରକାରର ଅଟେ, କିନ୍ତୁ ପରିପ୍ରକାଶର ମୂଲ୍ୟ float ଟାଇପ୍‌କୁ ଉନ୍ନତ ହୋଇଥାଏ।

କାଷ୍ଟ ଅପରେଟର୍‌ର ଗୋଟିଏ ବ୍ୟବହାର ହେଉଛି ଏହା ସୁନିଶ୍ଚିତ କରେ ଯେ ଇଣ୍ଟିଜର୍ ଅପେରାଣ୍ଡଗୁଡ଼ିକର ଏକ ବିଭାଜନର ପରିଣାମ ବାସ୍ତବ ସଂଖ୍ୟା ହୁଏ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଯଦି ଆମେ ହାରାହାରି ଗଣନା କରିବାକୁ ଚାହୁଁ, ଯେଉଁଥିରେ ଭଗ୍ନାଂଶଟି ଏକ କାଷ୍ଟ ଅପରେଟର ବିନା ଥାଏ, ତେବେ ଇଣ୍ଟିଜର୍ ବିଭାଜନ ହେତୁ ପରିଣାମ ଏକ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ହେବ।

ବାସ୍ତବ ସଂଖ୍ୟା ପ୍ରକାରର ଫଳାଫଳକୁ ବାଧ୍ୟ କରିବାକୁ, ଆମେ ହାରାହାରି ଗଣନା କରିବାକୁ ନିମ୍ନ ବିବୃତ୍ତି ଲେଖିପାରିବା

```
average = (float)total / n;
```

ଏହି ବିବୃତ୍ତିରେ, ଏଠାରେ total, float କୁ ପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତର ହୋଇଛି, ଏବଂ ତା'ପରେ n, float କୁ ଅପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତର ହୁଏ। ଚାହୁଁଥିବା ଅନୁଯାୟୀ, ବର୍ତ୍ତମାନ ବାସ୍ତବ ସଂଖ୍ୟା ବିଭାଜନ ହେବ ଏବଂ ଫଳାଫଳ ମଧ୍ୟ ଏକ ବାସ୍ତବ ସଂଖ୍ୟା ହେବ, ଏବଂ ହାରାହାରି (Average) କୁ ନ୍ୟସ୍ତ ହେବ।

2.4 ପ୍ରାଥମିକତା ଏବଂ ସହଯୋଗିତା (PRECEDENCE AND ASSOCIATIVITY)

ପ୍ରାଥମିକତାର ବ୍ୟବହାର ଏକ ପରିପ୍ରକାଶରେ ବିଭିନ୍ନ ଅପରେଟରଗୁଡ଼ିକରର ହେବାକୁ ଥିବା ମୂଲ୍ୟାଙ୍କନର କ୍ରମ ନିର୍ଦ୍ଧାରଣ କରିବାକୁ ବ୍ୟବହୃତ ହୋଇଥାଏ।

ସହଯୋଗିତାର ବ୍ୟବହାର ଏକ ପରିପ୍ରକାଶରେ ସମାନ ପ୍ରାଥମିକତା ବିଭିନ୍ନ ଅପରେଟରଗୁଡ଼ିକରର ମୂଲ୍ୟାଙ୍କନର କ୍ରମ ନିର୍ଦ୍ଧାରଣ କରିବାକୁ ବ୍ୟବହୃତ ହୁଏ।

କେଉଁ କ୍ରମରେ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ କରାଯିବ ତାହା ନିର୍ଦ୍ଧାରଣ କରିବା ପାଇଁ ସହଯୋଗିତା ପୂର୍ବରୁ ପ୍ରାଥମିକତାର ପ୍ରୟୋଗ କରାଯାଏ। ଯଦି ଆବଶ୍ୟକ ହୁଏ ତେବେ ପରେ ସହଯୋଗିତାର ପ୍ରୟୋଗ କରାଯାଏ।

ପ୍ରାଥମିକତା (Precedence)

ଗଣିତ ବିଷୟରେ ପ୍ରାଥମିକତାର ଧାରଣା ଭଲ ଭାବରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଛି। ତୁମେ ନିଶ୍ଚିତ ଭାବରେ ନିୟମ BODMAS(ବରହଗୁମିଫେ) ବିଷୟରେ ଅବଗତ ଥିବ – ବନ୍ଧନୀ(Brackets), ର(Of), ହରଣ(Division), ଗୁଣନ (Multiplication), ମିଶାଣ (Addition), ଏବଂ ଫେଡାଣ (Subtraction)। ବୀଜଗଣିତରେ, ଯୋଗ ଏବଂ ବିୟୋଗ ପୂର୍ବରୁ ବିଭାଜନ ଏବଂ ଗୁଣନ କରାଯାଏ।

ନିମ୍ନଲିଖିତ ପ୍ରାଥମିକତାର ଏକ ସରଳ ଉଦାହରଣ:

```
5 + 3 * 4
```

ଏହି ପରିପ୍ରକାଶ ଗୋଟିଏ ଯୋଗ ଏବଂ ଗୋଟିଏ ଗୁଣନ ଅପରେଟରକୁ ନେଇ ଗଠିତ। ଯେପରି ତୁମେ ଜାଣିଛ, ବୀଜଗଣିତରେ ଗୁଣନର ଯୋଗ ଅପେକ୍ଷା ଅଧିକ ପ୍ରାଥମିକତା ଥାଏ, ଅର୍ଥାତ ଯୋଗ କରିବା ପୂର୍ବରୁ ଗୁଣନ କରାଯାଏ।

ତେଣୁ, ପରିପ୍ରକାଶକୁ ଏହିପରି ମୂଲ୍ୟାଙ୍କନ କରାଯିବ

```
( 5 + ( 3 * 4 ) ) → ( 5 + 12 ) → 17
```

ସମଗ୍ର ପରିପ୍ରକାଶର ମୂଲ୍ୟ 17 ହେବ।

ସହଯୋଗିତା (Associativity)

ଯେତେବେଳେ ସମାନ ପ୍ରାଥମିକତାର ଏକାଧିକ ଅପରେଟର ଗୋଟିଏ ପରିପ୍ରକାଶରେ ବ୍ୟବହୃତ ହୁଏ ସେତେବେଳେ ସହଯୋଗିତା ପ୍ରୟୋଗ କରାଯାଏ। ସହଯୋଗିତା ବାମରୁ ଡାହାଣ କିମ୍ବା ଡାହାଣରୁ ବାମକୁ ହୋଇପାରେ। ବାମ-ଡାହାଣ ସହଯୋଗିତା ବାମ ପାର୍ଶ୍ୱରୁ ଆରମ୍ଭ କରି ଏବଂ ଡାହାଣକୁ ଯାଇ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ କରେ ଯେତେବେଳେ ଡାହାଣ-ବାମ ସହଯୋଗିତା ଡାହାଣରୁ ଆରମ୍ଭ କରି ବାମକୁ ଯାଇ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ କରେ।

ନିମ୍ନଲିଖିତଟି ହେଉଛି ସହଯୋଗିତାର ଏକ ସରଳ ଉଦାହରଣ:

```
2 + 5 + 7
```

ଏହି ପରିପ୍ରକାଶରେ ଦୁଇଟି ଯୋଗ ଅପରେଟର ରହିଛି। ଏଥିରେ ଉପ ପରିପ୍ରକାଶଗୁଡିକ କିପରି ଏକତ୍ର ଗୋଷ୍ଠୀଭୁକ୍ତ ହେବେ ତାହା ସହଯୋଗିତା ନିର୍ଦ୍ଧାରଣ କରେ। ଯେହେତୁ ଯୋଗ ଅପରେଟରର ସହଯୋଗିତା ବାମରୁ ଡାହାଣକୁ ଅଛି ତେଣୁ ପରିପ୍ରକାଶକୁ ନିମ୍ନମତେ ଗୋଷ୍ଠିଭୁକ୍ତ କରାଯିବ

```
( ( 2 + 5 ) + 7 ) → ( 7 + 7 ) → 14
```

ସମଗ୍ର ପରିପ୍ରକାଶର ମୂଲ୍ୟ 14 ହେବ।

ଟେବୁଲ୍ 2.10: ଅପରେଟରମାନଙ୍କର ପ୍ରାଥମିକତା ଏବଂ ସହଯୋଗିତା

ଅପରେଟର	ବର୍ଣ୍ଣନା	ପ୍ରାଥମିକତା ସ୍ତର (Precedence Level) ରାଙ୍କ୍ (Rank)	ସହଯୋଗିତା (Associativity)
() [] –> .	ଫଙ୍କସନ୍ କଲ୍ ଆରେ ଉପାଦାନ ରେଫରେନ୍ସ ଷ୍ଟ୍ରକ୍ଚର ଅପରେଟର୍ର ବ୍ୟବହାର ଷ୍ଟ୍ରକ୍ଚରକୁ ଏକ ପଏଁଟର୍ ସହିତ ଷ୍ଟ୍ରକ୍ଚର ଅପରେଟର୍ର ବ୍ୟବହାର ଷ୍ଟ୍ରକ୍ଚର ଚଳ ସହିତ	1	ବାମରୁ ଡାହାଣ
~ ~ ++ — + – * & (type) sigeof	ଲଜିକାଲ୍ NOT ଅପରେଟର୍ 1'ର ଅନୁପୂରକ ବୃଦ୍ଧି ହ୍ରାସ ୟୁନାରି ଯୁକ୍ତ ୟୁନାରି ବିଯୁକ୍ତ ପଏଁଟର୍ ରେଫରେନ୍ସ (ଇନ୍ଡାଇରେକ୍ସନ୍) ଠିକଣା ଟାଇପ୍ କାଷ୍ଟ ଅପେରାଣ୍ଡର ଆକାର	2	ଡାହାଣରୁ ବାମ
* / %	ଗୁଣନ ବିଭାଜନ ମୋଡୁଲସ୍ (ଅବଶିଷ୍ଟ)	3	ବାମରୁ ଡାହାଣ
+ –	ଯୋଗ ବିଯୋଗ	4	ବାମରୁ ଡାହାଣ
<< >>	ବାମ ସିଫ୍ଟ ଡାହାଣ ସିଫ୍ଟ	5	ବାମରୁ ଡାହାଣ

ଅପରେଟର	ବର୍ଣ୍ଣନା	ପ୍ରାଥମିକତା ସ୍ତର (Precedence Level) ରାଙ୍କ୍ (Rank)	ସହଯୋଗିତା (Associativity)
< <= > >=	ଠାରୁ କମ୍ କମ୍ କିମ୍ବା ସମାନ ଅଧିକ ଅଧିକ କିମ୍ବା ସମାନ	6	ବାମରୁ ଡାହାଣ
== !=	ସମାନ ସମାନ ନୁହେଁ	7	ବାମରୁ ଡାହାଣ
&	ବିଟ୍‌ୱାଇଜ୍ ଏବଂ	8	ବାମରୁ ଡାହାଣ
^	ବିଟ୍‌ୱାଇଜ୍ ଏକ୍ସକ୍ଲୁସିଭ୍ କିମ୍ବା	9	ବାମରୁ ଡାହାଣ
|	ବିଟ୍‌ୱାଇଜ୍ ଅନ୍ତର୍ଭୁକ୍ତ କିମ୍ବା	10	ବାମରୁ ଡାହାଣ
&&	ଲଜିକାଲ୍ ଏବଂ	11	ବାମରୁ ଡାହାଣ
||	ଲଜିକାଲ୍ କିମ୍ବା	12	ବାମରୁ ଡାହାଣ
? :	ଅବସ୍ଥା (ଟେର୍ନାରୀ)	13	ଡାହାଣରୁ ବାମ
= += –= *= /= %= &= ^= ~= <<= >>=	ଆସାଇନମେଣ୍ଟ ଅପରେଟର	14	ଡାହାଣରୁ ବାମ

2.5 ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ (LIBRARY FUNCTIONS)

ପ୍ରାୟତଃ ବାରମ୍ବାର ବ୍ୟବହୃତ ହେଉଥିବା ଅନେକ ଗାଣିତିକ ଫଙ୍କସନ୍‌ର ମୂଲ୍ୟାଙ୍କନ କରିବା ଆବଶ୍ୟକ ହୁଏ, ଯେପରିକି ବର୍ଗମୂଳ (square root), ଲଗାରିଦମ୍ (logarithms), ଘାତାଙ୍କୀ (exponentials), ତ୍ରିକୋଣମିତିକ (trigonometric) ଫଙ୍କସନ୍ ଇତ୍ୟାଦି । ପ୍ରତ୍ୟେକ ବ୍ୟବହାରକାରୀ ଏହି ଫଙ୍କସନ୍‌ଗୁଡିକ ଗଣନା କରିବାକୁ ସେମାନଙ୍କର ନିଜସ୍ୱ କୋଡ୍ ଲେଖିବା ପରିବର୍ତ୍ତେ, C କମ୍ପାଇଲର୍, ସୁବିଧାରେ ବ୍ୟବହାର କରି ହେଲା ସେଗୁଡ଼ିକୁ ବିଲ୍ଟ-ଇନ୍ ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ ଭାବରେ ପ୍ରଦାନ କରେ ।

ବ୍ୟବହାରକାରୀଙ୍କୁ ହେଡର୍ ଫାଇଲ୍ math.h କୁ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଅନ୍ତର୍ଭୁକ୍ତ କରିବାକୁ ପଡ଼ିବ। ଏହି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡିକ ବ୍ୟତୀତ, ଅନ୍ୟ ଅନେକ ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍ ଅଛି ଯାହା ବିଶେଷ କାର୍ଯ୍ୟଗୁଡିକ ସମ୍ପାଦନ କରେ, ଏବଂ ଏଗୁଡ଼ିକ ବିଭିନ୍ନ ହେଡର୍ ଫାଇଲଗୁଡିକରେ ବ୍ୟାଖ୍ୟା ହୋଇଛି। ଉଦାହରଣ ସ୍ୱରୂପ, ଷ୍ଟ୍ରିଙ୍ଗ୍ ନିୟନ୍ତ୍ରଣପାଇଁ ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡିକ string.h ହେଡର୍ ଫାଇଲ୍‌ରେ ବ୍ୟାଖ୍ୟା ହୋଇଛି।

କିଛି ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍‌ର ଚର ଫଙ୍କ୍‌ସନ୍‌ର ସଂଜ୍ଞା ଦ୍ୱାରା ସୀମିତ। ଉଦାହରଣ ସ୍ୱରୂପ, ଏକ ଋଣାତ୍ମକ ସଂଖ୍ୟାର ଲଗାରିଦମ୍ ଗାଣିତିକ ଭାବରେ ଅବର୍ଣ୍ଣିତ। ସେହିଭଳି, ଏକ ଋଣାତ୍ମକ ସଂଖ୍ୟାର ବର୍ଗମୂଳ ଗାଣିତିକ ଭାବରେ ଅବର୍ଣ୍ଣିତ, ଏବଂ ତେଣୁ, ଅନୁମୋଦିତ ନୁହେଁ।

ଟେବୁଲ୍ 2.11: ସାଧାରଣ ଭାବେ ବ୍ୟବହୃତ କିଛି ଗାଣିତିକ ଫଙ୍କ୍‌ସନ୍

ଫଙ୍କ୍‌ସନ୍	ବର୍ଣ୍ଣନା
`pow (x,y)`	x ର ଘାତ y, ଅର୍ଥାତ x^y ଫେରସ୍ତ କରେ।
`pow10(p)`	10^p ମୂଲ୍ୟ ଫେରସ୍ତ କରେ।
`exp (x)`	*e* ଘାତ *x*, ଅର୍ଥାତ, e^x ଫେରସ୍ତ କରେ।
`log (x)`	ଲଗ୍ *(x)* ମୂଲ୍ୟ ଫେରସ୍ତ କରେ ।
`log10 (x0`	ଲଗ୍ *10(x)* ମୂଲ୍ୟ ଫେରସ୍ତ କରେ
`sqrt (x)`	$\sqrt{x}$ ମୂଲ୍ୟ ଫେରସ୍ତ କରେ।
`sin (x)`	sin(x) ମୂଲ୍ୟ ଫେରସ୍ତ କରେ , ଯେଉଁଠାରେ କୋଣ x ରେଡିଆନରେ ଅଛି।
`ceil (x)`	ସବୁଠାରୁ ଛୋଟ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ଯାହା x ଠାରୁ ଅଧିକ କିମ୍ବା ସମାନ ହୁଏ।
`floor (x)`	ସବୁଠାରୁ ବଡ଼ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ଯାହା x ଠାରୁ କମ୍ କିମ୍ବା ସମାନ ହୁଏ।

ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଏହି ବିଭାଗରେ, ଆସ କିଛି ପରିସ୍ଥିତି ଏବଂ/କିମ୍ବା ସମସ୍ୟାଗୁଡିକ ବିଚାର କରିବା ଯାହା ଆମକୁ ଅପରେଟର୍ ଏବଂ ପରିପ୍ରକାଶ ବ୍ୟବହାର କରିବାର ସୁଯୋଗ ଦେବ। ଏହା ପୂର୍ବରୁ ଆସନ୍ତୁ ଡ୍ରାଇ ରନ୍ (dry run) ବିଷୟରେ ଜାଣିବା।

ଏକ ଡ୍ରାଇ ରନ୍‌ର ମୂଳତଃ ଅର୍ଥ ହେଉଛି ଆମେ ଏକ କମ୍ପ୍ୟୁଟରକୁ ଅନୁକରଣ କରି କାଗଜ ପେନସିଲ୍‌ରେ ବିବୃତ୍ତିଗୁଡ଼ିକର କାର୍ଯ୍ୟକାରିତା ପର୍ଯ୍ୟବେକ୍ଷଣ କରିବୁ। ଏହି ସମୟରେ, ଆମେ ଚଳଗୁଡିକର ମୂଲ୍ୟ ଲେଖି ରଖିବା କାରଣ ଚାଳନ ସମୟରେ ମୂଲ୍ୟ ପରିବର୍ତ୍ତନ ହୁଏ। ଯଦି ଏକ ଚଳର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇନଥାଏ, ତେବେ ଏହାର ମୂଲ୍ୟକୁ ବର୍ଣ୍ଣ **'?'** ସହିତ ଦର୍ଶାଇବାକୁ ହୁଏ।

ଉଦାହରଣରେ ଥିବା ମୁଖ୍ୟ ଧାରଣାଗୁଡ଼ିକ ବୁଝିବା ଆବଶ୍ୟକ । ତୁମେ ପ୍ରୋଗ୍ରାମ୍ ବା ପ୍ରୋଗ୍ରାମ୍ କୋଡ୍ଖଣ୍ଡର ଡ୍ରାଇ ରନ୍ ସେଠାରେ ଦେଖିପାରିବ ।

ଉଦାହରଣ 2.1: ତୃତୀୟ ଚଳ 't' ବ୍ୟବହାର କରି ଦୁଇଟି ଚଳ 'a' ଏବଂ 'b'ର ମୂଲ୍ୟ ଅଦଳବଦଳ କର ।

ସମାଧାନ:

```
int a=10,b=20,t;                /* 1 */
t = a;                          /* 2 */
a = b;                          /* 3 */
b = t;                          /* 4 */
```

ଡ୍ରାଇ ରନ୍

ବିବୃତ୍ତି ସଂଖ୍ୟା	*a*	*b*	*t*
1	10	20	?
2	10	20	10
3	20	20	10
4	10	20	10

କୋଡ୍‌ର ଶେଷ ତିନୋଟି ଧାଡିକୁ C ରେ ନିମ୍ନମତେ ମଧ୍ୟ ଲେଖାଯାଇପାରିବ ।

```
t = a, a = b, b = t;
```

ଉଦାହରଣ 2.2: ତୃତୀୟ ଚଳ ବ୍ୟବହାର ନ କରି ଏବଂ କେବଳ ଗାଣିତିକ ଯୋଗ (+) ଏବଂ ବିୟୋଗ (-) ବ୍ୟବହାର କରି ଦୁଇଟି ଚଳ 'a' ଏବଂ 'b' ର ମୂଲ୍ୟ ଅଦଳବଦଳ କର ।

ସମାଧାନ:

```
int a=10,b=20;                  /* 1 */
a = a + b;                      /* 2 */
b = a - b;                      /* 3 */
a = a - b;                      /* 4 */
```

ଡ୍ରାଇ ରନ୍

ବିବୃତ୍ତି ସଂଖ୍ୟା	*a*	*b*
1	10	20
2	30	20
3	30	10
4	20	10

ଏହି କାର୍ଯ୍ୟକୁ ନିମ୍ନପ୍ରଦତ୍ତ ଗୋଟିଏ ବିବୃତ୍ତି ଉପଯୋଗ କରି ମଧ୍ୟ କାର୍ଯ୍ୟାନ୍ୱିତ କରାଯାଇପାରିବ:

```
a = ( a + b ) -  ( b = a );
```

ଉଦାହରଣ 2.3: ତୃତୀୟ ଚଳର ବ୍ୟବହାର ନ କରି ଏବଂ କେବଳ ଗାଣିତିକ ଗୁଣନ (*) ଏବଂ ଡିଭିଜନ୍ (/) ଅପରେସନ୍ ବ୍ୟବହାର କରି ଦୁଇଟି ଚଳ 'a' ଏବଂ 'b' ର ମୂଲ୍ୟ ଅଦଳବଦଳ କର ।

ସମାଧାନ:

```
int a=10,b=20;     /* 1 */
a = a * b;             /* 2 */
b = a /  b;            /* 3 */
a = a / b;             /* 4 */
```

ଡ୍ରାଇ ରନ୍

ବିବୃତ୍ତି ସଂଖ୍ୟା	*a*	*b*
1	10	20
2	200	20
3	200	10
4	20	10

ଏହି କାର୍ଯ୍ୟକୁ ନିମ୍ନପ୍ରଦତ୍ତ ଗୋଟିଏ ବିବୃତ୍ତି ଉପଯୋଗ କରି ମଧ୍ୟ କାର୍ଯ୍ୟାନ୍ୱିତ କରାଯାଇପାରିବ:

```
a = ( a * b ) /  ( b = a );
```

ଉଦାହରଣ 2.4: ତିନୋଟି ସଂଖ୍ୟା *a, b,* ଏବଂ *c* ମଧ୍ୟରୁ ସର୍ବବୃହତଟିକୁ ଖୋଜିବା ପାଇଁ

ସମାଧାନ:

```
int a=10,b=30,c=25, big;                              /* 1 */

big = (a>b)?((a>c)?a:c)):((b>c)?b:c));            /* 2 */
```

ଏଠାରେ, ଯଦି *a, b* ଠାରୁ ଅଧିକ ହୁଏ ତେବେ ପରିପ୍ରକାଶ *((a>c)?a:c)* ର ମୂଲ୍ୟାଙ୍କନ କରାଯିବ; ଅନ୍ୟଥା ପରିପ୍ରକାଶ *(b>c)?b:c)* ର ମୂଲ୍ୟାଙ୍କନ କରାଯିବ ।

ପରିପ୍ରକାଶ *((a>c)?a:c)* ଟି *a* ଫେରସ୍ତ କରିବ ଯଦି *a, c* ଠାରୁ ଅଧିକ ହୁଏ ଅନ୍ୟଥା *c* ଫେରସ୍ତ କରିବ ।

ପରିପ୍ରକାଶ *((b>c)?b:c)* ଟି *b* ଫେରସ୍ତ କରିବ ଯଦି *b , c* ଠାରୁ ଅଧିକ ହୁଏ ଅନ୍ୟଥା *c* ଫେରସ୍ତ କରିବ ।

ଉଦାହରଣ 2.5: *n* ରେ ଗଛିତ ଥିବା ଏକ 4 ଅଙ୍କ ବିଶିଷ୍ଟ ସଂଖ୍ୟାର ଶେଷ ଡାହାଣ ଅଙ୍କ ଏବଂ ଶେଷ ବାମ ଅଙ୍କର ସମଷ୍ଟି ପାଇବା ପାଇଁ ଉପଯୁକ୍ତ ବିବୃତ୍ତିଗୁଡ଼ିକ ଲେଖ ।

ସମାଧାନ:

```
left_most_digit = n / 1000;
right_most_digit = n % 1000;
sum = left_most_digit + right_most_digit;
```

ଉଦାହରଣ 2.6: ସେଲସିୟସ୍ ସ୍କେଲରେ ତାପମାତ୍ରାକୁ ଫାରେନହାଇଟ୍ ସ୍କେଲରେ ପରିଣତ କରିବାକୁ ଉପଯୁକ୍ତ ବିବୃତ୍ତିଗୁଡ଼ିକ ଲେଖ ।

ସମାଧାନ:

ସେଲସିୟସ୍ ସ୍କେଲ୍ ଏବଂ ଫାରେନହାଇଟ୍ ସ୍କେଲ୍‌ରେ ତାପମାତ୍ରା ମଧ୍ୟରେ ସମ୍ପର୍କ ହେଉଛି

$$\frac{C}{100} = \frac{F-32}{180} \qquad F-32 = \frac{180}{100}C \qquad \Rightarrow \qquad F = 1.8C + 32$$

```
float c, f;

f = 1.8 * C + 32;
```

ୟୁନିଟ୍ ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଯାହା ଶିଖିଛୁ

- ଅପରେଟର୍‌ଗୁଡ଼ିକ ଏକ ଭାଷାର କ୍ରିୟା ଯାହା ବ୍ୟବହାରକାରୀଙ୍କୁ ମୂଲ୍ୟଗୁଡ଼ିକୁ ଗଣନା କରିବାକୁ ଦିଏ ।
- ଅପରେଟର୍ ୟୁନାରି କିମ୍ବା ବାଇନାରି ହୋଇପାରେ
- C ଭାଷା ହେଉଛି ପ୍ରଥମ ଭାଷା ଯେଉଁଠାରେ ଅପରେଟର୍‌ଗୁଡ଼ିକ ସହିତ ସମୃଦ୍ଧ ।
- ଅପରେଟର୍‌ଗୁଡ଼ିକ ମୁଖ୍ୟତଃ ଗାଣିତିକ ଅପରେଟର, ସମ୍ବନ୍ଧୀୟ ଅପରେଟର୍, ଲଜିକାଲ୍ ଅପରେଟର, ବିଟ୍-ୱାଇଜ୍ ଅପରେଟର ଏବଂ ସ୍ୱତନ୍ତ୍ର ଅପରେଟର୍‌ରେ ବିଭକ୍ତ ।
- ଗାଣିତିକ ଅପରେସନ୍‌ଗୁଡ଼ିକୁ ସଙ୍ଖ୍ୟାତ୍ମକ ଗଣିତ, ବାସ୍ତବ ସଂଖ୍ୟା ଗଣିତ, ଏବଂ ମିଶ୍ରିତ ମୋଡ୍ ଗଣିତ ଭାବରେ ଶ୍ରେଣୀଭୁକ୍ତ କରାଯାଇପାରିବ ।
- ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍‌ଗୁଡ଼ିକ ଚଳକୁ ମୂଲ୍ୟ ନ୍ୟସ୍ତ କରିବାରେ ବ୍ୟବହୃତ ହୋଇଥାନ୍ତି ।
- ଏକ ପରିପ୍ରକାଶ ହେଉଛି ଏକ ମୂଲ୍ୟ ଗଣନା କରିବା ପାଇଁ ଅପରେଟର ଏବଂ ଅପେରାଣ୍ଡ୍‌ଗୁଡ଼ିକୁ ଏକତ୍ର ନେଇ ଗଠିତ ଏକ ସୂତ୍ର ।
- ପରିପ୍ରକାଶଗୁଡିକ ମୁଖ୍ୟତଃ ଗାଣିତିକ ପରିପ୍ରକାଶ, ସମ୍ବନ୍ଧୀୟ ପରିପ୍ରକାଶ, ଲଜିକାଲ୍ ପରିପ୍ରକାଶ ଏବଂ ସର୍ତ୍ତମୂଳକ ପରିପ୍ରକାଶରେ ବିଭକ୍ତ ହୋଇଥାନ୍ତି ।
- ଗୋଟିଏ ଟାଇପ୍‌ର ମୂଲ୍ୟରୁ ଅନ୍ୟ ଟାଇପ୍‌ର ମୂଲ୍ୟ ରୂପାନ୍ତର କରିବାର ପ୍ରକ୍ରିୟା, ଟାଇପ୍ ରୂପାନ୍ତରଣ ଭାବରେ ଜଣାଶୁଣା ।
- ପ୍ରୋଗ୍ରାମରଙ୍କ ହସ୍ତକ୍ଷେପ ବିନା କମ୍ପାଇଲର୍ ଦ୍ୱାରା ଏକ ଅପ୍ରତ୍ୟକ୍ଷ ପ୍ରକାର ରୂପାନ୍ତରଣ ସ୍ୱତଃସ୍ଫୂର୍ତ୍ତ ଭାବେ ହୋଇଥାଏ ।
- ଏକ ପ୍ରତ୍ୟକ୍ଷ ଟାଇପ୍ ରୂପାନ୍ତରଣ ବ୍ୟବହାରକାରୀ-ବ୍ୟାଖ୍ୟା ଜନିତ ହୋଇଥାଏ ଯାହା ଏକ ଅପେରାଣ୍ଡ୍ କିମ୍ବା ପରିପ୍ରକାଶକୁ ନିର୍ଦ୍ଦିଷ୍ଟ ଟାଇପ୍ ହେବାକୁ ବାଧ୍ୟ କରେ ।
- ପ୍ରତ୍ୟକ୍ଷ ଟାଇପ୍ ରୂପାନ୍ତରଣ ପ୍ରକ୍ରିୟାକୁ ଟାଇପ୍ କାଷ୍ଟିଂ ମଧ୍ୟ କୁହାଯାଏ ।
- ଏକ ପରିପ୍ରକାଶରେ ବିଭିନ୍ନ ଅପରେଟରଗୁଡ଼ିକର ମୂଲ୍ୟାଙ୍କନର କ୍ରମ ନିର୍ଦ୍ଧାରଣ କରିବାକୁ ପ୍ରାଥମିକତାର ବ୍ୟବହାର ହୁଏ ।
- ଏକ ପରିପ୍ରକାଶରେ ସମାନ ପ୍ରାଥମିକତା ଥିବା ବିଭିନ୍ନ ଅପରେଟରଗୁଡ଼ିକର ମୂଲ୍ୟାଙ୍କନର କ୍ରମ ନିର୍ଦ୍ଧାରଣ କରିବା ପାଇଁ ସହଯୋଗିତା ବ୍ୟବହୃତ ହୋଇଥାଏ ।
- ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡିକ ବିଭିନ୍ନ ରୁଟିନ୍ ପ୍ରକାର ଅପରେସନ୍‌ର କାର୍ଯ୍ୟାନ୍ୱୟନପାଇଁ ପୂର୍ବ-ଲିଖିତ ଫଙ୍କ୍‌ସନ୍, ଏବଂ ଏହା କମ୍ପାଇଲର୍‌ର ଏକ ଅଂଶ ଭାବରେ ପ୍ରଦାନ କରାଯାଇଥାଏ । ସେଗୁଡିକ ବ୍ୟବହାର କରିବାକୁ, ତୁମକୁ ପ୍ରୋଗ୍ରାମରେ ଉପଯୁକ୍ତ ହେଡର୍ ଫାଇଲ୍ ଅନ୍ତର୍ଭୁକ୍ତ କରିବା ଆବଶ୍ୟକ ।

ଅନୁଶୀଳନୀ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. ଗୋଟିଏ ଅପରେଟର କ'ଣ? ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ବିଭିନ୍ନ ଗାଣିତିକ ଅପରେଟରଗୁଡ଼ିକର ବ୍ୟବହାର ବର୍ଣ୍ଣନା କର।
2. ଗାଣିତିକ ଅପରେଟରମାନଙ୍କ ମଧ୍ୟରେ, ଅପରେଟରମାନଙ୍କର ବ୍ୟାପକ ଶ୍ରେଣୀଗୁଡ଼ିକ କ'ଣ?
3. ମୋଡୁଲସ୍ ଅପରେଟର '%' ଉପରେ କୌଣସି ପ୍ରତିବନ୍ଧକ ଅଛି କି?
4. ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ସର୍ତ୍ତମୂଳକ ଅପରେଟରଗୁଡ଼ିକର କାର୍ଯ୍ୟପ୍ରଣାଳୀକୁ ବର୍ଣ୍ଣନା କର।
5. sizeof ଅପରେଟର୍‌ର କାର୍ଯ୍ୟ ବ୍ୟାଖ୍ୟା କର।
6. ବୀଜଗାଣିତିକ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନକୁ ନିୟନ୍ତ୍ରଣ କରୁଥିବା ନିୟମଗୁଡିକ ବର୍ଣ୍ଣନା କର।
7. ଅପରେଟରଗୁଡ଼ିକର ପ୍ରାଥମିକତାର ଅର୍ଥ ତୁମେ କ'ଣ ବୁଝ? ବିଭିନ୍ନ ଗାଣିତିକ ଅପରେଟରଗୁଡ଼ିକ ମଧ୍ୟରେ ପ୍ରାଥମିକତା କ'ଣ?
8. ସହଯୋଗିତା ଅପରେଟରଗୁଡ଼ିକର ବିଷୟରେ ତୁମେ ମତ କ'ଣ ?
9. ଅଲଗା ପ୍ରକାରର ଅପେରାଣ୍ଡ୍ ଥିବା ପରିପ୍ରକାଶପାଇଁ ପ୍ରଯୁଜ୍ୟ ନିୟମଗୁଡିକୁ ସଂକ୍ଷେପରେ ବର୍ଣ୍ଣନା କର।
10. ଗାଣିତିକ ପରିପ୍ରକାଶରେ ଚନ୍ଦ୍ର ବନ୍ଧନୀର ବ୍ୟବହାର କେବେ କରାଯିବା ଉଚିତ? ଅତି କମ୍‌ରେ ଗୋଟିଏ ଉଦାହରଣ ଦିଅ।
11. ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ପ୍ରକୃତରେ 'C' ଭାଷାର ଏକ ଅଂଶ କି? ବ୍ୟାଖ୍ୟା କର।
12. ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ସର୍ତ୍ତମୂଳକ ଅପରେଟରଗୁଡ଼ିକର କାର୍ଯ୍ୟକୁ ବର୍ଣ୍ଣନା କର।
13. ବୁଲିଆନ୍ False ସହିତ କେଉଁ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାର ମୂଲ୍ୟ ସମାନ?
14. ପରିପ୍ରକାଶ କ'ଣ?
15. ବୃଦ୍ଧି ଏବଂ ହ୍ରାସ ଅପରେଟରଗୁଡ଼ିକର କାର୍ଯ୍ୟକୁ ବ୍ୟାଖ୍ୟା କର ।
16. ପ୍ରକାର/ଟାଇପ୍ ରୂପାନ୍ତରଣ କ'ଣ?
17. ଯେଉଁ ପରିସ୍ଥିତିରେ ଅପ୍ରତ୍ୟକ୍ଷ ପ୍ରକାର ରୂପାନ୍ତରଣ ହୁଏ ସେହି ପରିସ୍ଥିତିକୁ ବର୍ଣ୍ଣନା କର।
18. ପ୍ରତ୍ୟକ୍ଷ ଟାଇପ୍ କାଷ୍ଟିଂ କିପରି କରାଯାଏ? ଉଦାହରଣ ଦିଅ।
19. ତୁମକୁ ଗାଣିତିକ ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍ ବ୍ୟବହାର କରିବାପାଇଁ ଅନ୍ତର୍ଭୁକ୍ତ କରିବାକୁ ଆବଶ୍ୟକ ଥିବା ହେଡର୍ ଫାଇଲ୍‌ର ନାମ ଦିଅ।
20. ନିମ୍ନଲିଖିତ ବୀଜଗାଣିତିକ ପରିପ୍ରକାଶଗୁଡ଼ିକପାଇଁ C ଭାଷାର ପରିପ୍ରକାଶ ଲେଖ:

(a) $a - \frac{a}{a^{-x}} + b$ (b) $1 + \frac{1}{a + \frac{1}{a}} + b$ (c) $x + \frac{y^{-z}}{k} - \sqrt{x}$

(d) $x^3 - \frac{y^7}{-z} + e^x$ (e) $\left(\frac{1-x}{1+x}\right) / \left(\frac{1+y}{1-y}\right)$ (f) $\frac{1}{r} = \frac{1}{r_1} + \frac{1}{r_2}$

(g) $\sqrt{\left(x^2 + y^2 + z^2\right)^3}$ (h) $\frac{-b - \sqrt{b^2 - 4ac}}{2a}$

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଗାଣିତିକ ଅପରେଟର୍ ନୁହେଁ?

   (a) + (b) &

   (c) % (d) *

2. ଅପରେଟରଗୁଡ଼ିକର ପ୍ରାଥମିକତା ନିର୍ଦ୍ଧାରଣ କରେ ଯେ କେଉଁ ଅପରେଟର ____________

   (a) ପ୍ରଥମେ ବ୍ୟବହୃତ ହୁଏ (b) ହେଉଛି ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ

   (c) ସର୍ବାଧିକ ସଂଖ୍ୟା ଉପରେ କାର୍ଯ୍ୟ କରେ (d) ଦ୍ରୁତ ଚାଳନ କରେ

3. ଅପରେଟର୍ % କାହା ସହିତ ପ୍ରୟୋଗ କରାଯାଇପାରିବ

   (a) float ମୂଲ୍ୟ (b) double ମୂଲ୍ୟ

   (c) ପୂର୍ଣ୍ଣାଙ୍କ ମୂଲ୍ୟ (d) ଉପରୋକ୍ତ ସମସ୍ତ

4. ପରିପ୍ରକାଶ x % y କାହା ସହିତ ସମାନ

   (a) (x–(x/y)) (b) (x–(x/y)*y)

   (c) (y–(y/x)) (d) (y–(x/y)*x)

5. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ବିଟ ୱାଇଜ (bit-wise) ଅପରେଟର୍ ନୁହେଁ?

   (a) > (b) >>

   (c) ^ d) &

6. ପରିପ୍ରକାଶର 5 / 6 / 3 + 8 / 3 ର ମୂଲ୍ୟ କ'ଣ ହେବ?

   (a) 4 (b) 2

   (c) 2.333 (ପ୍ରାୟ) (d) ଉପରୋକ୍ତ କୌଣସିଟି ନୁହେଁ

7. ପରିପ୍ରକାଶରେ x / y, ରେ y ମୂଲ୍ୟ କଣ ହେବା ଉଚିତ

   (a) ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା (b) ଅଣ-ଶୂନ୍ୟ

   (c) ଅଣ-ବିଯୁକ୍ତ (d) ଯୁକ୍ତ ସଂଖ୍ୟା

8. ଯଦି x ଏକ int ପ୍ରକାରର ଚଳ, ତେବେ ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଅଯୁଗ୍ମ?

   (a) x << 1 (b) x = x * 2

   (c) x *= 2 (d) x <<= 1

9. C ଭାଷାରେ ନିମ୍ନଲିଖିତ ଉକ୍ତିକୁ ବିଚାର କର

   x = (a > b) ? ((a > c) ? a : c) : (b > c) ? b : c;

   x ର ମୂଲ୍ୟ କ'ଣ ହେବ ଯଦି a = 3, b = – 5, c = 2?

   (a) 2 (b) 3

   (c) – 5 (d) ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ

10. ଯେତେବେଳେ ଏକ ସମ୍ବନ୍ଧୀୟ ପରିପ୍ରକାଶ False ଥାଏ, ତେବେ ଏହାର ମୂଲ୍ୟ _____

(a) 1 (b) 0

(c) ବିଯୁକ୍ତ (d) ଯୁକ୍ତ

11. ଯଦି a = 15 ତେବେ ବିବୃତ୍ତି b = a << 2; ର ଫଳ କଣ ହେବ

(a) 30 (b) 60

(c) 7 (d) ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ

12. ନିମ୍ନଲିଖିତ କୋଡ୍‌ଖଣ୍ଡକୁ ବିଚାର କର

```
int i, j = 10;
i = j++;
```

ଏହି ଖଣ୍ଡର ଶେଷରେ i ଏବଂ j ର ମୂଲ୍ୟ ହେବ ________

(a) 10, 10 b) 10, 11

(c) 11, 10 (d) 11, 11

13. ନିମ୍ନଲିଖିତ କୋଡ୍‌ଖଣ୍ଡକୁ ବିଚାର କର

```
int x = 5.234, y;
y = sizeof(x);
```

y ର ମୂଲ୍ୟ ହେବ ___________

(a) 4 (b) 2

(c) 8 (d) ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି

14. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଅପରେଟରକୁ ଏକ ପୂର୍ଣ୍ଣାଙ୍କ ଅପେରାଣ୍ଡକୁ 2 ଦ୍ୱାରା ବିଭାଜନ କରିବାକୁ ବ୍ୟବହାର କରାଯାଇପାରିବ?

(a) / (b) >>

(c) << (d) ଉଭୟ (a) & (b)

15. ++ ଅପରେଟର୍ ବିଷୟରେ ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଉକ୍ତିଟି ଠିକ୍ ନୁହେଁ?

(a) ଏହା ଏକ ୟୁନାରି ଅପରେଟର।

(b) ଅପେରାଣ୍ଡ ଅପରେଟରଗୁଡ଼ିକର ପୂର୍ବରୁ କିମ୍ବା ପରେ ଆସିପାରେ।

(c) ଏହା ଏକ ପରିପ୍ରକାଶରେ ପ୍ରୟୋଗ କରାଯାଇପାରିବ।

(d) ଏହାର ସହଯୋଗିତା ଡାହାଣରୁ ବାମକୁ ଅଟେ ।

ଉତ୍ତର									
1.	*(b)*	**2.**	*(a)*	**3.**	*(c)*	**4.**	*(b)*	**5.**	*(a)*
6.	*(b)*	**7.**	*(b)*	**8.**	*(d)*	**9.**	*(b)*	**10.**	*(b)*
11.	*(b)*	**12.**	*(b)*	**13.**	*(b)*	**14.**	*(b)*	**15.**	*(c)*

ପ୍ରାକ୍ଟିକାଲ୍ (PRACTICALS)

1. ସରଳ ଏବଂ ଚକ୍ରବୃଦ୍ଧି ସୁଧ ପାଇବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ସମାଧାନ: ସରଳ ଏବଂ ଚକ୍ରବୃଦ୍ଧି ସୁଧ ବାହାର କରିବାର ସୂତ୍ର ହେଉଛି

$$\text{Simple interest} = \frac{p \times r \times t}{100}$$ ଏବଂ

$$\text{Compound interest} = p\left[\left(1+\frac{r}{100}\right)^{t} - 1\right]$$

କେଉଁଠାରେ p ମୂଳ ଧନ, r ବାର୍ଷିକ ସୁଧ ହାର, ଏବଂ t ବର୍ଷ ଜମା ଅବଧି ।

ତାଲିକା 2.1

```
/* program to compute simple and compound interest */
 #include <stdio.h>
 #include <math.h>
 int main()
 {
    float p, r, si, ci;
    int t;
    printf("\nEnter principle amount : ");
    scanf("%f", &p);
    printf("Enter rate of interest : ");
    scanf("%f", &r);
    printf("Enter principle amount : ");
    scanf("%d", &t);
    si = (p*r*t)/100;
    ci = p*(pow(1+r/100,t)-1);
    printf("\nsimple interest = Rs. %.2f", si);
    printf("\ncompound interest = Rs. %.2f", ci);
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter principle amount : 1000
Enter rate of interest : 5
Enter principle amount : 4
Simple interest = Rs. 200.00
Compound interest = Rs. 215.51
```

2. ଏକ ତ୍ରିଭୂଜର କ୍ଷେତ୍ରଫଳ ପାଇବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହାର ତିନୋଟି ପାର୍ଶ୍ୱର ମାପ ଯଥାକ୍ରମେ a, b, ଏବଂ c, ଦିଆଯାଇଛି । a, b, ଏବଂ c, ର ମୂଲ୍ୟ ନିମ୍ନଲିଖିତ ସର୍ତ୍ତଗୁଡ଼ିକୁ ପୂରଣ କରିବା ଆବଶ୍ୟକ ।

$a+b>c$ ଏବଂ $b+c>a$ ଏବଂ $c+a>b$

ସମାଧାନ: ତ୍ରିଭୂଜର କ୍ଷେତ୍ରଫଳ ବାହାର କରିବାର ସୂତ୍ର ହେଉଛି

$$s=\frac{a+b+c}{2} \qquad \text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

ତାଲିକା 2.2

```
/* program to compute area of a triangle whose sides are given */
#include <stdio.h>
#include <math.h>
int main()
{
   float a, b, c, s, area;
   printf("\nEnter value for a : ");
   scanf("%f", &a);
   printf("Enter value for b : ");
   scanf("%f", &b);
   printf("Enter value for c : ");
   scanf("%f", &c);
   s = (a+b+c)/2;
   area = sqrt(s*(s-a)*(s-b)*(s-c));
   printf("\nArea of triangle = %.2f Sq. Units", area);
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter value for a : 5
Enter value for b : 7
Enter value for c : 6
Area of triangle = 14.70 Sq. Units
```

3. ଏକ କୋଠାର 10 ଟି ମହଲା ଅଛି ଯେଉଁଥିରେ ପ୍ରତ୍ୟେକ ମହଲାର ଉଚ୍ଚତା 3 ମିଟର । କୋଠାର ଉପରୁ ଏକ ବଲ୍ ପକାଯାଏ । ପ୍ରତ୍ୟେକ ମହଲାରେ ପହଞ୍ଚିବାପାଇଁ ବଲ୍ ଦ୍ୱାରା ନିଆଯାଇଥିବା ସମୟ ପାଇବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

$$s=ut+\frac{1}{2}at^2$$

ସମାଧାନ: ଯେହେତୁ ପ୍ରତ୍ୟେକ ମହଲାର ଉଚ୍ଚତା 3 ମିଟର, ତେଣୁ, 10 ମହଲା ସହିତ କୋଠାର ଉଚ୍ଚତା (h) 30 ମିଟର।

ଏଠାରେ ଆମେ ଗତିର ସମୀକରଣ ବ୍ୟବହାର କରୁ, $h = \frac{1}{2}gt^2$

ଏଠାରେ s ହେଉଛି h, ଏବଂ a ହେଉଛି g(9.8 m/s2).

ଯେହେତୁ ବଲ୍ ଶୀର୍ଷ (10^{th}) ମହଲାରୁ ଡ୍ରପ୍ ହୋଇଛି, u = 0. ତେଣୁ, ସୂତ୍ର ହେବ

$$\Rightarrow \quad t = \sqrt{\frac{2h}{g}}$$

ତାଲିକା 2.3

```
/* program to compute the time taken by ball to reach
   the ground floor, when it is released from the top floor
 */
 #include <stdio.h>
 #include <math.h>
 int main()
 {
    float h = 30, g = 9.8, t;
    t = sqrt(2*h/g);
    printf("\nTime taken by ball = %.2f seconosQ", t);
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Time taken by ball = 2.47 seconosQ
```

ଅଧିକ ଜାଣିବାପାଇଁ

ବିଜ୍ଞାନ ଏବଂ ଇଞ୍ଜିନିୟରିଂ ସମ୍ବନ୍ଧୀୟ ସମସ୍ୟାଗୁଡ଼ିକପାଇଁ ଗାଣିତିକ ପରିପ୍ରକାଶ ହେଉଛି ପ୍ରୋଗ୍ରାମ୍‌ର ମେରୁଦଣ୍ଡ। ଏହିପରି ସମସ୍ୟାଗୁଡ଼ିକ ଜଟିଳ ଗାଣିତିକ ଏବଂ ବୀଜଗାଣିତିକ ପରିପ୍ରକାଶ ସହିତ ଜଡିତ। ତେଣୁ, ଏହିପରି ପରିପ୍ରକାଶକୁ C ପରିପ୍ରକାଶରେ ପରିଣତ କରିବା ପାଇଁ, ଜଣେ ପ୍ରାଥମିକତା ଜାଣିବା ଉଚିତ ଏବଂ ଅପରେଟରଗୁଡ଼ିକର ସହଯୋଗିତା, ଗଣିତର ପ୍ରକାର, ଏବଂ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ ସମୟରେ କେଉଁ ପ୍ରକାରର ରୂପାନ୍ତର ହୁଏ ତାହା ଜାଣିବା ଉଚିତ।

ଶିକ୍ଷକ ମାନଙ୍କ ଠାରୁ ଆଶା କରାଯାଉଛି ଯେ ପରିପ୍ରକାଶର ଗଠନ ଏବଂ ମୂଲ୍ୟାଙ୍କନ ଏବଂ ଅନ୍ୟାନ୍ୟ ସମ୍ବନ୍ଧୀୟ ପ୍ରସଙ୍ଗ ବିଷୟରେ ଏକ ବୁଝାମଣା ବିକଶିତ କରିବେ।

ଶିକ୍ଷାର୍ଥୀମାନଙ୍କ ସକ୍ରିୟ ଅଂଶଗ୍ରହଣ ସହିତ ବିଭିନ୍ନ ପ୍ରକାରର ପରିପ୍ରକାଶର ଗଠନ ଏବଂ ମୂଲ୍ୟାଙ୍କନ ମଧ୍ୟ ଶିକ୍ଷକ ପ୍ରଦର୍ଶନ କରିବା ଉଚିତ।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

3 ସର୍ତ୍ତମୂଳକ ଶାଖା ଏବଂ ଲୁପ୍

ଏକକ ବିଶେଷତ୍ୱ

ଏହି ଏକକରେ ବିଭିନ୍ନ ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତି ସହିତ ଜଡିତ ବିଷୟଗୁଡ଼ିକ ବିଷୟରେ ଆଲୋଚନା କରାଯାଇଛି । ଏହି ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତିଗୁଡ଼ିକ ପ୍ରୋଗ୍ରାମର୍‌ଙ୍କୁ ବିବୃତ୍ତିର କ୍ରମପରିବର୍ତ୍ତନ କରିବାକୁ କିମ୍ବା ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍‌ରେ ବିବୃତ୍ତିର ପୁନରାବୃତ୍ତି ନିୟନ୍ତ୍ରଣ କରିବାକୁ ଅନୁମତି ଦିଏ । ଏଠାରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଥିବା ବିଭିନ୍ନ ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତି ଅନ୍ତର୍ଭୁକ୍ତ ଯଥା – if, if-else, if-else-if, switch, for, while, do-while, break, ଏବଂ continue ବିବୃତ୍ତି । ଏଗୁଡ଼ିକର ବ୍ୟବହାର ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ପ୍ରଦର୍ଶିତ ହୋଇଛି ।

ଭୂମିକା

ଗୋଟିଏ କମ୍ପ୍ୟୁଟର ପ୍ରୋଗ୍ରାମ୍ ହେଉଛି କମ୍ପ୍ୟୁଟର ପାଇଁ ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକର ଏକ ସେଟ୍ । ଏହି ନିର୍ଦ୍ଦେଶାବଳିର କ୍ରମାନୁସାରେ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଲେଖାହୋଇଥିବା ଗୋଟିଏ ପରେ ଗୋଟିଏ କାର୍ଯ୍ୟକାରୀ ହୋଇଥାଏ । ବିବୃତ୍ତି ଚାଳନର ଏହି କ୍ରମ ସରଳ ସମସ୍ୟା ପାଇଁ ଭଲ କାମ କରେ । ସେଥିରେ ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକର କୌଣସି ନିଷ୍ପତ୍ତି ନେବା ପ୍ରକ୍ରିୟା କିମ୍ବା କୌଣସି ପୁନରାବୃତ୍ତି ଜଡିତ ନଥାଏ ।

ସେ ଯାହାହେଉ, କାର୍ଯ୍ୟକ୍ଷେତ୍ରରେ, ନିର୍ଦ୍ଦିଷ୍ଟ କେତୋଟି ଥରପାଇଁ କିମ୍ବା କିଛି ସର୍ତ୍ତ ପୂରଣ କରିବା ପର୍ଯ୍ୟନ୍ତ ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକର କ୍ରମ ପରିବର୍ତ୍ତନ କରିବା କିମ୍ବା ପୁନରାବୃତ୍ତି କରିବା ପ୍ରାୟତଃ ଆବଶ୍ୟକ ପଡ଼େ । ଏହିପରି କ୍ଷେତ୍ରରେ, ଏହି ବିବୃତ୍ତିଗୁଡ଼ିକ କାର୍ଯ୍ୟକାରୀ କ୍ରମକୁ ନିୟନ୍ତ୍ରଣ କରିବା ଆବଶ୍ୟକ ।

ଏହି ଏକକରେ ବିଦ୍ୟାର୍ଥୀମାନଙ୍କୁ ସିଣ୍ଟାକ୍ସ(Syntax) ବୁଝିବାରେ ଏବଂ ବିଭିନ୍ନ ବିବୃତ୍ତିର କାର୍ଯ୍ୟ କରିବାରେ ସାହାଯ୍ୟ କରେ ଫଳରେ ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ସମାଧାନ ପାଇଁ ସର୍ତ୍ତମୂଳକ ଶାଖା (conditional branching) ଏବଂ ଲୁପିଂ (looping) କାର୍ଯ୍ୟାନ୍ୱୟନ ପାଇଁ ବ୍ୟବହୃତ ହୋଇପାରିବ ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- ସମ୍ବନ୍ଧୀୟ ଅପରେଟର
- ଲଜିକାଲ୍ ଅପରେଟର

- ବୃଦ୍ଧି/ହ୍ରାସ ଅପରେଟର
- ସମ୍ବନ୍ଧୀୟ / ଲଜିକାଲ୍ ପରିପ୍ରକାଶର ମୂଲ୍ୟାଙ୍କନ

ୟୁନିଟ୍‌ଲବ୍ଧ ଜ୍ଞାନ

ୟୁନିଟ୍ ସମାପ୍ତ ହେବା ପରେ, ବିଦ୍ୟାର୍ଥୀଗଣ ନିମ୍ନଲିଖିତ ବିଷୟଗୁଡ଼ିକରେ ଜ୍ଞାନ ଆହରଣରେ ସମର୍ଥ ହେବେ।

U3-O1: ସର୍ତ୍ତମୂଳକ ଶାଖା ଏବଂ ଲୁପିଂର ଆବଶ୍ୟକତା ବ୍ୟାଖ୍ୟା କରିବା

U3-O2: ଦିଆଯାଇଥିବା ସମସ୍ୟା ଆଧାରରେ ସର୍ତ୍ତମୂଳକ ଶାଖା ବିବୃତ୍ତିର ଠିକ୍ ଚୟନ

U3-O3: ଦିଆଯାଇଥିବା ସମସ୍ୟା ଆଧାରରେ ସଠିକ୍ ଲୁପ୍‌ର ଚୟନ

U3-O4: break ଓ continue କୀୱାର୍ଡଗୁଡ଼ିକର ବ୍ୟବହାର

U3-O5: ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ସମାଧାନପାଇଁ ସର୍ତ୍ତମୂଳକ ଶାଖା ଏବଂ ଲୁପ୍ ବ୍ୟବହାର କରି ପ୍ରୋଗ୍ରାମ୍‌ର ତିଆରି

ୟୁନିଟ୍-3 ୟୁନିଟ୍‌ଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ *(1 – ଦୁର୍ବଳ ସହବନ୍ଧନ; 2 – ମଧ୍ୟମ ସହବନ୍ଧନ; 3 – ଦୃଢ଼ ସହବନ୍ଧନ)*							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U3-O1	1	–	–	–	–	–	–	–
U3-O2	–	–	–	2	–	–	–	–
U3-O3	–	–	–	2	–	–	–	–
U3-O4	–	3	–	2	–	–	–	–
U3-O5	–	–	–	3	–	–	–	–

3.1 ଉପକ୍ରମ

ବିବୃତ୍ତି ଚାଳନର କ୍ରମ ପରିବର୍ତ୍ତନ କରିବା, ଅର୍ଥାତ ନିୟନ୍ତ୍ରଣର ପ୍ରବାହକୁ ବଦଳାଇବାପାଇଁ, ନିମ୍ନଲିଖିତ ପରିସ୍ଥିତି ମଧ୍ୟରୁ କୌଣସିଟି ସଂଶ୍ଲିଷ୍ଟ ହୋଇପାରେ:

- **ବ୍ରାଂଚିଙ୍ଗ୍ (Branching):** ନିଷ୍ପତ୍ତିର ଫଳାଫଳ ଉପରେ ନିର୍ଭର କରି ଗୋଟିଏ ଗ୍ରୁପ୍‌ର ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକୁ ଚାଳନ।
- **ଲୁପିଙ୍ଗ୍ (Looping):** ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ସଂଖ୍ୟକ ଥର କିମ୍ବା ଆବଶ୍ୟକ ସର୍ତ୍ତ ପୂରଣ ନ ହେବା ପର୍ଯ୍ୟନ୍ତ ଏକ ଗ୍ରୁପ୍ ନିର୍ଦ୍ଦେଶଗୁଡ଼ିକର ପୁନରାବୃତ୍ତି କରିବା।
- **ଜମ୍ପିଙ୍ଗ୍ (Jumping):** ସମାନ ପ୍ରୋଗ୍ରାମ୍ ୟୁନିଟ୍‌ରେ ପ୍ରୋଗ୍ରାମ୍-ନିୟନ୍ତ୍ରଣକୁ ଗୋଟିଏ ବିନ୍ଦୁରୁ ଅନ୍ୟ ବିନ୍ଦୁକୁ ସ୍ଥାନାନ୍ତର କରିବା।

C ଭାଷା ବିବୃତ୍ତି ଚାଳନର କ୍ରମକୁ ନିୟନ୍ତ୍ରଣ କରିବାପାଇଁ ସୁବିଧା ଦେଇଥାଏ, ଯାହାକୁ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତି (flow control statements) କିମ୍ବା କେବଳ ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତି କୁହାଯାଏ।

ବିଭିନ୍ନ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତି ନିମ୍ନଲିଖିତ ବର୍ଗଗୁଡ଼ିକରେ ଏକତ୍ର ହୋଇଛି:

1. ***ସର୍ତ୍ତମୂଳକ ଶାଖା*** – ଏହି ବର୍ଗରେ, C ଭାଷା ନିମ୍ନଲିଖିତ ଉକ୍ତିଗୁଡ଼ିକ ପ୍ରଦାନ କରେ
   - if ବିବୃତ୍ତି
   - if-else ବିବୃତ୍ତି
   - else-if ର ସିଢ଼ି(ladder)
   - switch ବିବୃତ୍ତି
2. ***ଲୁପିଂ*** – ଏହି ବର୍ଗରେ, C ଭାଷା ନିମ୍ନଲିଖିତ ବିବୃତ୍ତିଗୁଡ଼ିକ ପ୍ରଦାନ କରେ
   - for ବିବୃତ୍ତି
   - while ବିବୃତ୍ତି
   - do-while ବିବୃତ୍ତି
3. ***ଜମ୍ପିଙ୍ଗ୍*** – ଏହି ବର୍ଗରେ, C ଭାଷା ନିମ୍ନଲିଖିତ ବିବୃତ୍ତିଗୁଡ଼ିକ ପ୍ରଦାନ କରେ
   - break ବିବୃତ୍ତି
   - continue ବିବୃତ୍ତି

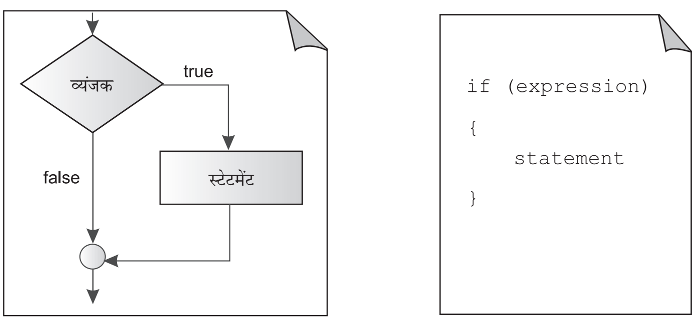
ଏହି ୟୁନିଟ୍‌ରେ, ଆମେ ଉପରୋକ୍ତ ବିବୃତ୍ତିଗୁଡ଼ିକ ବିଷୟରେ ଆଲୋଚନା କରିଛୁ । ଏହି ଆଲୋଚନା ଯଥେଷ୍ଟ ମାତ୍ରାରେ ସମାଧାନ ହୋଇଥିବା ପ୍ରୋଗ୍ରାମ ଦ୍ୱାରା ସମର୍ଥିତ ହୋଇଛି ।

3.2 ସର୍ତ୍ତମୂଳକ ଶାଖା (Conditional Branching)

ଏହି ବର୍ଗ ଅଧୀନରେ ଥିବା ବିଭିନ୍ନ ବିବୃତ୍ତିଗୁଡ଼ିକ ନିର୍ଦ୍ଦିଷ୍ଟ ନିଷ୍ପତ୍ତି ମାନଦଣ୍ଡ ଆଧାରରେ ପସନ୍ଦଯୋଗ୍ୟ ବିବୃତ୍ତିଗୁଡ଼ିକୁ ଚାଳନପାଇଁ ଅନୁମତି ଦିଏ । ଭଲଭାବେ ଡିଜାଇନ୍ ହୋଇଥିବା ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ ଦ୍ୱାରା ସେଗୁଡ଼ିକର କାର୍ଯ୍ୟ ଦେଖିବା ।

3.2.1 if ବିବୃତ୍ତି

if ବିବୃତ୍ତିର ସିଣ୍ଟାକ୍ସ (syntax) ହେଉଛି



(a) if ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ (b) if ବିବୃତ୍ତିର କୋଡ୍

ଚିତ୍ର 3.1: if ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍

ପରିପ୍ରକାଶ ବିଭିନ୍ନ ପ୍ରକାରର ହୋଇପାରେ; ଯଥା ସମ୍ବନ୍ଧୀୟ ପରିପ୍ରକାଶ, ଯୁକ୍ତିଯୁକ୍ତ ପରିପ୍ରକାଶ, ସଂଖ୍ୟାତ୍ମକ ଚଳ କିମ୍ବା ଏକ ସଂଖ୍ୟାତ୍ମକ ଧ୍ରୁବକ । ନିର୍ଦ୍ଦିଷ୍ଟ ପରିପ୍ରକାଶ ଏକ ସରଳ ପରିପ୍ରକାଶ କିମ୍ବା ଯୌଗିକ ପରିପ୍ରକାଶ ହୋଇପାରେ ।

C ରେ ପରିପ୍ରକାଶଟିଏ 0 କିମ୍ବା 1 କୁ ମୂଲ୍ୟାଙ୍କନ ହୁଏ । ଯଦି ପରିପ୍ରକାଶର ମୂଲ୍ୟ 1 ମୂଲ୍ୟାଙ୍କନ ହୁଏ, ତେବେ ବିବୃତ୍ତି-ବ୍ଲକରେ ଥିବା ବିବୃତ୍ତିଗୁଡ଼ିକ ଚାଳନ ହୁଏ; ଅନ୍ୟଥା ସେଗୁଡ଼ିକୁ ଏଡ଼ାଇ (bypass) ଦିଆଯାଏ ।

ଉଦାହରଣ 3.1: ଦିଆଯାଇଥିବା ଇଣ୍ଟିଜର ସଂଖ୍ୟାଟି ଯୁକ୍ତାତ୍ମକ କି ଋଣାତ୍ମକ ତାହା ପରଖ କରିବାକୁ ପ୍ରୋଗ୍ରାମ୍ ।

ତାଲିକା 3.1

```
#include<stdio.h>
 int main()
 {
    int n;
    printf( "\nEnter integer number : " );
    scanf( "%d", &n );

    if ( n < 0 ) {
       printf( "\nNumber is negative.\n" );
       return 0;
    }
    printf( "\nNumber is positive.\n" );
    return 0;
 }
```

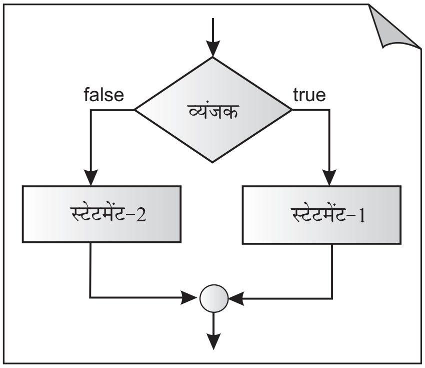
ଟେଷ୍ଟ ରନ୍

```
First Run
   Enter integer number : -25
   Number is negat ive.
Second Run
   Enter integer number : 30
   Number is positive.
```

3.2.2 if-else ବିବୃତ୍ତି

ଯଦି ବିବୃତ୍ତି ଗୋଟିଏ ଏକକ ହୁଏ, ତେବେ ବିବୃତ୍ତିକୁ (ସରଳ କିମ୍ବା ଯୌଗିକ) ଚାଳନ କରାଯାଏ, ନିର୍ଦ୍ଦିଷ୍ଟ ପରିପ୍ରକାଶଟି ଏକ ଅଣ-ଶୂନ୍ୟ ମୂଲ୍ୟକୁ ମୂଲ୍ୟାଙ୍କନ ହୁଏ । ଏହାର ଶୂନ୍ୟ ମୂଲ୍ୟାଙ୍କନ ହେଲେ ଏହା କିଛି କରେ ନାହିଁ । ଏପରି କୌଣସି ଉପାୟ ଅଛିକି ଯେଉଁଥିରେ ଗୋଟିଏ ବିବୃତ୍ତି ଚାଳନ ହୁଏ ଯଦି ଏକ ପରିପ୍ରକାଶ ଅଣ-ଶୂନ୍ୟ ମୂଲ୍ୟାଙ୍କନ ହୁଏ ଏବଂ ଅନ୍ୟ ଏକ ବିବୃତ୍ତି ଚାଳନ ହୁଏ ଯଦି ପରିପ୍ରକାଶ ଶୂନ୍ୟ ମୂଲ୍ୟାଙ୍କନ ହୁଏ ? ଉତ୍ତର ହେଉଛି ହଁ ।

ଏହି ଉଦ୍ଦେଶ୍ୟଟି if-else ବିବୃତ୍ତି ବ୍ୟବହାର କରି ହାସଲ ହୁଏ । ସିଣ୍ଟାକ୍ସ ହେଉଛି –



(a) if-else ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ

```
if (expression)
{
    statement-1
}
else
{
    statement-2
}
```

(b) if-else ବିବୃତ୍ତିର କୋଡ୍

ଚିତ୍ର . 3.2: if-else ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍

ଏଠାରେ ଯଦି ପରିପ୍ରକାଶ 1 ମୂଲ୍ୟାଙ୍କନ ହୁଏ, statement-1 ଚାଳନ ହୁଏ ଏବଂ statement-2 କୁ ଏଡ଼ାଇ ଦିଆହୁଏ । ପକ୍ଷାନ୍ତରରେ, ଯଦି ପରିପ୍ରକାଶ 0 ମୂଲ୍ୟାଙ୍କନ ହୁଏ, statement-1 ଏଡ଼ାଇ ଦିଆହୁଏ ଏବଂ statement -2 ଚାଳନ ହୁଏ।

ଉଦାହରଣ 3.2: ଗଣନ ସଂଖ୍ୟା ଯୁଗ୍ମ କି ଅଯୁଗ୍ମ ତାହା ପରଖପାଇଁ ପ୍ରୋଗ୍ରାମ୍।

ତାଲିକା 3.2

```
#include<stdio.h>
 int main()
 {
    int n;
    printf( "\nEnter any natural number : " );
    scanf( "%d", &n );
    if ( n % 2 == 0 )
       printf( "\nNumber entered is EVEN\n" );
    else
       printf( "\nNumber entered is ODD\n" );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

First Run

```
   Enter any whole number: 22
   Number entered is EVEN
```

ଦ୍ୱିତୀୟ ରନ୍

```
   Enter any number: 15
   Number entered is ODD
```

ଉଦାହରଣ 3.3: ଦୁଇଟି ସଂଖ୍ୟାର ବୃହତ୍ତର ଖୋଜିବା ପାଇଁ ପ୍ରୋଗ୍ରାମ୍ ।

ତାଲିକା 3.3

```
#include<stdio.h>
 int main()
 {
     int a, b;
     printf( "\nEnter value for a : " );
     scanf( "%d", &a );
     printf( "\nEnter value for b : " );
     scanf( "%d", &b );
     if ( a > b )
        printf( "\n%d is the larger.\n", a );
     else
        printf( "\n%d is the larger.\n", b );
     return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
First Run
   Enter value for a : 20
   Enter value for b : 10
   20 is the larger.
Second Run
   Enter value for a : 15
   Enter value for b : 25
   25 is the larger.
```

3.2.3 if ଏବଂ if-else ବିବୃତ୍ତିର ନୀଡ଼ନ (Nested if and if - else statements)

if ବିବୃତ୍ତିଗୁଡ଼ିକ ନୀଡ଼ନ (nested) ହୋଇ ପାରିବ । ଗୋଟିଏ if ବିବୃତ୍ତି ଅନ୍ୟ ଏକ if ବିବୃତ୍ତି ମଧ୍ୟରେ ଲେଖା ଯାଇପାରିବ । ଯଦି ଏକ ବାହ୍ୟ if ବିବୃତ୍ତିର ପରିପ୍ରକାଶ ଅଣ-ଶୂନ୍ୟ ମୂଲ୍ୟାଙ୍କନ ହୁଏ ତେବେ ଭିତର if ବିବୃତ୍ତିଟି ଚାଳନ ହୁଏ ।

ସେହିଭଳି, if- else ବିବୃତ୍ତିଗୁଡ଼ିକ ମଧ୍ୟ ନୀଡ଼ନ ହୋଇ ପାରିବ । ଏଠାରେ, if ବିବୃତ୍ତି if ଅଂଶ କିମ୍ବା else ଅଂଶରେ ନୀଡ଼ନ ହୋଇପାରିବ । ସେହିଭଳି, if-else ବିବୃତ୍ତି if ଅଂଶ କିମ୍ବା else ଅଂଶରେ ନୀଡ଼ନ ହୋଇପାରିବ ।

ଉଦାହରଣ 3.4: ଦିଆଯାଇଥିବା ବର୍ଷ ଏକ ଅଧିବର୍ଷ କି ନୁହେଁ ତାହା ନିର୍ଦ୍ଧାରଣ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ଦିଆଯାଇଥିବା ବର୍ଷଟି ଏକ ଅଧିବର୍ଷ ହେବ ଯଦି ନିମ୍ନଲିଖିତ ସର୍ତ୍ତଗୁଡ଼ିକ ମଧ୍ୟରୁ କୌଣସିଟି ପୂରଣ ହୁଏ:

1. ବର୍ଷଟି 4 ଦ୍ୱାରା ବିଭାଜ୍ୟ କିନ୍ତୁ 100 ଦ୍ୱାରା ବିଭାଜ୍ୟ ନୁହେଁ ।
2. ବର୍ଷଟି 4 ଦ୍ୱାରା, 100 ବିଭାଜ୍ୟ ଏବଂ 400 ଦ୍ୱାରା ବିଭାଜ୍ୟ ।

ଅନ୍ୟ ସମସ୍ତ କ୍ଷେତ୍ରରେ, ଦିଆଯାଇଥିବା ବର୍ଷଟି ଏକ ଅଧିବର୍ଷ ନୁହେଁ ।

ଉଦାହରଣ ବଶତଃ

1. 1988, 1992 ଏବଂ 1996 ବର୍ଷ ଗୁଡ଼ିକ ଅଧିବର୍ଷ, କାରଣ ଏଗୁଡ଼ିକ 4 ଦ୍ୱାରା ବିଭାଜ୍ୟ କିନ୍ତୁ 100 ଦ୍ୱାରା ଭାଜ୍ୟ ନୁହେଁ ।
2. 1700, 2100, ଏବଂ 2500 ବର୍ଷ ଗୁଡ଼ିକ ଅଧିବର୍ଷ ନୁହେଁ କାରଣ ସେମାନେ 4 ଏବଂ 100 ଦ୍ୱାରା ବିଭାଜ୍ୟ କିନ୍ତୁ 400 ଦ୍ୱାରା ନୁହେଁ ।
3. 1600, 2000 ଏବଂ 2400 ବର୍ଷ ଅଧିବର୍ଷ, କାରଣ ସେମାନେ 100 ଏବଂ 400 ଦ୍ୱାରା ବିଭାଜ୍ୟ ।

ତାଲିକା 3.4

```
#include <stdio.h>
 int main()
 {
    int year;
    printf("Enter given year : ");
    scanf("%d", &year);
    if ( year % 4 == 0 ) {
       if ( year % 100 == 0 ) {
           if ( year % 400 == 0 )
              printf("\nYear %d is a leap year.\n", year);
           else
              printf("\nYear %d is not a leap year.\n", year);
                  }
       else
       {
          printf("\nYear %d is a leap year.\n", year);
       }
    }
    else
    {
       printf("\nYear %d is not a leap year.\n", year);
    }
    return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
First Run
   Enter any year : 2000
   Year 2000 is a leap year.
Second Run
   Enter any year : 2008
   Year 2008 is a leap year.
Third Run
   Enter any year : 2006
   Year 2006 is not a leap year.
```

3.2.4 if-elseif ର ସିଡ଼ି (The if-else if Ladder)

if-elseif ର ସିଡ଼ି ହେଉଛି ଏକ ସମ୍ପ୍ରସାରିତ if-else ବିବୃତ୍ତି । ଏହା ଏପରି ଏକ ପରିସ୍ଥିତିରେ ବ୍ୟବହୃତ ହୁଏ ଯେତେବେଳେ ବିଭିନ୍ନ ସ୍ଥିତିଗୁଡ଼ିକ ପାଇଁ ଏକାଧିକ ମାମଲା କରିବାକୁ ପଡିବ ।

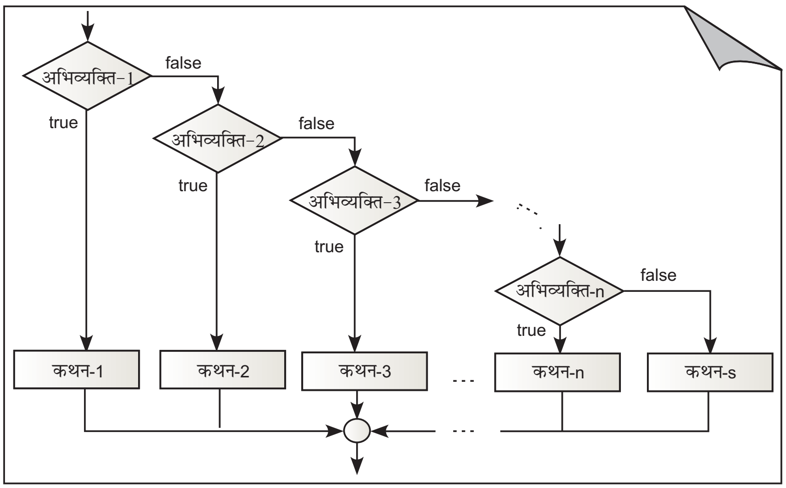
if-elseif ସିଡ଼ି ବିବୃତ୍ତିର ସିଣ୍ଟାକ୍ସ ଚିତ୍ର 3.3. ରେ ଦର୍ଶାଯାଇଛି

```
if ( expression-1 )
   statement-1
else if ( expression-2 )
   statement-2
else if ( expression-3 )
   statement-3
        :
else if ( expression-n )
   statement-n
else
   statement-s
```

ଚିତ୍ର 3.3: ଇଫ୍-ଏଲ୍ସ ସିଡ଼ି କୋଡ୍

ପରିପ୍ରକାଶଗୁଡ଼ିକ ଏକ କ୍ରମରେ ମୂଲ୍ୟାଙ୍କନ ହୁଏ, ଏବଂ ଯଦି କୌଣସି ପରିପ୍ରକାଶ ସତ୍ୟ ତେବେ ଏହାସହିତ ଜଡିତ ବିବୃତ୍ତି ବ୍ଲକ୍‌ଟି ଚାଳନ ହୁଏ, ଏବଂ ଏହା ସମଗ୍ର ଶୃଙ୍ଖଳାକୁ ସେହିଠାରୁ ସମାପ୍ତ କରିଦିଏ ।

ଶେଷ else ଅଂଶ ଚାଳନ ହୁଏ ଯେତେବେଳେ ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ କିମ୍ବା ଡିଫଲ୍ଟ ମାମଲା ଯେଉଁଠାରେ କୌଣସି ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ପରିପ୍ରକାଶ ସର୍ତ୍ତକୁ ପୂରଣ କରେ।ନାହିଁ।



ଚିତ୍ର 3.4: if-elseif ର ସିଡ଼ି ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ

ଉଦାହରଣ 3.5: ମାର୍କର ଶତକଡା ଦିଆଯାଇଅଛି । ନିମ୍ନଲିଖିତ ନିୟମାନୁଯାୟୀ ବିଦ୍ୟାର୍ଥୀର ଗ୍ରେଡ୍ ଗଣନା କରାଯାଏ ।

ମାର୍କର ଶତକଡା	ଗ୍ରେଡ୍
ଶତକଡା ≥ 90	A
75 ≥ 90 > ଶତକଡ଼ା	B
75 > ଶତକଡା ≥ 60	C
60 > ଶତକଡା ≥ 50	D
ଶତକଡା < 50	F

ଶତକଡା ମାର୍କ ଦିଆଯାଇଥିଲେ ବିଦ୍ୟାର୍ଥୀମାନଙ୍କ ଗ୍ରେଡ୍ ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 3.5

```
#include<stdio.h>
 int main()
 {
float percentage;
    printf("\nEnter percentage of marks : ");
    scanf("%f", &percentage);
    if ( percentage >= 90 )
       printf("\nGrade is A\n");
    else if ( percentage >= 75 )
       printf("\nGrade is B\n");
    else if ( percentage >= 60 )
       printf("\nGrade is C\n");
    else if ( percentage >= 50 )
       printf("\nGrade is D\n");
    else
       printf("\nGrade is E\n");
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter percentage of marks : 93
Grade is A
```

3.2.6 switch ବିବୃତ୍ତି

switch ବିବୃତ୍ତି ଗୋଟିଏ ଚଳ/ପରିପ୍ରକାଶର ବିଭିନ୍ନ ମୂଲ୍ୟ ପାଇଁ ଏକାଧିକ ମାମଲାର ବିକଳ୍ପ ପ୍ରଦାନ କରେ ।

switch ବିବୃତ୍ତିର ସିଣ୍ଟାକ୍ସ ଚିତ୍ର 3.5 ଦର୍ଶାଯାଇଛି।

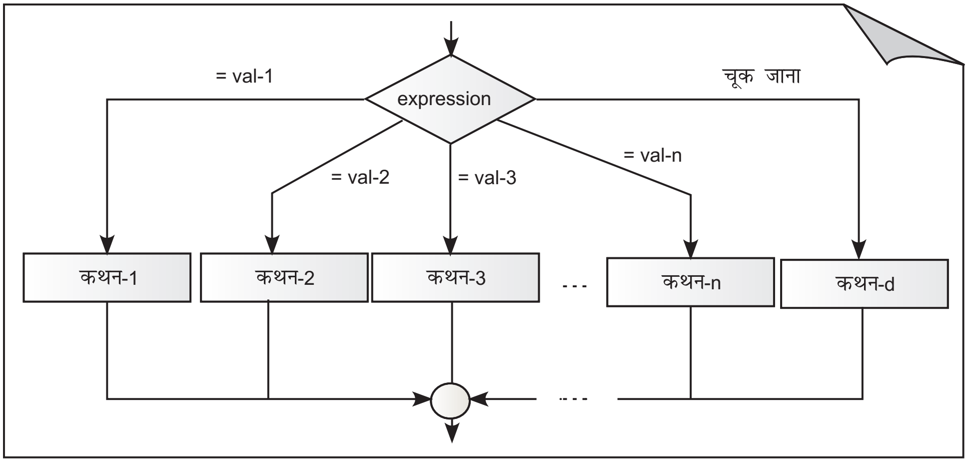
ଯଦି ପରିପ୍ରକାଶ *val-1, val-2, val-3, …, val-n* ମଧ୍ୟରୁ କୌଣସି ମୂଲ୍ୟ ନେଇଥାଏ, ତେବେ ନିୟନ୍ତ୍ରଣ ସେହି ମାମଲାକୁ ସ୍ଥାନାନ୍ତରିତ ହୋଇଥାଏ । ପ୍ରତ୍ୟେକ କ୍ଷେତ୍ରରେ, ବିବୃତ୍ତିଗୁଡ଼ିକ ଚାଳନ ହୁଏ ଏବଂ ତା'ପରେ break ବିବୃତ୍ତି ନିୟନ୍ତ୍ରଣକୁ switch ବିବୃତ୍ତିର ବାହାରକୁ ସ୍ଥାନାନ୍ତର କରେ ।

ଯଦି କୌଣସି ମାମଲା ଶେଷରେ break ବିବୃତ୍ତି ବ୍ୟବହାର କରା ନ ଯାଏ ତେବେ ଡିଫଲ୍ଟ କୀ'ୱାର୍ଡ଼ର ଅନୁପସ୍ଥିତିରେ କେବଳ ଶେଷ ମାମଲା ବ୍ୟତୀତ, ନିୟନ୍ତ୍ରଣ ପରବର୍ତ୍ତୀ ମାମଲାକୁ

```
switch ( expression )
{
   case val-1 :
                statement-1
                break;
   case val-2 :
                statement-2
                break;
        :
   case val-n :
                statement-n
                break;
   default    :
                statement-d
}
```

ଚିତ୍ର 3.5: ସୁଇଚ୍ କରନ୍ତୁ ବିବୃତ୍ତି ର କୋଡ୍

ଚାଲିଯିବ । ଯଦି ପରିପ୍ରକାଶର ମୂଲ୍ୟ କୌଣସି ମାମଲା ମୂଲ୍ୟ ସହିତ ମେଳ ଖାଉ ନଥାଏ, ତେବେ ନିୟନ୍ତ୍ରଣ ଡିଫଲ୍ଟ ବିବୃତ୍ତିକୁ ଚାଲିଯିବ ସାଧାରଣତଃ ପ୍ରତ୍ୟେକ ସୁଇଚ ବିବୃତ୍ତି ଶେଷରେ ଥାଏ । ଡିଫଲ୍ଟ କୀ'ୱାର୍ଡ଼ର ବ୍ୟବହାର ବଡ଼ ସୁବିଧା ଜନକ ହୋଇପାରେ । ଯଦି ଡିଫଲ୍ଟ କୀ'ୱାର୍ଡ଼ର ନ ଥାଏ ଏବଂ ପରିପ୍ରକାଶର ମୂଲ୍ୟ କୌଣସି ମାମଲା ମୂଲ୍ୟ ସହିତ ମେଳ ଖାଉ ନ ଥାଏ ତେବେ ସମ୍ପୂର୍ଣ୍ଣ ସୁଇଚ ବିବୃତ୍ତିଟି ସ୍କିପ୍ (skip) ହୋଇଯାଏ ।



ଚିତ୍ର . 3.6: switch ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ

ଉଦାହରଣ 3.6: କାଲକୁଲେଟର୍(calculator) ର ଚାରୋଟି ପ୍ରକାର୍ଯ୍ୟ ଚାଳନର ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ତାଲିକା 3.6

```
#include<stdio.h>
 int main()
 {
     int a, b;
     char op;
     printf("\nEnter expression such as 5 + 3 : ");
     scanf("%d %c %d", &a, &op, &b );
     switch ( op )
     {
        case '+' : printf("\n%d %c %d = %d\n", a, op, b, a+b);
                   break;
        case '-' : printf("\n%d %c %d = %d\n", a, op, b, a-b);
                   break;
        case '*' : printf("\n%d %c %d = %d\n", a, op, b, a*b);
                   break;
        case '/' : printf("\n%d %c %d = %d\n", a, op, b, a/b);
                   break;
        default :  printf("\nWrong input\n");
     }
     return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter expression such as 5 + 3 : 20 / 2
20 / 2 = 10
```

ଏହିପରି ପରିସ୍ଥିତି ଥାଇପାରେ, ଯେତେବେଳେ ଆମେ ଚାହୁଁ ଯେ ସେହି ସମାନ ବିବୃତ୍ତିଗୁଡ଼ିକ ପରିପ୍ରକାଶର ଏକାଧିକ ମୂଲ୍ୟ ପାଇଁ ଚାଳନ ହେବା ଉଚିତ । ସେହି ପରିସ୍ଥିତିର ପରିଚାଳନା ପାଇଁ, ନିମ୍ନ ପ୍ରଦର୍ଶିତ ମୁତାବକ ଆମେ ମାମଲା ଗୁଡ଼ିକ ଗୋଟିଏ ପରେ ଗୋଟିଏ କୋଡ଼ କରିଥାଉ(ବିବୃତ୍ତି ବିନା) ଏବଂ ଶେଷ ମାମଲାରେ ବିବୃତ୍ତିଗୁଡ଼ିକୁ ଲେଖିଥାଉ ।

```
switch ( expression )
 {
       :
    case val-4 :
    case val-5 :
    case val-6 : statement;
                 break;
       :
 }
```

ପରିପ୍ରକାଶର ମୂଲ୍ୟ *val*-4, *val*-5 କିମ୍ବା *val*-6 ହେବ, ହେଲେ ବିବୃତ୍ତି ବ୍ଲକଟି ଚାଳନ ହେବ ।

ଉଦାହରଣ 3.7: ଗୋଟିଏ ବର୍ଣ୍ଣ ସ୍ୱରବର୍ଣ୍ଣ କି ବ୍ୟଞ୍ଜନବର୍ଣ୍ଣ ତାହାର ପରୀକ୍ଷଣପାଇଁ ପ୍ରୋଗ୍ରାମ୍ ।

ତାଲିକା 3.7

```
#include<stdio.h>
 int main()
 {
     char ch;
     printf("\nEnter an alphabet : ");
     ch = getche();
     switch ( ch )
     {
        case 'a' :
        case 'A' :
        case 'e' :
        case 'E' :
        case 'i' :
        case 'I' :
        case 'o' :
        case 'O' :
        case 'u' :
        case 'U' : printf("\nAlphabet is vowel.\n");
                   break;
        default  :
                   printf("\nAlphabet is consonant.\n");
     }
     return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter an alphabet : A
Alphabet is vowel.
```

ସର୍ତ୍ତମୂଳକ ଶାଖା ପାଇଁ ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 3.8: ନିମ୍ନଲିଖିତ ଏକ ଦ୍ୱିଘାତ ସମୀକରଣର ମୂଳ ବାହାର କରିବାପାଇଁ ପ୍ରୋଗ୍ରାମ୍ –

$ax^2 + bx + c = 0$ iznku dh $a \neq 0$

ସମାଧାନ: ଦ୍ୱିଘାତ ସମୀକରଣର ମୂଳ ବାହାର କରିବାର ସୂତ୍ର ହେଉଛି –

$$a_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

ଯେଉଁଠାରେ ପରିପ୍ରକାଶ $b^2 - 4ac$ କୁ ବିଭେଦକ/ ଡିସ୍କ୍ରିମିନାଣ୍ଟ (discriminant) କୁହାଯାଏ ।

ଡିସ୍କ୍ରିମିନାଣ୍ଟର ଚିହ୍ନ(sign) ଉପରେ ନିର୍ଭର କରି, ମୂଳ ପାଇଁ ତିନୋଟି ସମ୍ଭାବନା ଅଛି:

1. ଯଦି $b^2 - 4ac < 0$, ତେବେ ମୂଳ ହେଉଛି କାଳ୍ପନିକ ସଂଖ୍ୟା, ଏବଂ ଆମେ ବାସ୍ତବ ଅଂଶ ଏବଂ କାଳ୍ପନିକ ଅଂଶକୁ ପୃଥକ ଭାବରେ ଗଣନା କରିପାରିବା ।

   ବାସ୍ତବ ଅଂଶ $= -\frac{b}{2a}$ କାଳ୍ପନିକ ଅଂଶ $= \frac{\sqrt{-(b^2 - 4ac)}}{2a}$

2. ଯଦି $b^2 - 4ac = 0$, ତେବେ ମୂଳ ହେଉଛି ବାସ୍ତବ ସଂଖ୍ୟା ଏବଂ ସେମାନେ ସମାନ, ଏବଂ ମୂଳ ଏହିପରି ଗଣନା କରାଯାଏ ।

   ମୂଳ $= -\frac{b}{2a}$

3. ଯଦି $b^2 - 4ac > 0$, ତେବେ ମୂଳ ହେଉଛି ବାସ୍ତବ ସଂଖ୍ୟା ଏବଂ ସେଗୁଡ଼ିକ ପୃଥକ୍ ତାହାର ଏହିପରି ଗଣନା କରାଯାଏ ।

   ମୂଳ ଏକ $= \frac{-b\sqrt{b^2 - 4ac}}{2a}$ ମୂଳ ଦୁଇ $= \frac{-b + \sqrt{b^2 - 4ac}}{2a}$

ତାଲିକା 3.8

```
#include<stdio.h>
 #include<math.h>

 int main()
 {
    float a, b, c, disc, root1, root2;
    float realPart, imagPart;
    printf( "\nEnter values of a,b,c: " );
    scanf( "%f,%f,%f", &a, &b, &c);
    if ( a == 0 ) {
       printf( "\nIt is not an quadratic equation\n" );
       return 0;
    }
    disc = b * b - 4.0 * a * c;
    if ( disc < 0 )
    {
       realPart = - b / (2*a);
       imagPart = sqrt((double)-disc) / (2*a);
       printf( "\nRoots are imaginary" );
   printf( "\nReal part = %.3f", realPart );
       printf( "\nImaginary part = %.3f\n", imagPart );
    }
    else if ( disc == 0 )
    {
      root1 = - b / (2*a);
   printf( "\nRoots are real and equal" );
   printf( "\nEach root = %.3f\n", root1 );
    }
    else
    {
      root1 = (-b-sqrt((double)disc )) / (2*a);
      root2 = (-b+sqrt((double)disc )) / (2*a);
      printf( "\nRoots are real and distinct" );
      printf( "\nRoot 1 = %.3f", root1 );
      printf( "\nRoot 2 = %.3f\n", root2 );
    }
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter values of a,b,c: 2,3,5
Roots are imaginary
Real part = -0.750
Imaginary part = 1.392
```

ଉଦାହରଣ 3.9: ଧରାଯାଉ ବ୍ୟକ୍ତିବିଶେଷଙ୍କପାଇଁ ନିମ୍ନ ପ୍ରଦତ୍ତ ଉପାୟରେ ଆୟକର ସ୍ଲାବ୍(slab) ହିସାବ ହୁଏ:

ଆୟ	ଟିକସ ଦେୟ
1,00,000 ଟଙ୍କା ପର୍ଯ୍ୟନ୍ତ/-	ଶୂନ୍ୟ
1,00,001/- ଟଙ୍କାରୁ 2,00,000/- ଟଙ୍କା ପର୍ଯ୍ୟନ୍ତ	1,00,000 ଟଙ୍କାରୁ ଅଧିକ ଅତିରିକ୍ତର 10%
2,00,001/- ଟଙ୍କାରୁ 3,00,000/- ଟଙ୍କା ପର୍ଯ୍ୟନ୍ତ	2,00,000 ଟଙ୍କାରୁ ଅଧିକ ଅତିରିକ୍ତର 20%
3,00,000 ଟଙ୍କାରୁ ଅଧିକ/-	3,00,000 ଟଙ୍କାରୁ ଅଧିକ ଅତିରିକ୍ତର 30%

ଆୟ ପଢ଼ି ପାରୁଥିବା ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଏବଂ ଧାର୍ଯ୍ୟ ଆୟକର ପ୍ରିଣ୍ଟ କରେ।

ସମାଧାନ: ନିମ୍ନ ଟିପ୍ପଣୀ ଦେଖ:

- ଯଦି ଆୟ କେବଳ 1,00,000 ଟଙ୍କା ପର୍ଯ୍ୟନ୍ତ ହୁଏ, ତେବେ ଟିକସ ହେଉଛି ଶୂନ୍ୟ.
- ଯଦି ଆୟ 1,00,001/- ରୁ 2,00,000/- ଟଙ୍କା ମଧ୍ୟରେ ଥାଏ, ତେବେ ଟିକସ ହେଉଛି 1,00,000 ଟଙ୍କାରୁ ଅଧିକ ପରିମାଣର 10%।
- ଯଦି ଆୟ ଟଙ୍କା 2,00,001/- ରୁ 3,00,000/- ଟଙ୍କା ମଧ୍ୟରେ ରହେ, ତା'ହେଲେ ଟିକସ 1,00,000/- ଟଙ୍କାରୁ ଅଧିକ ପରିମାଣର 10% (ଅର୍ଥାତ 10,000/-ଟଙ୍କା), ତା'ସହିତ 2,00,000/- ଟଙ୍କାରୁ ଅଧିକ ପରିମାଣର 20%
- ଯଦି ଆୟ 3,00,000 /-ଟଙ୍କାରୁ ଅଧିକ, ତେବେ ଟିକସ ହେଉଛି 1,00,001/- ରୁ 2,00,000/- ଟଙ୍କା ସ୍ଲାବ୍ ପାଇଁ 1,00,000 /- ଟଙ୍କାର 10% (10,000 ଟଙ୍କା/-) ତା'ସହିତ 2,00,001 ଟଙ୍କାରୁ 3,00,000/- ଟଙ୍କାର ସ୍ଲାବ୍ ପାଇଁ 1,00,000 /- ଟଙ୍କାର 20% (20,000 ଟଙ୍କା/-) ତା'ସହିତ 3,00,000/- ଟଙ୍କାରୁ ଅଧିକ ପରିମାଣର 30%।

ନିମ୍ନ ପ୍ରଦତ୍ତ ତଥ୍ୟ ଯଦି ଆୟ ହୁଏ ତେବେ ଆସ ଏହାର ଟିକସ ହିସାବ କରିବା।

(a) 2,50,000 ଟଙ୍କା/- (b) 1,75,000 ଟଙ୍କା/-

(c) 3,20,000 ଟଙ୍କା/-

a) 2,50,000/- ଟଙ୍କା ଆୟ ପାଇଁ ଟିକସ ହିସାବ

ପ୍ରଥମ 1,00,000/- ଟଙ୍କା ପାଇଁ ଟିକସ ଶୂନ୍ୟ

ପରବର୍ତ୍ତୀ 1,00,000 ଟଙ୍କା ପାଇଁ/-	ଟିକସ ହେଉଛି 10,000/- (@ 10%)
ପରବର୍ତ୍ତୀ 50,000 ଟଙ୍କା ପାଇଁ/-	ଟିକସ ହେଉଛି 10,000/- (@ 20%)
ମୋଟ ଟିକସ ଦେୟ	**₹ 20,000/-**

(b) 1,75,000/- ଟଙ୍କା ଆୟ ପାଇଁ ଟିକସ ହିସାବ

ପ୍ରଥମ 1,00,000/- ଟଙ୍କା ପାଇଁ	ଟିକସ ଶୂନ୍ୟ
ପରବର୍ତ୍ତୀ 75,000 ଟଙ୍କା ପାଇଁ/-	ଟିକସ ହେଉଛି 7,500/- (@ 10%)
ମୋଟ ଟିକସ ଦେୟ	**₹ 7,500/-**

(c) 3,20,000/- ଟଙ୍କା ଆୟ ପାଇଁ ଟିକସ ହିସାବ

ପ୍ରଥମ 1,00,000/- ଟଙ୍କା ପାଇଁ	ଟିକସ ଶୂନ୍ୟ
ପରବର୍ତ୍ତୀ 1,00,000 ଟଙ୍କା ପାଇଁ/-	ଟିକସ ହେଉଛି 10,000/- (@ 10%)
ପରବର୍ତ୍ତୀ 1,00,000 ଟଙ୍କା ପାଇଁ/-	ଟିକସ ହେଉଛି 20,000/- (@ 20%)
ପରବର୍ତ୍ତୀ 20,000 ଟଙ୍କା ପାଇଁ/-	ଟିକସ ହେଉଛି 6,000/- (@ 30%)
ମୋଟ ଟିକସ ଦେୟ	**₹ 36,000/-**

ତାଲିକା 3.9

```
#include<stdio.h>
void main()
{
float income, tax;
printf( "\nEnter gross income (in Rs.): " );
scanf( "%f", &income );
if ( income <= 100000.0 )
tax = 0;
else if ( income <= 200000.0 )
tax = ( income - 100000.0 ) * 0.1;
else if ( income <= 300000.0 )
tax = 10000 + ( income - 200000.0 ) * 0.2;
else
tax = 30000 + ( income - 300000.0 ) * 0.3;
printf( "\nTax due = Rs. %.2f\n", tax );
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter gross income : 250000
Tax due = Rs. 20000.00
```

ଉଦାହରଣ 3.10: ସପ୍ତାହର ଦିନକୁ ସଂଖ୍ୟା ଭାବରେ ଗ୍ରହଣ କରିପାରୁଥିବା ଏବଂ ସେହି ଦିନର ନାମ ପ୍ରିଣ୍ଟ୍ କରେ, ଯଥା, ରବିବାର ପାଇଁ 0, ସୋମବାର ପାଇଁ 1, ... ଶନିବାର ପାଇଁ 6 ।

ତାଲିକା 3.10

```
#include <stdio.h>
 int main()
 {
    int day;
    printf("\nEnter day of week as number ( 0 - 6 ) : ");
    scanf("%d", &day);
    switch ( day )
    {
       case 0  : printf("\nDay of week is Sunday." );
                 break;
       case 1  : printf("\nDay of week is Monday." );
                 break;
       case 2  : printf("\nDay of week is Tuesday." );
                 break;
       case 3  : printf("\nDay of week is Wednesday." );
                 break;
       case 4  : printf("\nDay of week is Thursday." );
                 break;
       case 5  : printf("\nDay of week is Friday." );
                 break;
       case 6  : printf("\nDay of week is Saturday." );
                 break;
       default : printf("\nWrong input" );
    }
    printf("\n");
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter day of week as number ( 0 - 6 ) : 2
Day of week is Tuesday.
```

ଉଦାହରଣ 3.11: 2 ଦ୍ୱାରା ମୋଡୁଲସ୍(modulus) ପ୍ରକ୍ରିୟା ଓ if ବିବୃତ୍ତି ବ୍ୟବହାର ନ କରି ଏକ ଗଣନ ସଂଖ୍ୟା ଯୁଗ୍ମ କିମ୍ବା ଅଯୁଗ୍ମ ତାହା ପରୀକ୍ଷା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 3.11

```
#include <stdio.h>
 int main()
 {
    int n;
    printf("\nEnter any natural number : ");
    scanf("%d", &n);
    switch ( n%10 )
    {
       case 0  :
       case 2  :
       case 4  :
       case 6  :
       case 8  : printf("\n%d is even.\n");
                 break;
       default : printf("\n%d is even.\n");
    }
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

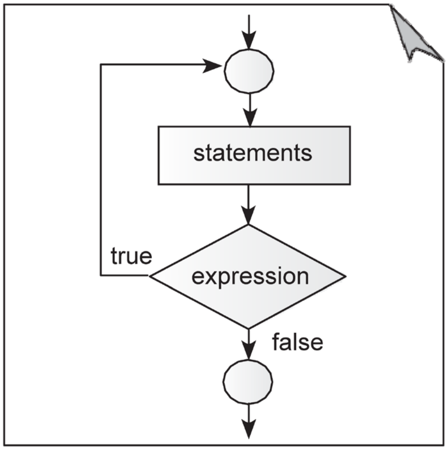
```
First Run
Enter any natural number : 120
20 is even.
Second Run
Enter any natural number : 75
75 is odd.
```

3.3 ଲୁପିଂ(Looping)

ଏହି ବର୍ଗ ଅଧୀନରେ ଥିବା ବିବୃତ୍ତିଗୁଡ଼ିକ ନିର୍ଦ୍ଦିଷ୍ଟ ସଂଖ୍ୟକ ଥର କିମ୍ବା କିଛି ସର୍ତ୍ତ ପୂରଣ ନ ହେବା ପର୍ଯ୍ୟନ୍ତ ଏକ ବିବୃତ୍ତି ଗ୍ରୁପ୍‌କୁ ବାରମ୍ବାର ଚାଳନର ଅନୁମତି ଦିଏ । ଆମେ ସେଗୁଡ଼ିକର କାର୍ଯ୍ୟ ଭଲ ଭାବେ ଡିଜାଇନ୍ ହୋଇଥିବା ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣଦ୍ୱାରା ଦେଇପାରିବା ।

3.3.1 for ବିବୃତ୍ତି

ଏକ ବିବୃତ୍ତି କିମ୍ବା ବିବୃତ୍ତି-ବ୍ଲକ୍ କେତେଥର ଚାଳନ ହୋଇପାରେ ତାହା ଆଗରୁ ଜଣାଥିଲେ for ବିବୃତ୍ତି ସମସ୍ୟାପାଇଁ ଉପଯୁକ୍ତ ହୋଇପାରେ ।




```
do
{
   Statement
}
while (expression);
```

(a) ଫର୍ ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ (b) ଫର୍ ବିବୃତ୍ତିର କୋଡ୍

ଚିତ୍ର . 3.7: ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍‌ପାଇଁ ବିବୃତ୍ତି

ଯେଉଁଠାରେ *init* ହେଉଛି କାଉଣ୍ଟର(ଗଣକ)କୁ ଆରମ୍ଭ କରିବାପାଇଁ ଏକ ପରିପ୍ରକାଶ, test ହେଉଛି ଏକ ପରିପ୍ରକାଶ ଯାହା କେବେ ପୁନରାବୃତ୍ତି ବନ୍ଦ କରିବ, ଏବଂ ଲୁପ୍‌ର ପ୍ରତ୍ୟେକ ପାସ୍ ପାଇଁ କାଉଣ୍ଟରକୁ ପରିବର୍ତ୍ତନ କରିବାପାଇଁ change ହେଉଛି ଏକ ପରିପ୍ରକାଶ । *init* ଏବଂ change ଯାଗାରେ ଏକାଧିକ ବିବୃତ୍ତି କମା ଦ୍ୱାରା ପୃଥକ ହୋଇ ରହିପାରେ ।

for ବିବୃତ୍ତିର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରିବାକୁ, ଆସ for ବିବୃତ୍ତି ବ୍ୟବହାର କରି କିଛି ପ୍ରୋଗ୍ରାମ୍ ତିଆରି କରିବା ।

ଉଦାହରଣ 3.12: ପ୍ରଥମ n ଗଣନ ସଂଖ୍ୟାଗୁଡ଼ିକର ସମଷ୍ଟି ଜାଣିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 3.12

```
#include<stdio.h>
int main(void)
 {
    int i, n, sum = 0;
    printf("\nEnter value for n : " );
    scanf("%d", &n);
    for ( i = 1; i <= n; i++ ) {
       sum = sum + i;
    }
    printf("\nSum of first %d natural numbers = %d\n", n, sum );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter value for n : 10
Sum of first 10 natural numbers = 55
```

ଉଦାହରଣ 3.13: ଏକ ଗଣନ ସଂଖ୍ୟା n ର ଫ୍ୟାକ୍ଟୋରିଆଲ୍(factorial) ଜାଣିବା ପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 3.13

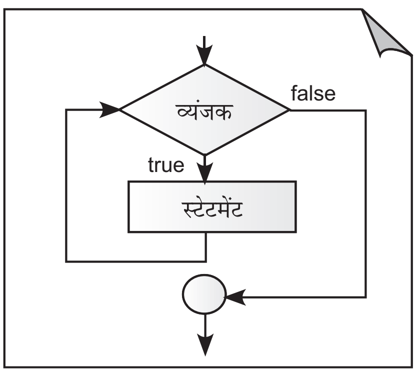
```
#include<stdio.h>
 int main(void)
 {
    int i, n, prod = 1;
    printf("\nEnter value for n : " );
    scanf("%d", &n);
    for ( i = 1; i <= n; i++ ) {
       prod = prod * i;
    }
    printf("\nFactorial %d = %d\n", n, prod );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

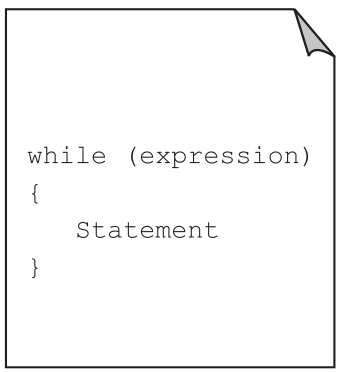
```
Enter value for n : 5
Factorial 5 = 120
```

3.3.2 while ବିବୃତ୍ତି

ଏକ ବିବୃତ୍ତି କିମ୍ବା ଏକ ବିବୃତ୍ତି-ବ୍ଲକ୍ କେତେଥର କାର୍ଯ୍ୟକାରୀ ହୋଇପାରେ ତାହା ଯଦି ଆଗୁଆ ଜଣା ନ ଥାଏ ତେବେ ତାହା while ବିବୃତ୍ତି ସମସ୍ୟା ପାଇଁ ଉପଯୁକ୍ତ ।



(a) ହ୍ୱାଇଲ୍ ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ



(b) ହ୍ୱାଇଲ୍ ବିବୃତ୍ତିର କୋଡ୍

ଚିତ୍ର . 3.8: while ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍

ଏଠାରେ ପରିପ୍ରକାଶଟି ଏକ ଧ୍ରୁବକ, ଏକ ଚଳ କିମ୍ବା ଏକ ପରିପ୍ରକାଶ ହୋଇପାରେ । ବିବୃତ୍ତି ଯେତେବେଳେ ପର୍ଯ୍ୟନ୍ତ ପରିପ୍ରକାଶ 1କୁ ମୂଲ୍ୟାଙ୍କନ ହେଉଥାଏ ସେତେବେଳେ ଯାଏ ବାରମ୍ବାର କାର୍ଯ୍ୟକାରୀ ହୋଇଥାଏ। ପରିପ୍ରକାଶ 0କୁ ମୂଲ୍ୟାଙ୍କନ ହେଲେ while ବିବୃତ୍ତିର ଚାଳନ ସମାପ୍ତ ହୁଏ ଏବଂ ତୁରନ୍ତ ଏହାକୁ ଅନୁସରଣ କରୁଥିବା ବିବୃତ୍ତିକୁ ନିୟନ୍ତ୍ରଣ ଯାଏ ।

while ବିବୃତ୍ତି ବ୍ୟବହାର କରିବାବେଳେ ନିମ୍ନଲିଖିତ ବିନ୍ଦୁଗୁଡ଼ିକୁ ଧ୍ୟାନରେ ରଖିବା ଆବଶ୍ୟକ –

- *while* ବିବୃତ୍ତି ପୂର୍ବରୁ ଏକ ବିବୃତ୍ତି ରହିବା ଆବଶ୍ୟକ ଯାହା ପରିପ୍ରକାଶର ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରେ ।
- ବିବୃତ୍ତି-ବ୍ଲକରେ, ଏକ ବିବୃତ୍ତି ରହିବା ଆବଶ୍ୟକ ଯାହା ପରିପ୍ରକାଶର ମୂଲ୍ୟ ପରିବର୍ତ୍ତନ କରେ ।

while ବିବୃତ୍ତିର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରିବାକୁ, ଆସ *while* ବିବୃତ୍ତି ବ୍ୟବହାର କରି କିଛି ପ୍ରୋଗ୍ରାମ୍ ତିଆରି କରିବା ।

ଉଦାହରଣ 3.14: ଏକ ଗଣନ ସଂଖ୍ୟା *n* ର ଅଙ୍କଗୁଡ଼ିକର ସମଷ୍ଟି ବାହାର କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 3.14

```
#include <stdio.h>
 int main()
 {
    int i, n, sum = 0, temp, digit;
    printf("\nEnter any natural number : " );
    scanf("%d", &n);
    temp = n;
    while ( temp > 0 )
    {
       digit = temp % 10;
       sum = sum + digit;
       temp = temp / 10;
    }
    printf( "\nSum of digits of %d = %d\n", n, sum );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter any natural number : 2375
Sum of digits of 2375 = 17
```

ଉଦାହରଣ 3.15: ଏକ ଅନୁଚ୍ଛେଦରେ ଅକ୍ଷର, ସ୍ୱରବର୍ଣ୍ଣ, ଏବଂ ଶବ୍ଦସଂଖ୍ୟା ଗଣନା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ତାଲିକା 3.15

```
#include <stdio.h>
 int main()
 {
    int vowelCount = 0,characterCount = 0,wordCount = 0;
    char ch;
    printf( "Type in the paragraph and terminate by ENTER key\n\n" );
    while ( ( ch = getche() ) != '\r' )
    {
       characterCount++;
```

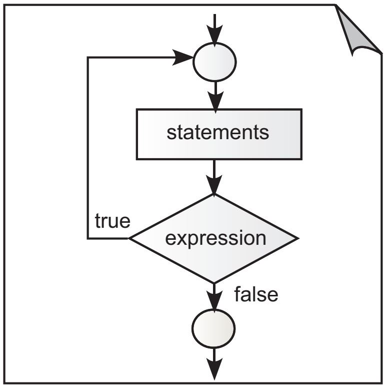
```
          switch ( ch )
       {
          case ' '  :
          case '\t' :
                      wordCount++;
                      break;
          case 'a'  :
          case 'A'  :
          case 'e'  :
          case 'E'  :
          case 'i'  :
          case 'I'  :
          case 'o'  :
          case 'O'  :
          case 'u'  :
          case 'U'  :
                      vowelCount++;
                      break;
       }
    }
    wordCount++;
    printf( "\nCharacter count = %d", characterCount);
    printf( "\nVowel count = %d", vowelCount);
    printf( "\nWord count = %d\n", wordCount);
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Type in the paragraph and terminate by ENTER key
Programming is the way to instruct computer to do a particular task.
Character count = 68
vowel count = 20
word count = 12
```

3.3.3 *do – while* ବିବୃତ୍ତି

do – while ବିବୃତ୍ତି, *while* ବିବୃତ୍ତି ପରି, ସେହି ସମସ୍ୟାଗୁଡ଼ିକ ପାଇଁ ମଧ୍ୟ ଉପଯୁକ୍ତ ଯେଉଁଠାରେ ଏହା ଆଗରୁ ଜଣା ନ ଥାଏ ଯେ କେତେ ଥର ଏକ ବିବୃତ୍ତି କାର୍ଯ୍ୟକାରୀ ହେବ ।



(a) ଡୁ-ହ୍ୱାଇଲ୍ ବିବୃତ୍ତି ର ତର୍କ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ

```
do
{
   Statement
}
while (expression);
```

(b) ଡୁ-ହ୍ୱାଇଲ୍ ବିବୃତ୍ତି ର କୋଡ୍

ଚିତ୍ର . 3.9: do - while ବିବୃତ୍ତିର ଲଜିକ୍ ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ଏବଂ କୋଡ୍

ଯେଉଁଠାରେ ପରିପ୍ରକାଶଟି ଏକ ଧ୍ରୁବକ କିମ୍ବା ଏକ ଚଳ କିମ୍ବା ଏକ ପରିପ୍ରକାଶ ତେବେ ପରିପ୍ରକାଶ 1 କୁ ମୂଲ୍ୟାୟନ କରୁଥିବା ପର୍ଯ୍ୟନ୍ତ ବିବୃତ୍ତି ବାରମ୍ବାର କାର୍ଯ୍ୟ କରୁଥାଏ ।

ଉଦାହରଣ 3.16: ଅଜ୍ଞାତ ସଂଖ୍ୟାଗୁଡ଼ିକର ହାରାହାରି ଗଣନା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ତାଲିକା 3.16

```
#include<stdio.h>
 int main()
 {
    int n = 0;
    char yesno;
    float number, sum = 0, average;
    do {
       printf( "\nEnter number %d: ", n+1 );
       scanf( "%f", &number );
       n++;
       sum += number;
       printf( "\nAny more number [yn]?: " );
       yesno = getchar();
    }
    while ( ( yesno == 'y' ) || ( yesno == 'Y' )  );
    average = sum / n;

    printf( "\n\nAverage of given numbers = %.2f\n", average );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter number 1: 2.5
Any more number [yn]?: y
Enter number 2: 12.0
Any more number [yn]?: y
Enter number 3: 10.25
Any more number [yn]?: y
Enter number 4: 8.75
Any more number [yn]?: y
Enter number 5: 11.0
Any more number [yn]?: n
Average of given numbers = 8.90
```

3.3.4 ନୀଡ଼ନ ହ୍ୱାଇଲ୍, *for* ଏବଂ *do – while* ବିବୃତ୍ତି

ଯେପରି *if* ବିବୃତ୍ତି ନୀଡ଼ନ ହୋଇପାରେ। ଏହି ବିବୃତ୍ତିଗୁଡ଼ିକ ମଧ୍ୟ ନୀଡ଼ନ ହୋଇପାରିବ। ବାହ୍ୟ ଲୁପ୍‌ର ପ୍ରତ୍ୟେକ ଆଇଟରେସନ୍(iteration) ପାଇଁ ଭିତର ଲୁପ୍ ଆରମ୍ଭରୁ ଚାଳନ ହୁଏ।

କୋଡ୍‌ର ନିମ୍ନଲିଖିତ ବିଭାଗଗୁଡ଼ିକ ପ୍ରତ୍ୟେକ ଲୁପିଂ ବିବୃତ୍ତିର ନୀଡ଼ନକୁ ଦର୍ଶାଉଛି।

```
for(i=0;i<m;i++)
{
   :
   for(j=0;j<n;j++)
   {
      :

   }
}
```

```
i=0;
while(i<m)
{
   :
   j=0;
   while(j<n)
   {
      :
     j++;
   }
   i++;
}
```

```
i=0;
do
{
   :
   j=0;
   do
   {
      :
     j++;
   } while(j<n);
   i++;
} while(i<m);
```

ସାଧାରଣତଃ, *for* ବିବୃତ୍ତି ଭିତରେ *while* ଏବଂ *do -while* ବିବୃତ୍ତିଗୁଡ଼ିକୁ, ସେହିପରି *while* ଏବଂ *do -while* ବିବୃତ୍ତିଗୁଡ଼ିକ ଭିତରେ *for* ବିବୃତ୍ତିକୁ ନୀଡ଼ନ କରିବା ସମ୍ଭବ, ଅର୍ଥାତ, ସମସ୍ତ ପ୍ରକାରର ନୀଡ଼ନ ସମାବେଶ ଅନୁମୋଦିତ।

ନୀଡ଼ନ ଯୁକ୍ତ ଲୁପ୍‌ର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରିବା ପାଇଁ ହେଲେ ଆସ ନୀଡ଼ନ for ବ୍ୟବହାର କରି କିଛି ପ୍ରୋଗ୍ରାମ୍ ବିକଶିତ କରିବା।

ଉଦାହରଣ 3.17: ନିମ୍ନଲିଖିତ ଶୈଳୀ ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

```
 1
2 2
3 3 3
```

ତାଲିକା 3.17

```
#include <stdio.h>
 int main()
 {
     int i, j;
     printf("\n");
     for ( i = i; i <= 3; i++ )
     {
         for ( j = 1; j <= i; j++ )
         {
             printf("%4d", i);
         }
         printf("\n");
     }
     return 0;
 }
```

ଉଦାହରଣ 3.18: ନିମ୍ନଲିଖିତ ଶୈଳୀ ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

```
3 3 3
2 2
1
```

ତାଲିକା 3.18

```
#include <stdio.h>
 int main()
 {
     int i, j;
     printf("\n");
     for ( i = 3; i >= 1; i-- )
     {
         for ( j = 1; j <= i; j++ )
         {
             printf("%4d", i);
         }
         printf("\n");
     }
     return 0;
 }
```

ଉଦାହରଣ 3.19: ନିମ୍ନଲିଖିତ ଶୈଳୀ ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

1

1 2

1 2 3

ତାଲିକା 3.19

```
#include <stdio.h>

 int main()
 {
     int i, j;
     printf("\n");
     for ( i = i; i <=3; i++ )
     {
         for ( j = 1; j <= i; j++ )
         {
             printf("%4d", j);
         }
         printf("\n");
     }
     return 0;
 }
```

ଉଦାହରଣ 3.20: ନିମ୍ନଲିଖିତ ଶୈଳୀ ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

```
        *
      *  *
    *  *  *
```

ତାଲିକା 3.20

```
#include <stdio.h>

 int main()
 {
     int i, j;
     printf("\n");
     for ( i = i; i <= 3; i++ )
     {
         for ( j = 1; j <= i; j++ )
         {
```

```
            printf("    *");
        }
        printf("\n");
    }
    return 0;
}
```

3.4 ଜମ୍ପିଂ ବିବୃତ୍ତି (Jumping Statements)

ଏହି ବିବୃତ୍ତିଗୁଡ଼ିକ ପ୍ରୋଗ୍ରାମ୍‌ର ଗୋଟିଏ ଭାଗରୁ ଅନ୍ୟ ଏକ ଭାଗକୁ ସ୍ଥାନାନ୍ତର କରେ। ଏହି ବିଭାଗରେ, ଆମେ ସେଗୁଡ଼ିକର କାର୍ଯ୍ୟ ଦେଖିବା।

3.4.1 break ବିବୃତ୍ତି

break ବିବୃତ୍ତି ସବୁବେଳେ *switch* ବିବୃତ୍ତି, ଏବଂ ଲୁପ୍ ବିବୃତ୍ତି ଭିତରେ ବ୍ୟବହୃତ ହୁଏ।

```
switch ( expression )
{
    case val-1 : statement-1
                 break;
    case val-2 : statement-2
                 break;
    case val-3 : statement-3
                 break;
       :
       :
    case val-n : statement-n
                 break;
    default    :
                statement-d
}
```

ଚିତ୍ର **3.10:** switch ବିବୃତ୍ତି ଭିତରେ break ବିବୃତ୍ତିର କାର୍ଯ୍ୟ

switch ବିବୃତ୍ତିରେ, break ବିବୃତ୍ତିଟି ଶେଷ case ବ୍ୟତୀତ ଅନ୍ୟ ପ୍ରତ୍ୟେକ caseର ଶେଷ ବିବୃତ୍ତି ଭାବରେ ବ୍ୟବହୃତ ହୁଏ। ଯେତେବେଳେ ଚାଳନ ହୁଏ, ଏହା switch ବିବୃତ୍ତିର ବାହାରକୁ ସ୍ଥାନାନ୍ତର କରେ ଏବଂ switch ବିବୃତ୍ତିର ପର ବିବୃତ୍ତିଠାରୁ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନକୁ ଜାରି ରଖେ। for, while ଏବଂ do –while ବିବୃତ୍ତିରେ, ଏହା ସର୍ବଦା if ବିବୃତ୍ତି ସହିତ ମିଳିତ ଭାବରେ ବ୍ୟବହୃତ ହୁଏ। ଧ୍ୟାନ ଦିଅ ଯେ ଲୁପ୍ ବିବୃତ୍ତିର ଏକ ଅଂଶ ହୋଇନଥିଲେ break ବିବୃତ୍ତି, if ବିବୃତ୍ତି ସହିତ ବ୍ୟବହୃତ ହୁଏ ନାହିଁ। ଏହା ଚାଳନା ହେଲେ ଲୁପ୍ ବିବୃତ୍ତିରୁ ନିୟନ୍ତ୍ରଣ ବାହାରକୁ ସ୍ଥାନାନ୍ତର କରେ ଏବଂ ଲୁପିଂ ବିବୃତ୍ତିର ପର ବିବୃତ୍ତିଠାରୁ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନକୁ ଜାରି ରଖେ।

```
for(i=0;i<m;i++)
{
   :
   if (expression)
      break;
   :
}
```

```
i=0;
while(i<m)
{
   :
   if (expression)
      break;
   :
}
```

```
i=0;
do
{
   :
   if (expression)
      break;
   :
} while(i<m);
```

ଚିତ୍ର **3.11:** for, while ଏବଂ do –while ବିବୃତ୍ତି ଭିତରେ break ବିବୃତ୍ତିର କାର୍ଯ୍ୟ

ସରଳ ଶବ୍ଦରେ, ଆମେ କହିପାରିବା ଯେ *break* ର ଚାଳନ ହେଲେ ଲୁପ୍‌ର ଚାଳନ ସମାପ୍ତ ହୁଏ ।

ଉଦାହରଣ 3.21: ଏକ ଲୁପ ଭିତରେ, *break* ବିବୃତ୍ତିର ଉପଯୋଗୀତା ପ୍ରଦର୍ଶନ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ତାଲିକା 3.21

```
#include <stdio.h>
 int main()
 {
    int i;
    for ( i = 1; i < 10; i++ )
    {
        if ( i % 5 == 0 )
            break;
        printf("%d ", i);
    }
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
1 2 3 4
```

break ବିବୃତ୍ତି ସହିତ *if* ବିବୃତ୍ତିର ବ୍ୟବହାର *for* ଲୁପ୍‌କୁ ସେତେବେଳେ ସମାପ୍ତ କରେ, ଯେତେବେଳେ *i* ର ମୂଲ୍ୟ 5 ଦ୍ୱାରା ବିଭାଜ୍ୟ ହୁଏ।

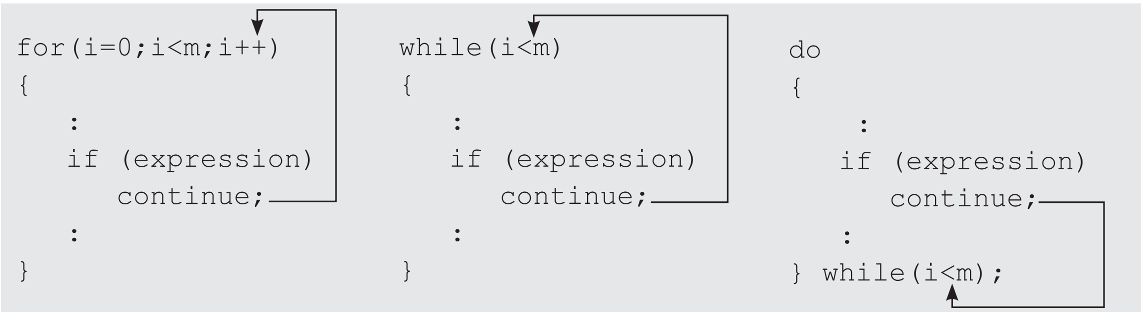
3.4.2 continue ବିବୃତ୍ତି

continue ବିବୃତ୍ତି ସର୍ବଦା ଲୁପିଂ ବିବୃତ୍ତି ଭିତରେ ବ୍ୟବହୃତ ହୁଏ । ଏମିତି ପରିସ୍ଥିତି ଥାଇପାରେ ଯେତେବେଳେ ଆମେ ଚାହୁଁ ଯେ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ବିବୃତ୍ତି ପରଠାରୁ, ଲୁପ୍‌ର ଶେଷ ବିବୃତ୍ତି ପର୍ଯ୍ୟନ୍ତ ସମସ୍ତ ବିବୃତ୍ତିଗୁଡ଼ିକୁ ଏଡ଼ାଇ ଦିଆଯିବା ଉଚିତ। ଏହି କାର୍ଯ୍ୟ *continue* ବିବୃତ୍ତି ବ୍ୟବହାର କରି ସମ୍ପନ୍ନ ହୁଏ ।

continue ବିବୃତ୍ତି, ନିୟନ୍ତ୍ରଣକୁ ଲୁପ୍‌ର ପରବର୍ତ୍ତୀ ପୁନରାବର୍ତ୍ତନକୁ (iteration) ଆରମ୍ଭକୁ ସ୍ଥାନାନ୍ତର କରେ, ତେଣୁ

ଏପର୍ଯ୍ୟନ୍ତ କାର୍ଯ୍ୟକାରୀ ହୋଇ ନ ଥିବା ବିବୃତ୍ତିଗୁଡ଼ିକୁ ବାଇପାସ୍ (bypass) କରାଯାଏ । ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ *continue* ବିବୃତ୍ତି ସର୍ବଦା if ବିବୃତ୍ତି ସହ ବ୍ୟବହୃତ ହୁଏ ।

ସରଳ ଶବ୍ଦରେ, ଆମେ କହିପାରିବା ଯେ continue ବିବୃତ୍ତି ଚାଳନ ହେଲେ ଲୁପ୍‌ର ସାମ୍ପ୍ରତିକ ଆଇଟରେସନ୍ ସମାପ୍ତ ହୋଇଥାଏ।



ଚିତ୍ର **3.12**: *for, while* ଏବଂ *do -while* ବିବୃତ୍ତି ଗୁଡ଼ିକରେ *continue* ବିବୃତ୍ତିର କାର୍ଯ୍ୟକାରୀତା

ଉଦାହରଣ 3.22: ଏକ ଲୁପ ଭିତରେ, *continue* ବିବୃତ୍ତିର ଉପଯୋଗୀତା ପ୍ରଦର୍ଶନ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ତାଲିକା 3.22

```
#include <stdio.h>
int main()
{
   int i;
   for ( i = 1; i < 10; i++ )
   {
      if ( i % 5 == 0 )
         continue;
      printf("%d ", i);
   }
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
1 2 3 4 6 7 8 9
```

if ବିବୃତ୍ତି ସହିତ *continue* ବିବୃତ୍ତିର ବ୍ୟବହାରଦ୍ୱାରା *for* ଲୁପ୍‌ରେ continue ବିବୃତ୍ତିର ପରଠାରୁ ବାକି ସବୁ ବିବୃତ୍ତିକୁ ସେତେବେଳେ ଏଡ଼ାଇ ଦିଆଯାଏ, ଯେତେବେଳେ i ର ମୂଲ୍ୟ 5 ଦ୍ୱାରା ବିଭାଜ୍ୟ ହୁଏ ।

ଲୁପ୍ ପାଇଁ ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 3.23: ପ୍ରଦତ୍ତ ସଂଖ୍ୟା ଏବଂ ଧାଡି ସଂଖ୍ୟାପାଇଁ ଏକ ଗୁଣନ ଖନ୍ଦା ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଯଦି ସଂଖ୍ୟା 5 ଏବଂ ଧାଡି ସଂଖ୍ୟା = 3, ତେବେ ଆଉଟପୁଟ୍ ହେବା ଉଚିତ୍:

$$5 \times 1 = 5$$
$$5 \times 2 = 10$$
$$5 \times 3 = 15$$

ତାଲିକା 3.23

```
#include <stdio.h>
 int main()
 {
    int num, rows, i, j;
    printf( "\nEnter number whose table to print : " );
    scanf( "%d", &num );
    printf( "\nEnter rows to print : " );
    scanf( "%d", &rows );
    for ( i = 1; i <= rows; i++ )
    {
        printf("\n%d x %d = %d", num, i, num*i) ;
    }
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter number whose table to print : 5
Enter rows to print : 3
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
```

ଉଦାହରଣ 3.24: ପ୍ରଦତ୍ତ ଗଣନ ସଂଖ୍ୟା n ଏକ ମୌଳିକ ସଂଖ୍ୟା କି ନୁହେଁ ଜାଣିବା ପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ଏକ ଗଣନ ସଂଖ୍ୟା ଯଦି କେବଳ 1 ଏବଂ ନିଜ ଦ୍ୱାରା ବିଭାଜ୍ୟ ହୁଏ ତେବେ ଏହାକୁ ମୌଳିକ ସଂଖ୍ୟା କୁହାଯାଏ, ଅର୍ଥାତ, ଏହାକୁ ବିଭାଜିତ କରାଯାଇପାରିବ ନାହିଁ। ଏହା ବ୍ୟତୀତ, ଏହି ସଂଜ୍ଞା ଅନୁଯାୟୀ 2 ବ୍ୟତୀତ ଅନ୍ୟ କୌଣସି ଯୁଗ୍ମ ସଂଖ୍ୟା ଏକ ମୌଳିକ ସଂଖ୍ୟା ନୁହେଁ।

ତେଣୁ, ଆମ ପରୀକ୍ଷଣର ମାନଦଣ୍ଡ ହେଉଛି –

1. ଯଦି n, 2 ରୁ ବଡ ଏବଂ ଯୁଗ୍ମ ସଂଖ୍ୟା ଅଟେ ତେବେ n ଏକ ମୌଳିକ ସଂଖ୍ୟା ନୁହେଁ।
2. ଯଦି ଷ୍ଟେପ୍ 1 ରେ ପରୀକ୍ଷା ବିଫଳ ହୁଏ, ତେବେ ଆମେ ସଂଖ୍ୟା n କୁ k = 3, 5, 7, ... ସଂଖ୍ୟକ ଭାଜକ

ଦ୍ୱାରା ଭାଗ କରିବାକୁ ଚେଷ୍ଟା କରୁ, ଏବଂ ଯଦି n, k ର କୌଣସି ମୂଲ୍ୟ ଦ୍ୱାରା ବିଭାଜ୍ୟ ହୁଏ, ତେବେ n ଏକ ମୌଳିକ ସଂଖ୍ୟା ନୁହେଁ।

3. ଯଦି ଷ୍ଟେପ୍ 2 ରେ ମଧ୍ୟ ପରୀକ୍ଷା ବିଫଳ ହୁଏ, ତେବେ n ଏକ ମୌଳିକ ସଂଖ୍ୟା।

ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍ ଏହି ମାନଦଣ୍ଡକୁ କାର୍ଯ୍ୟକାରୀ କରେ।

ତାଲିକା 3.24

```
#include<stdio.h>
 #include<math.h>
 int main()
 {
    int n, k, m;
    printf( "\nEnter a positive integer number: " );
    scanf("%d", &n);
    if ( ( n > 2 ) && ( ( n % 2 ) == 0 ) )
    {
       printf( "\n%d is not a prime number.\n", n );
       return 0;
    }
    m = sqrt( n );
    for ( k = 3; k <= m; k += 2 )
    {
        if ( n % k == 0 )
        {
            printf( "\n%d is not a prime number.\n", n );
            return 0;
        }
    }
    printf( "\n%d is a prime number.\n", n );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
First Run
  Enter a positive integer number: 43
  43 is a prime number.
Second Run
  Enter a positive integer number: 92
  92 is not a prime number.
```

ଉଦାହରଣ 3.25: 1991 ସଂଖ୍ୟା ଏକ ପାଲିଣ୍ଡ୍ରୋମ୍ (palindrome) କାରଣ ବାମରୁ କିମ୍ବା ଡାହାଣରୁ ପଢିଲେ ଏହା ସମାନ ସଂଖ୍ୟା ହୁଏ । ଦିଆଯାଇଥିବା ସଂଖ୍ୟାଟି ଏକ ପାଲିଣ୍ଡ୍ରୋମ୍ କି ନୁହେଁ ତାହା ଜାଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ଆମେ ପ୍ରଥମେ ଦିଆଯାଇଥିବା ସଂଖ୍ୟାର ଅଙ୍କକୁ ଓଲଟାଇ ଏକ ନୂତନ ସଂଖ୍ୟା ଗଠନ କରୁ ଏବଂ ତା'ପରେ ଦିଆଯାଇଥିବା ସଂଖ୍ୟାକୁ ଓଲଟା ନମ୍ବର ସହିତ ତୁଳନା କରୁ । ଯଦି ସେଗୁଡ଼ିକ ମେଳ ଖାଆନ୍ତି, ତେବେ ଦିଆଯାଇଥିବା ସଂଖ୍ୟା ପାଲିଣ୍ଡ୍ରୋମ୍ ହେବ । ଅନ୍ୟଥା ଏହା ଏକ ପାଲିଣ୍ଡ୍ରୋମ୍ ନୁହେଁ ।

ତାଲିକା 3.25

```
#include<stdio.h>
 int main()
 {
    int sum = 0, digit;
    int number, temp;
    printf( "\nEnter any positive integer number: " );
    scanf( "%d", &number );
    temp = number;

    while ( temp > 0 )
    {
       digit = temp % 10;
       temp /= 10;
       sum = sum * 10 + digit;
    }
    if ( number == sum )
       printf( "\n%d is a palindrome number.\n", number );
    else
       printf( "\n%d is not a palindrome number.\n", number );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
First Run
   Enter any positive integer number: 1991
   1991 is a palindrome number.
Second Run
   Enter any positive integer number: 1234
   1234 is not a palindrome number.
```

ଉଦାହରଣ 3.26: ଦିଆଯାଇଥିବା ଗଣନ ସଂଖ୍ୟା ଏକ ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ (Armstrong) ନମ୍ବର କି ନୁହେଁ ଜାଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ ନମ୍ବର ହେଉଛି ଏକ 3 ଅଙ୍କ ବିଶିଷ୍ଟ ସଂଖ୍ୟା ଯାହାର ଅଙ୍କଗୁଡ଼ିକର ଘନର ସମଷ୍ଟି ନିଜେ ସଂଖ୍ୟା ସହିତ ସମାନ । ଉଦାହରଣ ସ୍ୱରୂପ, 153 ନମ୍ବର ହେଉଛି ଏକ ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ ନମ୍ବର ।

$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

ତାଲିକା 3.26

```
#include<stdio.h>
 int main()
 {
   int n, t, s = 0, d;
   printf( "\nEnter 3-digit natural number : " );
   scanf("%d", &n);
   t = n;
   while ( t > 0 )
   {
      d = t % 10;
      s = s + d * d * d;
      t = t / 10;
   }
   if ( s == n )
      printf("\n%d is an Armstrong number.\n", n);
   else
      printf( "\n%d is not an Armstrong number,\n", n);
   return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
First Run
   Enter 3-dgit natural number : 153
   153 is an Armstrong number.
Second Run
   Enter 3-dgit natural number : 135
   135 is not an Armstrong number.
```

ଉଦାହରଣ 3.27: ଏକ ଫିବୋନାକି କ୍ରମ ନିମ୍ନଲିଖିତ ଭାବରେ ବ୍ୟାଖ୍ୟା ହୋଇଛି: କ୍ରମରେ ପ୍ରଥମ ଏବଂ ଦ୍ୱିତୀୟ ପଦଗୁଡ଼ିକ ହେଉଛି 0 ଏବଂ 1 । ପରବର୍ତ୍ତୀ ପଦଗୁଡ଼ିକ ପୂର୍ବବର୍ତ୍ତୀ ଦୁଇଟି ପଦକୁ ମିଶାଇ ମିଳିଥାଏ । ଉଦାହରଣ ସ୍ୱରୂପ, ଫିବୋନାକି କ୍ରମର ପ୍ରଥମ 10 ଟି ସର୍ତ୍ତାବଳୀ ହେଉଛି

0 1 1 2 3 5 8 13 21 34

କ୍ରମର ପ୍ରଥମ n ସର୍ତ୍ତାବଳୀ ସୃଷ୍ଟି କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

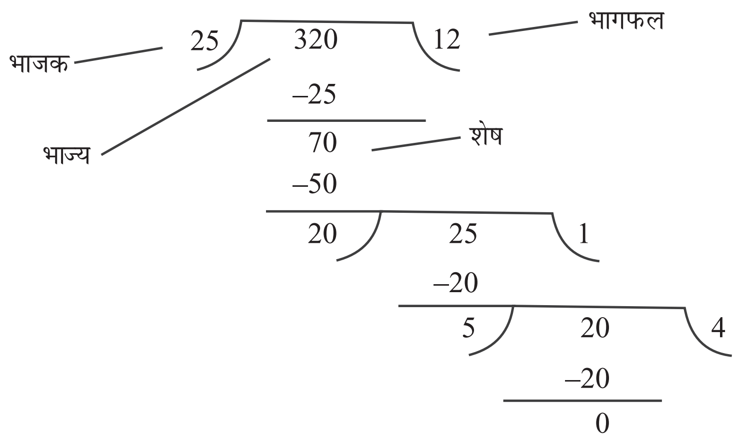
ତାଲିକା 3.27

```
#include<stdio.h>
 int main()
 {
    int prev = 0, curr = 1, next, n, count;
    printf( "\nEnter value for n(>2) : " );
    scanf( "%d", &n );
    printf( "%d  %d", prev, curr );
    count = 2;
    while ( count < n )
    {
        next = prev + curr;
        printf( "%d  ", next );
        count++;
        previ = curr;
        curr = next;
    }
    printf( "\n" );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter value for n(>2) : 10
0  1  1  2  3  5  8  13  21  34
```

ଉଦାହରଣ 3.28: ନିମ୍ନ ଚିତ୍ରଟି ଦୁଇଟି ଧନାତ୍ମକ ସଂଖ୍ୟା 25 ଏବଂ 320 ର ଗରିଷ୍ଠ ସାଧାରଣ ଗୁଣନୀୟକ (GCD) ଲମ୍ବା ବିଭାଜନ ପଦ୍ଧତି ବ୍ୟବହାର କରି, ଗଣନା କରିବାର ଉପାୟ ପ୍ରଦର୍ଶନ କରୁଛି ।



ଚିତ୍ର **3.13:** ଲମ୍ବା ବିଭାଜନ ବ୍ୟବହାର କରି GCD ଗଣନା ପ୍ରକ୍ରିୟାର ଚିତ୍ର

ଉପରୋକ୍ତ ଚିତ୍ରରୁ, ତୁମେ ଦେଖୁଥିବ ଯେ କ୍ରମାଗତ ବିଭାଜନରେ, ପୂର୍ବ ବିଭାଜନର ଭାଜକଟି, ଭାଜ୍ୟ ହୋଇଯାଏ, ଭାଗଶେଷଟି, ଭାଜକ ହୋଇଯାଏ, ଏବଂ ବିଭାଜନ ପ୍ରକ୍ରିୟା ପୁନର୍ବାର କରାଯାଏ । ଭାଗଶେଷ ଶୂନ ହେବା ପର୍ଯ୍ୟନ୍ତ ଏହି ପ୍ରକ୍ରିୟା କରାଯାଏ, ଏବଂ ଶେଷ ଭାଜକକୁ ଦିଆଯାଇଥିବା ସଂଖ୍ୟାଗୁଡ଼ିକର ଗରିଷ୍ଠ ସାଧାରଣ ଗୁଣନୀୟକ (GCD) ଭାବରେ ନିଆଯାଏ ।

ଦୁଇଟି ଯୁକ୍ତ ସଂଖ୍ୟା m ଏବଂ n ର ଗ.ସା.ଗୁ. (GCD) ବାହାର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 3.28

```
#include<stdio.h>
 void main()
 {
    int m, n, r;
    printf( "\nEnter value for m : " );
    scanf( "%d", &m );
    printf( "Enter value for n : " );
    scanf( "%d", &n );
    while ( 1 )
    {
       r = n % m;
       if ( r == 0 )
       {
          printf( "\nGCD = %d\n", n );
          return;
       }
       n = m;
       m = r;
    }
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter value for m : 25
Enter value for n : 320
GCD = 5
```

ୟୁନିଟ୍ ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଶିଖିଛୁ ଯେ

- ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତିର ବ୍ୟବହାରଦ୍ୱାରା ବିବୃତ୍ତିଗୁଡ଼ିକର ଚାଳନର କ୍ରମକୁ ନିୟନ୍ତ୍ରଣ କରି ହୁଏ ।
- ବିବୃତ୍ତିର ଏହି ନିୟନ୍ତ୍ରଣ, ବୈକଳ୍ପିକ ବିବୃତ୍ତିଗୁଡ଼ିକ ମଧ୍ୟରୁ ଗୋଟିଏ ବିବୃତ୍ତିର ଚୟନ କିମ୍ବା କିଛି ମନୋନୀତ ବିବୃତ୍ତିର ବାରମ୍ବାର ଚାଳନ ଅନ୍ତର୍ଭୁକ୍ତ ।
- if-else ଏବଂ switch ଉକ୍ତିଗୁଡିକୁ ନିଷ୍ପତ୍ତି (decision making) ବିବୃତ୍ତି କୁହାଯାଏ କାରଣ ସେଗୁଡ଼ିକ ଦିଆଯାଇଥିବା ସର୍ତ୍ତର ଫଳାଫଳ ଉପରେ ନିର୍ଭର କରି ବିକଳ୍ପଗୁଡ଼ିକ ମଧ୍ୟରୁ ଗୋଟିଏ ବିବୃତ୍ତି ବା ବିବୃତ୍ତି ଗ୍ରୁପ୍ ନିର୍ବାଚନ କରିବାକୁ ଅନୁମତି ଦିଅନ୍ତି ।
- ଏକ ସର୍ତ୍ତ କେବଳ ଏକ ପୂର୍ଣ୍ଣାଙ୍କ ମୂଲ୍ୟ ନେଇଥିଲେ switch ବିବୃତ୍ତି କେବଳ ସେହି ପରିସ୍ଥିତିରେ କାର୍ଯ୍ୟ କରେ ।
- for, while ଏବଂ do – while ବିବୃତ୍ତିଗୁଡ଼ିକୁ ଆଇଟରେଟିଭ୍ ବା ଲୁପିଂ ବିବୃତ୍ତି କୁହାଯାଏ କାରଣ ସେଗୁଡ଼ିକ ମନୋନୀତ ବିବୃତ୍ତିଗୁଡ଼ିକୁ ପୁନରାବୃତ୍ତି(iterate) କରିବାକୁ ଅନୁମତି ଦେଇଥାଏ ।
- for ବିବୃତ୍ତିକୁ ଏପରି ପରିସ୍ଥିତିରେ ପସନ୍ଦ କରାଯାଏ ଯେଉଁଥିରେ ଆମେ ବିବୃତ୍ତି ଚାଳନ କେତେଥରପାଇଁ ପୁନରାବୃତ୍ତି ହେବ ତାହା ଆଗୁଆ ଜାଣିଥାଉ ।
- while ଏବଂ do – while ବିବୃତ୍ତିଗୁଡ଼ିକ ପସନ୍ଦ କରାଯାଏ ଯେତେବେଳେ ବିବୃତ୍ତିଗୁଡ଼ିକର ଚାଳନ କେତେଥରପାଇଁ ପୁନରାବୃତ୍ତି ହେବ ତାହା କିଛି ସର୍ତ୍ତ ପୂରଣ ଉପରେ ନିର୍ଭର କରେ ।
- while ଏବଂ do – while ବିବୃତ୍ତିଗୁଡ଼ିକ ମଧ୍ୟରେ ଏକମାତ୍ର ପାର୍ଥକ୍ୟ ହେଉଛି do - while ବିବୃତ୍ତିଟି ସର୍ବଦା ଅତି କମରେ ଥରେ ଚାଳନ ହୁଏ ଫଳରେ କି while ଲୁପ୍‌ଟି ଚାଳନ ହୋଇପାରେ କିମ୍ବା ଆଦୌ ହୋଇ ନ ପାରେ।
- switch ବିବୃତ୍ତି ଏବଂ ସମସ୍ତ ଲୁପିଂ ବିବୃତ୍ତି ସହିତ break ବିବୃତ୍ତି ବ୍ୟବହୃତ ହୁଏ । ଏହାର ଚାଳନ ଅନୁଯାୟୀ, ଏହା ନିୟନ୍ତ୍ରଣକୁ ବ୍ଲକ୍‌ର ବାହାରକୁ ସ୍ଥାନାନ୍ତର କରେ ।
- break ବିବୃତ୍ତି ଯେତେବେଳେ switch ବିବୃତ୍ତି ସହିତ ବ୍ୟବହାର ହୁଏ, ଏହା ନିୟନ୍ତ୍ରଣକୁ switch ବିବୃତ୍ତି ପରିସର ବାହାରକୁ ନେଇ ଯାଇଥାଏ ।
- ଯେତେବେଳେ ଲୁପିଂ ବିବୃତ୍ତି ସହ ବ୍ୟବହୃତ ହୁଏ, break ବିବୃତ୍ତି ଲୁପ୍ ବାହାରକୁ ନିୟନ୍ତ୍ରଣ ନେଇ ଯାଇଥାଏ, ଅର୍ଥାତ, ଏହା ଲୁପ୍‌ର ଚାଳନକୁ ଶେଷ କରିଦିଏ ।
- continue ବିବୃତ୍ତି କେବଳ ଲୁପିଂ ବିବୃତ୍ତି ସହ ବ୍ୟବହୃତ ହୁଏ । ଏହାର ଚାଳନ ହେଲେ ଏହାକୁ ଅନୁସରଣ କରୁଥିବା ବିବୃତ୍ତିଗୁଡ଼ିକୁ ଚାଳନରୁ ବାଦ୍‌ଦିଏ, ଅର୍ଥାତ, ଏହା ସମ୍ପ୍ରତି ଚାଳନ ହେଉଥିବା ଆଇଟରେସନ୍‌କୁ ସମାପ୍ତ କରି ପରବର୍ତ୍ତୀ ଆଇଟରେସନ୍‌କୁ ଆରମ୍ଭ କରିଥାଏ ।

ଅଭ୍ୟାସ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. ଯେତେବେଳେ *if* ବିବୃତ୍ତିରେ କୌଣସି else ଜଡିତ ନାହିଁ, ତେବେ ସେତେବେଳେ କ'ଣ ହୁଏ ଯେବେ ସର୍ତ୍ତର ଶୂନ୍ୟ ମୂଲ୍ୟାଙ୍କନ ହୋଇଥାଏ ?
2. *if-else* ବିବୃତ୍ତି ଅପେକ୍ଷା *switch* ବିବୃତ୍ତିର ସୁବିଧା କ'ଣ?
3. ନୀଡ଼ନର*(nesting)* ଅର୍ଥ କ'ଣ?
4. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ରେ କିଛି ଭୁଲ୍ ଅଛି କି?

```
void main()
{
    char ch;
    if ( ( ch = getche() ) == 'a' )
        printf("\nYou typed character a\n");

}
```

5. *break* ବିବୃତ୍ତି କେଉଁଠାରେ ବ୍ୟବହୃତ ହୁଏ , ଏବଂ ଏହାର କାର୍ଯ୍ୟ କ'ଣ ?
6. *switch* ବିବୃତ୍ତି ବ୍ୟବହାର କରି ନିମ୍ନଲିଖିତ କୋଡ୍‌ଖଣ୍ଡକୁ ପୁନର୍ଲିଖନ କର:

```
char code;
code = getchar();
if ( code == 'A' )
   printf("\nAccountant\n");
else if ( code == 'C' || code == 'G' )
   printf("\nGrade IV\n");
else if ( code == 'F' )
   printf("\nFinancial Advisor\n");
else
   printf("\nIncorrect code\n");
```

7. *if-else* ବିବୃତ୍ତି ବ୍ୟବହାର କରି ନିମ୍ନଲିଖିତ କୋଡ୍‌ଖଣ୍ଡକୁ ପୁନର୍ଲିଖନ କର

```
int month;
scanf("%d", &month);
switch ( month )
{
   case 1 :
   case 3 :
```

```
        case 5 :
        case 7 :
        case 8 :
        case 10:
        case 12: printf("\nIt is a month having 31 days\n");
                break;
        case 4 :
        case 6 :
        case 9 :
        case 11: printf("\nIt is a month having 30 days\n");
                break;
        case 2:  printf("\nIt is a month having 28/29 days\n");
        default: printf("\nIncorrect month\n");

    }
```

8. switch ବିବୃତ୍ତି ବ୍ୟବହାର କରି ନିମ୍ନଲିଖିତ କୋଡ୍ ଖଣ୍ଡକୁ ପୁନର୍ଲିଖନ କର:

```
if ( ch == 'N' )            if (ch == 'O')
    north++;                    Outstanding++;
if ( ch == 'S' )          else if (ch == 'E')
    south++;                    Excellent++;
if ( ch == 'E' )          else if (ch == 'G')
    east++;                       Good++ ;
if ( ch == 'W' )          else if (ch == 'P')
    west++;                       Poor++;
 else                       else
   unknown++;                  Unknown++ ;
```

9. *switch* ଏବଂ *if-else* ର ଅପରେସନ୍ (operation) ଅନୁଯାୟୀ ଯେକୌଣସି ଦୁଇଟି ପାର୍ଥକ୍ୟ ଲେଖ ।
10. ଗୋଟିଏ *switch* ବିବୃତ୍ତି ରେ *break* ବିବୃତ୍ତି ଅନୁପସ୍ଥିତିର ପ୍ରଭାବ କ'ଣ?
11. ଯଦି ଗୋଟିଏ ପରିପ୍ରକାଶ ପ୍ରାରମ୍ଭରୁ ମିଥ୍ୟା ହୁଏ ତେବେ କ'ଣ ହେବ?
12. *while* ଏବଂ *do - while* ବିବୃତ୍ତି ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ କ'ଣ?
13. କେତେବେଳେ *while* ଅପେକ୍ଷା *do - while* ବିବୃତ୍ତିକୁ ଅଧିକ ପସନ୍ଦ କରାଯାଏ?
14. for ବିବୃତ୍ତିର ବୃଦ୍ଧି ଭାଗରେ ଏକାଧିକ ବୃଦ୍ଧି-ପରିପ୍ରକାଶ ରଖା ଯାଇପାରିବ କି?
15. କେତେବେଳେ *for* ଅପେକ୍ଷା *while* ବିବୃତ୍ତିକୁ ଅଧିକ ପସନ୍ଦ କରାଯାଏ?

16. କେଉଁଠାରେ *break* ବିବୃତ୍ତି ବ୍ୟବହୃତ ହୁଏ, ଏବଂ ଏହାର କାର୍ଯ୍ୟ କ'ଣ ?

17. *continue* ବିବୃତ୍ତିର କାର୍ଯ୍ୟ କ'ଣ?

18. *break* ଏବଂ *continue* ବିବୃତ୍ତି ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଦର୍ଶାଅ?

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. ନିମ୍ନଲିଖିତ କୋଡ୍‌ଖଣ୍ଡକୁ ବିଚାର କର:

```
switch(ch)
{
   case 'a': printf("a");
   case 'b': printf("b");
   default:  printf("error");

}
```

ଯଦି ଅକ୍ଷର ଚଳ ch କୁ ମୂଲ୍ୟ 'a' ଦିଆଯାଏ ତା'ପରେ ଆଉଟପୁଟ୍ (output) ହେବ ?

(*a*) a (*b*) ab

(*c*) aberror (*d*) error

2. ନିମ୍ନଲିଖିତ କୋଡ୍‌ଖଣ୍ଡକୁ ବିଚାର କର:

```
int a = 6, b = 6;

  if ( b == 6 || --a )

{

  statement1;

  statement2;

}
```

ଧରାଯାଉ ବିବୃତ୍ତି 1 ଏବଂ ବିବୃତ୍ତି 2, a ଏବଂ b ମୂଲ୍ୟ ପରିବର୍ତ୍ତନ କରେ ନାହିଁ, ତେବେ ଉପରୋକ୍ତ କୋଡ ଚାଳନ ପରେ a ର ମୂଲ୍ୟ କ'ଣ ହେବ?

(a) 6 (b) 5

(c) 7 (d) ତ୍ରୁଟି(error)

3. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଉକ୍ତି switch ବିବୃତ୍ତି ବିଷୟରେ True ଅଟେ?

(a) ଏଥିରେ ଶୂନ କିମ୍ବା ଅଧିକ ମାମଲା (case) ରହିପାରେ ।

(b) ଧ୍ରୁବକ ପରିପ୍ରକାଶ ମୂଲ୍ୟ ହେଉଛି ବୈଧ ମାମଲା ।

(c) ନିୟନ୍ତ୍ରଣ ପରବର୍ତ୍ତୀ ମାମଲାଗୁଡ଼ିକୁ ନ ଯିବାପାଇଁ, ପ୍ରତ୍ୟେକ ମାମଲା ବିବୃତ୍ତି ବ୍ଲକ୍‌ରେ break ବିବୃତ୍ତି ଶେଷ ବିବୃତ୍ତି ଭାବରେ ନିଶ୍ଚିତ ରହିବା ଆବଶ୍ୟକ ।

(d) ଉପରୋକ୍ତ ସମସ୍ତ।

4. ଯଦି switch ବିବୃତ୍ତିରେ ଡିଫଲ୍ଟ(default) ବିବୃତ୍ତିକୁ ବାଦ ଦିଆଯାଏ ଏବଂ ମାମଲା ମୂଲ୍ୟସହିତ ଏହାର କୌଣସି ମେଳ ନାହିଁ, ତେବେ

   (a) switch ବ୍ଲକରେ କୌଣସି ବିବୃତ୍ତି ଚାଳନ ହୋଇ ନ ଥାଏ ।

   (b) switch ବ୍ଲକରେ ସମସ୍ତ ବିବୃତ୍ତି ଚାଳନ ହୋଇଥାଏ ।

   (c) କେବଳ ଶେଷ ମାମଲା ବ୍ଲକ୍‌ର ଉକ୍ତିଗୁଡ଼ିକ ଚାଳନ ହୋଇଥାଏ ।

   (d) ଏକ ଚାଳନ ସମୟ(run time) ତ୍ରୁଟି ଘଟିଥାଏ।

5. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର କମ୍ପାଇଲ୍ ଏବଂ ଚାଳନ (run) ଚେଷ୍ଟା କରିବାର ଫଳାଫଳ କ'ଣ ହେବ?

```
#include <stdio.h>
void main()
{
  int c;
  printf( "\nEnter value of c: " );
  scanf( "%d", &c);
  switch(c);
{
case 1 : printf( "\nHello 1\n" );
         break;
case 2 : printf( "\nHello 2\n" );
         break;
default : printf( "\nInvaild value\n" );
         break;
case 3 : printf( "\nHello 3\n" );
         break;
case 4 : printf( "\nHello 4\n" );
}
}
```

   (a) ପ୍ରୋଗ୍ରାମ୍‌ଟି କମ୍ପାଇଲ୍ କରିବାରେ ବିଫଳ ହେବ କାରଣ default କେବଳ ସମସ୍ତ ବୈଧ ମାମଲାଗୁଡ଼ିକର ପରେ ଶେଷରେ ଦୃଶ୍ୟମାନ ହୋଇପାରିବ।

   (b) ପ୍ରୋଗ୍ରାମ୍‌ଟି କମ୍ପାଇଲ୍ ଏବଂ ଚାଳନ ହେବ ।

   (c) ପ୍ରୋଗ୍ରାମ୍‌ଟି କମ୍ପାଇଲ୍ ହେବାରେ ବିଫଳ ହେବ ଏବଂ କମ୍ପାଇଲର୍ ସମସ୍ତ ମାମଲା ଏବଂ break ବିବୃତ୍ତି ପାଇଁ ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ରିପୋର୍ଟ୍ କରିବ ଯେ "Case outside of switch statement in function main" ତାପରେ "Misplaced break in function main" ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ରିପୋର୍ଟ୍ କରିବ ।

   (d) ଉପରୋକ୍ତ କୌଣସିଟି ନୁହେଁ।

6. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌କୁ ବିଚାର କର

```
#include <stdio.h>
void main()
{
  int a, b = 10;
  scanf( "%d", &a );
  if ( a = 0 )
    b *= 2;
  else
      b /= 2;
   printf( "%d", b );
}
```

ବ୍ୟବହାରକାରୀର ଇନପୁଟ୍ ସଂଖ୍ୟା 6 ହେଲେ, ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନର ଫଳାଫଳ କ'ଣ ହେବ?

(*a*) 20 (*b*) 10
(*c*) 5 (*d*) କୌଣସିଟି ନୁହେଁ

7. switch ବିବୃତ୍ତିରେ ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ବୈଧ ପରିପ୍ରକାଶ ନୁହେଁ?

(a) ଅକ୍ଷର(character) (b) ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା (integer)
(c) ଫ୍ଲୋଟ୍ (float) (d) ଏନମ୍ (enum)

8. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ (output) କ'ଣ ହେବ?

```
void main() {
  char val=1;
   if(val--==0)
     printf("TRUE");
  else
     printf("FALSE");
}
```

(a) ସତ (TRUE) (b) ମିଥ୍ୟା(FALSE)
(c) ତ୍ରୁଟି (Error) (d) ଉପରୋକ୍ତ କୌଣସିଟି ନୁହେଁ

9. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ ?

```
#define TRUE 1
void main()
{
    if(TRUE)
        printf("1");
        printf("2");
    else
        printf("3");
        printf("4");
}
```

(*a*) ତ୍ରୁଟି (Error) (*b*) 1

(*c*) 12 (*d*) 34

10. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
   int a=10;
   switch(a) {
      case 5+5:
         printf("Hello\n");
      default:
         printf("OK\n");
}
}
```

(*a*) Hello (*b*) OK

(*c*) Hello OK (*d*) ତ୍ରୁଟି (Error)

11. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
   int a=2;
   switch(a)
   {
      printf("Message\n");
      default:
         printf("Default\n");
      case 2:
         printf("Case-2\n");
     case 3:
         printf("Case-3\n");
   }
   printf("Exit from switch\n");
}
```

(*a*) Case-2

(*b*) Message

(*c*) Message Case-2

(*d*) Case-2 Case-3 Exit from switch

12. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
    short day=2;
    switch(day)
    {
        case 2: || case 22:
            printf("%dnd",day);
        break;
```

```
            default:
                printf("%dth",day);
            break;
        }
    }
```

(*a*) 2nd (*b*) 22nd

(*c*) Error (*d*) 2nd 22nd

13. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
  int a=2;
  switch(a)
  {
    case 1L:
      printf("One\n"); break;
    case 2L:
      printf("Two\n"); break;
    default:
      printf("Else\n"); break;
  }
}
```

(*a*) One (*b*) Two

(*c*) Error (*d*) Else

14. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
#define TRUE 1
void main()
{
```

```
    switch(TRUE)
    {
        printf("Hello");
    }
}
```

(a) Hello (b) କୌଣସି ଆଉଟପୁଟ୍ ନାହିଁ

(c) ଅନିର୍ଦ୍ଦିଷ୍ଟ ମୂଲ୍ୟ (garbage) (d) ତ୍ରୁଟି

15. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
   int x;
   float y = 5.5;
   switch(x=y+1)
   {
      case 6: printf("It's Eight."); break;
      default: printf("Oops No choice here.");
   }
}
```

(*a*) Oops No choice here. (*b*) It's Eight. Oops No choice here.

(*c*) It's Eight (*d*) =kqfV

16. ନିମ୍ନଲିଖିତ କୋଡ୍ ବିଚାର କର:

```
while ( ++i <= n );
```

i ର ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ 1 ହେଲେ, ଲୁପ୍ ସମାପ୍ତ ହେଲାବେଳକୁ i ର ମୂଲ୍ୟ କ'ଣ ହେବ ?

(*a*) n (*b*) $n - 1$

(*c*) $n + 1$ (*d*) $n + 2$

17. ନିମ୍ନଲିଖିତ କୋଡ୍ ବିଚାର କର। ଏହାର ଉତ୍ପାଦନ କ'ଣ?

```
for ( i = 1;i <= 5; i++ )
{
   printf( "%d", i += 2 );
}
```

(*a*) 1 2 3 4 5 (*b*) 3 4 5 6 7

(*c*) 3 5 7 9 11 (*d*) 3 6

18. continue ବିବୃତ୍ତି ବିଷୟରେ ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଉକ୍ତିଟି ସତ୍ୟ ନୁହେଁ?

    (a) ଏହା switch ସହ ବ୍ୟବହାର କରାଯାଇପାରିବ ।

    (b) ଏହା ଲୁପ୍‌କୁ ସମାପ୍ତ କରେ ।

    (c) ଏହା ବର୍ତ୍ତମାନର ଆଇଟରେସନ୍‌କୁ ସମାପ୍ତ କରେ।

    (d) ଏହା ସମସ୍ତ ଲୁପିଂ ବିବୃତ୍ତି ଆଉ switch ସହିତ ମଧ୍ୟ ବ୍ୟବହୃତ ହୋଇପାରିବ ।

19. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଉକ୍ତି switch ବିବୃତ୍ତି ବିଷୟରେ ସତ୍ୟ ନୁହେଁ?

    (a) ଏଥିରେ ଶୂନ କିମ୍ବା ଅଧିକ ମାମଲା(case) ରହିପାରେ ।

    (b) ଧ୍ରୁବକ ପରିପ୍ରକାଶ ହେଉଛି ବୈଧ ମାମଲା ମୂଲ୍ୟ ।

    (c) switch ବିବୃତ୍ତିରେ ଥିବା ପରିପ୍ରକାଶ ଫ୍ଲୋଟ୍ ପ୍ରକାର ହୋଇପାରେ ।

    (d) ପରବର୍ତ୍ତୀ ମାମଲା ଗୁଡ଼ିକର ଚାଳନକୁ ଏଡାଇବା ପାଇଁ, ପ୍ରତ୍ୟେକ ମାମଲାର ବିବୃତ୍ତି ବ୍ଲକ୍‌ରେ ଶେଷ ବିବୃତ୍ତି ଭାବରେ break ବିବୃତ୍ତି ନିଶ୍ଚିତ ରହିବା ଆବଶ୍ୟକ ।

20. continue ବିବୃତ୍ତି ବ୍ୟବହୃତ ହୁଏ

    (a) ଲୁପ୍ ବିବୃତ୍ତିର ପରବର୍ତ୍ତୀ ଆଇଟରେସନ୍‌କୁ ଜାରି ରଖିବା ପାଇଁ ।

    (b) ଲୁପ୍ ବିବୃତ୍ତିର ବ୍ଲକରୁ ପ୍ରସ୍ଥାନ କରିବାପାଇଁ ।

    (c) ବାହ୍ୟତମ ବ୍ଲକରୁ ପ୍ରସ୍ଥାନ କରିବାପାଇଁ , ଏପରିକି ଭିତର ବ୍ଲକରେ ବ୍ୟବହୃତ ହେଲେ ମଧ୍ୟ ।

    (d) ତ୍ରୁଟି ଘଟିତ ହେଲେ ମଧ୍ୟ ପ୍ରୋଗ୍ରାମ୍‌ର ଚାଳନ ଜାରି ରଖିବା ପାଇଁ ।

21. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌କୁ ବିଚାର କର

```
void main()
{
    int i, j;
    for ( i = 0; j = 10; i < j;  i++, j-- );
        printf( "x" );
}
```

କେତେ ଥର ଅକ୍ଷର 'x' ମୁଦ୍ରିତ ହେବ?

(*a*) 5 (*b*) 1

(*c*) 10 (*d*) 4

22. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌କୁ ବିଚାର କର

```
void main()
{
    unsigned char ch;
    for ( ch = 0; ch <= 256; ch++ )
        printf( "%d = %c", ch, ch );
}
```

କେତେ ଥରପାଇଁ for ଲୁପ୍‌ଟି ଚାଳନ ହେବ?

(*a*) 256 (*b*) 255

(*c*) 257 (*d*) vlhe :i ls

23. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ରେ କେତେ ଥର ଲୁପ୍‌ଟି ଚାଳନ ହେବ?

```
void main()
{
   int j = 1;
   while ( j <= 100 );
   {
      printf("\nj = %d, j*j = %d", j, j*j );
    j++;
   }
}
```

(a) ଅସୀମ ସମୟପାଇଁ (b) 100 ଥର

(c) 99 ଥର (d) 101 ଥର

24. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
  int i;
  for( i = 0; i < 5; i++ )
   {
     int j = 3;
```

```
       printf("%d  ",i*j);
     }
     printf("%d”,j);
  }
```

(a) ପ୍ରୋଗ୍ରାମ୍‌ଟି ସଫଳତାର ସହ କମ୍ପାଇଲ୍ ଏବଂ ଚାଳନ ହେବ।

(b) ପ୍ରୋଗ୍ରାମ୍‌ଟି କମ୍ପାଇଲ୍ କରିବାରେ ବିଫଳ ହେବ।

(c) ପ୍ରୋଗ୍ରାମ୍ 0 3 6 9 12 3 ଆଉଟପୁଟ୍ ଦେବ

(d) ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ

25. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
   int i = 0;
   for(;;)
  {
    if ( i++ == 4 ) break;
    continue;
  }
  printf("i = %d",i);
}
```

(a) $i = 4$ (b) $i = 5$

(c) $i = 0$ (d) =kqfV

ଉତ୍ତର									
1.	(*c*)	**2.**	(*a*)	**3.**	(*d*)	**4.**	(*a*)	**5.**	(*c*)
6.	(*c*)	**7.**	(*c*)	**8.**	(*a*)	**9.**	(*a*)	**10.**	(*c*)
11.	(*d*)	**12.**	(*c*)	**13.**	(*b*)	**14.**	(*b*)	**15.**	(*c*)
16.	(*c*)	**17.**	(*d*)	**18.**	(*d*)	**19.**	(*c*)	**20.**	(*a*)
21.	(*b*)	**22.**	(*d*)	**23.**	(*a*)	**24.**	(*b*)	**25.**	(*b*)

ପ୍ରୋଗ୍ରାମିଂ ସମସ୍ୟା

1. କୌଣସି ଗାଣିତିକ ଅପରେଟର ବ୍ୟବହାର ନ କରି ଦିଆଯାଇଥିବା ଗଣନ ସଂଖ୍ୟା ଯୁଗ୍ମ କିମ୍ବା ଅଯୁଗ୍ମ ଜାଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
2. ତିନୋଟି ରେଖା ଖଣ୍ଡର ମାପ a, b, ଓ c ଦିଆଯାଇଛି । ଏହି ରେଖା ଖଣ୍ଡଗୁଡ଼ିକୁ ତ୍ରିଭୁଜ ଗଠନ କରିବାରେ ବ୍ୟବହାର କରି ହେବ କି ନାହିଁ ଜାଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
3. ଏକ ତ୍ରିଭୁଜ ABC ର ତିନୋଟି କୋଣର ମାପ ଯଥାକ୍ରମେ a, b, ଏବଂ c ଡିଗ୍ରୀ ଦିଆଯାଇଛି । ଏହି କୋଣ ସହିତ ଏକ ତ୍ରିଭୁଜ ଗଠନ କରି ହେବ କି ନାହିଁ ଜାଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
4. *switch* ବିବୃତ୍ତି ବ୍ୟବହାର କରି ଦିଆଯାଇଥିବା ବର୍ଣ୍ଣଟି ଏକ ସ୍ୱର ବର୍ଣ୍ଣ କି ବ୍ୟଞ୍ଜନ ବର୍ଣ୍ଣ ଜାଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
5. ଗୋଟିଏ କୋଣର ମାପ ଡିଗ୍ରୀରେ ଦିଆଯାଇଛି, କୋଣର ପ୍ରକାର ଜାଣିବା ପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
6. ତିନୋଟି ବିନ୍ଦୁ $A(x_1, y_1)$, $B(x_2, y_2)$ ଏବଂ $C(x_3, y_3)$ ଦିଆଯାଇଛି, ସେମାନେ ଏକ ସରଳରେଖାଭୁକ୍ତ (collinear), ଅର୍ଥାତ ସମାନ ଧାଡିରେ ଅବସ୍ଥାନ କରିଛନ୍ତି କି ନାହିଁ ନିର୍ଦ୍ଧାରଣ କର ।
7. ଦିଆଯାଇଛି ଯେ ବିନ୍ଦୁ (x_1, y_1) ଓ (x_2, y_2) ରେଖା AB ଉପରେ ଅବସ୍ଥିତ ଏବଂ ରେଖା CD ଉପରେ ବିନ୍ଦୁ (x_3, y_3) ଓ (x_4, y_4) ଅବସ୍ଥିତ, ତେବେ AB ଏବଂ CD ପରସ୍ପରକୁ ଛେଦ କରନ୍ତି କି ନାହିଁ ତାହା ନିର୍ଦ୍ଧାରଣ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
8. ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ତାରିଖରେ ସପ୍ତାହର ଦିନ ଜାଣିବା ପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
9. ତୁମର ଜନ୍ମ ତାରିଖ ଏବଂ ଆଜିର ତାରିଖ ଉଭୟ dd/mm/yyyy ଫର୍ମାଟରେ ଦିଆଯାଇଅଛି। ତୁମର ବୟସ ଜାଣିବା ପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
10. 12 ଘଣ୍ଟା ସିଷ୍ଟମରୁ 24 ଘଣ୍ଟା ସିଷ୍ଟମକୁ ସମୟ ପରିବର୍ତ୍ତନ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
11. 24 ଘଣ୍ଟା ସିଷ୍ଟମରୁ 12 ଘଣ୍ଟା ସିଷ୍ଟମକୁ ସମୟ ପରିବର୍ତ୍ତନ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
12. ଦିଆଯାଇଥିବା ତଥ୍ୟ ଅନୁଯାୟୀ ଆରମ୍ଭ ସମୟ ଏବଂ ଶେଷ ସମୟ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଜାଣିବା ପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ (ଉଭୟ ସମୟ hh:mm:ss ଫର୍ମାଟରେ ଦିଆଯାଇଅଛି) ।
13. n ତମ ମୌଳିକ ସଂଖ୍ୟାଟି ବାହାର କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
14. ପ୍ରଥମ n ଟି ମୌଳିକ ସଂଖ୍ୟା ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
15. ଫିବୋନାକି କ୍ରମର n ତମ ପଦ ବାହାର କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
16. ତୁମେ ଜାଣିଛ ଯେ ଏକ ଯୁଗ୍ମ ସଂଖ୍ୟା ହେଉଛି ଏକ ସଂଖ୍ୟା ଯାହା 2 ଦ୍ୱାରା ବିଭାଜ୍ୟ । ତଥାପି ଅନ୍ୟ ଏକ ଉପାୟ ଅଛି ଯାହା ଦିଆଯାଇଥିବା ସଂଖ୍ୟା ଯୁଗ୍ମ କି ନାହିଁ ଜାଣିବାକୁ ବ୍ୟବହୃତ ହୋଇପାରିବ । ଉପାୟଟି ହେଉଛି ଏକକ ଅଙ୍କକୁ ଯାଞ୍ଚ କରିବା - ଯଦି ଏକକ ଅଙ୍କଟି 0, 2, 4, 6, କିମ୍ବା 8 ହୋଇଥାଏ, ତେବେ ସଂଖ୍ୟାଟି ଯୁଗ୍ମ ଅଟେ । ତେଣୁ 2 ଦ୍ୱାରା ଭାଗ ନ କରି ଦିଆଯାଇଥିବା ସଂଖ୍ୟାଟି ଯୁଗ୍ମ କି ନୁହେଁ ଜାଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ପ୍ରାକ୍ଟିକାଲ୍ (Practical)

1. ଏକ ଦ୍ୱିଘାତ(Quadratic) ସମୀକରଣର ମୂଳ ବାହାର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   ଉଦାହରଣ 3.8 ଦେଖ ।

2. ଗଣନ ସଂଖ୍ୟାଟି ମୌଳିକ କି ନୁହେଁ ପରୀକ୍ଷା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

   ଉଦାହରଣ 3.24 ଦେଖ ।

3. ଦିଆଯାଇଥିବା ଗଣନ ସଂଖ୍ୟା ପାଲିଣ୍ଡ୍ରୋମ୍ (palindrome) କି ନୁହେଁ ପରୀକ୍ଷା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   ଉଦାହରଣ 3.25 ଦେଖ ।

4. dd/mm/yyyy ଫର୍ମାଟରେ ଦିଆଯାଇଥିବା ତାରିଖଟି ବୈଧ କି ନୁହେଁ ଜାଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 3.29

```
/*
     Program to check whether the date given in format dd/mm/yyyy
     is valid or not.
 */

 #include<stdio.h>
 int main()
 {
    int mm, dd, yyyy;
    char ch;
    printf( "\nEnter date in format dd/mm/yyyy : ");
    scanf( "%2d%c%2d%c%4d", &dd, &ch, &mm, &ch, &yyyy );

    if ( mm < 1 || mm > 12 ) {
        printf( "\nDate is Invald.\n");
        return 0;
    }
    if ( mm == 2 ) {
        /* condition to check for leap year  */
        if ( (yyyy%400==0)||( (yyyy%4==0)&&(yyyy%100!=0) ) ) {
            if ( dd > 1 && dd <= 29 )
                printf( "\nDate is valid.\n");
            else
                printf( "\nDate is invald.\n");
        } else {
            if ( dd > 1 && dd <= 28 )
                printf( "\nDate is valid.\n");
            else
                printf( "\nDate is invald.\n");
        }
    } else if ( mm == 4 || mm == 6 || mm == 9 || mm == 11 ) {
        if ( dd > 1 && dd <= 30 )
            printf( "\nDate is valid.\n");
        else
            printf( "\nDate is invald.\n");
    } else {
        if ( dd > 1 && dd <= 31 )
            printf( "\nDate is valid.\n");
        else
            printf( "\nDate is invald.\n");
    }
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

ପ୍ରଥମ ରନ୍

```
Enter date in format dd/mm/yyyy : 29/2/2020
Date is valid.
```

ଦ୍ୱିତୀୟ ରନ୍

```
Enter date in format dd/mm/yyyy : 29/2/2021
Date is invalid.
```

5. ସମସ୍ତ ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ ସଂଖ୍ୟା ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ଉଲ୍ଲେଖ ଯୋଗ୍ୟ ଯେ ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ୍ ସଂଖ୍ୟା ଏକ 3-ଅଙ୍କ ବିଶିଷ୍ଟ ସଂଖ୍ୟା। ତେଣୁ, ଆମେ 100 (ପ୍ରଥମ 3-ଅଙ୍କ ସଂଖ୍ୟା) ରୁ ଆରମ୍ଭ କରି 999 (ଶେଷ 3-ଅଙ୍କ ସଂଖ୍ୟା) ପର୍ଯ୍ୟନ୍ତ ଜାରି ରଖିବା, ଏବଂ ଏହା ଏକ ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ ସଂଖ୍ୟା କି ନୁହେଁ ତାହା ଜାଣିବା ପାଇଁ ପ୍ରତ୍ୟେକ ସଂଖ୍ୟାକୁ ପରୀକ୍ଷା କରିବା ।

ତାଲିକା 3.30

```
/*
     Program to find prime numbers in the range m..n.
     Value of m and n will be supplied by user at execution time
 */
 #include<stdio.h>
 int main()
 {
    int i, t, s, d;
    printf( "\nFollowing is list of all Armstrong numbers.\n\n");
    for ( i = 100; i <= 999; i++ )
    {
       t = i;
       s = 0;
       while ( t > 0 )
       {
          d = t % 10;
          s = s + d * d * d;
          t = t / 10;
       }
       if ( s == i )
           printf("%d  ", i);
    }
    printf( "\n");
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Following is list of all Armstrong numbers.
153  370  371  407
```

6. *m..n* ବିସ୍ତାର ମଧ୍ୟରେ ଥିବା ସମସ୍ତ ମୌଳିକ ସଂଖ୍ୟାଗୁଡ଼ିକୁ ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ । ଚାଳନ ସମୟରେ ବ୍ୟବହାରକାରୀଙ୍କ ଦ୍ୱାରା m ଏବଂ n ର ମୂଲ୍ୟ ଯୋଗାଇ ଦିଆଯିବ।

ତାଲିକା 3.31

```
/*
     Program to find prime numbers in the range m..n.
     Value of m and n will be supplied by user at execution time
 */
 #include<stdio.h>
 #include<math.h>
 int main()
 {
    int i, m, n, k, mm;
     printf( "\nProgram to find prime numbers in the range m..n\
n\n");
    printf( "\nEnter value of m(>=2) : ");
    scanf( "%d", &m );
    printf( "\nEnter value of n : ");
    scanf( "%d", &n );

   printf( "\n\nPrime numbers in the range %d..%d are\n\n", m, n);
    for ( i = m; i <= n; i++ )
    {
        if ( ( i > 2 ) && ( ( i % 2 ) == 0 ) )
            continue;

        mm = sqrt(i);
        for ( k = 3; k <= mm; k += 2 )
        {
            if ( i % k == 0 )
                break;
        }
        if ( k > mm )
            printf( "%d  ", i );
    }
    printf( "\n");
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Program to find prime numbers in the range m..n
Enter value of m(>=2) : 10
Enter value of n : 50
Prime numbers in the range 10..50 are
11  13  17  19  23  29  31  37  41  43  47
```

ଅଧିକ ଜାଣିବାପାଇଁ

ସର୍ତ୍ତମୂଳକ ଶାଖା ଏବଂ ଲୁପିଂ ବିଷୟଗୁଡ଼ିକ ପ୍ରୋଗ୍ରାମିଂର ମୁଖ୍ୟ ଭାଗ। ତେଣୁ, ବିଦ୍ୟାର୍ଥୀମାନେ ତାଙ୍କର ଜ୍ଞାନ ଏହି ପ୍ରସଙ୍ଗଗୁଡ଼ିକରେ ଦୃଢ଼ କରିବେ ବୋଲି ଆଶା କରାଯାଉଛି।

ଶିକ୍ଷକଙ୍କଠାରୁ ଆଶା କରାଯାଏ ଯେ ବିଦ୍ୟାର୍ଥୀମାନଙ୍କ ସକ୍ରିୟ ଅଂଶଗ୍ରହଣ ସହିତ ଲଜିକ୍ ବିକଶିତ କରିବାପାଇଁ ନିଆଯାଇଥିବା ସମସ୍ୟା ବିଷୟରେ ଆଲୋଚନା କରିବେ।

ଶିକ୍ଷକ ସମସ୍ୟାର ସମାଧାନଗୁଡ଼ିକୁ କହି/ଲେଖିଦେବା ଉଚିତ ନୁହେଁ, ବରଂ ବିଦ୍ୟାର୍ଥୀମାନେ କିପରି ପୁସ୍ତକରେ ତାଲିକାଭୁକ୍ତ ପ୍ରୋଗ୍ରାମ୍‌ଗୁଡ଼ିକୁ ହୃଦୟଙ୍ଗମ କରି ନିଜେ ନିଜର ପ୍ରୋଗ୍ରାମ୍ ବିକଶିତ କରି ପାରିବେ ସେଥିପାଇଁ ସୁବିଧା ସୁଯୋଗ ଯୋଗାଇଦେବା ଉଚିତ୍।

ପ୍ରୋଗ୍ରାମ୍‌ର ସଠିକତା ସୁନିଶ୍ଚିତ କରିବାପାଇଁ ଶିକ୍ଷକ, ପରୀକ୍ଷଣ ଏବଂ ଡିବଗିଂ (debugging) ପ୍ରକ୍ରିୟା ମଧ୍ୟ ପ୍ରଦର୍ଶନ କରିବା ଉଚିତ୍।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

4 ଆରେ (Array)

ଏକକ ବିଶେଷତ୍ୱ

ଏହି ଏକକରେ ଆରେ(Array) ସମ୍ବନ୍ଧୀୟ ବିଷୟ ଆଲୋଚନା ହୋଇଛି। ଏହି ଆରେଗୁଡ଼ିକ ସଂଖ୍ୟାତ୍ମକ ଡାଟା କିମ୍ବା ଅକ୍ଷର/ବର୍ଣ୍ଣ ଡାଟାର ହୋଇପାରେ। C ଭାଷାରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପରିଚାଳନା କରିବାପାଇଁ ଅକ୍ଷର ଆରେ ବ୍ୟବହୃତ ହୁଏ। ଏହି ଏକକରେ ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ର ବିଭିନ୍ନ ଦିଗ ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ବ୍ୟବହାରଗୁଡ଼ି ପ୍ରଦର୍ଶନ କରାଯାଇଛି।

ଭୂମିକା

ବାସ୍ତବ ଜୀବନରେ ଆମକୁ ଏକ କିମ୍ବା ଅନେକ ପ୍ରକାରର ତଥ୍ୟର ସଂଗ୍ରହ କରିବାକୁ ପଡ଼େ। ଆମର ମଧ୍ୟ ସଂଗୃହୀତ ତଥ୍ୟଗୁଡ଼ିକ ପ୍ରକୃତିରେ ସଂଖ୍ୟାତ୍ମକ କିମ୍ବା ଅଣ-ସଂଖ୍ୟାତ୍ମକ ହୋଇପାରେ।

ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ସମାଧାନ କ୍ଷେତ୍ରରେ ସଂଗୃହୀତ ତଥ୍ୟର ସଂପର୍କ ରହିଛି। ସେହି ସଂଗୃହୀତ ତଥ୍ୟ ସବୁକୁ ମେମୋରିରେ ଭଣ୍ଡାରଣପାଇଁ ଆମକୁ ଉପଯୁକ୍ତ ଡାଟା ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଆବଶ୍ୟକ, ଏବଂ ସମସ୍ୟା ସମାଧାନରେ ପହଞ୍ଚିବାକୁ ସେଗୁଡ଼ିକର ପ୍ରକ୍ରିୟାକରଣ ସୁଗମ କରିବା ଆବଶ୍ୟକ।

ଆରେ ହେଉଛି ଏକ ଡାଟା ଷ୍ଟ୍ରକ୍‌ଚର୍ ଏହା ଗୋଟିଏ ପ୍ରକାରର ତଥ୍ୟକୁ ଗଚ୍ଛିତ କରିବା ଏବଂ ଏଗୁଡ଼ିକ ସହ ପ୍ରକ୍ରିୟା କରିବାପାଇଁ ବହୁତ ସହଜ କରିଥାଏ।

ଛାତ୍ରଛାତ୍ରୀଙ୍କୁ ଆରେ ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ଦିଗ ବୁଝିବାରେ ଏବଂ ଗୋଟିଏ ପ୍ରକାରର ତଥ୍ୟ ସଂଗ୍ରହ ଦରକାର କରୁଥିବା ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାପାଇଁ ପ୍ରୋଗ୍ରାମ୍ ବିକଶିତ କରିବାରେ ସାହାଯ୍ୟ କରିପାରେ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- ମୌଳିକ ଡାଟା ଟାଇପ୍ (Basic data types)
- ସର୍ତ୍ତମୂଳକ ଶାଖାକରଣ (Conditional branching)
- ଲୁପ୍ ଏବଂ ନୀଡ଼ନ ଲୁପ୍ (Loops and nested loops)

ଏକକ ଲବ୍ଧ ଜ୍ଞାନ

ଏକକ ସମାପ୍ତ ହେବା ପରେ, ବିଦ୍ୟାର୍ଥୀଗଣ ନିମ୍ନ ବିଷୟଗୁଡ଼ିକରେ ଜ୍ଞାନ ଆହରଣରେ ସମର୍ଥ ହେବେ

U4-O1: ଆରେ ଧାରଣାର ବ୍ୟାଖ୍ୟା

U4-O2: ଘୋଷଣା କରବା, ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରିବା, ଏବଂ ଆରେକୁ ଇନପୁଟ୍ /ଆଉଟପୁଟ୍ କରିବା

U4-O3: ଆଲଗୋରିଦମ୍ ଏବଂ ପ୍ରୋଗ୍ରାମ୍ ପ୍ରସ୍ତୁତ କରିବାପାଇଁ ଆରେର ବ୍ୟବହାର

U4-O4: ମ୍ୟାଟ୍ରିକ୍ସ ସମ୍ବନ୍ଧୀୟ ସମସ୍ୟାର ସମାଧାନପାଇଁ ଆରେର ବ୍ୟବହାର

U4-O5: ଅକ୍ଷରର ଏକ ଆରେ ଭାବରେ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ର ବ୍ୟାଖ୍ୟା

U4-O6: ମାନକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପରିଚାଳନା ଫଙ୍କ୍ସନ୍‌ଗୁଡ଼ିକର ବ୍ୟବହାରର ପ୍ରଦର୍ଶନ

ୟୁନିଟ୍ -4 ୟୁନିଟ୍ ଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ *(1 – ଦୁର୍ବଳ ସମ୍ପର୍କ; 2 – ମଧ୍ୟମ ସମ୍ପର୍କ; 3 – ଦୃଢ ସମ୍ପର୍କ)*							
	CO-1	**CO-2**	**CO-3**	**CO-4**	**CO-5**	**CO-6**	**CO-7**	**CO-8**
U4-O1	1	–	–	–	–	–	–	–
U4-O2	–	–	1	–	–	–	–	–
U4-O3	–	–	–	–	–	3	–	–
U4-O4	–	–	–	–	–	3	–	–
U4-O5	–	–	1	–	–	–	–	–
U4-O6	–	–	–	–	–	2	–	–

4.1 ଉପକ୍ରମ

ଏକ 'ଆରେ' ହେଉଛି ଗୋଟିଏ ନାମ ଦ୍ୱାରା ବର୍ଣ୍ଣିତ ଏକ ପ୍ରକାରର ତଥ୍ୟ ଉପାଦାନଗୁଡ଼ିକର (ଅର୍ଥାତ୍ ସମାନ ଡାଟା ଟାଇପ୍‌ର) ଏକ ସଂଗ୍ରହ। ଏକ ଆରେର ପ୍ରତ୍ୟେକ ଉପାଦାନକୁ ଏକ ସବ୍‌ସ୍କ୍ରିପ୍ଟ(subscript) ହୋଇଥିବା ଚଳଦ୍ୱାରା ଲେଖାଯାଏ। ଏହା ଆରେ ନାମ ସହିତ ଏକ ସବ୍‌ସ୍କ୍ରିପ୍ଟକୁ ଯୋଡ଼ି ବା ଏକ ଇଣ୍ଡେକ୍ସକୁ(index) ବନ୍ଧନୀରେ ଆବଦ୍ଧ କରି ଗଠନ ହୋଇଥାଏ ।

ଏଠାରେ ସବ୍‌ସ୍କ୍ରିପ୍ଟ ଶବ୍ଦର ଅର୍ଥ ଗାଣିତିକ ସଂକେତ ବ୍ୟବହୃତ ଅର୍ଥ ସହ ସମାନ ।

ଆରେର ପ୍ରକାରଭେଦ

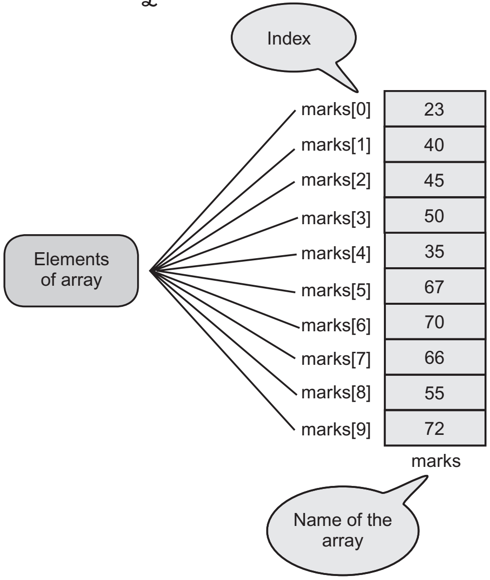
- **ଏକ-ପରିସରଯୁକ୍ତ (One-dimensional(1D)) ଆରେ** – ଏହା ଏକ ପ୍ରକାର ଆରେ ଯେଉଁଠାରେ ଏହାର ଉପାଦାନଗୁଡ଼ିକରେ ପହଞ୍ଚିବାକୁ (access) ଆମକୁ କେବଳ ଗୋଟିଏ ସବସ୍କ୍ରିପ୍ଟର ଆବଶ୍ୟକ ହୁଏ । ଗୋଟିଏ ଏକ-ପରିସରଯୁକ୍ତ ଆରେକୁ ମଧ୍ୟ ଏକ ରୈଖିକ (linear) ଆରେ କୁହାଯାଏ । ଏହା ହେଉଛି ସର୍ବାଧିକ ନିତ୍ୟବ୍ୟବହୃତ ଆରେ।
- **ଦୁଇ-ପରିସରଯୁକ୍ତ (2D) ଆରେ** – ଏହା ଏକ ପ୍ରକାର ଆରେ ଯେଉଁଠାରେ ଏହାର ଉପାଦାନଗୁଡ଼ିକରେ ପହଞ୍ଚିବାକୁ ଆମକୁ ଦୁଇଟି ସବସ୍କ୍ରିପ୍ଟର ଆବଶ୍ୟକ ହୁଏ । ଏହି ଆରେ, ସେହି ପ୍ରକାର ଡାଟା ଗଚ୍ଛିତ କରିବାକୁ ବ୍ୟବହୃତ ହୁଏ ଯେଉଁ ଡାଟାର ଉପାଦାନଗୁଡ଼ିକ ଆୟତାକାର ଢଙ୍ଗରେ ରଖାହୁଏ, ଅର୍ଥାତ୍, ଧାଡି (rows) ଏବଂ ସ୍ତମ୍ଭରେ (columns)। ସାଧାରଣ ଶବ୍ଦରେ ଆମେ ଏହାକୁ ଏକ ଟେବୁଲ୍ ବୋଲି କହିଥାଉ, ଏବଂ ଗାଣିତିକ ଶବ୍ଦରେ, ଏହାକୁ ଏକ ମ୍ୟାଟ୍ରିକ୍ସ ବୋଲି କହିଥାଉ ।
- **ବହୁ-ପରିସରଯୁକ୍ତ ଆରେ** – ଏହା ଏକ ପ୍ରକାର ଆରେ ଯେଉଁଠାରେ ଏହାର ଉପାଦାନରେ ପହଞ୍ଚିବାକୁ ଆମକୁ ଦୁଇରୁ ଅଧିକ ସବସ୍କ୍ରିପ୍ଟର ଆବଶ୍ୟକତା ପଡ଼େ । ଜଟିଳ ଇଞ୍ଜିନିୟରିଂ ସମସ୍ୟା ସମ୍ବନ୍ଧୀୟ ତଥ୍ୟ ଗଚ୍ଛିତ କରିବାକୁ ଏହି ପ୍ରକାରର ଆରେ ବ୍ୟବହାର କରାଯାଏ ।

ଏହି ଏକକରେ ଆମର ଆଲୋଚନା 1D ଏବଂ 2ଡି ରେ ସୀମିତ ରହିବ ।

ଏହା ବ୍ୟତୀତ, ଆମେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ବିଷୟରେ ମଧ୍ୟ ଆଲୋଚନା କରିବୁ ଯାହା ଅକ୍ଷର ଆରେର ଏକ ବିଶେଷ ଅଂଶ ।

4.2 ଏକ-ପରିସରଯୁକ୍ତ ଆରେ

- ଚିତ୍ର 4.1 ମାର୍କ ନାମକ ଏକ-ପରିସରଯୁକ୍ତ ଆରେର ବିଭିନ୍ନ ଅଂଶ ଦର୍ଶାଉଛି। ଏହା 10 ଜଣ ଛାତ୍ରଙ୍କ ମାର୍କ କୁ ଗଚ୍ଛିତ କରିପାରେ ।



ଚିତ୍ର. 4.1: *ମାର୍କ ଆରେ*

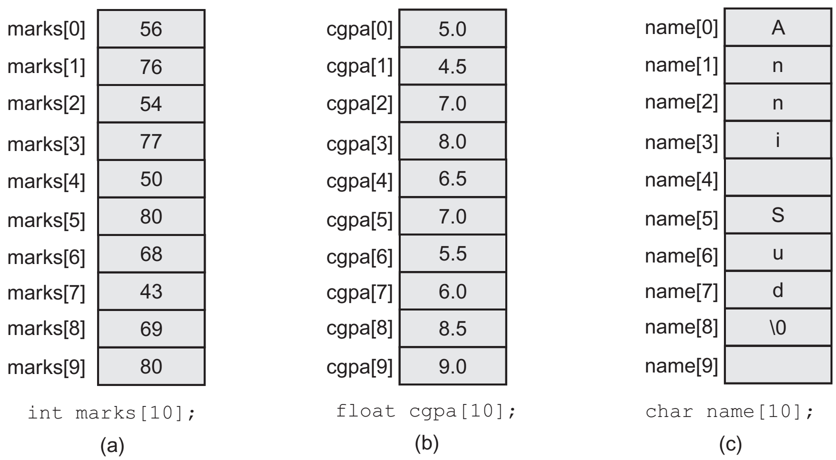
4.2.1 ଘୋଷଣା

ଆରେ ବ୍ୟବହାର କରିବା ପୂର୍ବରୁ ଗୋଟିଏ ଘୋଷଣା କରାଯିବା ଆବଶ୍ୟକ। ଆରେ ଘୋଷଣାନାମା କମ୍ପାଇଲର୍‌କୁ ଆରେର ନାମ, ଏହାର ଉପାଦାନର ପ୍ରକାର, ଏବଂ ଉପାଦାନର ଆକାର/ସଂଖ୍ୟା ସୂଚାଇଥାଏ। ଆରେର ଆକାର ଏକ ଧ୍ରୁବକ ଏବଂ ସଂକଳନ ସମୟରେ ଏହାର ଏକ ମୂଲ୍ୟ ଥିବା ଆବଶ୍ୟକ।

ଏକ-ପରିସରଯୁକ୍ତ ଆରେର ଘୋଷଣାପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
type arrayName[arraySize];
```

ଚିତ୍ର 4.2 ତିନୋଟି ଭିନ୍ନଭିନ୍ନ ଆରେର ଘୋଷଣା ଦର୍ଶାଉଛି ଯେ ଗୋଟିଏ ଇଣ୍ଟିଜର୍ (ସଙ୍ଖ୍ୟାତ୍ମକ) ପାଇଁ, ଗୋଟିଏ ଫ୍ଲୋଟିଂ-ପଏଣ୍ଟ ନମ୍ବରପାଇଁ, ଏବଂ ଗୋଟିଏ ବର୍ଣ୍ଣପାଇଁ। ଏବେ କେବଳ ବୁଝିବାପାଇଁ ଆରେର ଉପାଦାନଗୁଡ଼ିକ ଦର୍ଶାଯାଇଛି।



ଚିତ୍ର . 4.2: 1-D ଆରେର ଘୋଷଣା

ଘୋଷଣା ବିବୃତ୍ତି

```
int marks[10];
```

ଏହା 20 ବାଇଟ୍‌ର ଏକ ସଂଲଗ୍ନ *(contiguous)* ମେମୋରି ବ୍ଲକ୍ (ଏହା Turbo C/C++ କମ୍ପାଇଲର୍‌ପାଇଁ, ଯାହା ଏକ 16-ବିଟ୍ କମ୍ପାଇଲର୍), *int* ପ୍ରକାରର ପ୍ରତ୍ୟେକ ଉପାଦାନପାଇଁ 2-ବାଇଟ୍ ମେମୋରି ବ୍ଲକ୍ ଆବଣ୍ଟନ କରେ। ପ୍ରଥମ 2-ବାଇଟ୍ ଆରେର ପ୍ରଥମ ଉପାଦାନ (marks [0]) ପାଇଁ ବ୍ୟବହୃତ ହୁଏ, ଆରେର ଦ୍ୱିତୀୟ ଉପାଦାନ (marks [1]) ପାଇଁ ପରବର୍ତ୍ତୀ 2-ବାଇଟ୍, ଏବଂ ଏହିପରି ଶେଷ 2-ବାଇଟ୍ ଉପାଦାନ (marks [9]) ପାଇଁ ବ୍ୟବହୃତ ହୋଇଥାଏ।

ଘୋଷଣା ବିବୃତ୍ତି

```
float cgpa[10];
```

40 ବାଇଟ୍‌ର ଏକ ସଂଲଗ୍ନ ମେମୋରି ବ୍ଲକ୍ ଆବଣ୍ଟନ କରେ ଯେଉଁଠାରେ *float* ପ୍ରକାରର ପ୍ରତ୍ୟେକ ଉପାଦାନପାଇଁ, 4-ବାଇଟ୍ ମେମୋରି ବ୍ଲକ୍ ଆବଣ୍ଟନ ହୁଏ। ପ୍ରଥମ 4-ବାଇଟ୍ ଆରେର ପ୍ରଥମ ଉପାଦାନପାଇଁ ବ୍ୟବହୃତ ହୁଏ (*cgpa*[0]), ଆରେର ଦ୍ୱିତୀୟ ଉପାଦାନପାଇଁ ପରବର୍ତ୍ତୀ 4-ବାଇଟ୍ (*cgpa*[1]), ଏବଂ ଏହିପରି ଶେଷ 4-ବାଇଟ୍ *cgpa*[9] ଉପାଦାନପାଇଁ ବ୍ୟବହୃତ ହୁଏ।

ଘୋଷଣା ବିବୃତ୍ତି

```
char name[10];
```

ଏଠାରେ 10 ବାଇଟ୍‌ର ଏକ ସଂଲଗ୍ନ ମେମୋରି ବ୍ଲକ୍ ଆବଣ୍ଟନ କରେ ଯେଉଁଠାରେ ପ୍ରତ୍ୟେକ char ପ୍ରକାରର ଉପାଦାନପାଇଁ 1-ବାଇଟ୍ ମେମୋରି ବ୍ଲକ୍ ଆବଣ୍ଟନ ହୁଏ। ପ୍ରଥମ ବାଇଟ୍, ଆରେର ପ୍ରଥମ ଉପାଦାନ (*name*[0]) ପାଇଁ ବ୍ୟବହୃତ ହୁଏ, ଆରେର ଦ୍ୱିତୀୟ ଉପାଦାନ(*name*[1]) ପାଇଁ, ପରବର୍ତ୍ତୀ ବାଇଟ୍ ଏବଂ ଏହିପରି ଶେଷ ବାଇଟ୍ ଉପାଦାନ *name*[9]ପାଇଁ ବ୍ୟବହୃତ ହୁଏ।

4.2.2 ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟକରଣ

ଯେପରି ଘୋଷଣା ସହିତ ସାଧାରଣ ଚଳ ଗୁଡ଼ିକର ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ଦିଆଯାଇପାରିବ, ସେହିପରି ଆରେର ଉପାଦାନ ଗୁଡ଼ିକୁ ମଧ୍ୟ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ଦେଇହେବ। ଆରେରେ ଥିବା ପ୍ରତ୍ୟେକ ଉପାଦାନପାଇଁ, ଆମେ ଏକ ମୂଲ୍ୟ ପ୍ରଦାନ କରୁ। ଏକମାତ୍ର ପାର୍ଥକ୍ୟ ହେଉଛି ମୂଲ୍ୟଗୁଡ଼ିକ କୁଟିଳ ବନ୍ଧନୀରେ ଆବଦ୍ଧ ହେବା ଆବଶ୍ୟକ, ଏବଂ ଯଦି ଏକରୁ ଅଧିକ ମୂଲ୍ୟ ଥାଏ ତେବେ ଏଗୁଡ଼ିକ କମା ଦ୍ୱାରା ଅଲଗା କରି ଲେଖା ଯାଇପାରିବ।

ନିମ୍ନଲିଖିତ ଉଦାହରଣଗୁଡ଼ିକ ଏକ-ପରିସରଯୁକ୍ତ (1-D) ଆରେର ଆରମ୍ଭ କରିବାର ସମସ୍ତ ସମ୍ଭାବ୍ୟ ଉପାୟଗୁଡ଼ିକ ପ୍ରଦାନ କରେ।

ନିମ୍ନଲିଖିତ ବିବୃତ୍ତି

```
int b[10]={12, 0, 14, -4, 7, 8, 10, 11, 9, 15};
```

ଏହା ଉପାଦାନ *b*[0] କୁ 12, *b*[1]କୁ 0, *b*[2] କୁ 14, *b*[3] କୁ –4, *b*[4] କୁ 7, *b*[5] କୁ 8, *b*[6] 10, *b*[7] କୁ 11, *b*[8] କୁ 9, *b*[9] କୁ 15 ଲେଖାଁଏ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ପ୍ରଦାନ କରେ।

ଯଦି ନିମ୍ନମତେ ଆରେର ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇଥାଏ

```
int b[] = {12, 0, 14, -4, 7};
```

ଯେହେତୁ ଆରେର ଆକାର ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇନାହିଁ, କମ୍ପାଇଲର୍ ପ୍ରାରମ୍ଭିକକରଣ ତାଲିକାରେ ଥିବା ଉପାଦାନର ସଂଖ୍ୟା ଗଣି ଏହାକୁ ଆରେର ଆକାର ଭାବେ ନିର୍ଦ୍ଦିଷ୍ଟ କରିଥାଏ।

ନିମ୍ନଲିଖିତ ଘୋଷଣା ବିବୃତ୍ତିରୁ ତୁମେ କ'ଣ ଆଶା କରିପାରିବ?

```
int b[10] = {1, 2, 3};
```

ଏହା ଉପାଦାନ *b*[0] କୁ 1, *b*[1] କୁ 2, *b*[2] କୁ 3 ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ଦେଇଥାଏ, ଏବଂ ଅବଶିଷ୍ଟ ସମସ୍ତ ଉପାଦାନଗୁଡ଼ିକୁ ମୂଲ୍ୟ 0 ଦିଆଯାଏ। ଆମେ କହିପାରିବା ଯେ ଆଂଶିକ ପ୍ରାରମ୍ଭିକ ଆରେରେ, ଅବଶିଷ୍ଟ ସମସ୍ତ ଉପାଦାନ କୁ ଆରମ୍ଭରେ 0 ମୂଲ୍ୟ ଦିଆଯାଏ।

ନିମ୍ନଲିଖିତ ଘୋଷଣାବିବୃତ୍ତିରୁ ତୁମେ କ'ଣ ଆଶା କରିପାରିବ?

```
int a[6]={12, 0, 14, -4, 7, 15, -20, 22, 25};
```

ଏଠାରେ କମ୍ପାଇଲର୍ ଏକ ତ୍ରୁଟି ସନ୍ଦେଶ ଫ୍ଲାଗ୍ କରିବ, କାରଣ ପ୍ରାରମ୍ଭିକ କରଣ ତାଲିକାରେ ଘୋଷିତ ଆକାର ଅପେକ୍ଷା ଅଧିକ ଉପାଦାନ ରହିଛି।

4.2.3 ଉପାଦାନକୁ ପହଞ୍ଚ (Accessing Elements)

ଏକ-ପରିସରଯୁକ୍ତ ଆରେର ନିର୍ଦ୍ଦିଷ୍ଟ ଉପାଦାନଗୁଡ଼ିକରେ ପହଞ୍ଚିବାକୁ ଏକ ସୂଚକାଙ୍କ(index) ବ୍ୟବହୃତ ହୋଇଥାଏ । ସୂଚକାଙ୍କ ନିଶ୍ଚିତ ଭାବରେ ଏକ ପୂର୍ଣ୍ଣାଙ୍କ ମୂଲ୍ୟ କିମ୍ବା ଏକ ପରିପ୍ରକାଶ ଯାହା ଏକ ପୂର୍ଣ୍ଣାଙ୍କ ମୂଲ୍ୟକୁ ମୂଲ୍ୟାଙ୍କନ ହୁଏ ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଚିତ୍ର 4.2 (a), ରେ ଦିଆଯାଇଥିବା *marks* ଆରେର ପ୍ରଥମ ଉପାଦାନକୁ ପହଞ୍ଚିବାକୁ ବା ଆକ୍ସେସ୍ କରିବାପାଇଁ ନିମ୍ନ ବିବୃତ୍ତି ବ୍ୟବହାର କରିବା

```
marks[0]
```

ସେହିପରି *marks* ର ସମସ୍ତ ଉପାଦାନଗୁଡ଼ିକୁ ପ୍ରକ୍ରିୟା କରିବାକୁ ନିମ୍ନଲିଖିତ କୋଡ୍ ପରି ଏକ ଲୁପ୍ ବ୍ୟବହାର କରାଯାଇ ପାରିବ:

```
for (i = 0; i < 10; i++ )
    process ( marks[i] );
```

4.2.4 ଇନପୁଟ୍

ନିମ୍ନ ଘୋଷଣାନାମାକୁ ବିଚାର କର

```
int a[50];
```

ଧରାଯାଉ, ଏକ 1D ଆରେରେ ଉପାଦାନଗୁଡ଼ିକର ପ୍ରକୃତ ସଂଖ୍ୟା ହେଉଛି n (<=50)। ତା'ହେଲେ ବ୍ୟବହାରକାରୀ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ସମୟରେ n ର ମୂଲ୍ୟ ପ୍ରଦାନ କରିବେ ।

ନିମ୍ନ କୋଡ୍‌ଖଣ୍ଡ ଗୋଟିଏ ଏକ-ପରିସରଯୁକ୍ତ ଆରେରେ ଡାଟା ଇନ୍‌ପୁଟ୍ କରିବାର ଉପାୟ ଦର୍ଶାଏ ।

```
for ( i = 0; i < n; i++ )
 {
    scanf( "%d", &a[i] );
 }
```

ନିମ୍ନଲିଖିତ ଜିନିଷଗୁଡ଼ିକ ନିରୀକ୍ଷଣ କର:

1. ଆମେ ସୂଚକାଙ୍କ i କୁ 0 ରୁ ଆରମ୍ଭ କରୁ ଏବଂ ଏହା $(n-1)$ ପର୍ଯ୍ୟନ୍ତ ଉପରକୁ ଯାଏ ।
2. ଯଦିଓ ଆମେ ଆରେ ଉପାଦାନ ସହିତ କାର୍ଯ୍ୟ କରୁଛୁ, ତଥାପି *scanf()* ଫଙ୍କସନ୍ କଲ୍‌ରେ ଠିକଣା ଅପରେଟର୍‌ର (&) ଆବଶ୍ୟକତା ରହିଛି ।

ଏକା ଧାଡ଼ିରେ ଶୂନ୍ୟସ୍ଥାନ / ଭୂସମାନ୍ତର-ଟ୍ୟାବ୍‌ଦ୍ୱାରା ଅଲଗା କରି କିମ୍ବା ଅଲଗା ଧାଡ଼ିରେ ଗୋଟିଏ ଲେଖାଁଏ ଉପାଦାନ କରି ଆରେର ଉପାଦାନଗୁଡ଼ିକ ଇନ୍‌ପୁଟ୍ କରାଯାଇପାରିବ ।

4.2.5 ଆଉଟପୁଟ୍

ଆରେ ଉପାଦାନଗୁଡ଼ିକର ମୂଲ୍ୟ ଆଉଟପୁଟ୍ କରିବାକୁ, ଆମେ ଲୁପ୍‌ର ବ୍ୟବହାର ନିମ୍ନମତେ କରୁ:

```
for ( i = 0; i < n; i++ ) {
    printf( "%d  ", a[i] );
 }
 printf( "\n" );
```

ପ୍ରତ୍ୟେକ ମୂଲ୍ୟ ଦୁଇ ଶୂନ୍ୟସ୍ଥାନ ସହ ପୃଥକ ହୋଇ ଗୋଟିଏ ଧାଡ଼ିରେ ପ୍ରଦର୍ଶିତ ହେବ ।

ତଥାପି, ସେମାନଙ୍କର ମୂଲ୍ୟର ପରିମାଣ କିମ୍ବା ସଂଖ୍ୟା ହେତୁ ଯଦି ସମସ୍ତ ମୂଲ୍ୟ ଗୋଟିଏ ଧାଡ଼ିରେ ପ୍ରଦର୍ଶିତ ହୋଇପାରୁ ନାହିଁ, ତେବେ ସେଗୁଡ଼ିକ ପରବର୍ତ୍ତୀ ଧାଡ଼ିରେ ପ୍ରଦର୍ଶିତ ହୁଏ । for ଲୁପ୍ ସମ୍ପୂର୍ଣ୍ଣ ହେବା ପରେ, ଏକ *printf()* ଫଙ୍କସନ୍ କଲ୍, କର୍ସର୍‌କୁ ପରବର୍ତ୍ତୀ ଲାଇନକୁ ନେଇଯାଏ ।

ନିମ୍ନଲିଖିତ ଫର ଲୁପ୍‌ଟି ପ୍ରତି ଲାଇନରେ ଗୋଟିଏ ଲେଖାଁଏ ମୂଲ୍ୟ ପ୍ରଦର୍ଶନ କରିଥାଏ ।

```
for ( i = 0; i < n; i++ ) {
    printf( "\n%d", a[i] );
 } printf( "\n" );
```

1D ଆରେର ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଏକ-ପରିସରଯୁକ୍ତ ଆରେର ବିଭିନ୍ନ ଦିଗ ବୁଝିବା ପରେ, ଏବେ ଜଟିଳ ସମସ୍ୟା ପରିଚାଳନା କରିବାରେ ଏକ-ପରିସରଯୁକ୍ତ ଆରେର ସାମର୍ଥ୍ୟ ପ୍ରଦର୍ଶନ କରିବାପାଇଁ, କିଛି ଉଦାହରଣ ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବା ।

ଉଦାହରଣ 4.1: *a* ନାମକ ଏକ ଆରେ ଦିଆଯାଇଛି, ଯାହାର ଆକାର $n(\leq 50)$ ଏବଂ ଯାହାର ଉପାଦାନଗୁଡ଼ିକ int ପ୍ରକାରର । ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ଆରେର କେବଳ ଯୁଗ୍ମ ଉପାଦାନଗୁଡ଼ିକୁ ପ୍ରଦର୍ଶନ କରେ ।

ତାଲିକା 4.1

```
/*
    Program that outputs those elements of a 1D array that are EVEN
 */

 #include<stdio.h>

 int main()
 {
    int a[50], i, n;
    printf("\nEnter size of array n(<=50) : ");
    scanf("%d", &n);
    printf("\nEnter %d natural numbers as elements of array\n", n);
    for ( i = 0; i < n; i++ )
       scanf("%d", &a[i]);
    printf("\nEVEN elements of array are\n");
    for ( i = 0; i < n; i++ )
```

```
    {
       if ( a[i] % 2 == 0 )
           printf("%d  ", a[i]);
    }
    printf("\n");
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter size of array n(<=50) : 10
Enter 10 natural numbers as elements of array
10 7 15 14 24 23 77 35 70 12
EVEN elements of the array are
10 14 24 70 12
```

ଉଦାହରଣ 4.2: a ନାମକ ଏକ ଆରେ ଦିଆଯାଇଛି, ଯାହାର ଆକାର *n(≤50)* ଏବଂ ଯାହାର ଉପାଦାନଗୁଡ଼ିକ int ପ୍ରକାରର। 1D ଆରେର ସର୍ବବୃହତ ଉପାଦାନ, କ୍ଷୁଦ୍ରତମ ଉପାଦାନ, ଏବଂ ହାରାହାରି ଉପାଦାନ ଜାଣିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

ତାଲିକା 4.2

```
 /*
    Program to find the largest element, smallest element,
    and the average of elements of 1D array
 */
 #include <stdio.h>
 int main()
{
int a[20];
int i, j, n, largest, smallest, sum;
    float average;
    printf("\nEnter size of array n(<=20) : ");
    scanf("%d", &n);
    printf("\nEnter %d element of 1D array\n", n);
    for ( i = 0; i < n; i++ )
        scanf("%d", &a[i]);
    /*  code segment to find largest element  */
    largest = a[0];
    for ( i = 1; i < n; i++ ) {
        if ( a[i] > largest )
```

```
                        largest = a[i];
                   }
     /*  code segment to find smallest element  */
     smallest = a[0];
     for ( i = 1; i < n; i++ ) {
         if ( a[i] < smallest )
                        smallest = a[i];
                   }
     /*  code segment to find average of elements  */
     sum = 0;
for ( i = 0; i < n; i++ )
                    sum = sum + a[i];
average = (float) sum/n;
      /*  code segment to output the results  */
  printf("\nLargest element    = %d", largest );
  printf("\nSmallest element   = %d", smallest );
  printf("\nAverage of elements = %.2f\n", average );
    return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter size of array n(<=20) : 10
Enter 10 element of 1D array
10 24 33 15 19 21 35 20 40 16
Largest element    = 40
Smallest element   = 10
Average of elements = 23.30
```

ଉଦାହରଣ 4.3: ଏକ ଡେସିମାଲ୍ ନମ୍ବର *n* ଦିଆଯାଇଛି । *n* କୁ ଏହାର ସମକକ୍ଷ ବାଇନାରି ନମ୍ବରରେ ରୂପାନ୍ତର କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 4.3

```
/*
    Program to convert a decimal number 'n' into its equivalent
    binary number. The program uses 1D array to store remainders during
    the process of conversion and then prints this array in reverse
    order.
*/
```

```
#include <stdio.h>
int main()
{
   int a[20];
   int i, j, t, n;
   printf("\nEnter decimal number n : ");
  scanf("%d", &n);
   t = n;
   i = 0;
   while ( t > 0 )
   {
       a[i] = t % 2;
       i++;
       t = t / 2;
   }
   printf("\nDecimal number %d is equivalent to ", n);
   for ( j = i-1; j >= 0; j-- )
    printf("%d", a[j]);
   printf(" binary.\n");
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter decimal number n : 10
Decimal number 10 is equivalent to 1010 binary.
```

ଉଦାହରଣ 4.4: *a* ନାମକ ଏକ ଆରେ ଦିଆଯାଇଛି, ଯାହାର ଆକାର *n(≤50)* ଏବଂ ଯାହାର ଉପାଦାନଗୁଡ଼ିକ *int* ପ୍ରକାରର । ଏକ 1D ଆରେର ପାର୍ଶ୍ୱବର୍ତ୍ତୀ ଉପାଦାନଗୁଡ଼ିକ ଅଦଳବଦଳ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 4.4

```
/*
    Program to swap adjacent elements of a 1D array.
    Size of array is even number.
 */
 #include <stdio.h>
 int main()
 {
    int a[20];
       int i, n, t;
```

```
    printf("\nEnter size of array n(<=20) : ");
    scanf("%d", &n);
    printf("\nEnter %d element of 1D array\n", n);
    for ( i = 0; i < n; i++ )
        scanf("%d", &a[i]);
   /*  code segment to swap adjacent elements  */
    for ( i = 0; i < n-1; i += 2 ) {
        t = a[i];
                    a[i] = a[i+1];
                    a[i+1] = t;
                  }
   printf("\nElements of an array after swapping adjacent elements\n");
    for ( i = 0; i < n; i++ )
       printf("%d  ", a[i]);
    printf("\n\n");

    return 0;
 }
```

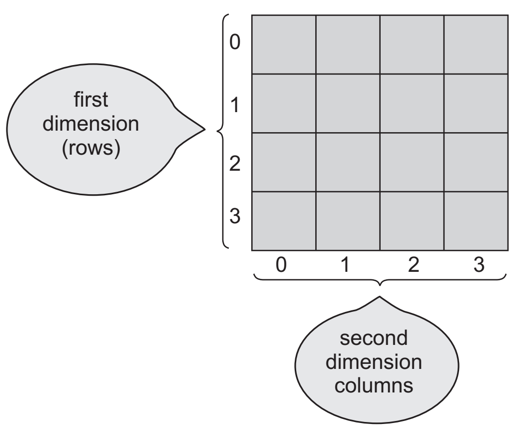
ଟେଷ୍ଟ ରନ୍

```
Enter size of array n(<=20) : 10
Enter 10 element of 1D array
10 24 33 15 19 21 35 20 40 16
Elements of an array after swapping adjacent elements
14 10 15 33 21 19 20 35 16 40
```

4.3 ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ

ଏପର୍ଯ୍ୟନ୍ତ ଆମେ ଆଲୋଚନା କରିଥିବା ଆରେକୁ ଏକ-ପରିସରଯୁକ୍ତ ଆରେ କୁହାଯାଏ କାରଣ ଏଥିରେ ଡାଟା କେବଳ ଗୋଟିଏ ଦିଗରେ (dimension) ସଂଗଠିତ। କିନ୍ତୁ ଅନେକ ଆପ୍ଲିକେସନ୍‌ପାଇଁ ଡାଟା ଏକାଧିକ ପରିସରରେ ସଂଗଠିତ ହେବା ଦରକାର। ଗୋଟିଏ ସାଧାରଣ ଉଦାହରଣ ହେଉଛି ଏକ ମ୍ୟାଟ୍ରିକ୍ସ (ଏକ ଟେବୁଲ୍), ଯାହା ଧାଡ଼ି ଏବଂ ସ୍ତମ୍ଭକୁ ନେଇ ଗଠିତ ଏକ ଆରେ।

ଚିତ୍ର 4.3, 4 ଧାଡ଼ି ଏବଂ 4 ସ୍ତମ୍ଭ ସହିତ ଏକ ମ୍ୟାଟ୍ରିକ୍ସ (ଟେବୁଲ୍) କୁ ଦର୍ଶାଏ ଯାହା ସାଧାରଣତଃ ଏକ ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ ଭାବରେ ବ୍ୟବହୃତ ହୁଏ।



ଚିତ୍ର . 4.3: ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ

4.3.1 ଘୋଷଣାନାମା

ଏକ-ପରିସରଯୁକ୍ତ ଆରେ ପରି, ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ ବ୍ୟବହାର ହେବା ପୂର୍ବରୁ ଘୋଷଣା କରାଯିବା ଆବଶ୍ୟକ। ଘୋଷଣାନାମା କମ୍ପାଇଲର୍‌କୁ ଆରେର ନାମ, ଏହାର ଉପାଦାନଗୁଡ଼ିକର ପ୍ରକାର, ଏବଂ ପ୍ରତ୍ୟେକ ଡାଇମେନ୍‌ସନ୍‌ର ଆକାର କହିଥାଏ ।

ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେର ଘୋଷଣାପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
type arrayName[RowSize][ColSize];
```

ପ୍ରଥାସମ୍ମତ ଭାବେ, ଆରେରେ ପ୍ରଥମ ଡାଇମେନ୍‌ସନ୍‌ଟି ଧାଡ଼ି ସଂଖ୍ୟା ନିର୍ଦ୍ଦିଷ୍ଟ କରେ ଏବଂ ଦ୍ୱିତୀୟ ଡାଇମେନ୍‌ସନ୍ ପ୍ରତ୍ୟେକ ଧାଡ଼ିର ସ୍ତମ୍ଭ ସଂଖ୍ୟା ନିର୍ଦ୍ଦିଷ୍ଟ କରେ ।

ନିମ୍ନ ଲିଖିତ ଘୋଷଣାନାମା ଦେଖ

```
int a[3][3];
```

ଏହି ଘୋଷଣାନାମା ବିବୃତ୍ତି 18 ବାଇଟ୍‌ର ଏକ ସଂଲଗ୍ନ ମେମୋରି ବ୍ଲକ୍ ଆବଣ୍ଟନ କରେ। ଏଥିରେ ପ୍ରତ୍ୟେକ ଧାଡିପାଇଁ, 6-ବାଇଟ୍ ରହିଥାଏ । 6-ବାଇଟ୍‌ର ପ୍ରଥମ ସେଟ୍ ପ୍ରଥମ ଧାଡିର ଉପାଦାନଗୁଡ଼ିକ ଗଚ୍ଛିତ କରିବାକୁ ବ୍ୟବହୃତ ହୁଏ, ଦ୍ୱିତୀୟ ଧାଡିର ଉପାଦାନଗୁଡ଼ିକ ଗଚ୍ଛିତ କରିବାକୁ ଆଉ 6-ବାଇଟ୍‌ର ସେଟ୍ ବ୍ୟବହୃତ ହୁଏ, ଏବଂ ତୃତୀୟ (ଶେଷ) ଧାଡିର ଉପାଦାନଗୁଡ଼ିକ ଗଚ୍ଛିତ କରିବାକୁ ମଧ୍ୟ 6-ବାଇଟ୍‌ର ତୃତୀୟ ସେଟ୍ ବ୍ୟବହୃତ ହୁଏ। ପ୍ରତ୍ୟେକ ଧାଡ଼ି ମଧ୍ୟରେ, ପ୍ରଥମ ସ୍ତମ୍ଭର ଉପାଦାନ ଗଚ୍ଛିତ କରିବାକୁ ପ୍ରଥମ 2-ବାଇଟ୍ ବ୍ୟବହୃତ ହୁଏ, ପରବର୍ତ୍ତୀ 2-ବାଇଟ୍ ଦ୍ୱିତୀୟ ସ୍ତମ୍ଭର ଉପାଦାନ ଗଚ୍ଛିତ କରିବାକୁ ବ୍ୟବହୃତ ହୁଏ, ଏବଂ ତୃତୀୟ (ଶେଷ) ସ୍ତମ୍ଭର ଉପାଦାନ ଗଚ୍ଛିତ କରିବାକୁ ଅନ୍ତିମ 2-ବାଇଟ୍ ବ୍ୟବହୃତ ହୁଏ ।

> ଏକ ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେର ଉପାଦାନଗୁଡ଼ିକ ଧାଡ଼ି ଅନୁଯାୟୀ ଗଚ୍ଛିତ ହୋଇଥାଏ , ଅର୍ଥାତ୍, ମେମୋରିର ଆବଣ୍ଟନ ସଂଲଗ୍ନ ବ୍ଲକ ଅନୁଯାୟୀ, ପ୍ରଥମ ଧାଡିର ପ୍ରଥମ ଉପାଦାନଗୁଡ଼ିକ, ତା'ପରେ ଦ୍ୱିତୀୟ ଧାଡିର ଉପାଦାନ, ଏବଂ ଶେଷରେ ତୃତୀୟ ଧାଡିର ଉପାଦାନ, ଇତ୍ୟାଦି ଗଚ୍ଛିତ ହୋଇଥା'ନ୍ତି ।

4.3.2 ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟକରଣ

ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେର ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରିବାର ଗୋଟିଏ ଉପାୟ ନିମ୍ନରେ ଦର୍ଶାଯାଇଛି:

```
int a[3][3] = { 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3 };
```

ଏହା ପ୍ରଥମ ଧାଡିର ସମସ୍ତ ଉପାଦାନର ମୂଲ୍ୟକୁ 0 ସହିତ, ଦ୍ୱିତୀୟ ଧାଡିର ଉପାଦାନଗୁଡ଼ିକର ମୂଲ୍ୟକୁ 1 ସହିତ, ତୃତୀୟ ଧାଡିର ଉପାଦାନଗୁଡ଼ିକର ମୂଲ୍ୟକୁ 2 ସହିତ ଏବଂ ଚତୁର୍ଥ ଧାଡିର ଉପାଦାନଗୁଡ଼ିକର ମୂଲ୍ୟକୁ 3 ସହିତ ନିର୍ଦ୍ଦିଷ୍ଟ କରେ ।

ଯଦିଓ ଉପରୋକ୍ତ ପ୍ରାରମ୍ଭିକସ୍ତରରେ କୌଣସି ସମସ୍ୟା ନାହିଁ, କିନ୍ତୁ ଏହା ସୁପାରିଶ କରାଯାଇଛି ଯେ ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେର ଠିକ୍ ପ୍ରକୃତି ଦେଖାଇବାପାଇଁ ନୀଡ଼ନ ବନ୍ଧନୀ ବ୍ୟବହାର କରାଯିବା ଦରକାର ।

ଆରେ *a* ର ଉତ୍ତମ ରୂପେ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟକରଣ ନିମ୍ନରେ ପ୍ରଦର୍ଶିତ:

```
int a[3][3] = { { 0, 0, 0 },
                 { 1, 1, 1 },
                 { 2, 2, 2 },
                 { 3, 3, 3 },
               };
```

ଉପରୋକ୍ତ ପ୍ରାରମ୍ଭିକସ୍ତରର ପ୍ରକ୍ରିୟାରେ, ଆମେ ପ୍ରତ୍ୟେକ ଧାଡିକୁ ବନ୍ଧନୀରେ ଆବଦ୍ଧ ତିନୋଟି ଉପାଦାନର ଏକ-ପରିସରଯୁକ୍ତ ଆରେ ଭାବରେ ପ୍ରାରମ୍ଭିକ କରୁ । ତିନି ଧାଡିର ଆରେପାଇଁ ଆଉ ଏକ ବନ୍ଧନୀ ସେଟ୍ ଅଛି । ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ ଆମେ ସ୍ତମ୍ଭରେ ଥିବା ଉପାଦାନଗୁଡ଼ିକ ମଧ୍ୟରେ କମା ବ୍ୟବହାର କରାଯାଏ। ଏବଂ ଧାଡିଗୁଡ଼ିକ ମଧ୍ୟରେ ମଧ୍ୟ କମା ବ୍ୟବହାର କରାଯାଇପାରେ ।

4.3.3 ଉପାଦାନଗୁଡ଼ିକୁ ପହଞ୍ଚ

i ଧାଡ଼ିର *j* ତମ ଉପାଦାନକୁ ଆକସେସ୍ କରିବାକୁ, ଆମେ ଲେଖିଥାଉ

```
a[i][j]
```

a ର ସମସ୍ତ ଉପାଦାନକୁ ଧାଡ଼ି ଅନୁଯାୟୀ ପ୍ରକ୍ରିୟା କରିବାପାଇଁ, ନିମ୍ନଲିଖିତ କୋଡ୍ ପରି ନୀଡ଼ନ ଲୁପ୍ ବ୍ୟବହାର କରାଯିବ:

```
for (i = 0; i < 3; i++ )
 {
     for (j = 0; j < 3; j++ )
     {
         process ( a[i][j] );
     }
 }
```

4.3.4 ଇନପୁଟ୍

ନିମ୍ନଲିଖିତ କୋଡ୍ଖଣ୍ଡ ଦର୍ଶାଏ ଯେ ନୀଡ଼ନ for ଲୁପ୍ ବ୍ୟବହାର କରି ଧାଡ଼ି ଅନୁଯାୟୀ ଉପାଦାନଗୁଡ଼ିକ ପଢିପାରିବ:

```
for (i = 0; i < 3; i++ )
 {
    for (j = 0; j < 3; j++ )
    {
       scanf( "%d", &a[i][j] );
    }
 }
```

ସାଧାରଣତଃ, ଯଦି ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେର ଆକାର $m \times n$ ଅଟେ, ପ୍ରଥମ ଲୁପ୍ 0 ରୁ (m-1) ପର୍ଯ୍ୟନ୍ତ ଚାଲେ ଏବଂ ଦ୍ୱିତୀୟ ଲୁପ୍ 0 ରୁ (n-1) ପର୍ଯ୍ୟନ୍ତ ଚାଲେ ।

4.3.5 ଆଉଟପୁଟ୍

ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ ଉପାଦାନଗୁଡ଼ିକର ମୂଲ୍ୟ ଆଉଟ୍ପୁଟ୍ କରିବାକୁ, ଆମେ ନୀଡ଼ନ ଲୁପ୍ ନିମ୍ନମତେ ବ୍ୟବହାର କରିଥାଉ:

```
for ( i = 0; i < 3; i++ )
 {
     for ( j = 0; j < 3; j++ )
     {
        printf( "%d  ", a[i][j] );
     }
     printf( "\n" );
 }
```

ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ୍‌ଖଣ୍ଡ ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ ଉପାଦାନଗୁଡ଼ିକୁ ଧାଡି-ପରେ ଧାଡ଼ି ଆଉଟପୁଟ୍ କରେ। ଏହିପରି, ପ୍ରଥମ ଲୁପ୍ ଧାଡିକୁ ନିୟନ୍ତ୍ରଣ କରୁଥିବାବେଳେ ଦ୍ୱିତୀୟ ଲୁପ୍ ସ୍ତମ୍ଭଗୁଡ଼ିକୁ ନିୟନ୍ତ୍ରଣ କରେ।

2-D ଆରେର ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 4.5: A ଏକ $n \times m$, ଅର୍ଡରର ମ୍ୟାଟ୍ରିକ୍ସ ଟ୍ରାନ୍ସପୋଜ୍ କରିବାକୁ ପ୍ରୋଗ୍ରାମ୍।

ଆମେ ଜାଣୁ ଯଦି ଆମେ ସ୍ତମ୍ଭ ସହିତ ଧାଡିକୁ ଅଦଳବଦଳ କରୁ, ତେବେ ଆମେ ଏକ ଦିଆଯାଇଥିବା ମ୍ୟାଟ୍ରିକ୍ସର ଟ୍ରାନ୍ସପୋଜ୍ ପାଇଥାଉ, ଧରାଯାଉ B ଯାହାର ଅର୍ଡର $n \times m$, ,ଅର୍ଥାତ୍,

$$bji = aij$$

ତାଲିକା 4.5

```
/*
    Program to compute transpose of matrix A(mxn)
 */
 #include<stdio.h>
 int main()
 {
     int a[10][10], b[10][10];
     int n, m, i, j;
    printf( "Enter the size of matrix A as m,n: " );
     scanf( "%d,%d", &m, &n);
printf("\nEnter %d elements of matrix A row-wise\n",m*n);
     for ( i = 0 ; i < m; i++ ) {
         for ( j = 0; j < n; j++ ) {
            scanf( "%d", &a[i][j] );
         }
     }
     for ( i = 0 ; i < m ; i++ ) {
         for ( j = 0 ; j < n ; j++ ) {
```

```
            b[j][i] = a[i][j];
         }
     }
    printf( "\nTranspose is\n\n" );
     for ( i = 0 ; i < n; i++ ) {
         for ( j = 0; j < m; j++ ) {
            printf( "%5d", b[i][j] );
         }
         printf ( "\n" );
     }
     return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter the size of matrix A as m,n: 3,3
Enter 9 elements of matrix A row-wise
 1 2 3
 4 5 6
 7 8 9
 Transpose is
   1   4   7
   2   5   8
   3   6   9
```

ଉଦାହରଣ 4.6: ଦୁଇଟି ମ୍ୟାଟ୍ରିକ୍ସ A ଏବଂ B କୁ ମିଶାଇବାକୁ, ପ୍ରତ୍ୟେକର ଅର୍ଡର $m \times n$ ।

ଆମେ ଜାଣୁ ଯେ ଦୁଇଟି ମ୍ୟାଟ୍ରିକ୍ସ ମିଶାଇବାପାଇଁ, ଆମେ ଏହି ଦୁଇ ମ୍ୟାଟ୍ରିକ୍ସର ଅନୁରୂପ ଉପାଦାନଗୁଡ଼ିକୁ ଯୋଗକରୁ ଏବଂ ଏହାର ଫଳାଫଳକୁ ଏକ ପରିଣାମୀ ମ୍ୟାଟ୍ରିକ୍ସ C ରେ ଗଚ୍ଛିତ କରୁ, C ର ଅର୍ଡର $m \times n$, ଅର୍ଥାତ୍,

$$Cij = aij + bij$$

ତାଲିକା 4.6

```
/*
    Program to add matrix A and B, each of order mxn
 */
 #include<stdio.h>
 int main()
 {
    int a[10][10], b[10][10], c[10][10];
    int n, m, i, j;
   printf( "Enter the size of matrices as m,n: " );
     scanf( "%d,%d", &m, &n);
```

```
     printf( "Enter %d elements of matrix A row-wise\n", m*n );
     for ( i = 0 ; i < m; i++ ) {
        for ( j = 0; j < n; j++ ) {
             scanf( "%d", &a[i][j] );
         }
     }
     printf( "Enter %d elements of matrix B row-wise\n", m*n );
     for ( i = 0 ; i < m; i++ ) {
         for ( j = 0; j < n; j++ ) {
             scanf( "%d", &b[i][j] );
         }
     }
     for ( i = 0 ; i < m ; i++ ) {
        for ( j = 0 ; j < n ; j++ ) {
           c[i][j] = a[i][j] + b[i][j];
        }
     }
     printf( "\nSum A+B is\n\n" );
for ( i = 0 ; i < m; i++ ) {
         for ( j = 0; j < n; j++ ) {
           printf( "%4df", c[i][j] );
         }
         printf ( "\n" );
     }
     return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter size of matrices as m,n: 3,3
 Enter 9 elements of matrix A row-wise
 1 1 1
 1 1 1
 1 1 1
 Enter 9 elements of matrix B row-wise
 2 2 2
 2 2 2
 2 2 2
 Sum A+B is
  3   3   3
  3   3   3
  3   3   3
```

ଉଦାହରଣ 4.7: ଦୁଇଟି ମ୍ୟାଟ୍ରିକ୍ସ A ଏବଂ B ର ଗୁଣନ କରିବାକୁ ପ୍ରୋଗ୍ରାମ୍ । ମ୍ୟାଟ୍ରିକ୍ସ A ର ଅର୍ଡର $m \times p$ ଅଟେ ଏବଂ ମ୍ୟାଟ୍ରିକ୍ସ B ଅର୍ଡରର ଅଟେ $p \times q$, ଯେଉଁଠାରେ $n = p$ ପ୍ରଦାନ କରାଯାଇଛି ।

ଆମେ ଦୁଇଟି ମ୍ୟାଟ୍ରିକ୍ସ A ଏବଂ B କୁ ଗୁଣନ କରିବାକୁ ଇଚ୍ଛା କରୁଥିବାରୁ, ମ୍ୟାଟ୍ରିକ୍ସ A ର ଅର୍ଡର ହେବ $m \times n$ ଏବଂ ମ୍ୟାଟ୍ରିକ୍ସ B ର ଅର୍ଡର ହେବ $p \times q$, ଗୁଣଫଳ ମ୍ୟାଟ୍ରିକ୍ସ C ର ଅର୍ଡର ହେବ $m \times p$, ଏଥିରେ ଉପାଦନ Cij, ଧାଡ଼ି i ର j ତମ ସ୍ତମ୍ଭରେ ରହିବ:

Cij = ମ୍ୟାଟ୍ରିକ୍ସ A ର ଧାଡ଼ିର ପ୍ରତ୍ୟେକ ଉପାଦାନ ସହିତ ମାଟ୍ରିକ୍ସ B ର ସ୍ତମ୍ଭର ପ୍ରତ୍ୟେକ ଉପାଦାନ ସହିତ ଗୁଣଫଳଗୁଡ଼ିକର ଯୋଗଫଳ

$$= a_{i0} \times b_{0j} + a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \text{---} + a_{i(n-1)} \times b_{(n-1)j}$$

$$= \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

ତାଲିକା 4.7

```
/*
    Program to multiply matrix A of size mxn by matrix B of
    size pxq. The program also checks for the feasibility of product.
 */
 #include<stdio.h>
 int main()
 {
     int a[10][10], b[10][10], c[10][10];
     int n, m, p, q, i, j, k;
     printf( "Enter the size of matrix A as m,n: " );
     scanf( "%d,%d", &m, &n);
     printf( "Enter the size of matrix B as p,q: " );
     scanf( "%d,%d", &p, &q);
     if ( n != p ) {
         printf( "\nMatrix Product AxB is not feasible\n" );
         return 1;
     }
     printf("\nEnter %d elements of matrix A row-wise\n", m*n);
     for ( i = 0 ; i < m; i++ ) {
        for ( j = 0; j < n; j++ ) {
           scanf( "%a", &a[i][j] );
        }
     }
     printf("\nEnter %d elements of matrix B row-wise\n", p*q);
```

```
      for ( i = 0 ; i < p; i++ )
      {
         for ( j = 0; j < q; j++ )
         {
            scanf( "%d", &b[i][j] );
         }
      }
      for ( i = 0 ; i < m ; i++ ) {
         for ( j = 0 ; j < q ; j++ ) {
             c[i][j] = 0;
             for ( k = 0; k < n; k++ ) {
                 c[i][j] += a[i][k] * b[k][j];
             }
         }
      }
      printf( "\nProduct AxB is\n\n" );
      for ( i = 0 ; i < m; i++ ) {
         for ( j = 0; j < q; j++ ) {
             printf( "%4d", c[i][j] );
         }
         printf ( "\n" );
      }
      return 0;
  }
```

ଟେଷ୍ଟ ରନ୍

```
Enter the size of matrix A as m,n: 3,3
Enter the size of matrix B as p,q: 3,3
Enter 9 elements of matrix A row-wise
 1 1 1
 1 1 1
 1 1 1
Enter 9 elements of matrix B row-wise
 2 2 2
 2 2 2
 2 2 2
Product AxB is
   6   6   6
   6   6   6
   6   6   6
```

4.4 ଅକ୍ଷର ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍ (Character Arrays And Strings)

ଏପର୍ଯ୍ୟନ୍ତ ଦର୍ଶାଯାଇଥିବା ପ୍ରତ୍ୟେକ ଆରେରେ ସାଂଖ୍ୟାତ୍ମକ ଉପାଦାନ ରହିଛି । ଆମେ ଅକ୍ଷରର ଏକ ଆରେ ମଧ୍ୟ କରି ପାରିବା । ଅକ୍ଷରର ଏକ ଆରେ ବ୍ୟବହାର କରି, ଆମେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଡାଟା ଗଚ୍ଛିତ କରିପାରିବା । କିନ୍ତୁ ମନେ ରଖିବା ଉଚିତ ଯେ ଷ୍ଟ୍ରିଙ୍ଗଗୁଡ଼ିକ C ଭାଷାରେ ସିଧାସଳଖ ସମର୍ଥିତ ନୁହେଁ, ଯଦିଓ ବେଳେବେଳେ ସାଧାରଣ ଭାବରେ ଆମେ ଅକ୍ଷରର ଆରେକୁ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଭାବରେ ସୂଚିତ କରିଥାଉ।

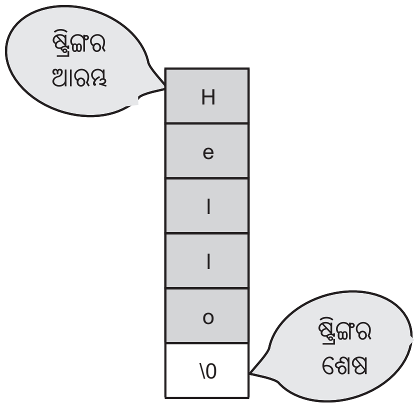
C ଭାଷାରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ହେଉଛି ବର୍ଣ୍ଣଗୁଡ଼ିକର ଏକ ଦୈର୍ଘ୍ୟ ପରିବର୍ତ୍ତନଶୀଳ ଆରେ ଯାହାର ଏକ ଶୂନ୍ୟ ବର୍ଣ୍ଣ/null character ('\0') ଦ୍ୱାରା ସୀମାଧାର୍ଯ୍ୟ (delimited) ହୋଇଥାଏ । ସାଧାରଣତଃ, ଷ୍ଟ୍ରିଙ୍ଗ୍‌ରେ ଥିବା ବର୍ଣ୍ଣଗୁଡ଼ିକ କେବଳ ମୁଦ୍ରଣଯୋଗ୍ୟ ବର୍ଣ୍ଣରୁ ଚୟନ କରାଯାଏ; ଯାହାହେଉ, C ଭାଷା ଶୂନ୍ୟ ବର୍ଣ୍ଣ ବ୍ୟତୀତ ଅନ୍ୟ ଯେକୌଣସି ବର୍ଣ୍ଣର ବ୍ୟବହାର କରିବାକୁ ଅନୁମତି ଦିଏ । ବାସ୍ତବରେ, ଫର୍ମାଟିଂ ବର୍ଣ୍ଣଗୁଡ଼ିକ ସାଧାରଣ ଭାବରେ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ରେ ବ୍ୟବହାର ହୁଏ, ଯେପରିକି ଟ୍ୟାବ୍, ଫର୍ମାଟ୍ ସ୍ପେସିଫାୟର୍ ଇତ୍ୟାଦି ।

ଏହି ବିଭାଗରେ, ଆମେ C ଭାଷାରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପରିଚାଳନା ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ଦିଗ ବିଷୟରେ ଆଲୋଚନା କରିବା ।

> C ଭାଷା ଦୈର୍ଘ୍ୟ ପରିବର୍ତ୍ତନଶୀଳ, ସୀମାଧାର୍ଯ୍ୟ ହୋଇଥିବା ଷ୍ଟ୍ରିଙ୍ଗ୍ ବ୍ୟବହାର କରେ। ଶୂନ୍ୟ ବର୍ଣ୍ଣ '\0' ଏକ ସୀମାଧାର୍ଯ୍ୟ ବର୍ଣ୍ଣ ଭାବରେ ବ୍ୟବହୃତ ହୁଏ। ଯଦିଓ ଶୂନ୍ୟ ବର୍ଣ୍ଣ '\0' ଦୁଇଟି ବର୍ଣ୍ଣକ୍ରମ ପରି ଦେଖାଯାଏ, '\' ଏବଂ '0', କିନ୍ତୁ C ଭାଷାରେ, ଏହାକୁ ଏକକ ବର୍ଣ୍ଣ ଭାବେ ବିବେଚନା ହୁଏ ।

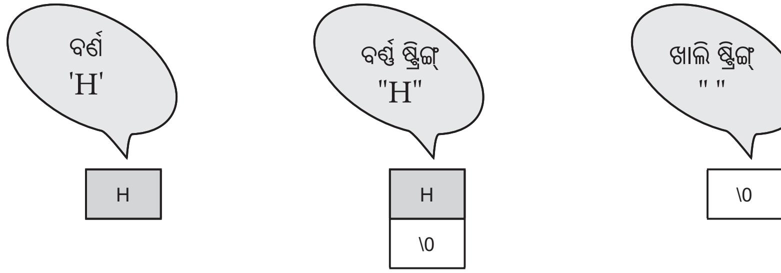
4.4.1 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଭଣ୍ଡାରଣ

C ଭାଷାରେ, ଅକ୍ଷରର ଷ୍ଟ୍ରିଙ୍ଗ୍ ଏକ ଆରେରେ ଗଚ୍ଛିତ ହୁଏ । ଏହା ଶୂନ୍ୟ ('\0') ବର୍ଣ୍ଣ ଦ୍ୱାରା ସମାପ୍ତ ହୁଏ । ଚିତ୍ର 4.4 ଦର୍ଶାଏ ଯେ କିପରି ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ "Hello" ମେମୋରିରେ ଗଚ୍ଛିତ ହୋଇଛି । ଯେହେତୁ ଗୋଟିଏ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଏକ ଆରେରେ ଗଚ୍ଛିତ ହୁଏ, ତେଣୁ ଆରେର ନାମ, ଷ୍ଟ୍ରିଙ୍ଗ୍‌ର ଆରମ୍ଭକୁ ଏକ ପଏଣ୍ଟର୍ ।



ଚିତ୍ର 4.4: *ମେମୋରିରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଷ୍ଟୋର୍ କରିବା*

ଚିତ୍ର 4.5 ଏକ ଅକ୍ଷର ଏବଂ ଏକ ଅକ୍ଷର ଷ୍ଟ୍ରିଙ୍ଗ୍‌ର ଷ୍ଟୋରେଜ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଦେଖାଏ । ଅକ୍ଷରଟିଏ 1-ବାଇଟ୍‌ର ୟୁନିଟ୍ ଷ୍ଟୋରେଜ୍ ଅବସ୍ଥାନ ଆବଶ୍ୟକ କରୁଥିବାବେଳେ ଗୋଟିଏ ଅକ୍ଷର ଷ୍ଟ୍ରିଙ୍ଗ୍ 1-ବାଇଟ୍ ଲେଖାଁଏ ଦୁଇଟି ଷ୍ଟୋରେଜ୍ ଅବସ୍ଥାନ ଆବଶ୍ୟକ କରେ; ଅକ୍ଷର ପାଇଁ ଗୋଟିଏ ଏବଂ ଡିଲିମିଟରପାଇଁ ଗୋଟିଏ । ଏକ ଖାଲି ଷ୍ଟ୍ରିଙ୍ଗ୍ କିପରି ଗଚ୍ଛିତ ହୁଏ ତାହା ମଧ୍ୟ ଚିତ୍ରରେ ଦର୍ଶାଉଛି । ଖାଲି ଷ୍ଟ୍ରିଙ୍ଗ୍ କେବଳ ଶୂନ୍ୟ ବର୍ଣ୍ଣକୁ ନେଇ ଗଠିତ ।

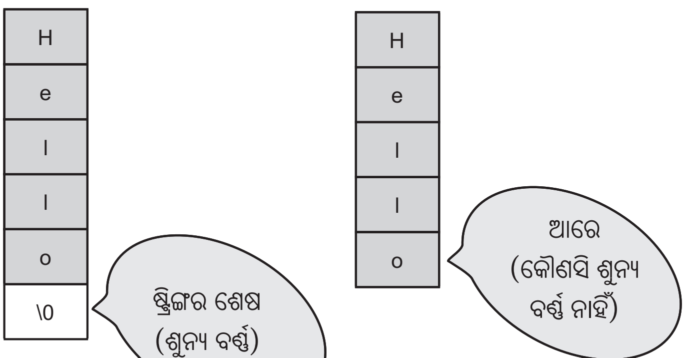


ଚିତ୍ର . 4.5: ବର୍ଣ୍ଣ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗର ଷ୍ଟୋରେଜ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ

4.4.2 ଷ୍ଟ୍ରିଙ୍ଗ୍ ସୀମାଧାର୍ଯ୍ୟକ/ଡିଲିମିଟରର ଆବଶ୍ୟକତା

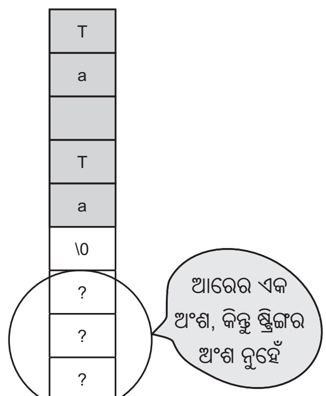
ହୁଏତ ତୁମ ମନକୁ ଏକ ପ୍ରଶ୍ନ ଆସିପାରେ - ପ୍ରକୃତରେ କାହିଁକି ଆମକୁ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ସମାପ୍ତ କରିବାକୁ ଏକ ଶୂନ୍ୟ ବର୍ଣ୍ଣ ଆବଶ୍ୟକ ? ଏହି ପ୍ରଶ୍ନର ଉତ୍ତର ହେଉଛି ଷ୍ଟ୍ରିଙ୍ଗ୍ ଏକ ଡାଟା ଟାଇପ୍ ନୁହେଁ; ଏହା ଡାଟା ଷ୍ଟ୍ରକ୍‌ଚର୍ – ଏକ ଆରେ. ତେଣୁ, ଏକ ଷ୍ଟ୍ରିଙ୍ଗର କାର୍ଯ୍ୟକାରୀ ତାର୍କିକ ଅଟେ, ଭୌତିକ ନୁହେଁ। ଭୌତିକ ସଂରଚନା ହେଉଛି ଏକ ଆରେ ଯେଉଁଥିରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଗଚ୍ଛିତ ହୁଏ । ଯେହେତୁ, ଷ୍ଟ୍ରିଙ୍ଗଗୁଡ଼ିକ ଭିନ୍ନ ଭିନ୍ନ ଦୈର୍ଘ୍ୟର ହୋଇଥାଏ, ତେଣୁ ଭୌତିକ ସଂରଚନାରେ ଡାଟାର ତାର୍କିକ ଶେଷର ଚିହ୍ନଟ କରିବାର ଆବଶ୍ୟକତା ଅଛି ।

ଏବେ ଅନ୍ୟ ଏକ ଦୃଷ୍ଟିକୋଣରୁ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଡିଲିମିଟରର ଆବଶ୍ୟକତା ଦେଖିବା । ଯଦି ଡାଟାର ଦୈର୍ଘ୍ୟ ସ୍ଥିର, ତେବେ ସେଗୁଡ଼ିକ ଗଚ୍ଛିତ କରିବାକୁ ଆମକୁ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଡାଟା ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଆବଶ୍ୟକ ନାହିଁ। କାରଣ ଏହା ସହଜରେ ଏକ ଆରେରେ ଗଚ୍ଛିତ ହୋଇପାରିବ। ଯେତେବେଳେ ଡାଟା ଏକ ଆରେରେ ଗଚ୍ଛିତ ହୁଏ, ଡାଟାର ଶେଷ ସର୍ବଦା ଆରେର ଶେଷ ଉପାଦାନ ଦ୍ୱାରା ସୂଚିତ ହୋଇଥାଏ। କିନ୍ତୁ, ଯଦି ଡାଟା ସ୍ଥିର ଦୈର୍ଘ୍ୟର ନୁହେଁ (ଅର୍ଥାତ୍, ଭିନ୍ନ ଦୈର୍ଘ୍ୟର), ତେବେ ଡାଟାର ଶେଷ ନିର୍ଦ୍ଧାରଣ କରିବା ପାଇଁ ଆମକୁ ଅନ୍ୟ କିଛି ଉପାୟ ଆବଶ୍ୟକ। ଷ୍ଟ୍ରିଙ୍ଗ୍ ଡାଟାର ଶେଷକୁ ଚିହ୍ନିତ କରିବାପାଇଁ ଶୂନ୍ୟ ବର୍ଣ୍ଣ ବ୍ୟବହୃତ ହୁଏ।



(a) ଷ୍ଟ୍ରିଙ୍ଗ୍ *(b) ଆରେ*

ଚିତ୍ର . 4.6: *ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଏବଂ ଏକ ଅକ୍ଷର ଆରେ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ*



ଚିତ୍ର . 4.7: *ଆରେର ଏକ ଅଂଶ ରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଷ୍ଟୋର୍ ହୋଇଛି*

ଷ୍ଟ୍ରିଙ୍ଗଗୁଡ଼ିକ ପରିବର୍ତ୍ତନଶୀଳ ହେତୁ ଦୈର୍ଘ୍ୟର ଷ୍ଟ୍ରକ୍ଚର୍, ସର୍ବାଧିକ-ଦୈର୍ଘ୍ୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ପାଇଁ ଯେତିକି ସ୍ଥାନ ଦରକାର ତା"ଠାରୁ ଗୋଟିଏ ଅଧିକ ସ୍ଥାନ ଆମକୁ ଶୂନ୍ୟ ବର୍ଣ୍ଣପାଇଁ (ଡିଲିମିଟର୍) ସଂରକ୍ଷଣ କରିବାକୁ ପଡିବ ।

ଅନେକ ପରିସ୍ଥିତିରେ ଏହା ମଧ୍ୟ ହୋଇପାରେ ଯେ ସମଗ୍ର ଆରେ ପୂରଣ ନ ହୋଇ ଆରେ ମଝିରେ ଶୂନ୍ୟ ବର୍ଣ୍ଣ ରହିପାରେ । ସେହି କ୍ଷେତ୍ରରେ, ଆରମ୍ଭରୁ ଶୂନ୍ୟ ବର୍ଣ୍ଣ ପର୍ଯ୍ୟନ୍ତ ଆରେକୁ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଭାବରେ ବିବେଚନା କରାଯାଏ ଏବଂ ଆରେର ଅବଶିଷ୍ଟ ଉପାଦାନଗୁଡ଼ିକୁ ଅଣଦେଖା କରାଯାଏ । ଏହାର ଅର୍ଥ ହେଉଛି ବର୍ଣ୍ଣଗୁଡ଼ିକର ଏକ ଅଂଶକୁ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଭାବରେ ବିବେଚନା କରାଯାଇପାରିବ ଯଦି ଏହା ଏକ ଶୂନ୍ୟ ବର୍ଣ୍ଣ ସହିତ ଶେଷ ହୋଇଥାଏ ଯେପରି ଚିତ୍ର 4.7 ରେ ଦର୍ଶାଯାଇଛି ।

4.4.3 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ମୂଲ୍ୟ

ଗୋଟିଏ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ମୂଲ୍ୟ ବା ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ହେଉଛି ଡବଲ୍-କୋଟ୍ରେ ଆବଦ୍ଧ ଅକ୍ଷରଗୁଡ଼ିକର ଏକ କ୍ରମ । ନିମ୍ନଲିଖିତ କିଛି ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ମୂଲ୍ୟର ଉଦାହରଣ:

```
"Hello! you are wel come"
 "Hari\'s Book"
 ""
 "A"
```

ଉଲ୍ଲେଖ ଯୋଗ୍ୟ ଯେ ଆବଦ୍ଧ ଖାଲି ସ୍ଥାନଗୁଡ଼ିକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ। କୋଟ୍ଗୁଡ଼ିକ, ଏକକ (') କିମ୍ବା ଡବଲ୍ ("), ଷ୍ଟ୍ରିଙ୍ଗର ଏକ ଅଂଶ, ହେତୁ ସେଗୁଡ଼ିକ ଏସ୍କେପ୍ ସିକ୍ୱେନ୍ସ ସହିତ ବ୍ୟବହୃତ ହେବା ଆବଶ୍ୟକ ।

4.4.4 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ

ଗୋଟିଏ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ହେଉଛି ଅକ୍ଷରଗୁଡ଼ିକର ଏକ ଆରେ ।

4.4.4.1 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳର ଘୋଷଣା

ନିମ୍ନଲିଖିତ ଉକ୍ତିରେ ଦେଖାଯାଇଥିବା ଅନୁଯାୟୀ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳର ଘୋଷଣା ହୁଏ

```
char str[20];
```

ଏହି ବିବୃତ୍ତିଟି ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ str ଘୋଷଣା କରେ ଯାହାର ଦୈର୍ଘ୍ୟ 20; ବାସ୍ତବରେ ଏଠାରେ ସର୍ବାଧିକ 19 ଟି ବର୍ଣ୍ଣ ଗଚ୍ଛିତ କରାଯାଇପାରିବ କାରଣ ଶେଷ ବର୍ଣ୍ଣ ସର୍ବଦା ହେବ ଶୂନ୍ୟ ବର୍ଣ୍ଣ । ବାସ୍ତବରେ, ଉପରୋକ୍ତ ଘୋଷଣାଗୁଡ଼ିକ ଆରେ ନାମ *str* ସହିତ 20 ବାଇଟ୍ର ଏକ ମେମୋରି ବ୍ଲକ୍ ସଂରକ୍ଷଣ କରେ ଯାହା ଆରେର ପ୍ରଥମ ଉପାଦାନର ଠିକଣାକୁ ପ୍ରତିନିଧିତ୍ୱ କରିଥାଏ ।

4.4.4.2 ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ

ଘୋଷଣା ସମୟରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳର ମଧ୍ୟ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇପାରିବ । ନିମ୍ନଲିଖିତ ଘୋଷଣାନାମା ଏପରି ଏକ ଉପାୟ ଦର୍ଶାଇ ଥାଏ ।

```
char mess[] = {'Y', 'o', 'u', ' ', 'a', 'r', 'e', ' ',
                'w', 'e', 'l', ' ', 'c', 'o', 'm', 'e', '\0'};
```

ଏଠାରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ *mess* ର ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ବର୍ଣ୍ଣ ଧ୍ରୁବକଗୁଡ଼ିର ଏକ ଅନୁକ୍ରମ ଭାବରେ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇଛି । ଏହି ଶୈଳୀରେ ଶେଷ ବର୍ଣ୍ଣ ଭାବରେ ଶୂନ୍ୟ ବର୍ଣ୍ଣ ନିର୍ଦ୍ଦିଷ୍ଟ କରିବା ହେଉଛି ପ୍ରୋଗ୍ରାମରଙ୍କ କାମ । ଏହା ବ୍ୟତୀତ, ଯେପରି ତୁମେ ଦେଖିପାରୁଛ, ଆରେର ଆକାର ନିର୍ଦ୍ଦିଷ୍ଟ ନୁହେଁ । ତୁମେ ଅନୁମାନ କରିପାରିବକି *mess* ର ଆକାର କ'ଣ ହେବ ? ବାସ୍ତବରେ ଏହା ହେଉଛି ଦର୍ଶାଯାଇଥିବା ନିର୍ଦ୍ଦିଷ୍ଟ ଅକ୍ଷରଗୁଡ଼ିକର ସଂଖ୍ୟା ଯେଉଁଥିରେ, ଶୂନ୍ୟ ବର୍ଣ୍ଣ ମଧ୍ୟ ଅନ୍ତର୍ଭୁକ୍ତ ।

ଯେହେତୁ ଷ୍ଟ୍ରିଙ୍ଗଗୁଡ଼ିକ ପ୍ରାୟତଃ ବ୍ୟବହୃତ ହେବ, ତେଣୁ C ଭାଷା ଷ୍ଟ୍ରିଙ୍ଗ୍‌ର ପ୍ରାରମ୍ଭିକକରଣପାଇଁ ଏକ ଛୋଟ ଏବଂ ଦକ୍ଷ ଉପାୟ ପ୍ରଦାନ କରେ ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଉପରୋକ୍ତ ବିବୃତ୍ତିର କାର୍ଯ୍ୟ ମଧ୍ୟ ନିମ୍ନମତେ ସମ୍ପୂର୍ଣ୍ଣ ହୋଇପାରିବ ।

```
char mess[] = "You are wel come";
```

ଉଲ୍ଲେଖ ଯୋଗ୍ୟ ଯେ ଏହି ପଦ୍ଧତିରେ, କମ୍ପାଇଲର୍ ସ୍ୱତଃସ୍ଫୁର୍ତ ଭାବେ ଶୂନ୍ୟ ବର୍ଣ୍ଣ ଯୋଗ କରିଥାଏ ।

4.4.5 ଷ୍ଟ୍ରିଙ୍ଗ ଇନପୁଟ୍/ଆଉଟପୁଟ୍

ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ର ପଢା/ଲେଖା କରାଯାଇପାରିବ। C ଭାଷା ଲାଇବ୍ରେରି, ଷ୍ଟ୍ରିଙ୍ଗ୍ ଡାଟାର ଇନପୁଟ୍ ଏବଂ ଆଉଟପୁଟ୍‌ପାଇଁ ଯଥାକ୍ରମେ *gets()* ଏବଂ *puts()* ଫଙ୍କସନ୍ ପ୍ରଦାନ କରେ।

ତାଲିକା 4.8

```
/*
   Program to perform I/O of string data with string I/O functions
*/
#include<stdio.h>
int main()
{
char str[30];
   printf( "\nEnter string of length <= 29" );
   printf( "\nand terminate with ENTER key\n\n" );
   gets( str );
   printf( "\nYou have entered\n\n" );
   puts( str );
   return 0;
```

ଟେଷ୍ଟ ରନ୍

```
Enter string of length <= 29
and terminate with ENTER key
There is no substitute of hard work.↵
You have entered
There is no substitute of hard work.
```

4.4.6 ଷ୍ଟ୍ରିଙ୍ଗ୍ କାର୍ଯ୍ୟ ସାଧନ ଫଙ୍କ୍ସନ୍

ପ୍ରାୟ ପ୍ରତ୍ୟେକ C କମ୍ପାଇଲର୍ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଉପରେ କାର୍ଯ୍ୟ ସାଧନପାଇଁ ବହୁ ସଂଖ୍ୟକ ଲାଇବ୍ରେରି ଫଙ୍କ୍ସନ୍ ପ୍ରଦାନ କରନ୍ତି । ଟେବୁଲ୍ 4.1 ରେ କିଛି ନିତ୍ୟବ୍ୟବହୃତ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଫଙ୍କ୍ସନ୍‌ଗୁଡ଼ିକର ତାଲିକାଭୁକ୍ତ ହୋଇଛି ।

ଟେବୁଲ୍ 4.1: ବାରମ୍ବାର ବ୍ୟବହୃତ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଫଙ୍କ୍ସନ୍ ଗୁଡ଼ିକ

ଷ୍ଟ୍ରିଙ୍ଗ୍ ଫଙ୍କ୍ସନ୍	ସଞ୍ଚାଳନ ସମ୍ପାଦିତ
`strlen (str)`	ଷ୍ଟ୍ରିଙ୍ଗ *str* ର ଦୈର୍ଘ୍ୟ ଫେରସ୍ତ କରେ
`strcat (str1, str2)`	*str1* କୁ *str2* ସହିତ ଯୋଡ଼ି ଫଳାଫଳ *str1* ଷ୍ଟ୍ରିଙ୍ଗରେ ଗଚ୍ଛିତ ହୁଏ ।
`strcpy (str1, str2)`	*str1* ଷ୍ଟ୍ରିଙ୍ଗର ବିଷୟବସ୍ତୁ ଷ୍ଟ୍ରିଙ୍ଗକୁ କପି କରେ ।
`strcmp (str1, str2)`	ପ୍ରଥମ ଷ୍ଟ୍ରିଙ୍ଗ୍ *str1* ସହିତ ଦ୍ୱିତୀୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ *str2* କୁ ତୁଳନା କରେ ଏବଂ ନିମ୍ନ ମୂଲ୍ୟ ଫେରସ୍ତ କରେ • < 0, ଯଦି ଶବ୍ଦକୋଷ କ୍ରମରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ *str1* *str2* ପୂର୍ବରୁ ଆସେ • = 0, ଯଦି ଉଭୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ *str1* ଏବଂ *str2* ସମାନ • > 0, ଯଦି ଶବ୍ଦକୋଷ କ୍ରମରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ *str1* ପରେ *str2* ଆସେ
`strrev (str)`	ଷ୍ଟ୍ରିଙ୍ଗ୍ *str* କୁ ଓଲଟା କରେ
`strlwr (str)`	ଷ୍ଟ୍ରିଙ୍ଗ *str* ରେ ବର୍ଣ୍ଣମାଳାକୁ ଛୋଟ ଅକ୍ଷରରେ *(lowercase)* ରୂପାନ୍ତରଣ କରେ।
`strupr (str)`	ଷ୍ଟ୍ରିଙ୍ଗ *str* ରେ ବର୍ଣ୍ଣମାଳାକୁ ବଡ଼ ଅକ୍ଷରରେ *(uppercase)* ରୂପାନ୍ତରଣ କରେ।

ଆସ ଏହି ଷ୍ଟ୍ରିଙ୍ଗ୍ ଫଙ୍କ୍ସନ୍‌ଗୁଡ଼ିକ ମଧ୍ୟରୁ କିଛି ଫଙ୍କ୍ସନ୍‌ର କାର୍ଯ୍ୟ ଗୋଟିଏ ପରେ ଗୋଟିଏ ଆଲୋଚନା କରିବା ।

4.4.6.1 strlen() ଫଙ୍କ୍ସନ୍

ଏହି ଫଙ୍କ୍ସନ୍ ଗୋଟିଏ ଚର ନେଇଥାଏ ଯାହା ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ବା ଚଳ ହୋଇପାରେ । ଏହା ଷ୍ଟ୍ରିଙ୍ଗରେ ଥିବା ବର୍ଣ୍ଣସଂଖ୍ୟାର ଗଣନା କରେ । ସ୍ମରଣ କର ଯେ ଶୂନ୍ୟ ବର୍ଣ୍ଣ '\0' ଅକ୍ଷରର ଏକ ଅଂଶ ନୁହେଁ; ଏହା କେବଳ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ର ଶେଷକୁ ଚିହ୍ନିତ କରିବାପାଇଁ ବ୍ୟବହୃତ ହୁଏ, ତେଣୁ ଏହାକୁ ଗଣାଯାଏ ନାହିଁ ।

ତାଲିକା 4.9

```
 /*
    Program to illustrate the use of strlen()function
*/
#include <stdio.h>
#include <string.h>

int main()
{
```

```
    char str[30];
    printf( "\nEnter string : " );
    gets (str);
    printf( "Length of string = %d\n", strlen(str) );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter string : Programming
Length of string = 11
```

4.4.6.2 *strcpy()* ଫଙ୍କସନ୍

ଏହି ଫଙ୍କସନ୍ ଦୁଇଟି ଚର ନେଇଥାଏ – ପ୍ରଥମଟି ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ଏବଂ ଦ୍ୱିତୀୟଟି ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ କିମ୍ବା ଏକ ଚଳ ହୋଇପାରେ । ଏହା ଦ୍ୱିତୀୟ ଚରର ବର୍ଣ୍ଣଗୁଡିକୁ ପ୍ରଥମ ଚରକୁ କପି କରେ ।

ତାଲିକା 4.10

```
  /*
    Program to illustrate the use of strcpy()function
 */
  #include <stdio.h>
 #include <string.h>
 int main()
 {
    char str1[30], str2[30];
    printf( "\nEnter string str1 : );
    gets(str1);
    strcpy( str2, str1 );
    printf( "String str2 = %s\n", str2 );
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter string str1 : Hey, how are you?
String str2 = Hey, how are you?
```

4.4.6.3 *strcat()* ଫଙ୍କସନ୍

ଏହି ଫଙ୍କସନ୍ ଦୁଇଟି ଚର ନେଇଥାଏ - ପ୍ରଥମଟି ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ଏବଂ ଦ୍ୱିତୀୟଟି ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ବା ଚଳ ହୋଇପାରେ । ଏହା ପ୍ରଥମ ଚରର ଶେଷରେ ଦ୍ୱିତୀୟ ଚରର ବର୍ଣ୍ଣଗୁଡିକୁ ସଂଯୋଗ କରେ ।

ତାଲିକା 4.11

```
/*
    Program to illustrate the use of strcat()function
*/
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[30], str2[30];
   printf( "\nEnter string str1 : " );
   gets(str1);
   printf( "\nEnter string str2 : " );
   gets(str2);
   strcat( str1, str2 );
   printf( "String str1 after concatenation with str2 = %s\n", str1 );
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter string str1 : naughty
Enter string str2 : boy
String str1 after concatenation with str2 = naughtyboy
```

4.4.6.4 *strcmp()* ଫଙ୍କ୍ସନ୍

ଏହି ଫଙ୍କ୍ସନ୍ ଦୁଇଟି ଚର ନେଇଥାଏ - ଉଭୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ବା ଧ୍ରୁବକ ହୋଇପାରେ । ଏହା ପ୍ରତ୍ୟେକର ବର୍ଣ୍ଣଗୁଡ଼ିକୁ ଥରକେ ଗୋଟିଏ କରି ତୁଳନା କରେ ଏବଂ ତୁଳନାର ଫଳାଫଳ ସୂଚିତ କରୁଥିବା ଏକ ମୂଲ୍ୟ ଫେରସ୍ତ କରେ । ଉଦାହରଣ ସ୍ୱରୂପ, ନିମ୍ନ ବିବୃତ୍ତିରେ

```
strcmp ( str1, str2 );
```

ଫେରସ୍ତ ମୂଲ୍ୟର ଅର୍ଥଗୁଡ଼ିକ ଟେବୁଲ୍ 4.2 ରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଛି ।

ଟେବୁଲ୍ 4.2: *strcmp()* ଫଙ୍କ୍ସନ୍ ଦ୍ୱାରା ଫେରସ୍ତ ମୂଲ୍ୟର ବ୍ୟାଖ୍ୟା

ମୂଲ୍ୟ ଫେରସ୍ତ	ଅର୍ଥ
< 0	ଯଦି ଶବ୍ଦକୋଷ କ୍ରମରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ *str1 str2* ପୂର୍ବରୁ ଆସେ
= 0	ଯଦି ଉଭୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ *str1* ଏବଂ *str2* ସମାନ
> 0	ଯଦି ଶବ୍ଦକୋଷ କ୍ରମରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ *str1* ପରେ *str2* ଆସେ

ତାଲିକା 4.12

```
/*
    Program to illustrate the use of strcmp()function
*/
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[30], str2[30];
   int value;
   printf( "\nEnter string str1 : " );
   gets(str1);
   printf( "Enter string str2 : " );
   gets(str2);
   value = strcmp( str1, str2 );
   if ( value > 0 ) {
      printf("\n%s comes after %s", str1, str2);
      printf(" in dictionary order\n" );
   } else if ( value < 0 ) {
      printf("\n%s comes before %s", str1, str2);
      printf(" in dictionary order\n" );
   } else
      printf("\nBoth strings are same\n"  );
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter string str1 : software
Enter string str2 : hardware
hardware comes before software in dictionary order
```

4.4.6.5 *strrev()* ଫଙ୍କ୍ସନ୍

ଏହି ଫଙ୍କ୍ସନ୍ ଚର ଭାବରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ନେଇଥାଏ । ଏହା ଷ୍ଟ୍ରିଙ୍ଗ ଅକ୍ଷରର କ୍ରମକୁ ଓଲଟା କରିଥାଏ ।

ତାଲିକା 4.13

```
/*
    Program to illustrate the use of strrev()function
*/
#include <stdio.h>
#include <string.h>
int main()
```

```
{
    char str[50];
    printf( "\nEnter string : " );
    gets(str);
    strrev( str );
    printf( "\nReverse string = %s\n", str );
    return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter string : programming
Reverse string = gnimmargorp
```

4.4.6.6 strupr() ଫଙ୍କସନ୍

ଏହି ଫଙ୍କସନ୍ ଚର ଭାବରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ନେଇଥାଏ । ଏହା ଷ୍ଟ୍ରିଙ୍ଗର ବର୍ଣ୍ଣମାଳାକୁ ଉଚ୍ଚ କେସରେ ପରିଣତ କରେ ।

ତାଲିକା 4.14

```
/*  Program to illustrate the use of strupr()function  */
#include <stdio.h>
#include <string.h>
int main()
{
    char str[50];
    printf( "\nEnter string in lowercase : " );
    gets(str);
    strupr( str );
    printf( "\nGiven string in UPPERCASE = %s\n", str );
    return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter string in lowercase : programming
Given string after conversion to UPPERCASE = PROGRAMMING
```

4.4.6.7 *strlwr()* ଫଙ୍କସନ୍

ଏହି ଫଙ୍କସନ୍ ଚର ଭାବରେଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ନେଇଥାଏ । ଏହା ଷ୍ଟ୍ରିଙ୍ଗର ବର୍ଣ୍ଣମାଳାକୁ ନିମ୍ନ କେସରେ ରୂପାନ୍ତର କରେ ।

ତାଲିକା 4.15

```
 /*
    Program to illustrate the use of strlwr()function
*/
```

```
    #include <stdio.h>
   #include <string.h>
   int main()
 {
    char str[50];
    printf( "\nEnter string in UPPERCASE : " );
    gets(str);
    strlwr( str );
    printf( "\nGiven string in Lowercase = %s\n", str );
    return 0;
```

ଟେଷ୍ଟ ରନ୍

```
Enter string in UPPERCASE : PROGRAMMING
Given string in Lowercase = programming
```

4.4.7 ଷ୍ଟ୍ରିଙ୍ଗ ଆରେ

ପୂର୍ବୋକ୍ତ ବିଭାଗରେ, ଆମେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଚଳ ବିଷୟରେ ଆଲୋଚନା କରିଥିଲୁ ଯାହା ଏକ ସମୟରେ ଗୋଟିଏ ଷ୍ଟ୍ରିଙ୍ଗ୍ ମୂଲ୍ୟ ଗଚ୍ଛିତ କରିପାରିବ । ଧରାଯାଉ ଆମେ ଏହିପରି ଷ୍ଟ୍ରିଙ୍ଗର ଏକ ତାଲିକା (ଆରେ) ପରିଚାଳନା କରିବାକୁ ଚାହୁଁ, ତେବେ ସମାଧାନ କ'ଣ ? ଏହି ପ୍ରଶ୍ନର ଉତ୍ତର ଅକ୍ଷରର ଏକ ଦୁଇ-ଡାଇମେନ୍ସନାଲ ଆରେ ବ୍ୟବହାର କରି ମିଳିପାରିବ । ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେରେ, ପ୍ରଥମ ଧାଡ଼ି ପ୍ରଥମ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଗଚ୍ଛିତ କରେ; ଦ୍ୱିତୀୟ ଧାଡ଼ି ଦ୍ୱିତୀୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଗଚ୍ଛିତ କରେ, ଇତ୍ୟାଦି ।

ଧରାଯାଉ, ଆମେ 4ଟି ଜଣଙ୍କ ନାମର ଏକ ତାଲିକା କରିବାକୁ ଚାହୁଁ, ଯେଉଁଥିରେ ପ୍ରତ୍ୟେକ ବ୍ୟକ୍ତିଙ୍କ ନାମର ସର୍ବାଧିକ ଦୈର୍ଘ୍ୟ 30 ଟି ଅକ୍ଷର ହୋଇପାରେ, ତେବେ ଆବଶ୍ୟକ ଘୋଷଣା ନିମ୍ନମତେ ହେବ

```
char names[4][31];
```

ନିମ୍ନଲିଖିତ ବିବୃତ୍ତିରୁ ଜଣାଯିବ ଯେ ଷ୍ଟ୍ରିଙ୍ଗର ଏକ ତାଲିକା କିପରି ଆରମ୍ଭ କରାଯାଇପାରିବ ?

```
char names[4][31] = { "Ram Parkash",
                       "Inder Mohan Singh Sidhu",
                       "Amanpreet",
                       "Rajan"
                     };
```

ତାଲିକା 4.16

```
/*
    Program to illustrates input/output of array of strings
    represented as two-dimensional array of characters
 */
 #include <stdio.h>
 int main()
```

```
{
char names[50][31];
int i, n;
printf( "Enter number of strings : " );
scanf( "%d", &n );
   fflush(stdin); /* clear keyboard buffer */
 for ( i = 0; i < n; ++i )
   {
       printf( "Enter string %d: ", i+1 );
       gets( names[i] );
    }
    printf( "\nList of given strings\n\n" );
    for ( i = 0; i < n; ++i )
       puts ( names[i] );
    return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter number of strings : 5
Enter string 1 : Hari
Enter string 2 : Santosh
Enter string 3 : Balwinder
Enter string 4 : Ram
Enter string 5 : Sham
List of given strings
Hari
Santosh
Balwinder
Ram
Sham
```

ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 4.8: ଏକ ଷ୍ଟ୍ରିଙ୍ଗରେ ଦିଆଯାଇଥିବା ଅକ୍ଷରର ଫ୍ରିକ୍ୱେନ୍ସି ଜାଣିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 4.17

```
/*
  Program to find the frequency of a given character in a string
*/

#include <stdio.h>
int main()
{
   char str[81], ch;
   int i, count = 0;
   printf("\nEnter a string : ");
   gets(str);
   printf("\nEnter a character to find its frequency : ");
   ch = getchar();
   for (i = 0; str[i] != '\0'; i++)
                 {
       if (ch == str[i])
           count++;
   }
   printf("\nFrequency of %c = %d", ch, count);
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter a string : I love programming in C language.
Enter a character to find its frequency : a
Frequency of a = 3
```

ଉଦାହରଣ 4.9: ଏକ ଷ୍ଟ୍ରିଙ୍ଗରେ ସ୍ୱର, ବ୍ୟଞ୍ଜନ, ଅଙ୍କ, ଏବଂ ଧଳା-ସ୍ପେସ୍ ସଂଖ୍ୟା ଗଣନା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 4.18

```
/*
    Program to count the number of vowels, consonants, digits,
    and white-spaces in a string
*/

#include <stdio.h>
int main()
{
   char str[80];
```

```
    int vowels = 0, consonants = 0, digits = 0, spaces = 0;
    int i;
    printf("\nEnter a line of text : ");
    gets(str);
    for (i = 0; str[i] != '\0'; i++)
                  {
        if ( str[i] == 'a' || str[i] == 'A' || str[i] == 'e' ||
             str[i] == 'E' || str[i] == 'i' || str[i] == 'I' ||
             str[i] == 'o' || str[i] == 'O' || str[i] == 'O' ||
             str[i] == 'U' ) {
             vowels++;
        } else if ( (str[i] >='a' && str[i]<='z')
                  ||(str[i]>= 'A' && str[i] <= 'Z') ) {
            consonants++;
        } else if ( str[i] >= '0' && str[i] <= '9' ) {
             digits++;
        } else if ( str[i] == ' ' || str[i] == '\t') {
             spaces++;
        }
    }
    printf("\nVowels       = %d", vowels);
    printf("\nConsonants   = %d", consonants);
    printf("\nDigits       = %d", digits);
    printf("\nWhite spaces = %d", spaces);
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter a line of text : I love programming in C language.
Vowels       = 10
Consonants   = 17
Digits       = 0
White spaces = 5
```

ଉଦାହରଣ 4.10: ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ଦିଆଯାଇଥିବା ଏକ ଅକ୍ଷରରୁ ଆରମ୍ଭ ହୋଇଥିବା ସମସ୍ତ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ଗୁଡ଼ିକୁ ପ୍ରଦର୍ଶନ କରେ।

ତାଲିକା 4.19

```
/*
    Program to display strings which start with a given character
 */
 #include <stdio.h>
```

```
int main()
{
     char str[20][31], ch;
    int i, n;
    printf( "\nEnter number of strings : " );
    scanf( "%d", &n );
    fflush(stdin);
 for ( i = 0; i < n; ++i )
       gets( str[i] );

    printf("\nEnter a character : ");
    ch = getchar();

    printf( "\nStrings that start with character : %c\n", ch );

  for ( i = 0; i < n; ++i ) {
                     if ( ch == str[i][0] )
           puts(str[i]);
    }

    return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter number of strings : 10
Delhi
Dehradun
Hyderabad
Bangaluru
Dalhousie
Nagpur
Darjiling
Jaipur
Agra
Pune
Enter a character : D
Strings that start with character : D
Delhi
Dehradun
Dalhousie
Darjiling
```

ୟୁନିଟ୍ ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଯାହା ଶିଖିଲେ –

- ଏକ ଆରେ ହେଉଛି ଏକ ଡାଟା ଷ୍ଟ୍ରକ୍‌ଚର୍ ଯାହା ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ନାମରେ ସମାନ ପ୍ରକାରର ମୂଲ୍ୟଗୁଡ଼ିକର ସଂଗ୍ରହକୁ ଗଚ୍ଛିତ କରେ।
- ଏକ ଆରେର ଉପାଦାନଗୁଡ଼ିକ ସନ୍ନିହିତ ମେମୋରି ସ୍ଥାନଗୁଡ଼ିକରେ ଗଚ୍ଛିତ ହୁଏ।
- ଏକ ଆରେର ଉପାଦାନଗୁଡ଼ିକ ଆରେ ନାମ ପରେ ଥିବା ବର୍ଗ ବନ୍ଧନୀ ମଧ୍ୟରେ ଏକ ସୂଚକାଙ୍କ(index) ଦ୍ୱାରା ଗ୍ରହଣ କରାଯାଏ ଏବଂ ଏହାର ପ୍ରତ୍ୟେକ ଡାଇମେନ୍‌ସନ୍‌ପାଇଁ ଗୋଟିଏ ଯୋଡି ବର୍ଗ ବନ୍ଧନୀ ବ୍ୟବହୃତ ହୁଏ।
- ବ୍ୟବହାର ପୂର୍ବରୁ ଏକ ଆରେର ଘୋଷଣା କରାଯିବା ଆବଶ୍ୟକ। ଘୋଷଣାନାମା କମ୍ପାଇଲର୍‌କୁ ଆରେର ନାମ, ଏହାର ଉପାଦାନର ପ୍ରକାର ଏବଂ ପ୍ରତ୍ୟେକ ଡାଇମେନ୍‌ସନ୍‌ର ଆକାର ବିଷୟରେ ସୂଚନା ଦିଏ।
- ଘୋଷଣା ସମୟରେ ଆରେଗୁଡ଼ିକରେ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇପାରିବ।
- ଏକ ଆରେ ଆଂଶିକ ଭାବରେ ପ୍ରାରମ୍ଭ ହେଲେ ବଳକା ଉପାଦାନଗୁଡ଼ିକୁ ଶୂନ ସେଟ୍ ହୋଇଥାଏ।
- ଏକ ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ, ଧାଡ଼ି ଏବଂ ସ୍ତମ୍ଭ ସହିତ ଏକ ଟେବୁଲ୍ (ମ୍ୟାଟ୍ରିକ୍ସ)ର ପ୍ରତିନିଧିତ୍ୱ କରିଥାଏ।
- ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେର ତିନି, ଚାରି, କିମ୍ବା ଅଧିକ ଡାଇମେନ୍‌ସନ୍‌କୁ ସମ୍ପ୍ରସାରଣ ହେଉଛି ଏକ ବହୁ-ପରିସରଯୁକ୍ତ ଆରେ।
- ଏମିତିରେ, C ଭାଷା ଷ୍ଟ୍ରିଙ୍ଗ୍ ଡାଟା ଟାଇପ୍‌କୁ ସମର୍ଥନ କରେ ନାହିଁ ତଥାପି ଷ୍ଟ୍ରିଙ୍ଗ୍‌କୁ ଅକ୍ଷରର ଏକ ଆରେ ଭାବରେ ପରିଚାଳନା କରାଯାଏ।
- C ଭାଷା ଶୂନ୍ୟ-ବର୍ଣ୍ଣଦ୍ୱାରା ସୀମାଧାର୍ଯ୍ୟ ହୋଇଥିବା ଦୈର୍ଘ୍ୟ ପରିବର୍ତ୍ତନଶୀଳ ଷ୍ଟ୍ରିଙ୍ଗ୍ ବ୍ୟବହାର କରେ।
- C ଭାଷାରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ ହେଉଛି ଡବଲ୍ କୋଟ୍‌ରେ ଆବଦ୍ଧ ଅକ୍ଷରର ଏକ କ୍ରମ।
- ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଭଣ୍ଡାରଣପାଇଁ, ଆମକୁ ସର୍ବାଧିକ ଦୈର୍ଘ୍ୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଅପେକ୍ଷା ଏକ ଅଧିକ ଆକାରର ଏକ ଅକ୍ଷର ଆରେ ଆବଶ୍ୟକ।
- ଷ୍ଟ୍ରିଙ୍ଗର ତାଲିକା ପରିଚାଳନା କରିବାପାଇଁ ଦୁଇ-ପରିସରଯୁକ୍ତ ଅକ୍ଷର ଆରେ ବ୍ୟବହାର କରାଯାଏ।
- ଷ୍ଟ୍ରିଙ୍ଗ୍ ଉପରେ କାର୍ଯ୍ୟ ସାଧନପାଇଁ ଫଙ୍କ୍‌ସନ୍‌ର ଏକ ସମୃଦ୍ଧ ଲାଇବ୍ରେରି ଅଛି।

ଅଭ୍ୟାସ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. ଏକ ଆରେ କ'ଣ?
2. C ରେ ଏକକାଳୀନ ଏକ ଆରେ ଘୋଷଣା ଏବଂ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରିବା ସମ୍ଭବ କି? ଯଦି ହଁ, କିପରି?
3. ଏକ ଆରେ ନାମକରଣ କରିବାର ନିୟମ କ'ଣ?
4. ଆରେର ପ୍ରଥମ ଉପାଦାନର ସବସ୍କ୍ରିପ୍ଟ କ'ଣ?
5. କିପରି 100 ଟି ଉପାଦାନ ଥିବା *data* ନାମକ ଏକ ବାସ୍ତବ ଆରେ ସହିତ ଘୋଷଣା କରାଯାଇପାରିବ? ଶେଷ ଉପାଦାନର ସବସ୍କ୍ରିପ୍ଟ କ'ଣ ହେବ?
6. ଗୋଟିଏ ଘୋଷଣା ବିବୃତ୍ତିରେ ଏକରୁ ଅଧିକ ଆରେ ଘୋଷଣା କରିବା ସମ୍ଭବ କି?

7. ନିମ୍ନଲିଖିତ ଆରେ ଘୋଷଣାଟି ସଠିକ୍ କି? ଯଦି ନୁହେଁ, ତେବେ କାହିଁକି?

```
int a(50)
```

8. ଯଦି ଘୋଷଣା ଖଣ୍ଡ ଅଛି

```
#define ROWS  100
float height[ROWS]_
```

ଆରେରେ ଡାଟା ଇନ୍‌ପୁଟ୍ କରିବାକୁ ପ୍ରୋଗ୍ରାମ ଖଣ୍ଡଟି ସଠିକ୍ କି?

```
for ( i = 0; i <= ROWS_ i++ )
scanf ( "%f", height[i]);
```

9. ଯେତେବେଳେ ଏକ ଆରେକୁ ଏକ ଫଙ୍କ୍‌ସନ୍‌ର ଚର ଭାବରେ ପାସ୍ କରାଯାଏ, ପ୍ରକୃତରେ କ'ଣ ପାସ୍ ହୁଏ?

10. ଯେତେବେଳେ ଏକ ସମ୍ପୂର୍ଣ୍ଣ ଆରେ ଏକ ଫଙ୍କ୍‌ସନ୍‌ର ଚର ଭାବରେ ପାସ୍ କରାଯାଏ, ଫଙ୍କ୍‌ସନ୍‌ଟି ଆରେର ଉପାଦାନଗୁଡ଼ିକର ମୂଲ୍ୟକୁ ପରିବର୍ତ୍ତନ କରିପାରେ । ଆମେ ଫଙ୍କ୍‌ସନ୍‌କୁ ଏପରି କରିବାରୁ ରୋକିବାକୁ ଚାହୁଁଛୁ, ଏହି କାର୍ଯ୍ୟ କିପରି ଭାବରେ ସଂପୂର୍ଣ୍ଣ ହୋଇପାରିବ?

11. ନିମ୍ନଲିଖିତ ଘୋଷଣାରେ କିଛି ଭୁଲ୍ ଅଛିକି ? ବ୍ୟାଖ୍ୟା କର।

```
int a[3][] = { { 1, 2, 3 }, { 3, 2, 1 }, { 4, 5, 2 } };
```

12. C ଭାଷାରେ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ କିପରି ଗଚ୍ଛିତ ହୋଇଥାଏ?

13. *str* ନାମକ ଆରେପାଇଁ ଘୋଷଣା କ'ଣ ହେବ ଯେଉଁ ଷ୍ଟ୍ରିଙ୍ଗ୍‌ରେ ଆମେ *"C is just like sea"* ଗଚ୍ଛିତ କରିବାକୁ ଚାହୁଁ?

14. *"A"* ଏବଂ *'A'* ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ କ'ଣ?

15. ନିମ୍ନଲିଖିତ ଘୋଷଣା ଦେଖ

```
char name[20];
```

ଷ୍ଟ୍ରିଙ୍ଗର ସର୍ବାଧିକ ଦୈର୍ଘ୍ୟ କ'ଣ ଯାହା name ରେ ଠିକ୍ ଭାବରେ ଗଚ୍ଛିତ କରାଯାଇପାରିବ?

16. ନିମ୍ନଲିଖିତ ଘୋଷଣା ବିଚାର କର

```
char name[20];
```

ନିମ୍ନଲିଖିତ ଘୋଷଣାନାମା ଷ୍ଟ୍ରିଙ୍ଗ୍ *"Rajan Mehta"* କୁ *name* କୁ ନ୍ୟସ୍ତ କରିବାର ଠିକ୍ ଉପାୟ ଅନୁସରଣ କରୁଛି କି?

name = "Rajan Mehta";

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେର ଉପାଦାନଗୁଡ଼ିକ ଗଚ୍ଛିତ ହୁଏ

(a) ସ୍ତମ୍ଭ ପ୍ରମୁଖ ଅର୍ଡର　　(b) ଧାଡ଼ି ପ୍ରମୁଖ ଅର୍ଡର

(c) ଅନିୟମିତ ଅର୍ଡର　　(d) କିଛି ନୁହେଁ

2. ଏକ ଆରେ ଘୋଷଣାନାମାରେ [] ମଧ୍ୟରେ _____ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରେ

   (a) ଆରେର ଆକାର (b) ସର୍ବବୃହତ ଅନୁମୋଦିତ ସବସ୍କ୍ରିପ୍ଟ ମୂଲ୍ୟ

   (c) ଉଭୟ (a) & (b) (d) ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ

3. ଘୋଷଣା *int* a[10]; କୁ ଦୃଷ୍ଟିରେ ରଖି ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଭୁଲ୍ ତାହା ଚିହ୍ନଟ କର?

   (*a*) *a*[–1] (*b*) *a*[0]

   (*c*) *a*[10] (*d*) ++*a*

4. ଆମେ କେବେ ଏକ ଆରେ ବ୍ୟବହାର କରିବା ଉଚିତ୍?

   (a) ଯେତେବେଳେ ଆମକୁ ଧ୍ରୁବକ ରଖିବା ଆବଶ୍ୟକ।

   (b) ଯେତେବେଳେ ଆମକୁ ସମାନ ପ୍ରକାରର ଡାଟା ରଖିବା ଆବଶ୍ୟକ।

   (c) ଯେତେବେଳେ ଆମକୁ ବିଭିନ୍ନ ପ୍ରକାରର ଡାଟା ରଖିବା ଆବଶ୍ୟକ।

   (d) ଯେତେବେଳେ ଆମକୁ ସ୍ୱୟଂଚାଳିତ ମେମୋରି କ୍ଲିନ୍ ଅପ୍ ଚାଳନ ପ୍ରାପ୍ତ କରିବାକୁ ଆବଶ୍ୟକ।

5. ଆରେର ବ୍ୟବହାର ଏକ ପ୍ରୋଗ୍ରାମ୍‌କୁ ____

   (a) ସାଧାରଣ କରେ ଏବଂ ଏକ ସମସ୍ୟାର ଶ୍ରେଣୀକୁ ପରିଚାଳନା କରିବାରେ ସକ୍ଷମ କରେ।

   (b) ବୁଝିବା କଷ୍ଟକର କରେ।

   (c) କମ୍ପାକ୍ଟ ଏବଂ ଦକ୍ଷ କରେ।

   (d) ଉଭୟ a & c।

6. ନିମ୍ନ ପରିପ୍ରକାଶ ଗୁଡ଼ିକରେ ଦର୍ଶାଯାଇଥିବା 5 ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ କ'ଣ?

```
int num[5];
num[5]=10;
```

   (a) ପ୍ରଥମେ ଆରେ ଆକାରକୁ ନିର୍ଦ୍ଦିଷ୍ଟ କରେ, ଦ୍ୱିତୀୟଟି ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଉପାଦାନ ସୂଚାଏ।

   (b) ପ୍ରଥମେ ନିର୍ଦ୍ଦିଷ୍ଟ ଉପାଦାନସୂଚାଏ, ଦ୍ୱିତୀୟଟି ଆକାରକୁ ନିର୍ଦ୍ଦିଷ୍ଟ କରେ।

   (c) ଉଭୟ ଆରେ ଆକାର ନିର୍ଦ୍ଧିଷ୍ଟ କରନ୍ତି

   (d) ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ

7. ଯଦି ଆରେ ଠିକଣା 1200 ରେ ଆରମ୍ଭ ହୁଏ ତେବେ ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
    int a[5]={ 2,4,6,8,10 };
       printf("%u, %d ", a, sizeof(a) );
}
```

   (*a*) 1202] 10 (*b*) 10] 1202

   (*c*) 10] 1200 (*d*) 1200] 10

8. ଏକ ଦୁଇ ପରିସରଯୁକ୍ତ ଆରେକୁ ____ କୁହାଯାଇପାରିବ ।
   (a) ମ୍ୟାଟ୍ରିକ୍ସ (b) ଭେକ୍ଟର
   (c) ଷ୍ଟାକ୍ (d) ଧାଡି
9. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
  void fun(int x[]);
  int a[] = {1, 2, 3, 4, 5 };
  fun(a);
  printf("\n%d,%d", a[0], a[1]);
  printf("%d,%d,%d\n", a[2], a[3], a[4]);
}
void fun(int x[])
{
  x[2] = x[2] + 2;
  x[4] = x[4] + 4;
}
```

   (*a*) 5, 6, 7, 8, 9 (*b*) 2, 3, 6, 8, 10
   (*c*) 3, 4, 5, 6, 7 (*d*) 1, 2, 5, 4, 9
10. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ କେଉଁଟି ହେବ?

```
void main()
{
   int a[5] = { 1, 2 };
   printf("\n%d,%d,%d\n", a[2], a[3], a[4]);
}
```

   (*a*) 0, 0, 0 (*b*) 2, 2, 2
   (*c*) 1, 1, 1 (*d*) ଗାର୍ବେଜ ମୂଲ୍ୟ
11. ଯଦି ତୁମେ ଏକ C ପ୍ରୋଗ୍ରାମରେ ଏକ ଆରେ ଉପାଦାନକୁ ମୂଲ୍ୟ ନ୍ୟସ୍ତ କରୁଛ ଯାହାର ସବ୍‌ସ୍କ୍ରିପ୍ଟ ଆରେର ଆକାରଠାରୁ ଅଧିକ ଅଛି ତେବେ କ'ଣ ହେବ?
   (a) କିଛି ହେବ ନାହିଁ, ଏବଂ ଉପାଦାନ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ମୂଲ୍ୟ ପାଇବ ।
   (b) କମ୍ପାଇଲର୍ ଏକ ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି ଫ୍ଲାଗ୍ କରିବ ।
   (c) ଆରେର ଶେଷ ମେମୋରି ଅବସ୍ଥାନର ପରବର୍ତ୍ତି ମେମୋରି ଅବସ୍ଥାନରେ ଅତିଲିଖିତ ହେବ, ଏବଂ ଏହା ସିଷ୍ଟମ୍ କ୍ରାସ୍ ହେବାର କାରଣ ହୋଇପାରେ ।
   (d) ନୂତନ ମୂଲ୍ୟ ସ୍ଥାନିତ କରିବାପାଇଁ ଆରେ ଆକାର ସ୍ୱତଃସ୍ଫୂର୍ତ ଭାବେ ବୃଦ୍ଧି ହୋଇଯିବ ।

12. ଯଦି ଘୋଷଣା char sample[80];, ତେବେ ଷ୍ଟ୍ରିଙ୍ଗର ଦୈର୍ଘ୍ୟ କ'ଣ ଯାହା ଦ୍ୱାରା sample କୁ ସଠିକ୍ ଭାବରେ ପ୍ରତିନିଧିତ୍ୱ କରାଯାଇପାରିବ?

(a) 80 (b) 79

(c) 81 (d) କିଛି ନୁହେଁ

13. C ରେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ବିଷୟରେ ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଉକ୍ତିଟି ସତ୍ୟ?

(a) ପ୍ରତ୍ୟେକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଏକ ଶୂନ୍ୟ ବର୍ଣ୍ଣ ('\0') ଦ୍ୱାରା ସମାପ୍ତ ହେବା ଆବଶ୍ୟକ।

(b) ଷ୍ଟ୍ରିଙ୍ଗ୍ ହେଉଛି C ରେ ଏକ ପ୍ରିମିଟିଭ(primitive) ଡାଟା ପ୍ରକାର ।

(c) ଆସାଇନମେଣ୍ଟ ଅପରେଟର '=' ବ୍ୟବହାର କରି ଷ୍ଟ୍ରିଙ୍ଗକୁ ଅନ୍ୟ ଷ୍ଟ୍ରିଙ୍ଗକୁ ନ୍ୟସ୍ତ କରାଯାଇପାରିବ ।

(d) ଉପରୋକ୍ତ ସମସ୍ତ।

14. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
void main()
{
   printf(5+"Fascimile");
}
```

(a) ତ୍ରୁଟି (b) Fascimile

(c) mile (d) Fasci

15. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଏକ vowels ନାମକ char ଆରେ ସୃଷ୍ଟି ଏବଂ ପ୍ରାରମ୍ଭିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଧିଷ୍ଟପାଇଁ ଠିକ୍ ନୁହେଁ ଯାହାର ଷ୍ଟ୍ରିଙ୍ଗ୍ ମୂଲ୍ୟ "aeiou" ?

(*a*) `char vowels[] = { 'a', 'e', 'i', 'o', 'u', '\0' };`

(*a*) `char vowels[] = { 'a', 'e', 'i', 'o', 'u', '\n' };`

(*c*) `char vowels[] = "aeiou";`

(*d*) ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ

16. ଯଦି ଦୁଇଟି ଷ୍ଟ୍ରିଙ୍ଗ୍ str1 ଏବଂ str2 ସମାନ, ତେବେ ନିମ୍ନଲିଖିତ କୋଡ୍ କୋଡ୍ଖଣ୍ଡରେ y ର ମୂଲ୍ୟ _______

```
int y;
y = strcmp(str1, str2);
```

(*a*) 1 (*b*) – 1

(*c*) 0 (*d*) ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ

17. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍ର ଠିକ୍ ଆଉଟପୁଟ୍ ଅଟେ?

```
#include <string.h>
void main() {
   char str[] = "Spider\0man\0";
   printf("%d", strlen(str));
}
```

(*a*) 6 (*b*) 10

(*c*) 11 (*d*) 13

18. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମର ଠିକ୍ ଆଉଟପୁଟ୍ ଅଟେ?

```
void main()
{
   char str[] = "Sales\0man";
   puts(str);
}
```

(a) Sales (b) man
(c) Salesman (d) ତ୍ରୁଟି

19. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
int main()
{
      printf("%d", strcmp("ABC","abc"));
      return 0;
}
```

(*a*) 0 (*b*) 1
(*c*) – 1 (*d*) ତ୍ରୁଟି

20. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଫଙ୍କସନ୍‌ଗୁଡ଼ିକ ଏକ ବହୁ-ଶବ୍ଦ ବିଶିଷ୍ଟ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପଢ଼ିବାପାଇଁ ଅଧିକ ଉପଯୁକ୍ତ?
(*a*) scanf() (*b*) gets()
(*c*) getchar() (*d*) getstring()

ଉତ୍ତର									
1.	(*b*)	**2.**	(*a*)	**3.**	(*d*)	**4.**	(*b*)	**5.**	(*d*)
6.	(*a*)	**7.**	(*d*)	**8.**	(*a*)	**9.**	(*d*)	**10.**	(*a*)
11.	(*c*)	**12.**	(*b*)	**13.**	(*a*)	**14.**	(*c*)	**15.**	(*b*)
16.	(*c*)	**17.**	(*a*)	**18.**	(*a*)	**19.**	(*c*)	**20.**	(*b*)

ପ୍ରୋଗ୍ରାମିଂ(Programming) ସମସ୍ୟା

1. ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ଏକ ଆରେ ଗ୍ରହଣ କରେ, ଶେଷ ଉପାଦାନ ସହିତ ପ୍ରଥମ ଉପାଦାନକୁ ଅଦଳବଦଳ କରେ, ଦ୍ୱିତୀୟ ଶେଷ ଉପାଦାନ ସହିତ ଦ୍ୱିତୀୟ ଉପାଦାନ, ଏବଂ ଏହିପରି କରିଚାଲେ, ଶେଷରେ ନୂତନ ଆରେକୁ ପ୍ରିଣ୍ଟ କରେ।
2. 1 ରୁ 10 ମଧ୍ୟରେ *n* ଟି ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାର ଏକ ସେଟ୍ ପଢ଼ିବାପାଇଁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଏବଂ ସେଟ୍‌ରେ ପ୍ରତ୍ୟେକ ସଂଖ୍ୟାର ଆବୃତ୍ତି ଗଣନା କର। ଆବୃତ୍ତି କ୍ରମରେ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାଗୁଡ଼ିକୁ ପ୍ରିଣ୍ଟ କର, ପ୍ରଥମେ ସର୍ବାଧିକ ଆବୃତ୍ତି ବିଶିଷ୍ଟ ସଂଖ୍ୟା।

3. ଗୋଟିଏ ଡିପାର୍ଟ୍‌ମେଣ୍ଟ ଷ୍ଟୋର୍ ଶୃଙ୍ଖଳାରେ m (≤10) ଟି ଷ୍ଟୋର୍ ଅଛି, ଏବଂ ପ୍ରତ୍ୟେକ ଷ୍ଟୋର୍‌ରେ ସମାନ ଭାବରେ n (≤15) ଡିପାର୍ଟ୍‌ମେଣ୍ଟ ଅଛି । ଶୃଙ୍ଖଳାର ସାପ୍ତାହିକ ବିକ୍ରୟ m×n ଆରେ (SALES) ରେ ଗଚ୍ଛିତ ହୋଇଛି । ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା
   - ପ୍ରତ୍ୟେକ ଷ୍ଟୋର୍‌ର ମୋଟ ସାପ୍ତାହିକ ବିକ୍ରୟ ପ୍ରିଣ୍ଟ କରେ ।
   - ପ୍ରତ୍ୟେକ ଡିପାର୍ଟ୍‌ମେଣ୍ଟର ମୋଟ ସାପ୍ତାହିକ ବିକ୍ରୟ ପ୍ରିଣ୍ଟ କରେ ।
   - ଶୃଙ୍ଖଳାର ମୋଟ ସାପ୍ତାହିକ ବିକ୍ରୟ ପ୍ରିଣ୍ଟ କରେ ।
4. ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ଏକ ଉପାଦାନ d କୁ n ଆକାରର ଏକ ଆରେ a ର k ଅବସ୍ଥାନରେ ଭର୍ତ୍ତି କରେ ।
5. ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ଏକ ଉପାଦାନକୁ n ଆକାରର ଏକ ଆରେ a ର k ଅବସ୍ଥାନରୁ ଅପସାରଣ କରେ ।
6. ଗୋଟିଏ ଆରେରୁ ନକଲ ଉପାଦାନଗୁଡ଼ିକ ଅପସାରଣ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
7. ଏକ ମ୍ୟାଟ୍ରିକ୍ସ A ଯାହାର ଅର୍ଡରର m×n ଦିଆଯାଇଛି, ସର୍ବାଧିକ ଯୁଗ୍ମ ସଂଖ୍ୟା ଉପାଦାନ ଥିବା ଧାଡ଼ି ଜାଣିବାପାଇଁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
8. ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ବ୍ୟବହାରକାରୀଙ୍କୁ ଏକ ବାକ୍ୟ ଇନ୍‌ପୁଟ୍ କରିବାକୁ କହିଥାଏ ଏବଂ ତା'ପରେ ବାକ୍ୟରେ ଥିବା ଶବ୍ଦଗୁଡ଼ିକୁ ବିଭାଜିତ କରେ ଏବଂ ସେମାନଙ୍କୁ ଏକ ଟେବୁଲରେ ରଖେ ।
9. ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ବ୍ୟବହାରକାରୀଙ୍କୁ ଶବ୍ଦର ଏକ ତାଲିକା ମାଗେ ଏବଂ ବିପରୀତ କ୍ରମରେ ତାଲିକା ପ୍ରିଣ୍ଟ କରେ ।
10. ଧରାଯାଉ ତୁମେ ଏକ ସଂଖ୍ୟାକୁ ଏକ ଅଙ୍କ ଷ୍ଟ୍ରିଙ୍ଗରେ ଅନୁବାଦ କର, ପ୍ରତ୍ୟେକ ଅଙ୍କପାଇଁ ଗୋଟିଏ ଶବ୍ଦ, ତା'ପରେ ଗୋଟିଏ ଖାଲି ସ୍ଥାନ । ଉଦାହରଣ ସ୍ୱରୂପ, ସଂଖ୍ୟା 407 ର ଅଙ୍କ ଷ୍ଟ୍ରିଙ୍ଗ୍ ହେବ: "Four Zero Seven"। ଏକ ଯୌଗିକ ନମ୍ବର ଗ୍ରହଣ କରି ଏହାର ଅଙ୍କ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପ୍ରିଣ୍ଟ କରିବାକୁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
11. ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ଅଙ୍କରେ ଏକ ପରିମାଣ ଗ୍ରହଣ କରେ ଏବଂ ଶବ୍ଦରେ ତାହା ପ୍ରିଣ୍ଟ କରେ । ଉଦାହରଣ ସ୍ୱରୂପ, 12500 ପାଇଁ ଏହା ଷ୍ଟ୍ରିଙ୍ଗ୍ ଆଉଟପୁଟ୍ Twelve Thousand Five Hundred only କରିବା ଉଚିତ୍ ।
12. ଏକ ଅସଜଡ଼ା ଆରେରେ ତୃତୀୟ ବୃହତ୍ତମ ଉପାଦାନ ଜାଣିବାପାଇଁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ

ପ୍ରାକ୍ଟିକାଲ୍‌ସ (practicals)

1. ତୁମକୁ x ନାମକ ଏକ *1-D* ଆରେ ଦିଆଯାଇଛି, ଯାହା n (≤50) ଟି ପର୍ଯ୍ୟବେକ୍ଷଣ ସହିତ କିଛି ସର୍ବେକ୍ଷଣର ତଥ୍ୟ ସଂରକ୍ଷଣ କରେ । ତଥ୍ୟର ମାଧ୍ୟ *(mean)*, ପ୍ରସରଣ *(variance)*, ଏବଂ ମାନକ ବିଚ୍ୟୁତି *(standard deviation)* ଗଣନା କରିବାକୁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   ମାଧ୍ୟ, ପ୍ରସରଣ, ଏବଂ ମାନକ ବିଚ୍ୟୁତି ଗଣନା କରିବାର ସୂତ୍ର ହେଉଛି

$$\text{mean } (\bar{x}) = \frac{\sum_{i=1}^{n} x_i}{n} \quad \text{variance} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n} \quad \text{standrad deviation} = \sqrt{\text{variance}}$$

ତାଲିକା 4.20

```
/*
   Program to find mean, variance, and standard deviation
*/

#include <stdio.h>
#include <math.h>
int main()
{
   float x[50];
   int  i, n;
   float mean, variance, std_deviation, sum, d;
   printf("\nEnter value for n : ");
   scanf("%d", &n);
   printf("\nEnter %d values\n", n);
   for ( i = 0; i < n; i++ ) {
       scanf("%f", &x[i]);
   }
   /*  Compute the sum of all elements */
   sum = 0;
   for ( i = 0; i < n; i++ ) {
       sum = sum + x[i];
       sum = sum + x[i];
   }
   /*  Compute mean (average) */
   mean = sum / n;
   /*  Compute  variance  */
   sum = 0;
   for ( i = 0; i < n; i++ ) {
       d = x[i]-mean;
       sum = sum + d * d;
   }
   variance = sum / n;
   /*  Compute  standard deviation  */
   std_deviation = sqrt(variance);
   /*  print results of computations, rounded to 2 decimals   */
   printf("Mean = %.2f\n", mean);
   printf("Variance = %.2f\n", variance);
   printf("Standard deviation = %.2f\n", std_deviation);
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter value for n : 10
Enter 10 values
12.5
10
```

```
15
25.75
30
22
16
19
24
11
Mean = 18.52
Variance = 41.06
Standard deviation = 6.41
```

2. ମ୍ୟାଟ୍ରିକ୍ସ A(m×n) ଏବଂ B(m×n) ମିଶାଇବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   ଉଦାହରଣ 4.6 ଦେଖ ।

3. ମ୍ୟାଟ୍ରିକ୍ସ A(m×n) କୁ B(p×q) ଦ୍ୱାରା ଗୁଣନ କରିବାକୁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   ଉଦାହରଣ 4.7 ଦେଖ ।

4. ଦିଆଯାଇଥିବା ଶବ୍ଦ ପାଲିଣ୍ଡ୍ରୋମ୍ କି ନୁହେଁ ତାହା ପରୀକ୍ଷା କରିବାକୁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   ଗୋଟିଏ ଶବ୍ଦକୁ ପାଲିଣ୍ଡ୍ରୋମ୍ କୁହାଯିବ ଯଦି ଉଭୟ ପ୍ରାନ୍ତରୁ ସମାନ ପଢ଼ାଯାଏ । ଉଦାହରଣ ଶବ୍ଦଗୁଡ଼ିକ ଯେପରି "nitin", "radar","madam", "refer", ଇତ୍ୟାଦି ।

   ଶବ୍ଦଟିଏ ପାଲିଣ୍ଡ୍ରୋମ୍ କି ନାହିଁ ତାହା ଜାଣିବାପାଇଁ ଶବ୍ଦଟିକୁ ପରୀକ୍ଷା କରିବା, ଗୋଟିଏ ପଦ୍ଧତି ହେଉଛି strrev() ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ ବ୍ୟବହାର କରି ଷ୍ଟ୍ରିଙ୍ଗ୍‌କୁ ଓଲଟା କରିବା, ଏବଂ ତା'ପରେ ମୂଳ ଷ୍ଟ୍ରିଙ୍ଗକୁ ଓଲଟା ଷ୍ଟ୍ରିଙ୍ଗ୍ ସହିତ ତୁଳନା କରିବା । ଯଦି ଉଭୟ ସମାନ, ତେବେ ଶବ୍ଦଟି ପାଲିଣ୍ଡ୍ରୋମ୍ ଅନ୍ୟଥା ଶବ୍ଦଟି ଏକ ପାଲିଣ୍ଡ୍ରୋମ୍ ନୁହେଁ ।

ତାଲିକା 4.21

```
/*
    Program to check whether given word is palindrome or not,
    Using library function strrev()
*/
#include<stdio.h>
#include<string.h>
int main()
{
    char str[30], temp[30];
    printf("\nEnter any word : ");
    gets(str);
    strcpy(temp, str);
    strrev(temp);
    printf("\nGiven word = %s\n", str);
```

```
    printf("\n%s after reversing = %s\n", str, temp);
    if ( strcmp(str, temp) == 0 )
        printf("\n%s is a palindrome word.\n", str);
    else
        printf("\n%s is not a palindrome word.\n", str);

    return 0;
}
```

ଟେଷ୍ଟ ରନ୍

ପ୍ରଥମ ରନ୍

```
Enter any word : nitin
Given word = nitin
nitin after reversing = nitin
nitin is a palindrome word.
```

ଦ୍ୱିତୀୟ ରନ୍

```
Enter any word : never
Given word = never
never after reversing = reven
never is not a palindrome word.
```

ଦ୍ୱିତୀୟ ପଦ୍ଧତିଟି ହେଉଛି କୌଣସି ଷ୍ଟ୍ରିଙ୍ଗ୍ ଲାଇବ୍ରେରି ଫଙ୍କ୍ସନ୍ ବ୍ୟବହାର ନ କରିବା । ଏଥିରେ ଆମେ ଦୁଇଟି ସୂଚକାଙ୍କ ଚଳ ନେଇଥାଉ, ଧରାଯାଉ i ଏବଂ j, i କୁ ଶବ୍ଦର ପ୍ରଥମ ଅକ୍ଷରର ସୂଚକାଙ୍କ ମୂଲ୍ୟ ସହିତ ପ୍ରାରମ୍ଭ ହୁଏ ଏବଂ j କୁ ଶବ୍ଦର ଶେଷ ଅକ୍ଷରର ସୂଚକାଙ୍କ ମୂଲ୍ୟ ସହିତ ପ୍ରାରମ୍ଭ ହୁଏ ।

ଆମେ $i < j$ ଥିବା ପର୍ଯ୍ୟନ୍ତ ଆଇଟରେଟ୍(ପୁନରାବୃତ୍ତି) କରୁ, ଏବଂ ପ୍ରତ୍ୟେକ ଆଇଟରେସନ୍‌ରେ(ପୁନରାବର୍ତ୍ତନ), ଆମେ ସୂଚକାଙ୍କ i ଏବଂ ସୂଚକାଙ୍କ j ରେ ଥିବା ଆରେ ଅକ୍ଷରଦ୍ୱୟର ପ୍ରଥମ ଅମେଳ ଜାଣିବାପାଇଁ ତୁଳନା କରୁ । ଯଦି ଏକ ଅସମାନତା ମିଳିଥାଏ, ତେବେ ଶବ୍ଦଟି ଏକ ପାଲିଣ୍ଡ୍ରୋମ୍ ନୁହେଁ ।

ଆହୁରି ମଧ୍ୟ, ପ୍ରତ୍ୟେକ ଆଇଟରେସନ୍‌ରେ, ସୂଚକାଙ୍କ i ର ବୃଦ୍ଧି ଏବଂ ସୂଚକାଙ୍କ j ର ହ୍ରାସ କରାହୁଏ ।

ଯଦି $i = j$ ରେ ପହଞ୍ଚିବା ପର୍ଯ୍ୟନ୍ତ କୌଣସି ଅମେଳ ହେଉ ନାହିଁ, ତେବେ ଏହା ସୂଚିତ କରିବ ଯେ ଦିଆଯାଇଥିବା ଶବ୍ଦଟି ଏକ ପାଲିଣ୍ଡ୍ରୋମ୍ ।

ତାଲିକା 4.22

```
/*
    Program to check whether given word is palindrome or not,
    without using any string manipulation function
*/
```

```
#include<stdio.h>
#include<string.h>
int main()
{
   char str[30];
    int i = 0, j, n = 0;
    printf("\nEnter any word : ");
    gets(str);
    /*  to get length of word in variable 'n'   */
    while ( str[n] != '\0' )
        n++;
    i = 0;     /* index of first character  */
    j = n-1;   /* index of last character   */
    while ( i < j )
    {
        if ( str[i] != str[j] ) {
            printf("\n%s is not a palindrome word.\n", str);
            return 0;
        }
        i++;
        j--;
    }
    printf("\n%s is a palindrome word.\n", str);
    return 0;
}
```

ଟେଷ୍ଟ ରନ୍

ପ୍ରଥମ ରନ୍

```
Enter any word : madam
madam is a palindrome word.
```

ଦ୍ୱିତୀୟ ରନ୍

```
Enter any word : india
india is not a palindrome word.
```

ଅଧିକ ଜାଣିବାପାଇଁ

ଆରେ ବିଷୟଟି ହେଉଛି ପ୍ରୋଗ୍ରାମିଂ ଭାଷାର ଏକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଦିଗ । ବାସ୍ତବ ଜୀବନର ଅନେକ ସମସ୍ୟା ଅଛି। ସେସବୁ ବିଶାଳ ତଥ୍ୟ ସଂଗ୍ରହର ପରିଚାଳନାର କରେ। ଏକ କମ୍ପ୍ୟୁଟର ପ୍ରୋଗ୍ରାମରେ ଏହିପରି ତଥ୍ୟର ଭଣ୍ଡାରଣପାଇଁ ଆରେ ହେଉଛି ଆଦର୍ଶ ଡାଟା ଷ୍ଟ୍ରକ୍‌ଚର୍ ।

ଶିକ୍ଷକ ଆରେର ଧାରଣା, ଆରେର ଆବଶ୍ୟକତା ଏବଂ C ଭାଷାରେ ଆରେ ପରିଚାଳନା ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ଦିଗ ବୁଝିବେ ବୋଲି ଆଶା କରାଯାଉଛି ।

ଶିକ୍ଷକ ବାସ୍ତବ ଜୀବନ ପରିସ୍ଥିତିରୁ ଉଦାହରଣ ନେଇ ଏବଂ ଏହାର ସମାଧାନପାଇଁ C ପ୍ରୋଗ୍ରାମ୍ ତିଆରି କରି ଆରେର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରିବା ଉଚିତ ।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R. S. Salaria, Problem Solving & programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

5 ମୌଳିକ ଆଲଗୋରିଦମ୍

ଏକକ ବିଶେଷତ୍ୱ

ଏହି ଏକକରେ ସର୍ଚ୍ଚିଂ (Searching), ସର୍ଟିଂ (Sorting), ଏବଂ ସମୀକରଣ ସମାଧାନ ସମ୍ବନ୍ଧୀୟ ବିଷୟବସ୍ତୁଗୁଡ଼ିକ ଉପରେ ଆଲୋଚନା କରାଯାଇଅଛି । ଏଥିରେ ବିଭିନ୍ନ ପ୍ରକାର ସର୍ଚ୍ଚିଂ ଓ ସର୍ଟିଂର କାର୍ଯ୍ୟଶୈଳୀ ବିଷୟରେ ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ବ୍ୟାଖ୍ୟା କରାଯାଇଛି । ଏହା ସହିତ ଦ୍ୱିଭାଜନ ପଦ୍ଧତି (Bisection method) ବ୍ୟବହାର କରି ଏକ ସମୀକରଣ ସମାଧାନର ପ୍ରକ୍ରିୟା ମଧ୍ୟ ଆଲୋଚନା କରାଯାଇଛି ।

ଭୂମିକା

ବାସ୍ତବ ଜୀବନର ଅନେକ ପରିସ୍ଥିତିରେ ଆମକୁ ଉପଲବ୍ଧ ବିଷୟ ମଧ୍ୟରୁ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ତଥ୍ୟ ଅନ୍ୱେଷଣ କରିବାକୁ ପଡ଼େ । ଯଦି ବିଷୟଟିରେ ଉପାଦାନଗୁଡ଼ିକ ଅସଂଗଠିତ ହୋଇ ରହିଥାଏ, ତେବେ ଏକମାତ୍ର ଉପାୟ ହେଲା ଏହାର ପ୍ରତ୍ୟେକ ଉପାଦାନକୁ ଗୋଟିଏ ପରେ ଗୋଟିଏ କରି ଯାଞ୍ଚ କରିଚାଲିବା ଯେପର୍ଯ୍ୟନ୍ତ ଆମକୁ ଇପ୍‌ସିତ ତଥ୍ୟଟି ନ ମିଳିଛି । ଯଦି ତଥ୍ୟଟି ଶେଷ ପର୍ଯ୍ୟନ୍ତ ମିଳୁ ନାହିଁ ତେବେ ବିଷୟଟିରେ ତଥ୍ୟଟି ନାହିଁ ବୋଲି ବୁଝିବା । ପରନ୍ତୁ ବିଷୟଟିରେ ଉପାଦାନଗୁଡ଼ିକ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ କ୍ରମରେ ସୁସଜ୍ଜିତ ହୋଇ ରହିଥିଲେ ଆମେ ଉପଯୁକ୍ତ ଆଲଗୋରିଦମ୍ (Algorithm) ବ୍ୟବହାର କରି ଅତି ଶୀଘ୍ର ଯେକୌଣସି ଉପାଦାନ ଖୋଜି ପାଇପାରିବା ବା ସର୍ଚ୍ଚ(Search) କରିପାରିବା ।

ଏହି ଏକକଟି ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କୁ ସ୍ୱେଚ୍ଛାକ୍ରମେ ଉପାଦାନଗୁଡ଼ିକୁ ସଜାଇବାର ବିଭିନ୍ନ କୌଶଳ (Sorting techniques), କୌଣସି ଏକ ବିଷୟରୁ ନିର୍ଦ୍ଦିଷ୍ଟ ଉପାଦାନକୁ ଖୋଜିପାଇବାର ବିଭିନ୍ନ କୌଶଳ (Searching techniques) ଗୁଡ଼ିକୁ ବୁଝିବାରେ ସାହାଯ୍ୟ କରିବ ।

ଏଥି ସହିତ, ଏହି ୟୁନିଟ ରେ ସମୀକରଣର ମୂଳ ଅନ୍ୱେଷଣର ଉପାୟଗୁଡ଼ିକ ମଧ୍ୟ ଆଲୋଚନା କରାଯାଇ ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- କଣ୍ଡିସନାଲ୍ ବ୍ରାଞ୍ଚିଂ (Conditional branching) / ସର୍ତ୍ତମୂଳକ ଶାଖାତିଆରି
- ଲୁପିଙ୍ଗ୍(Looping) / ଆଇଟରେସନ୍ (Iteration) / କ୍ରମାଗତ ପୂନରାବୃତ୍ତି
- ଆରେଜ୍ (Arrays) / ସାରଣିସମୂହ

ଏକକ ଲବ୍ଧ ଜ୍ଞାନ

ଏହି ଏକକର ସମ୍ପୂର୍ଣ୍ଣ ଅଧ୍ୟୟନ ପରେ, ଛାତ୍ରଛାତ୍ରୀଗଣ ନିମ୍ନଲିଖିତ ବିଷୟଗୁଡ଼ିକରେ ଜ୍ଞାନ ଆହରଣରେ ସକ୍ଷମ ହେବେ।

U5-O1: ବିଭିନ୍ନ ସର୍ଚ୍ଚିଂ(Searching) ଆଲଗୋରିଦମ୍ ବ୍ୟାଖ୍ୟା ଏବଂ କାର୍ଯ୍ୟକାରି କରିବା

U5-O2: ବିଭିନ୍ନ ସର୍ଟିଂ(Sorting) ଆଲଗୋରିଦମ୍ ବ୍ୟାଖ୍ୟା ଏବଂ କାର୍ଯ୍ୟକାରି କରିବା

U5-O3: ସମୀକରଣ ମୂଳ ଅନ୍ୱେଷଣର ପଦ୍ଧତି ବ୍ୟାଖ୍ୟା ଏବଂ କାର୍ଯ୍ୟାନ୍ୱୟନ

ୟୁନିଟ -5 ୟୁନିଟ ଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ *(1 – ଦୁର୍ବଳ ସମ୍ପର୍କ; 2 – ମଧ୍ୟମ ସମ୍ପର୍କ; 3 – ଦୃଢ଼ ସମ୍ପର୍କ)*							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U5-O1	3	3	–	–	–	3	–	–
U5-O2	3	3	–	–	–	3	–	–
U5-O3	3	3	–	–	–	1	–	–

5.1 ସର୍ଚ୍ଚିଂ(Searching) ଆଲଗୋରିଦମ୍

ସର୍ଚ୍ଚିଂ(Searching) ହେଉଛି ଦିଆଯାଇଥିବା ଏକ ଆରେ(Array) ବା ସାରଣୀ ମଧ୍ୟରେ ପ୍ରଦତ୍ତ ତଥ୍ୟର ଅବସ୍ଥିତି ଅନ୍ୱେଷଣର ଉପାୟ। ସର୍ଚ୍ଚିଂ ସଫଳ ହୁଏ ଯଦି ଦରକାରୀ ତଥ୍ୟଟି ଆରେ ମଧ୍ୟରୁ ମିଳିଯାଏ। ଅର୍ଥାତ, ତଥ୍ୟଟି ଆରେ (Array) ରେ ମହଜୁଦ୍ ଅଛି। ଅନ୍ୟଥା ସର୍ଚ୍ଚିଂ ଅସଫଳ ହୁଏ।

ସର୍ଚ୍ଚିଂ କାର୍ଯ୍ୟ ସମ୍ପାଦନପାଇଁ ଦୁଇଟି ଉପାୟ ଅଛି

- ଲିନିୟର ସର୍ଚ୍ଚ (Linear Search)
- ବାଇନାରି ସର୍ଚ୍ଚ (Binary Search)

ଉପରୋକ୍ତ କୌଣସି ଏକ ଆଲଗୋରିଦମ୍‌ର ଚୟନ, ଆରେ ଭିତରର ଉପାଦାନ ବିନ୍ୟାସ ଉପରେ ନିର୍ଭର କରେ। ଯଦି ଉପାଦାନଗୁଡିକ ଅନିୟମିତ କ୍ରମରେ ଥାଏ, ତେବେ ଲିନିୟର ସର୍ଚ୍ଚିଂ କୌଶଳ ବ୍ୟବହାର କରିବାକୁ ପଡ଼ିବ। କିନ୍ତୁ ଯଦି ଆରେର ଉପାଦାନଗୁଡିକ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟକ୍ରମରେ ସଜାହୋଇଥାଏ, ତେବେ ବାଇନାରି ସର୍ଚ୍ଚିଂ କୌଶଳ ବ୍ୟବହାର କରିବା ବାଞ୍ଛନୀୟ। ଏହି ଦୁଇଟି ସର୍ଚ୍ଚିଂ କୌଶଳ ପରବର୍ତ୍ତୀ ଅନୁଚ୍ଛେଦରେ ବର୍ଣ୍ଣନା କରାଯାଇଛି।

5.1.1 ଲିନିୟର ସର୍ଚ୍ଚ (Linear Search)

ଆରେ a ର ଉପାଦାନଗୁଡ଼ିକର ସର୍ଜନା ବିଷୟରେ କୌଣସି ସୂଚନା ନ ଥିଲେ ଦିଆଯାଇଥିବା ଆଇଟମ୍ (item) କୁ ସର୍ଚ୍ଚ କରିବାର ଏକମାତ୍ର ଉପାୟ ହେଉଛି ଆଇଟମକୁ a ର ଗୋଟିଏ ପରେ ଗୋଟିଏ ପ୍ରତ୍ୟେକ ଉପାଦାନ ସହିତ ତୁଳନା କରିଚାଲିବା। ଏହି ପଦ୍ଧତି, ଯାହା ଆଇଟମର ଅବସ୍ଥିତି ନିର୍ଣ୍ଣୟପାଇଁ ଏକ କ୍ରମାନ୍ୱୟରେ ଗତି କରେ, ଏହାକୁ ଲିନିୟର ସର୍ଚ୍ଚ ବା ସିକ୍ୱେନସିୟାଲ ସର୍ଚ୍ଚ କୁହାଯାଏ।

ତାଲିକା 5.1

```
/*
 *   Program to demonstrate the use of linear search technique
 *   to search a given element in an un-sorted array
 */
 #include <stdio.h>
int main()
 {
int a[50], i, n, item, flag;
printf("\nEnter size of array n(<=50) : " );
scanf("%d", &n );
printf("\nEnter %d elements of array\n", n);
for ( i = 0; i < n; i++ )
scanf( "%d", &a[i] );
printf( "\nEnter element to search : " );
scanf( "%d", &item );
flag = 0;
for ( i = 0; i < n; i++ )
    {
if( item == a[i] )      /* match found  */
       {
flag = 1;
break;
       }
    }
if ( flag == 1 )
printf( "\nElement found at index: %d\n", i );
else
printf( "\nElement not found...\n" );
return 0;
 }
```

ଟେଷ୍ଟ ରନ୍ 1

```
Enter size of array n(<=50) : 10
Enter 10 elements of array
12 5 18 9 11 10 25 36 22 100
Enter element to search : 11
Element found at index: 4
```

ଟେଷ୍ଟ ରନ୍ 2

```
Enter size of array n(<=50) : 10
Enter 10 elements of array
12 5 18 9 11 10 25 36 22 100
Enter element to search : 55
Element not found...
```

ଜଟିଳତା ବିଶ୍ଳେଷଣ (Complexity Analysis)

ସର୍ବୋତ୍ତମ ପରିସ୍ଥିତିରେ, ଉପାଦାନଟି ଆରେର ପ୍ରଥମ ସ୍ଥାନରେ ଥାଇପାରେ। ଏହି କ୍ଷେତ୍ରରେ, ସର୍ଚ୍ଚିଂ କାର୍ଯ୍ୟ କେବଳ ଗୋଟିଏ ତୁଳନା ସହିତ ସଫଳତାର ସହ ସମାପ୍ତ ହୁଏ। ଯାହା ହେଉ, ସବୁଠାରୁ ଖରାପ ପରିସ୍ଥିତି ସେତେବେଳେ ହୁଏ ଯେତେବେଳେ ଉପାଦାନଟି ଆରେର ଶେଷ ସ୍ଥାନରେ ଥାଏ କିମ୍ବା ଆରେରେ ନଥାଏ। ପ୍ରଥମୋକ୍ତ ପରିସ୍ଥିତିରେ, n ଟି ତୁଳନା ସହ ସର୍ଚ୍ଚିଂ ସଫଳତାର ସହ ସମାପ୍ତ ହୋଇଥାଏ। ଶେଷୋକ୍ତ ପରିସ୍ଥିତିରେ, n ଟି ତୁଳନା ସହିତ ସର୍ଚ୍ଚିଂ ବିଫଳତାର ସହ ସମାପ୍ତ ହୁଏ। ଏହିପରି, ଆମେ ପାଇଲୁ ଯେ ସବୁଠାରୁ ଖରାପ ପରିସ୍ଥିତିରେ ଲିନିୟର ସର୍ଚ୍ଚ O(n) ଅପରେସନ୍(operation) ଆବଶ୍ୟକ କରେ।

5.1.2 ବାଇନାରି ସର୍ଚ୍ଚ (Binary Search)

ଧରାଯାଉ ଆରେ a ର ଉପାଦାନଗୁଡିକ ଆରୋହଣ କ୍ରମରେ (ଯଦି ଉପାଦାନଗୁଡିକ ଗଣନ ସଂଖ୍ୟା) କିମ୍ବା ଅଭିଧାନ କ୍ରମ (ଯଦି ଉପାଦାନଗୁଡିକ ଷ୍ଟ୍ରିଙ୍ଗ୍) ସଜାଯାଇଛି। ଦିଆଯାଇଥିବା ଉପାଦାନର ଅବସ୍ଥିତି ଅନ୍ୱେଷଣପାଇଁ ବାଇନାରି ସର୍ଚ୍ଚ ନାମକ ସର୍ବୋତ୍ତମ ସର୍ଚ୍ଚିଂ ଆଲଗୋରିଦମ ବ୍ୟବହୃତ ହୁଏ।

ଆମେ ଆମର ଦୈନନ୍ଦିନ ଜୀବନରେ ଏହି ଉପାୟର ବ୍ୟବହାର କରୁ। ଉଦାହରଣ ସ୍ୱରୂପ, ଧରାଯାଉ ଆମେ କୌଣସି ଶବ୍ଦର ଅର୍ଥ ଏକ କମ୍ପ୍ୟୁଟର ଅଭିଧାନରେ ଅନ୍ୱେଷଣ କରିବାକୁ ଚାହୁଁଛୁ। ନିଶ୍ଚିତ ଭାବେ, ଆମେ ପୃଷ୍ଠା ପରେ ପୃଷ୍ଠା ସର୍ଚ୍ଚ କରୁନାହୁଁ। ପ୍ରଥମେ ଆମେ ଅଭିଧାନ ମଝିରୁ (ପ୍ରାୟ) ଖୋଲିଥାଉ, ତା'ପରେ କେଉଁ ଅର୍ଦ୍ଧେକରେ ଶବ୍ଦଟି ଅଛି ତାହା ମଝି ପୃଷ୍ଠାର ଶବ୍ଦଗୁଡ଼ିକୁ ଦେଖି କଳନା କରୁ ଏବଂ ପରବର୍ତ୍ତୀ ସର୍ଚ୍ଚପାଇଁ ଗୋଟିଏ ଅଧା ପରିତ୍ୟାଗ କରି ଅନ୍ୟ ଅଧାରେ ସର୍ଚ୍ଚ କରୁ। ବାରମ୍ବାର ଏହି ପ୍ରକ୍ରିୟା ଅବ୍ୟାହତ ରଖୁ ଯେପର୍ଯ୍ୟନ୍ତ ଆମେ ଆବଶ୍ୟକ ଶବ୍ଦଟି ପାଇନାହୁଁ। ସେହି ଶବ୍ଦଟି ଅଭିଧାନରେ ନାହିଁ ବୋଲି ଜଣାପଡ଼ିଯିବ ଯେତେବେଳେ ଶେଷରେ କେବଳ ଗୋଟିଏ ମାତ୍ର ପୃଷ୍ଠା ରହିଯିବ।

ଉଦାହରଣ 5.1: ବାଇନାରି ସର୍ଚ୍ଚ କୌଶଳର କାର୍ଯ୍ୟ ବୁଝିବାପାଇଁ ନିମ୍ନ ଦତ୍ତ 7 ଟି ଉପାଦାନ ବିଶିଷ୍ଟ ଏବଂ ଆରୋହଣକ୍ରମେ ସଜ୍ଜିତ ଆରେ a କୁ ବିଚାରକୁ ନିଅନ୍ତୁ।

3, 10, 15, 20, 35, 40, 60

ଆଉ ଆମେ ଉପାଦାନ 15 କୁ ସର୍ଚ୍ଚ କରିବାକୁ ଚାହୁଁଛୁ।

ସମାଧାନ:

ଦିଆଯାଇଥିବା ଆରେ a

[0]	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
3	10	15	20	35	40	60

ଆରମ୍ଭ କରିବାପାଇଁ, ଆମେ ନେବା beg = 0, end = 6, ଏବଂ ମଧ୍ୟମ ଉପାଦାନର ଅବସ୍ଥିତି ହିସାବ କରିବା ଯେମିତିକି

$$mid = (beg + end) / 2 = (0 + 6) / 2 = 3$$

ବର୍ତ୍ତମାନ a[mid], ଅର୍ଥାତ $a[3] \neq 15$, ଏବଂ $beg <= end$ ତେଣୁ ଆମେ ପରବର୍ତ୍ତୀ ଆଇଟରେସନ ଆରମ୍ଭ କରିବା।

ଯେହେତୁ a[mid] = 20 > 15, ତେଣୁ, ଆମେ ନେବା $end = mid - 1 = 3 - 1 = 2$, ଯେତେବେଳେକି beg ଅପରିବର୍ତ୍ତିତ ରହିଛି। ଏବେ ନିମ୍ନମତେ mid ଉପାଦାନର ଅବସ୍ଥିତି ବାହାର କରିବା

$$mid = (beg + end) / 2 = (0 + 2) / 2 = 1$$

ଏବେ $a[mid]$, ଅର୍ଥାତ $a[1] \neq 15$, ଏବଂ $beg <= end$ ତେଣୁ ପୁଣି ଆମେ ପରବର୍ତ୍ତୀ ଆଇଟରେସନ ଆରମ୍ଭ କରିବା ।

ଯେହେତୁ $a[mid] = 10 < 15$, ତେଣୁ, ଆମେ ନେବା $beg = mid + 1 = 1 + 1 = 2$, ଯେତେବେଳେକି end ଅପରିବର୍ତ୍ତିତ ରହିଛି ।

ବର୍ତ୍ତମାନ ଯେହେତୁ beg = end, ଏବେ ମଧ୍ୟମ ଉପାଦାନର ଅବସ୍ଥିତି ଗଣନା କରିବା

$$mid = (beg + end) / 2 = (2 + 2) / 2 = 2$$

ଯେହେତୁ $a[mid]$, *i.e.*, $a[2] = 15$, ଇପ୍ସିତ ଉପାଦାନଟି ମିଳିଗଲା, ତେଣୁ ସର୍ଚ୍ଚ ସଫଳତାର ସହ ସମାପ୍ତ ହେଲା ଆଉ ଉପାଦାନଟି ଆରେ ଇଣ୍ଡେକ୍ସ (index) 2 ରେ ମିଳିଲା ।

ତାଲିକା 5.2

```
/*
/*
 *   Program to demonstrate the use of binary search technique
 *   to search a given element in a sorted array
 */
 #include <stdio.h>
int main()
 {
int a[50], i, n, item, beg, end, mid, flag;
printf("\nEnter size of array n(<=50) : " );
scanf("%d", &n );
printf("\nEnter %d elements of array in ascending order\n", n);
for ( i = 0; i < n; i++ )
scanf( "%d", &a[i] );
printf( "\nEnter element to search : " );
scanf( "%d", &item );
flag = 0;
beg = 0;
end = n - 1;
while ( beg < end )
    {
mid = ( beg + end ) / 2;
if( item == a[mid] ) {     /* match found  */
flag = 1;
break;
       }
if( item < a[mid] )
end = mid - 1;
else
beg = mid + 1;
    }
if ( flag == 1 )
```

```
printf( "\nElement found at index: %d\n", mid );
else
printf( "\nElement not found...\n" );
return 0;
 }
```

ଟେଷ୍ଟ ରନ୍ 1

```
Enter size of array n(<=50) : 10↵
Enter 10 elements of array in ascending order
5 9 10 11 12 18 25 36 44 100
Enter element to search : 44
Element found at index: 8
```

ଟେଷ୍ଟ ରନ୍ 2

```
Enter size of array n(<=50) : 10
Enter 10 elements of array in ascending order
5 9 10 11 12 18 25 36 44 100
Enter element to search : 35
Element not found...
```

ଜଟିଳତା ବିଶ୍ଳେଷଣ

ପ୍ରତ୍ୟେକ ଆଇଟରେସନରେ, ସର୍ଚ୍ଚ ଟି ଆରେର ଅର୍ଦ୍ଧେକ ଭାଗକୁ ହ୍ରାସ ପାଇଥାଏ । ତେଣୁ, n ଉପାଦାନ ବିଶିଷ୍ଟ ଆରେପାଇଁ log2n ଆଇଟରେସନ ଦରକାର ହେବ। ଏଣୁ, ବାଇନାରି ସର୍ଚ୍ଚର ଜଟିଳତା ହେଉଛି O(log2n) । ଉପାଦାନର ସ୍ଥିତି ନିର୍ବିଶେଷରେ ଏପରିକି ଉପାଦାନଟି ଆରେରେ ଉପସ୍ଥିତ ନ ଥିଲେ ମଧ୍ୟ ଏହି ଜଟିଳତା ସମାନ ହେବ ।

5.2 ସର୍ଟିଂ (Sorting) ଆଲଗୋରିଦମ୍

ସର୍ଟିଂ (Sorting) ହେଉଛି ଉପାଦାନଗୁଡ଼ିକୁ କୌଣସି ଏକ ତାର୍କିକକ୍ରମରେ ସଜାଇବାର ପ୍ରକ୍ରିୟା । ଉପାଦାନଗୁଡ଼ିକ ସଂଖ୍ୟାତ୍ମକ ମାନ ହୋଇଥିଲେ ତାର୍କିକକ୍ରମଟି ଆରୋହଣ କିମ୍ବା ଅବରୋହଣକ୍ରମ ହୋଇଥାଏ କିନ୍ତୁ ଉଭୟ ବର୍ଣ୍ଣମାଳା ଓ ସଂଖ୍ୟାର ପ୍ରତୀକ ଥାଇ ଉପାଦାନଗୁଡ଼ିକ ଥିଲେ ଏହା ଅଭିଧାନ କ୍ରମରେ କରାଯାଇଥାଏ ।

ଆମେ ପ୍ରାୟତଃ, ଆମର ନିତିଦିନିଆ କାର୍ଯ୍ୟକଳାପରେ ଏହି ପ୍ରକ୍ରିୟା ପ୍ରତିଦିନ ବ୍ୟବହାର କରୁ । ଉଦାହରଣ ସ୍ୱରୂପ, ଆମେ ଆମର ଶ୍ରେଣୀନୋଟଗୁଡ଼ିକୁ ତାରିଖ କ୍ରମରେ ସଜାଇ ରଖୁ ଯଦ୍ୱାରା ପରେ ଦରକାର ବେଳେ ଶୀଘ୍ର ପାଇପାରିବା । ଏପରି ଅନେକ ପରିସ୍ଥିତିର ଉଦାହରଣ ତୁମେମାନେ ମଧ୍ୟ ଚିନ୍ତା କରିପାରିବ ।

ଏହି ଅନୁଚ୍ଛେଦରେ, ଆମେ ନିମ୍ନଲିଖିତ ସର୍ଟିଂ (Sorting) ଆଲଗୋରିଦମଗୁଡିକ ବିଷୟରେ ଆଲୋଚନା କରିବା:

- ବବଲ୍ ସର୍ଟ (Bubble sort)
- ଇନ୍ସର୍ସନ ସର୍ଟ (Insertion sort)
- ସିଲେକସନ ସର୍ଟ (Selection sort)

ଏହି ସମସ୍ତ ଆଲଗୋରିଦମ ଗୁଡ଼ିକର ଆଲୋଚନାପାଇଁ ଆମେ n ଉପାଦାନ ବିଶିଷ୍ଟ ଏକ ଆରେ ନେବା ଯାହାର ଉପାଦାନଗୁଡ଼ିକ int ପ୍ରକାରର ଅଟେ ।

5.2.1 ବବଲ୍ ସର୍ଟ (Bubble sort)

ବବଲ୍ ସର୍ଟ (sort) ପଦ୍ଧତିରେ ଏକ ଆରେ (Array) କୁ ସର୍ଟ (sort) କରିବାପାଇଁ (n-1) ଟି ପାସ୍ (pass) ଆବଶ୍ୟକ ହୋଇଥାଏ । ପ୍ରତ୍ୟେକ ପାସ୍ (pass) ରେ, ପ୍ରତ୍ୟେକ ଉପାଦାନ a[i] କୁ a[i + 1] ସହିତ ତୁଳନା କରାଯାଏ, i = 0 ଠାରୁ (n–k) ପର୍ଯ୍ୟନ୍ତ, ଯେଉଁଠାରେ k ହେଉଛି ପାସ୍ ସଂଖ୍ୟା, ଏବଂ ଯଦି ଉପାଦାନଗୁଡ଼ିକ କ୍ରମାନୁସାରେ ନାହିଁ, ଅର୍ଥାତ ଯଦି a[i] > a[i + 1], ତେବେ ସେ ଦୁଇ ଉପାଦାନକୁ ଅଦଳବଦଳ କରିଦିଅ । ଏହାଦ୍ୱାରା ସର୍ବବୃହତ ଉପାଦାନଟି ଶେଷ ଆଡ଼କୁ ଘୁଞ୍ଚି ଘୁଞ୍ଚି ଯିବ କିମ୍ବା ବବଲ୍ ଅପ୍ କରିବ ।

ପ୍ରଥମ ପାସ୍ ଶେଷ ହେବା ପରେ, ଆରେରେ ଥିବା ସର୍ବବୃହତ୍ତମ ଉପାଦାନଟି (n-1)ତମ ସ୍ଥିତିରେ ଥିବ ଏବଂ ପ୍ରତ୍ୟେକ କ୍ରମାଗତ ପାସ୍‌ରେ, ପରବର୍ତ୍ତୀ ବୃହତ୍ତମ ଉପାଦାନଟି ଯଥାକ୍ରମେ (n-2), (n-3)), ..., 1 ସ୍ଥିତିରେ ରହିବ ।

ଅଧିକ ସ୍ୱଷ୍ଟତାପାଇଁ, ନିମ୍ନଲିଖିତ ପଦକ୍ଷେପଗୁଡ଼ିକୁ ଯତ୍ନର ସହ ଯାଞ୍ଚ କର।

ପାସ୍ (Pass) 1:

ସୋପାନ 1 — ଯଦି a[0] > a[1] ତେବେ a[0] ଏବଂ a[1] କୁ ଅଦଳବଦଳ କର ।

ସୋପାନ 2 — ଯଦି a[1] > a[2] ତେବେ a[1] ଏବଂ a[2] କୁ ଅଦଳବଦଳ କର ।

⋮

ସୋପାନ n–1 — ଯଦି a[n-–2] > a[n-1] ତେବେ a[n-2] ଏବଂ a[n-1] କୁ ଅଦଳବଦଳ କର ।

ପାସ୍ (Pass) 2:

ସୋପାନ 1 — ଯଦି a[0] > a[1] ତେବେ a[0] ଏବଂ a[1] ଅଦଳବଦଳ କର ।

ସୋପାନ 2 — ଯଦି a[1] > a[2] ତେବେ a[1] ଏବଂ a[2] ଅଦଳବଦଳ କର ।

⋮

ସୋପାନ n-2 — ଯଦି a[n-3] > a[n-2] ତେବେ a[n-3] ଏବଂ a[n-2] କୁ ଅଦଳବଦଳ କର ।

ପାସ୍ (Pass) k:

ସୋପାନ 1 — ଯଦି a[0] > a[1] ତେବେ a[0] ଏବଂ a[1] କୁ ଅଦଳବଦଳ କର

ସୋପାନ 2 — ଯଦି a[1] > a[2] ତେବେ a[1] ଏବଂ a[2] କୁ ଅଦଳବଦଳ କର ।

⋮

ସୋପାନ n-k — ଯଦି a[n-k+1] > a[n-k] ତେବେ a[n-k+1] ଏବଂ a[n-k] କୁ ଅଦଳବଦଳ କର ।

ପାସ୍ (Pass) (n-1):

ସୋପାନ 1 — ଯଦି a[0] > a[1] ତେବେ a[0] ଏବଂ a[1] କୁ ଅଦଳବଦଳ କର ।

(n -1) ପାସ୍ ସରିବା ପରେ, ଆରେର ଉପାଦାନଗୁଡିକ ଆରୋହଣ କ୍ରମରେ ସଜେଇହୋଇଯିବ ।

ଉଦାହରଣ 5.2: ବବଲ୍ ସର୍ଟ (Bubble sort) ପଦ୍ଧତିର କାର୍ଯ୍ୟ ବର୍ଣ୍ଣନା କରିବାକୁ, ନିମ୍ନ ପ୍ରଦତ୍ତ ଆରେର ଉପାଦାନଗୁଡିକୁ ଆରୋହଣ କ୍ରମରେ ସଜାଇବାକୁ ବିଚାର କର ।

12 40 3 2 15

ସମାଧାନ: ମନେପକାଅ ଯେ, ଦିଆଯାଇଥିବା ଏକ ପାସ୍‌ର ଆଉଟପୁଟ୍ (output) ପରବର୍ତ୍ତୀ ପାସ୍‌ପାଇଁ ଇନପୁଟ୍ (input) ହୋଇଥାଏ । ସର୍ଟିଂ (sorting) ପ୍ରକ୍ରିୟା ସ୍ୱୟଂ ବ୍ୟାଖ୍ୟାତ୍ମକ ଅଟେ । ପ୍ରତ୍ୟେକ ପାସ୍‌ରେ, ଗୋଟିଏ ଉପାଦାନ ଅନ୍ତିମ ସ୍ଥିତିରେ ପହଞ୍ଚିଥାଏ, ଏହା ନିମ୍ନ ଚିତ୍ରରେ ଛାୟାଙ୍କିତ ଭାବେ ଦର୍ଶାଯାଇଛି ।

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]
12	40	3	2	15

Pass-1:

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]
(12)	(40)	3	2	15

(a) a[0] ଏବଂ a[1] ମଧ୍ୟରେ ତୁଳନା କର। ଯେହେତୁ a[0] < a[1] (12 < 40), କୌଣସି ପରିବର୍ତ୍ତନ ହୁଏ ନାହିଁ।

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]
12	(40)	(3)	2	15

(b) a[1] ଏବଂ a[2] ମଧ୍ୟରେ ତୁଳନା କର । ଯେହେତୁ a[1] > a[2] (40 > 3), ଏହି ଉପାଦାନଗୁଡିକ ଅଦଳବଦଳ କର ।

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]
12	3	(40)	(2)	15

(c) a[2] ଏବଂ a[3] ମଧ୍ୟରେ ତୁଳନା କର । ଯେହେତୁ a[2] > a[3] (40 > 2), ଏହି ଉପାଦାନଗୁଡିକ ଅଦଳବଦଳ କର ।

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]
12	3	2	(40)	(15)

(d) a[3] ଏବଂ a[4] ମଧ୍ୟରେ ତୁଳନା କର। ଯେହେତୁ a[3] > a[4] (40 > 15), ଏହି ଉପାଦାନଗୁଡିକ ଅଦଳବଦଳ କର।

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]
12	3	2	15	40

ଏହିପରି, ପ୍ରଥମ ପାସ୍ ପରେ, ସର୍ବ ବୃହତ ଉପାଦାନ, 40, ଅନ୍ତିମ ସ୍ଥିତିକୁ ଚାଲିଯାଇଛି । ଅବଶିଷ୍ଟ ସଂଖ୍ୟାଗୁଡିକ ନିଜର ପୂର୍ବସ୍ଥିତିକୁ ବଦଳାଇଥିଲେ ମଧ ଏପର୍ଯ୍ୟନ୍ତ ସଜା ହୋଇନାହି । ଛାୟାଙ୍କିତ ଅଂଶ ସଜାଯାଇଥିବା ଉପାଦାନଗୁଡ଼ି ଦର୍ଶାଉଛି। ଅବଶିଷ୍ଟ ପାସ୍‌ଗୁଡ଼ିକପାଇଁ, ଯଦି ଆବଶ୍ୟକ ହୁଏ, କେବଳ ତୁଳନା ଏବଂ ଅଦଳବଦଳ, କରାଯାଇଥାଏ ।

Pass-2:

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
(12)	(3)	2	15	40	Swap
3	(12)	(2)	15	40	Swap
3	2	(12)	(15)	40	No swap
3	2	12	15	40	

ଦ୍ୱିତୀୟ ପାସ୍ ପରେ, ଦ୍ୱିତୀୟ ବୃହତ୍ତମ ଉପାଦାନ, 15, ଏହାର ଅନ୍ତିମ ସ୍ଥିତିକୁ ଚାଲିଯାଇଛି ।

Pass-3:

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
(3)	(2)	12	15	40	Swap
2	(3)	(12)	15	40	No swap
2	3	12	15	40	

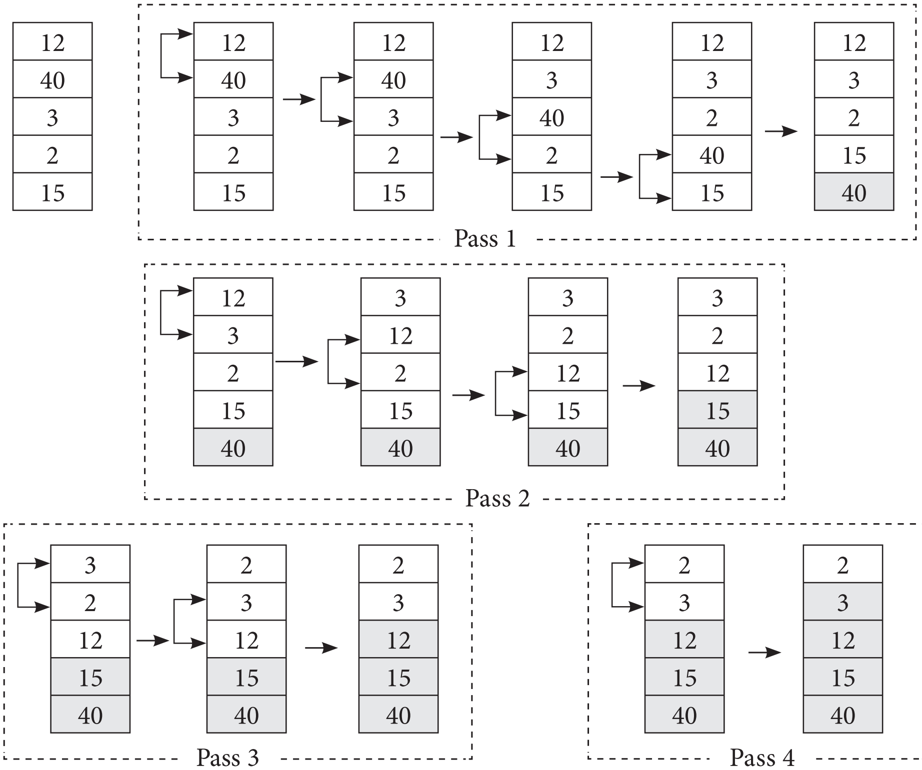
ତୃତୀୟ ପାସ୍ ପରେ, ତୃତୀୟ ବୃହତ୍ତମ ଉପାଦାନ, 12, ଏହାର ଅନ୍ତିମ ସ୍ଥିତିକୁ ଚାଲିଯାଇଛି ।

Pass-4:

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	Action
(2)	(3)	12	15	40	No swap
(2)	3	12	15	40	

ଚତୁର୍ଥ ପାସ୍ ପରେ, ଚତୁର୍ଥ ବୃହତ୍ତମ ଉପାଦାନ, 3, ଏହାର ଅନ୍ତିମ ସ୍ଥିତିକୁ ଚାଲିଯାଇଛି । ଏହିପରି, ଆରେଟି 4ଟି ପାସ୍‌ପରେ ସମ୍ପୂର୍ଣ୍ଣ ଭାବରେ ସଜା ହୋଇଯାଇଛି ।

ଉପରୋକ୍ତ ସର୍ଟିଂ ପ୍ରକ୍ରିୟାକୁ ନିମ୍ନଦର୍ଶିତ ଉପାୟରେ ମଧ୍ୟ ବୁଝାଯାଇପାରିବ:



ଚିତ୍ର - 5.1: ବବଲ୍ ସର୍ଟ *(Bubble sort)* ପଦ୍ଧତିର ଚିତ୍ରଣ

ଜଟିଳତା ବିଶ୍ଳେଷଣ

(n-1) ପାସ୍‌ପରେ, ଆରେଟି ଆରୋହଣକ୍ରମେ ସର୍ଟିଂ ହୋଇଥିବ ତୁମେ ଲକ୍ଷ୍ୟ କରିଥିବ ଯେ, ପ୍ରଥମ ପାସ୍ *(n-1)* ଟି ତୁଳନା ଆବଶ୍ୟକ କରେ, ଦ୍ୱିତୀୟ ପାସ୍ *(n-2)*ଟି ତୁଳନା ଆବଶ୍ୟକ କରେ, ..., *kth* ପାସ୍ *(n-k)* ଟି ଆବଶ୍ୟକ କରେ, ଏବଂ ଶେଷ ପାସ୍ କେବଳ ଗୋଟିଏ ତୁଳନା ଆବଶ୍ୟକ କରେ । ତେଣୁ, ସମୁଦାୟ ତୁଳନାଗୁଡ଼ିକ ହେଉଛି

$f(n) = (n-1) + (n-2) + (n-3) + \ldots + (n-k) + \ldots + 3 + 2 + 1 = n(n-1)/2 = O(n^2)$

ତାଲିକା 5.3

```
/*
 *   Program to demonstrate the use of Bubble sort method
 *   to sort a given array in ascending order
 */
 #include <stdio.h>
int main()
 {
int a[50], i, j, n, temp;
printf("\nEnter size of array n(<=50) : " );
scanf("%d", &n );
printf("\nEnter %d elements of array\n", n);
for ( i = 0; i < n; i++ )
scanf( "%d", &a[i] );
for ( i = 0; i < n-1; i++ )
    {
for ( j = 0; j < n-i-1; j++ )
       {
if ( a[j] > a[j+1] ) {
temp = a[j];
a[j] = a[j+1];
a[j+1] = temp;
          }
       }
    }
printf( "\nArray after sorting...\n\n" );
for ( i = 0; i < n; i++ )
printf( "%d ", a[i] );
printf( "\n" );
return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter size of array n(<=50) : 10
Enter 10 elements of array
5 12 10 15 22 18 2 -10 7 16
Array after sorting...
-10 2 5 7 10 12 15 16 18 22
```

5.2.2 ସିଲେକ୍ସନ ସର୍ଟ (Selection sort)

ଏକ ଆରେକୁ ସର୍ଟ(Sort) କରିବାପାଇଁ ସିଲେକ୍ସନ ସର୍ଟ ପଦ୍ଧତି ମଧ୍ୟ *(n-1)* ପାସ୍ ଆବଶ୍ୟକ କରେ । ପ୍ରଥମ ପାସ୍‌ରେ, *a[0], a[1], a[2], ……, a[n-1]* ଉପାଦାନଗୁଡ଼ିକ ମଧ୍ୟରୁ କ୍ଷୁଦ୍ରତମ ଉପାଦାନଟିକୁ ଖୋଜି ପ୍ରଥମ ଉପାଦାନ ଅର୍ଥାତ, *a[0]* ସହିତ ଅଦଳବଦଳ କର । ଦ୍ୱିତୀୟ ପାସ୍ ରେ, *a[1], a[2], a[3], ……, a[n-1]* ଉପାଦାନଗୁଡ଼ିକରୁ ଛୋଟ ଉପାଦାନ ଖୋଜି *a[1]* ସହ ଅଦଳବଦଳ କର । ଏବଂ ଏହିପରି କରିଚାଲ ।

ଅଧିକ ସ୍ୱଷ୍ଟତାପାଇଁ, ନିମ୍ନଲିଖିତ ସୋପାନଗୁଡିକ ଯତ୍ନର ସହ ଯାଞ୍ଚ କର:

ପାସ୍ (Pass) 1:

- ଆରେର କ୍ଷୁଦ୍ରତମ ଉପାଦାନର ଅବସ୍ଥିତି *loc* ନିରୂପଣ କର, ଅର୍ଥାତ, *a[0], a[1], a[2], …, a[n-1]* ।
- *a[0] ଓ a[loc]* ଅଦଳବଦଳ କର । ଏବେ *a[0]* ସହଜରେ ସର୍ଟ ହୋଇଗଲା ।

⋮

ପାସ୍ (Pass) 2:

- ପୁଣି ଉପ-ଆରେ *a[1], a[2], …, a[n-1]* ରୁ କ୍ଷୁଦ୍ରତମ ଉପାଦାନର ଅବସ୍ଥିତି loc ନିରୂପଣ କର ।
- *a[1]* ଏବଂ *a[loc]*କୁ ଅଦଳବଦଳ କର । ଏବେ *a[0]* ଏବଂ *a[1]* ସର୍ଟ ହୋଇଯାଇଛି, ଯେହେତୁ *a[0] - a[1]* ।

⋮

ପାସ୍ (Pass) k:

- ଉପ-ଆରେ *a[k], a[k+1], a[k+2], …, a[n-1]* ରୁ କ୍ଷୁଦ୍ରତମ ଉପାଦାନର ଅବସ୍ଥିତି loc ନିରୂପଣ କର ।
- *a[k] ଓ a[loc]*କୁ ଅଦଳବଦଳ କର। ଏବେ *a[0], a[1], …, a[k]* ଉପାଦାନଗୁଡ଼ିକ ସର୍ଟ ହୋଇଯାଇଛି ।

⋮

ପାସ୍ (Pass) n-1:

- *a[n-2]* ଏବଂ *a[n-1]* ରୁ କ୍ଷୁଦ୍ରତର ଉପାଦାନର ଅବସ୍ଥିତି loc ନିରୂପଣ କର ।
- *a[n-2]* ଏବଂ *a[loc]* କୁ ଅଦଳବଦଳ କର । ଏବେ *a[0], a[1], a[2], ..., a[n-1]* ସର୍ଟ ହୋଇଯାଇଛି ।

(n-1) ପାସ୍ *(Pass)* ହେବା ପରେ, ଆରେର *(array)* ଉପାଦାନଗୁଡିକ ଆରୋହଣ କ୍ରମରେ ସଜେଇହୋଇଯିବ ।

ଉଦାହରଣ 5.3: ସିଲେକସନ ସର୍ଟ ପଦ୍ଧତିର କାର୍ଯ୍ୟକାରିତା ବର୍ଣ୍ଣନା କରିବାକୁ, ନିମ୍ନ ଆରେ a କୁ 7 ଟି ଉପାଦାନ ସହିତ ବିଚାର କର।

20, 35, 40, 100, 3, 10, 15

ସମାଧାନ: ପ୍ରତ୍ୟେକ ଆଇଟରେସନ i ରେ, ଆରେଟିର ଅଣସଜ୍ଜିତ ଅଂଶରେ କ୍ଷୁଦ୍ରତମ ଉପାଦାନର ଅବସ୍ଥିତି loc ଆମେ ଖୋଜିଥାଉ। ଯଦି $loc \neq i$ ତେବେ *i* ଏବଂ loc ସ୍ଥାନରେ ଥିବା ଉପାଦାନଗୁଡ଼ିକର ଅଦଳବଦଳ କରାହୁଏ ।

ଦିଆଯାଇଥିବା ଆରେ a

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
20	35	40	100	3	10	15

ପାସ୍ (Pass) 1:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
20	35	40	100	3	10	15

$i = 0$
$loc = 4$

ଉପାଦାନ a[0] & a[4], ଅର୍ଥାତ 20 ଏବଂ 3 ର ଅଦଳବଦଳ କର, ଯଦ୍ଵାରା ନିମ୍ନ ଆରେ ମିଳିବ ।

ପାସ୍ (Pass) 2:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	35	40	100	20	10	15

$i = 1$
$loc = 5$

ଉପାଦାନ a[1] & a[5], ଅର୍ଥାତ 35 ଏବଂ 10 ର ଅଦଳବଦଳ କର, ଯଦ୍ୱାରା ନିମ୍ନ ଆରେ ମିଳିବ ।

ପାସ୍ (Pass) 3: $i = 2$, $loc = 6$

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
3	10	40	100	20	35	15

ଉପାଦାନ a[2] & a[6], ଅର୍ଥାତ 40 ଏବଂ 15 ର ଅଦଳବଦଳ କର, ଯଦ୍ୱାରା ନିମ୍ନ ଆରେ ମିଳିବ ।

ପାସ୍ (Pass) 4: $i = 3$, $loc = 4$

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
3	10	15	100	20	35	40

ଉପାଦାନ a[3] & a[4], ଅର୍ଥାତ 100 ଏବଂ 20 ର ଅଦଳବଦଳ କର, ଯଦ୍ୱାରା ନିମ୍ନ ଆରେ ମିଳିବ ।

ପାସ୍ (Pass) 5: $i = 4$, $loc = 5$

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
3	10	15	20	100	35	40

ଉପାଦାନ a[4] & a[5], ଅର୍ଥାତ 100 ଏବଂ 35 ର ଅଦଳବଦଳ କର, ଯଦ୍ୱାରା ନିମ୍ନ ଆରେ ମିଳିବ ।

ପାସ୍ (Pass) 6: $i = 5$, $loc = 6$

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
3	10	15	20	35	100	40

ଉପାଦାନ a[5] & a[6], ଅର୍ଥାତ 100 ଏବଂ 40 ର ଅଦଳବଦଳ କର, ଯଦ୍ୱାରା ନିମ୍ନ କ୍ରମବଦ୍ଧ ଆରେ ମିଳିବ ।

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
3	10	15	20	35	40	100

ଚିତ୍ର . 5.2: ସିଲେକସନ ସର୍ଟ ପଦ୍ଧତିର ଚିତ୍ରଣ

ଆରେର ଛାୟାଙ୍କିତ ଅଂଶଗୁଡ଼ିକ ପ୍ରତ୍ୟେକ ଆଇଟରେସନ ପରେ ସେପର୍ଯ୍ୟନ୍ତ ହୋଇଥିବା କ୍ରମବଦ୍ଧିତ ଭାଗକୁ ଦର୍ଶାଉଛି । ପ୍ରତ୍ୟେକ ଆଇଟରେସନ ପରେ ପୂର୍ବ କ୍ରମବଦ୍ଧ ଅଂଶର ଦକ୍ଷିଣପାର୍ଶ୍ୱରେ ଗୋଟିଏ ଲେଖାଏଁ ଉପାଦାନ ଯୋଡି ହୋଇଥାଏ ।

k ତମ ପାସ୍‌ରେ $a[k-1], a[k+1], a[k+2], \ldots, a[n-1]$, ମଧ୍ୟରୁ କ୍ଷୁଦ୍ରତମ ଉପାଦାନର ଅବସ୍ଥିତି loc ନିରୂପଣପାଇଁ ଆମକୁ ଏକ ରୁଟିନ୍ (routine)ର ଆବଶ୍ୟକ ।

ତାଲିକା 5.4

```
/*
*   Program to demonstrate the use of selection sort method
*   to sort a given array in ascending order
*/
#include <stdio.h>
int main()
{
  int a[50], i, j, n, temp, min, loc;
  printf("\nEnter size of array n(<=50) : " );
   scanf("%d", &n );
   printf("\nEnter %d elements of array\n", n);
   for ( i = 0; i < n; i++ )
      scanf( "%d", &a[i] );
   for ( i = 0; i < n-1; i++ )
   {
      min = a[i];
      loc = i;
      for ( j = i+1; j < n; j++ )
      {
             if ( a[j] < min ) {
             min = a[j];
             loc = j;
         }
      }
      if ( loc != i ) {
         temp = a[i];
         a[i] = a[loc];
         a[loc] = temp;
      }
   }
   printf( "\nArray after sorting...\n\n" );
   for ( i = 0; i < n; i++ )
       printf( "%d ", a[i] );
   printf( "\n" );
   return 0;
}
```

ଟେଷ୍ଟ ରନ୍

```
Enter size of array n(<=50) : 6
Enter 6 elements of array
12 5 10 15 22 18
Array after sorting...
5 10 12 15 18 22
```

ଜଟିଳତା ବିଶ୍ଳେଷଣ

ତୁମେ ନିଶ୍ଚିତ ଭାବରେ ଲକ୍ଷ୍ୟ କରିଥିବ ଯେ, କ୍ଷୁଦ୍ରତମ ଉପାଦାନର ଅବସ୍ଥିତି ଅନ୍ୱେଷଣପାଇଁ ପ୍ରଥମ ପାସ୍‌ରେ *(n-1)*ଟି ତୁଳନା, ଦ୍ୱିତୀୟ ପାସ୍‌ରେ *(n-2)*ଟି ତୁଳନା, ..., *k* ତମ ପାସ୍‌ରେ *(n-k)*ଟି ତୁଳନା, ଏବଂ ଶେଷ ପାସ୍‌ରେ କେବଳ ଗୋଟିଏ ତୁଳନା ଆବଶ୍ୟକ ହେଉଛି।

ତେଣୁ ସମୁଦାୟ ତୁଳନାଗୁଡ଼ିକ ହେବ

5.2.3 ଇନ୍‌ସର୍‌ସନ ସର୍ଟ (Insertion Sort)

ଏହି ଆଲଗୋରିଦମ୍ ବ୍ରିଜ୍ ଖେଳାଳିମାନଙ୍କ ମଧ୍ୟରେ ଲୋକପ୍ରିୟ ଯେତେବେଳେ ସେମାନେ ସେମାନଙ୍କର ତାସ୍‌ମୁଠାକୁ ପ୍ରଥମେ ଥରପାଇଁ ସର୍ଟ କରନ୍ତି ବା ସଜାଇଥାନ୍ତି।

ଏହି ପଦ୍ଧତିରେ, ଆମେ ପ୍ରତ୍ୟେକ ଥର ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ମୂଲ୍ୟ ନେଇଥାଉ ଏବଂ ତା'ପରେ ଏହାକୁ ସଜାଯାଇଥିବା ଉପ-ଲିଷ୍ଟ(list) ରେ ଉପଯୁକ୍ତ ସ୍ଥାନରେ ଭର୍ତ୍ତି କରିଥାଉ, ଅର୍ଥାତ *k* ତମ ଆଇଟରେସନ ସମୟରେ ଏକ ଉପାଦାନ a[k] ସଜାଯାଇଥିବା ଉପ-ଆରେ *(Sub-Array) a[1], a[2], a[3] , ..., a[k-1]* ର ଉପଯୁକ୍ତ ସ୍ଥାନରେ ଭର୍ତ୍ତି କରାଯାଇଥାଏ।

ଏଥିପାଇଁ *a[k]* କୁ *a[k-1], a[k-2], a[k-3], ...* ଇତ୍ୟାଦି ସହିତ ଏଭଳି ଏକ *a[j]* ମିଳିଲା ପର୍ଯ୍ୟନ୍ତ ତୁଳନା କରିଚାଲିବା ଯେପରିକି $a[j] \le a[k]$ ହେଉଥିବ। ଏହାପରେ *a[k-1], a[k-2], ..., a[j + 1]* ପର୍ଯ୍ୟନ୍ତ ସମସ୍ତ ଉପାଦାନଗୁଡ଼ିକୁ ଗୋଟିଏ ଲେଖାଏଁ ସ୍ଥାନ ଦକ୍ଷିଣକୁ ଘୁଞ୍ଚାଇଦେବା, ଏବଂ ତାପରେ ଆରେର *(j + 1)* ସ୍ଥାନରେ *a[k]*ରେ ଥିବା ଉପାଦାନକୁ ଭର୍ତ୍ତି କରିଦେବା।

ଅଧିକ ସ୍ୱଷ୍ଟତାପାଇଁ, ନିମ୍ନ ପ୍ରଦତ୍ତ ସୋପାନ ଗୁଡିକ ଯତ୍ନର ସହ ଯାଞ୍ଚ କର।

ପାସ୍ 1: *a[1]*କୁ *a[0]*ର ପୂର୍ବରୁ କିମ୍ବା ପରେ ଭର୍ତ୍ତି କରାଯାଏ ଯଦ୍ୱାରା *a[0]* ଏବଂ *a[1]* କ୍ରମବଦ୍ଧ ହୁଏ।

ପାସ୍ 2: *a[2]*କୁ *a[0]*ର ପୂର୍ବରୁ କିମ୍ବା *a[0]* ଏବଂ *a[1]* ମଧ୍ୟରେ ବା *a[1]* ପରେ ଭର୍ତ୍ତି କରାଯାଏ ଯାହାଦ୍ୱାରା *a[0], a[1], a[2]* ଉପାଦାନଗୁଡ଼ିକ କ୍ରମବଦ୍ଧ ହୁଏ।

⋮

ପାସ୍ 3: *a[3]*କୁ *a[0]*ର ପୂର୍ବରୁ କିମ୍ବା *a[0]* ଏବଂ *a[1]* ମଧ୍ୟରେ କିମ୍ବା *a[1]* ଏବଂ *a[2]* ମଧ୍ୟରେ ବା *a[2]* ପରେ ଭର୍ତ୍ତି କରାଯାଏ ଯାହାଦ୍ୱାରା *a[0], a[1], a[2], a[3]* ଉପାଦାନଗୁଡ଼ିକ କ୍ରମବଦ୍ଧ ହୁଏ।

⋮

ପାସ୍ k: *a[k]*କୁ କ୍ରମବଦ୍ଧ ଉପ-ଆରେର *a[0], a[1], a[2], ..., a[k-1]* ଏପରି ଉପଯୁକ୍ତ ସ୍ଥାନରେ ଭର୍ତ୍ତି

କରାଯାଏ ଯେପରି *a[0], a[1], a[2] , ..., a[k-1], a[k]* ଉପାଦାନଗୁଡ଼ିକ କ୍ରମବଦ୍ଧ ହେବ ।

⋮

ପାସ୍ n-1: *a[n-1]*କୁ ସଜାଯାଇଥିବା ଉପ-ଆରେର *a[0], a[1], a[2], ..., a[n-2]* ଏପରି ଉପଯୁକ୍ତ ସ୍ଥାନରେ ଭର୍ତ୍ତି କରାଯାଏ, ଯାହାଦ୍ୱାରା, *a[0], a[1], a[2], . . ., a[n-1]* ଉପାଦାନଗୁଡିକ କ୍ରମବଦ୍ଧ ହେବ ।

ଏହିପରି ଭାବରେ, ଆରେକୁ *(n-1)* ପାସ୍ ପରେ, ଆରୋହଣ କ୍ରମରେ ସଜାଯାଇପାରିବ ବା ସର୍ଟ କରାଯାଇ ପାରିବ ।

ଉଦାହରଣ 5.4: ଇନ୍‌ସର୍‌ସନ ସର୍ଟ *(Insertion Sort)* ପଦ୍ଧତିର କାର୍ଯ୍ୟକୁ ବର୍ଣ୍ଣନା କରିବାକୁ, ନିମ୍ନଲିଖିତ ଆରେ a କୁ 7 ଟି ଉପାଦାନ ସହିତ ବିଚାର କର

35, 20, 40, 100, 3, 10, 15

ସମାଧାନ: ଦିଆଯାଇଥିବା ଆରେ *a*

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]	*a*[5]	*a*[6]
35	20	40	100	3	10	15

ପାସ୍ 1:

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]	*a*[5]	*a*[6]
35	20	40	100	3	10	15

ଯେହେତୁ a[1] < a[0], ତେଣୁ a[0] ପୂର୍ବରୁ a[1]କୁ ଭର୍ତ୍ତି କର ଯେପରିକି ନିମ୍ନ ପ୍ରକାର ଆରେ ମିଳିବ ।

ପାସ୍ 2:

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]	*a*[5]	*a*[6]
20	35	40	100	3	10	15

ଯେହେତୁ a[2] > a[1], ଅର୍ଥାତ a[0], a[1] ଓ a[2] ପୂର୍ବରୁ ସର୍ଟ ଅଛି, ତେଣୁ କିଛି କରିବା ନାହିଁ ।

ପାସ୍ 3:

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]	*a*[5]	*a*[6]
20	35	40	100	3	10	15

ଯେହେତୁ a[3] > a[2], ଅର୍ଥାତ ପୂର୍ବରୁ ସର୍ଟ ଅଛି ତେଣୁ କୌଣସି ପ୍ରକ୍ରିୟା ଦରକାର ନାହିଁ ।

ପାସ୍ 4:

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]	*a*[5]	*a*[6]
20	35	40	100	3	10	15

ଏବେ ଯେହେତୁ a[4] ଉପାଦାନଟି a[3], a[2], a[1] ଏବଂ a[0] ଠାରୁ କମ୍, ତେଣୁ a[4] କୁ a[0] ପୂର୍ବରୁ ଭର୍ତ୍ତି କର, ଯାହାଦ୍ୱାରା ନିମ୍ନ କ୍ରମରେ ଆରେ ମିଳିବ ।

ପାସ୍ 5:

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]	*a*[5]	*a*[6]
3	20	35	40	100	10	15

ପୁଣି ଯେହେତୁ a[5] ଉପାଦାନଟି a[4], a[3], a[2], ଏବଂ a[1] ଠାରୁ କମ୍, ତେଣୁ a[5]କୁ a[1] ପୂର୍ବରୁ ଭର୍ତ୍ତି

କର, ଯାହାଦ୍ୱାରା ନିମ୍ନ କ୍ରମରେ ଆରେ ମିଳିବ ।

ପାସ୍ 6:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	20	35	40	100	15

ଶେଷରେ ଯେହେତୁ a[6] ଉପାଦାନଟି a[5], a[4], a[3], ଏବଂ a[2] ଠାରୁ କମ୍, ତେଣୁ a[6] କୁ a[2] ପୂର୍ବରୁ ଭର୍ତ୍ତି କର, ଯାହାଦ୍ୱାରା ନିମ୍ନ ସର୍ଟେଡ ଆରେ(sorted array) ବା ପୂର୍ଣ୍ଣ ସଜ୍ଜିତ ଆରେ ମିଳିଯିବ ।

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	10	15	20	35	40	100

ଚିତ୍ର. 5.3: ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ପଦ୍ଧତିର ଚିତ୍ରଣ

ତାଲିକା 5.5

```
/*  Program to demonstrate the use of insertion sort method
 *   to sort a given array in ascending order
 */
 #include <stdio.h>
 void main()
 {
    int a[50], i, j, k, n, temp;
    printf("\nEnter size of array n(<=50) : " );
    scanf("%d", &n );
    printf("\nEnter %d elements of array\n", n);
    for ( i = 0; i < n; i++ )
       scanf( "%d", &a[i] );
    for ( k = 1; k < n; k++ ) {
       temp = a[k];
       j = k - 1;
       while ( ( temp < a[j] ) && ( j >= 0 ) ) {
           a[j+1] = a[j];
           j--;
       }
       a[j+1] = temp;
    }
    printf( "\nArray after sorting...\n\n" );
    for ( i = 0; i < n; i++ )
        printf( "%d ", a[i] );
    printf( "\n" );
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter size of array n(<=50) : 7
Enter 7 elements of array
5 12 10 15 22 18 16
Array after sorting...
5 10 12 15 16 18 22
```

5.3 ସମୀକରଣର ମୂଳ ଅନ୍ୱେଷଣ

ସାଧାରଣତଃ, ଏକ ସମୀକରଣର ମୂଳ ଅନ୍ୱେଷଣପାଇଁ ଦୁଇ ପ୍ରକାରର ପଦ୍ଧତି ଅଛି।

ପ୍ରତ୍ୟକ୍ଷ ପଦ୍ଧତି (Direct methods)

ପ୍ରତ୍ୟକ୍ଷ ବା ସିଧାସଳଖ ପଦ୍ଧତିଗୁଡ଼ିକ ଏକ ସୀମିତ ସଂଖ୍ୟକ ପଦକ୍ଷେପରେ ଏକ ସମୀକରଣର ମୂଳ ଦେଇଥାଏ। ଏହା ସହିତ, ଏହି ପଦ୍ଧତିଗୁଡ଼ିକ ସମସ୍ତ ମୂଳକୁ ଏକ ସମୟରେ ଦେବାରେ ସମର୍ଥ ହୁଏ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଦ୍ୱିଘାତ ସମୀକରଣର ମୂଳ

$ax^2 + bx + c = 0$ ଯେଉଁଠାରେ $a \neq 0$

ତେବେ ମୂଳ ଦ୍ୱୟ

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

ଆଇଟରେଟିଭ୍ ପଦ୍ଧତି (Iterative methods)

ଆଇଟରେଟିଭ୍ ପଦ୍ଧତି, ଯାହାକୁ ପରୀକ୍ଷାମୂଳକ ପଦ୍ଧତି ମଧ୍ୟ କୁହାଯାଏ, ତାହା କ୍ରମାନ୍ୱୟ ଆସନ୍ନ ରାଶି ଧାରଣା (successive approximations) ଉପରେ ଆଧାରିତ। ଏଥିରେ ମୂଳପାଇଁ ଏକ ବା ଏକାଧିକ ଆନୁମାନିକ ରାଶି ନେଇ ଆରମ୍ଭ କରାଯାଏ। ତା'ପରେ ଏହାର ଏକ ସଂଗତ ସମାଧାନ ମିଳିବା ପର୍ଯ୍ୟନ୍ତ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଅନୁକ୍ରମର ପୁନରାବୃତ୍ତି କରି ଆସନ୍ନ ରାଶିର ଏକ ଅନୁକ୍ରମ ହାସଲ କରାଯାଏ। ସାଧାରଣତଃ, ଆଇଟରେଟିଭ୍ ପଦ୍ଧତି, ଥରକେ ଗୋଟିଏ ମୂଳ ଦେଇଥାଏ।

ମାନ୍ୟୁଆଲି(manually) ବା ଖାତା-କଲମରେ ସମୀକରଣକୁ ସମାଧାନ କରିବାପାଇଁ ଆଇଟରେଟିଭ୍ ପଦ୍ଧତିଗୁଡ଼ିକ ଅତ୍ୟନ୍ତ ଅସୁବିଧାଜନକ ଏବଂ ସମୟ ସାପେକ୍ଷ। କିନ୍ତୁ ଏହା କମ୍ପ୍ୟୁଟରରେ ବ୍ୟବହାରପାଇଁ ସର୍ବୋତ୍ତମ ଉପଯୁକ୍ତ। ଏହାର କାରଣ:

1. ଆଇଟରେଟିଭ୍ ପଦ୍ଧତି ସଂକ୍ଷିପ୍ତ ଭାବେ କମ୍ପ୍ୟୁଟେସନାଲ(computational) ଆଲଗୋରିଦମ ରୂପରେ ବ୍ୟକ୍ତ କରାଯାଇପାରେ।
2. ଏପରି ଆଲଗୋରିଦମ୍ ପ୍ରସ୍ତୁତ କରିବା ସମ୍ଭବ ଯାହା ସଦୃଶ-ସମସ୍ୟାଶ୍ରେଣୀପାଇଁ ବ୍ୟବହାର ହୋଇପାରିବ। ଉଦାହରଣ ସ୍ୱରୂପ, n ଘାତର ଏକ ପଲିନୋମିଆଲ୍ ସମୀକରଣ ସମାଧାନପାଇଁ ଏକ ଆଲଗୋରିଦମ୍ ବିକଶିତ କରାଯାଇପାରିବ।
3. ଆଇଟରେଟିଭ୍ ପଦ୍ଧତିରେ, ପ୍ରତ୍ୟକ୍ଷ ପଦ୍ଧତି ତୁଳନାରେ ରାଉଣ୍ଡ-ଅଫ୍ ତ୍ରୁଟିଗୁଡିକ ନଗଣ୍ୟ।

ଏକ ସମୀକରଣର ମୂଳ ଅନ୍ୱେଷଣପାଇଁ ନିମ୍ନଲିଖିତ ପଦ୍ଧତିଗୁଡିକ ଲୋକପ୍ରିୟ ଆଇଟରେଟିଭ୍ ପଦ୍ଧତି।

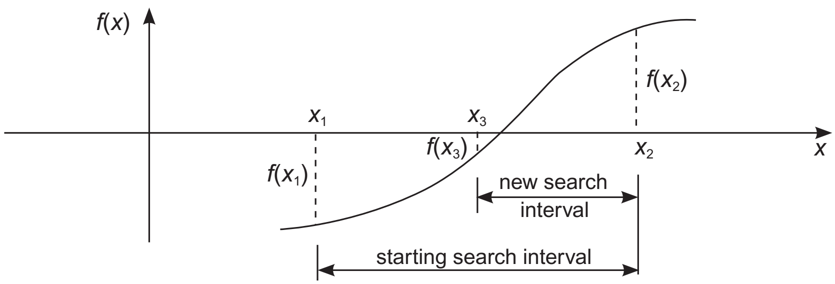
- ଦ୍ୱିଭାଜନ ପଦ୍ଧତି (Bisection Method)

- ରେଗୁଲା-ଫାଲ୍ସି ପଦ୍ଧତି (Regula-falsi Method)
- ସେକାଣ୍ଟ ପଦ୍ଧତି (Secant Method)
- ନ୍ୟୁଟନ୍-ରାଫସନ୍ ପଦ୍ଧତି (Newton-Raphson Method)
- କ୍ରମାନ୍ୱୟ ଆସନ୍ନ ରାଶି ପଦ୍ଧତି (Method of Successive approximations)

ଏହି ଅନୁଚ୍ଛେଦରେ ଆମେ ଦ୍ୱିଭାଜନ ପଦ୍ଧତି (Bisection Method) ବ୍ୟବହାର କରି ନିମ୍ନ ପ୍ରକାରର ସମୀକରଣର ମୂଳ ନିରୂପଣ ବିଷୟରେ ଜାଣିବା।

$f(x) = 0$

ଦ୍ୱିଭାଜନ ପଦ୍ଧତି ହେଉଛି ଅନ୍ୟତମ ସରଳ ଆଇଟରେସନ ପଦ୍ଧତି। ଆରମ୍ଭ କରିବାକୁ, ଏପରି ଦୁଇଟି ପ୍ରାରମ୍ଭିକ ଆସନ୍ନ ରାଶି *x1* ଓ *x2* ନିଆଯାଏ, ଯେପରିକି $f(x1)\times f(x2) < 0$, ଏବଂ ସୁନିଶ୍ଚିତ କରେ ଯେ ମୂଳ *x1* ଓ *x2* ମଧ୍ୟରେ ଅଛି। ମନେକର ପରବର୍ତ୍ତୀ *x*-ମୂଲ୍ୟ, *x3*, ଯାହାକି *[x1, x2]* ଅନ୍ତରାଳର ମଧ୍ୟ-ବିନ୍ଦୁ ଭାବରେ ଗଣନା କରାଯାଏ।



ଚିତ୍ର. 5.4: ଦ୍ୱିଭାଜନ ପଦ୍ଧତିଦ୍ୱାରା ମୂଳ ଆନୁମାନିକତା

ବର୍ତ୍ତମାନ ତିନୋଟି ସମ୍ଭାବନା ଉପୂନ୍ନ ହୋଇପାରେ:

i) ଯଦି f(x3) = 0, ତା'ହେଲେ x3 ରେ ଆମର ଏକ ମୂଳ ଅଛି

ii) ଯଦି f(x1) ଏବଂ f(x3), ଦୁଇ ବିପରୀତ ସଙ୍କେତଯୁକ୍ତ ରାଶି, ତା'ହେଲେ ମୂଳ (x1, x3) ଅନ୍ତରାଳରେ ଅଛି। ତେଣୁ x2 କୁ x3 ଦ୍ୱାରା ବଦଳାଯାଇ ଏକ ନୁଆ ଅନ୍ତରାଳର ଗଠନ ହୁଏ, ଯାହା କି ବର୍ତ୍ତମାନର ଅନ୍ତରାଳର ହେବ ଅର୍ଦ୍ଧେକ ଭାଗ। ଏବେ ନୂତନ ଅନ୍ତରାଳର ପୁନର୍ବାର ଦ୍ୱିଭାଜନ କରାଯାଏ।

iii) ଯଦି f(x1) ଏବଂ f(x3) ସମାନ ସଙ୍କେତଯୁକ୍ତ ରାଶି, ତା'ହେଲେ ମୂଳ (x3, x2) ଅନ୍ତରାଳରେ ଅଛି। ତେଣୁ x1 କୁ x3 ଦ୍ୱାରା ବଦଳାଯାଏ ଏବଂ ନୂତନ ଅନ୍ତରାଳକୁ ପୁନର୍ବାର ଦ୍ୱିଭାଜନ କରାଯାଏ।

ତଦନୁଯାୟୀ ଏହି ଅନ୍ତରାଳ ଦ୍ୱିଭାଜନ ପଦ୍ଧତିର ପୁନରାବୃତ୍ତି କରି, ପ୍ରତ୍ୟେକ ଥର ଆମେ ମୂଳକୁ ଏକ ନୂତନ ଅନ୍ତରାଳରେ ଆବଦ୍ଧ କରିଥାଉ, ଏବଂ ନୂତନ ଅନ୍ତରାଳଟି ପ୍ରତ୍ୟେକ ଆଇଟରେସନରେ ଅଧା ହୋଇଚାଲେ।

ଏହି ପୁନରାବୃତ୍ତି ଚକ୍ର ସମାପ୍ତ ହେବ ଯେତେବେଳେ ସର୍ଚ୍ଚ ଅନ୍ତରାଳଟି ନିର୍ଦ୍ଧାରିତ ସହନ ଅଶୁଦ୍ଧିଠାରୁ (ମୂଳପାଇଁ ଅନୁମୋଦିତ ଅଶୁଦ୍ଧି) ଛୋଟ ହୋଇଯିବ। ଏଣୁ, ଯଦି ଇପ୍‌ସିଲନ୍ (epsilon) ନିର୍ଦ୍ଧାରିତ ଆବଶ୍ୟକ ସହନ ଅଶୁଦ୍ଧି, ତା'ହେଲେ ପୁନରାବୃତ୍ତି ଚକ୍ର ସମାପ୍ତ ହେବ ଯେତେବେଳେ ପରମ ଅଶୁଦ୍ଧି ଇପ୍ସିଲନ୍‌ଠାରୁ (epsilon) କମ୍ କିମ୍ବା ସମାନ ହୁଏ। ଅର୍ଥାତ,

$$|x_1 - x_2| \le epsilon$$

ଆମେ ଶେଷ ସର୍ତ୍ତ ଅନ୍ତରାଳର ମଧ୍ୟ-ବିନ୍ଦୁକୁ ମୂଳପାଇଁ ଆସନ୍ନମାନ ଭାବେ ଗ୍ରହଣ କରୁ ।

ତାଲିକା 5.6

```
/*
 *   Program to implement Bisection method to find one of the
 *   root of equation x^3 - 4x - 9 = 0
 */
 #include<stdio.h>
 #include<stdlib.h>
 #include<math.h>
 double f( double x )
{
     return pow((double)x,(double)3.0)-4*x-9;
 }
int main()
 {
    float x1, x2, epsilon, x3;
    printf("Enter first point of the search interval : ");
    scanf("%f", &x1);
    printf("Enter second point of the search interval : ");
    scanf("%f", &x2);
    if ( ( f(x1) * f(x2) ) > 0 ) {
       printf("\nInitial approximations are unsuitable\n");
       return 1;
    }
    printf("Enter prescribed tolerance : ");
    scanf("%f", &epsilon);
    do {
       x3 = (x1+x2)/2;
       if ( f(x1) * f(x3) < 0 )
           x2 = x3;
       else
           x1 = x3;
    }
    while( fabs((double)(x1-x2)) > epsilon);
    printf("\nApproximate root = %8.4f\n", x3);
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍ 1

```
Enter first point of the search interval : 2.5
Enter second point of the search interval : 2.6
Initial approximations are unsuitable
```

ଟେଷ୍ଟ ରନ୍ 2

```
Enter first point of the search interval : 2.6
Enter second point of the search interval : 2.8
Enter prescribed tolerance : 0.0001
Approximate root = 2.7065
```

ୟୁନିଟ ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଯାହା ଶିଖିଛୁ

- ସର୍ଚ୍ଚିଂ ହେଉଛି ଏକ ଆରେରେ ଦିଆଯାଇଥିବା ଉପାଦାନ ଅବସ୍ଥିତିର ଅନ୍ୱେଷଣ ପ୍ରକ୍ରିୟା ।
- ବିଭିନ୍ନ ସର୍ଚ୍ଚିଂ କୌଶଳରେ ଲିନିୟର ସର୍ଚ୍ଚ ଏବଂ ବାଇନାରି ସର୍ଚ୍ଚ ଅନ୍ତର୍ଭୁକ୍ତ ।
- ଯଦି ଆରେ(Array) ସର୍ଟେଡ ନଥାଏ, ତା'ହେଲେ କୌଣସି ଏକ ଉପାଦାନ ସର୍ଚ୍ଚ କରିବାର ଏକମାତ୍ର ବିକଳ୍ପ ହେଉଛି ଲିନିୟର ସର୍ଚ୍ଚ । ଯଦି ଆରେ (Array) ସର୍ଟେଡ ଥାଏ, ତା'ହେଲେ ବାଇନାରି ସର୍ଚ୍ଚ ଏକ ଉତ୍ତମ ବିକଳ୍ପ ।
- ସର୍ଟିଂ(Sorting) ହେଉଛି କୌଣସି ଏକ ତାର୍କିକ କ୍ରମରେ ଉପାଦାନଗୁଡିକୁ ସଜାଇବାର ପ୍ରକ୍ରିୟା ।
- ବିଭିନ୍ନ ସର୍ଟିଂ(Sorting) କୌଶଳ ମଧ୍ୟରେ ବବଲ୍ ସର୍ଟ(bubble sort), ସିଲେକସନ୍ (selection), ଏବଂ ଇନ୍‌ସର୍‌ସନ ସର୍ଟ (insertion sort) ଅନ୍ତର୍ଭୁକ୍ତ ।
- ଅନ୍ୟ ପଦ୍ଧତିଗୁଡ଼ିକ ମଧ୍ୟରେ $f(x) = 0$ ଭଳି ପଲିନୋମିଆଲ୍(polynomial) ସମୀକରଣର ମୂଳ ଅନ୍ୱେଷଣପାଇଁ ଦ୍ୱିଭାଜନ ପଦ୍ଧତି ହେଉଛି ଏକ ସରଳ ପଦ୍ଧତି ।

ଅଭ୍ୟାସ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. ସର୍ଚ୍ଚିଂ କ'ଣ ?
2. ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ଲିନିୟର ସର୍ଚ୍ଚ ପଦ୍ଧତିର ବର୍ଣ୍ଣନା କର ।
3. ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ବାଇନାରି ସର୍ଚ୍ଚ ପଦ୍ଧତିର ବର୍ଣ୍ଣନା କର ।
4. ସର୍ଟିଂ କ'ଣ ? ସର୍ଟିଂର ଆବଶ୍ୟକତା କ'ଣ ?
5. ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ବବଲ୍ ସର୍ଟ (bubble sort) ପଦ୍ଧତିର ବର୍ଣ୍ଣନା କର ।
6. ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ସିଲେକସନ ସର୍ଟ (selection sort) ପଦ୍ଧତିର ବର୍ଣ୍ଣନା କର।
7. ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ଇନ୍‌ସର୍‌ସନ ସର୍ଟ (insertion sort) ପଦ୍ଧତିର ବର୍ଣ୍ଣନା କର।
8. ଏକ ସମୀକରଣର ମୂଳ ଅନ୍ୱେଷଣପାଇଁ ଦ୍ୱିଭାଜନ ପଦ୍ଧତିର ବର୍ଣ୍ଣନା କର ।
9. ଏକ ଆରେରେ ଦିଆଯାଇଥିବା ଉପାଦାନଗୁଡିକ ହେଲା 12, 7, 13, 9, 10, 77, 2 , 8 । ବବଲ୍ ସର୍ଟ ପଦ୍ଧତିର ପ୍ରଥମ ପାସ୍ ପରେ ଉପାଦାନଗୁଡ଼ିକର ଅବସ୍ଥିତି କ'ଣ ହେବ ?

10. ଏକ ଆରେର ଦିଆଯାଇଥିବା ଉପାଦାନଗୁଡିକ ହେଲା 11, 70, 13, 99, 5, 17, 21, 38 । ସିଲେକସନ ସର୍ଟ ପଦ୍ଧତିର ତିନୋଟି ପାସ୍ ପରେ ଉପାଦାନଗୁଡ଼ିକର ଅବସ୍ଥିତି କ'ଣ ହେବ?
11. ଏକ ଆରେର ଦିଆଯାଇଥିବା ଉପାଦାନଗୁଡିକ ହେଲା 12, 7, 11, 92, 13, 71, 21, 38 । ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ପଦ୍ଧତିର ଚାରୋଟି ପାସ୍ ପରେ ଉପାଦାନଗୁଡ଼ିକର ଅବସ୍ଥିତି କ'ଣ ହେବ?
12. ଏକ ସଜାଯାଇନଥିବା ଆରେର ଉପାଦାନଗୁଡିକ 12, 7, 11, 92, 13, 71, 21, 38 ଭାବରେ ଦିଆଯାଇଅଛି । ସେହି ଉପାଦାନଗୁଡିକରୁ 71 ଏବଂ 100 ସର୍ଚ୍ଚ କରିବାପାଇଁ ପଦକ୍ଷେପଗୁଡିକ ବର୍ଣ୍ଣନା କର।
13. ଏକ ଅଣ-ସଜ୍ଜିତ ଆରେର ଉପାଦାନଗୁଡିକ 82, 70, 61, 52, 43, 31, 24, 18 ଭାବରେ ଦିଆଯାଇଅଛି । ସେହି ଉପାଦାନଗୁଡିକରୁ 24 ଏବଂ 99 ସର୍ଚ୍ଚ କରିବାପାଇଁ ପଦକ୍ଷେପ ଗୁଡିକ ବର୍ଣ୍ଣନା କର।

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. ଏକ ଆରେ ମଧ୍ୟରୁ କୌଣସି ଉପାଦାନ ଅନ୍ୱେଷଣର ଲିନିୟର ସର୍ଚ୍ଚପାଇଁ ସବୁଠାରୁ ଖରାପ-ସ୍ଥିତି କ'ଣ ?

   (a) ସ୍ଥିର ସମୟ (Constant time) (b) ଲଗାରିଦମିକ୍ ସମୟ (Logarithmic time)

   (c) ଲିନିୟର ସମୟ (Linear time) (d) ଦ୍ୱିଘାତ ସମୟ (Quadratic time)

2. ଏକ ଆରେ ମଧ୍ୟରୁ କୌଣସି ଉପାଦାନ ଅନ୍ୱେଷଣର ବାଇନାରି ସର୍ଚ୍ଚପାଇଁ ସବୁଠାରୁ ଖରାପ-ସ୍ଥିତି କ'ଣ?

   (a) ସ୍ଥିର ସମୟ (Constant time) (b) ଲଗାରିଦମିକ୍ ସମୟ (Logarithmic time)

   (c) ଲିନିୟର ସମୟ (Linear time) (d) ଦ୍ୱିଘାତ ସମୟ (Quadratic time)

3. ଏକ ଆରେପାଇଁ କେଉଁ ଅତିରିକ୍ତ ଆବଶ୍ୟକତା ଅଛି, ଯାହା ଫଳରେ ଏକ ପ୍ରବିଷ୍ଟ ଉପାଦାନ ଅନ୍ୱେଷଣରେ ବାଇନାରି ସର୍ଚ୍ଚ ଉପଯୋଗ କରାଯାଇପାରେ?

   (a) ଆରେ ଉପାଦାନଗୁଡିକ ନିଶ୍ଚିତ ଭାବେ ଏକ ସ୍ତୁପ(heap) ଗଠନ କରୁଥିବେ,

   (b) ଆରେରେ ଅତି କମରେ 2ଟି ଉପାଦାନ ଥିବା ଆବଶ୍ୟକ,

   (c) ଆରେଟି ନିଶ୍ଚିତ ଭାବେ କ୍ରମବଦ୍ଧ ଥିବ,

   (d) ଆରେର ଆକାର ନିଶ୍ଚିତ ଭାବରେ 2ର ଘାତ ହେଉଥିବା।

4. ଏକ ଅଣ-କ୍ରମବଦ୍ଧ ଆରେ ବ୍ୟବହାର କରି ଏକ ଉପାଦାନ ଅନ୍ୱେଷଣର ସର୍ବୋତ୍ତମ ଉପାୟ ହେଉଛି ____

   (a) ଲିନିୟର ସର୍ଚ୍ଚ (Linear search), (b) ବାଇନାରି ସର୍ଚ୍ଚ (Binary search),

   (c) ରାଣ୍ଡମ୍ ସର୍ଚ୍ଚ(Random search), (d) ପ୍ରତ୍ୟକ୍ଷ ସର୍ଚ୍ଚ (Direct search)।

5. ବିଗ୍-O ସଙ୍କେତନ ବ୍ୟବହାର କରି, ବାଇନାରି ସର୍ଚ୍ଚଦ୍ୱାରା କେତେ ସଂଖ୍ୟକ ତୁଳନା ଆବଶ୍ୟକ।

   (a) O (log2n) (b) O (n) (c) O (n2) (d) O (nlog2n)

6. nଟି ଉପାଦାନର ସିଲେକ୍‌ସନ ସର୍ଟ ପ୍ରକ୍ରିୟାରେ, ଆଲଗୋରିଦମ୍‌ର ସମ୍ପୂର୍ଣ୍ଣ ନିଷ୍ପାଦନପାଇଁ ସର୍ବାଧିକ କେତେଥର ସ୍ୱାପ ଫଙ୍କସନ(swap function)କୁ ଡାକିବାକୁ ପଡେ?

   (a) 1 (b) n-1 (c) nlog2n (d) n2

7. ଧରାଯାଉ 100ଟି ଉପାଦାନର ସିଲେକସନ ସର୍ଟ ପଦ୍ଧତିରେ ମୁଖ୍ୟ ଲୁପ୍ 42 ଥର ଆଇଟରେସନ ସମାପ୍ତ କରିସାରିଛି। ବର୍ତ୍ତମାନ କେତୋଟି ଉପାଦାନ ସେମାନଙ୍କର ଉଦ୍ଦିଷ୍ଟ ସ୍ଥାନରେ ପହଞ୍ଚି ଯାଇଥିବାର ନିର୍ଭରୋକ୍ତି ଅଛି?

   (a) 21 (b) 41 (c) 42 (d) 43

8. ଧରାଯାଉ ଆମେ କିଛି ସର୍ଟିଂ ଆଲଗୋରିଦମ୍ ବ୍ୟବହାର କରି ଆଠଟି ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା(integers) ବିଶିଷ୍ଟ ଏକ ଆରେକୁ ଆରୋହଣ କ୍ରମରେ ସଜାଉଛୁ। ଆଲଗୋରିଦମ୍ ମୁଖ୍ୟ ଲୁପ୍‌ର ଚାରୋଟି ଆଇଟରେସନ ପରେ, ଯଦି ଆରେ ଉପାଦାନଗୁଡିକ ନିମ୍ନ ଲିଖିତ କ୍ରମରେ ଥାଏ:

   52 54 55 57 58 51 53 56

   ତେବେ ନିମ୍ନ ପ୍ରଦତ୍ତ କେଉଁ ବିବୃତ୍ତିଟି ଠିକ୍?

   (a) ଆଲଗୋରିଦମ୍‌ଟି ହୁଏତ ସିଲେକସନ ସର୍ଟ କିମ୍ବା ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ପଦ୍ଧତି ହୋଇପାରେ,

   (b) ଆଲଗୋରିଦମ୍‌ଟି ସିଲେକସନ ସର୍ଟ ପଦ୍ଧତି ହୋଇପାରେ, କିନ୍ତୁ ଏହା ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ପଦ୍ଧତି ନୁହେଁ,

   (c) ଆଲଗୋରିଦମ୍‌ଟି ସିଲେକସନ ସର୍ଟ ପଦ୍ଧତି ନୁହେଁ, କିନ୍ତୁ ଏହା ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ପଦ୍ଧତି ହୋଇପାରେ,

   (d) ଆଲଗୋରିଦମ୍‌ଟି ସିଲେକସନ ସର୍ଟ ପଦ୍ଧତି କିମ୍ବା ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ପଦ୍ଧତି ନୁହେଁ।

9. ଧରାଯାଉ ଆମେ କିଛି ସର୍ଟିଂ ଆଲଗୋରିଦମ୍ ବ୍ୟବହାର କରି ଦଶଟି ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା(integers) ବିଶିଷ୍ଟ ଏକ ଆରେକୁ ଆରୋହଣ କ୍ରମରେ ସଜାଉଛୁ। ଆଲଗୋରିଦମ୍ ମୁଖ୍ୟ ଲୁପ୍‌ର ଚାରୋଟି ଆଇଟରେସନ ପରେ, ଯଦି ଆରେ ଉପାଦାନଗୁଡିକ ନିମ୍ନ ଲିଖିତ କ୍ରମରେ ଥାଏ:

   11 22 33 42 55 50 66 87 98 80

   ତେବେ ନିମ୍ନ ପ୍ରଦତ୍ତ କେଉଁ ବିବୃତ୍ତିଟି ଠିକ୍?

   (a) ଆଲଗୋରିଦମ୍‌ଟି ହୁଏତ ସିଲେକସନ ସର୍ଟ କିମ୍ବା ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ହୋଇପାରେ,

   (b) ଆଲଗୋରିଦମ୍‌ଟି ସିଲେକସନ ସର୍ଟ ହୋଇପାରେ, କିନ୍ତୁ ଏହା ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ନୁହେଁ,

   (c) ଆଲଗୋରିଦମ୍‌ଟି ସିଲେକସନ ସର୍ଟ ନୁହେଁ, କିନ୍ତୁ ଏହା ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ହୋଇପାରେ,

   (d) ଆଲଗୋରିଦମ୍‌ଟି ସିଲେକସନ ସର୍ଟ କିମ୍ବା ଇନ୍‌ସର୍‌ସନ ସର୍ଟ ନୁହେଁ।

10. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଆଲଗୋରିଦମ୍‌ଟିର କାର୍ଯ୍ୟଦକ୍ଷତା ବୃଦ୍ଧି କରିବାକୁ ସଂଶୋଧନ କରାଯାଇପାରିବ?

   (a) ସିଲେକସନ ସର୍ଟ, (b) ବବଲ୍ ସର୍ଟ,

   (c) ଇନ୍‌ସର୍‌ସନ ସର୍ଟ, (d) ଉପରୋକ୍ତ ମଧ୍ୟରୁ କୌଣସିଟି ନୁହେଁ।

ଉତ୍ତର									
1.	(*c*)	**2.**	(*b*)	**3.**	(*c*)	**4.**	(*a*)	**5.**	(*a*)
6.	(*b*)	**7.**	(*c*)	**8.**	(*c*)	**9.**	(*b*)	**10.**	(*b*)

ପ୍ରୋଗ୍ରାମିଂ (Programming) ସମସ୍ୟା

1. ଅବତରଣ କ୍ରମେ ସଜ୍ଜିତ ଦିଆଯାଇଥିବା ଉପାଦାନଗୁଡିକ ମଧ୍ୟରୁ ଏକ ଉପାଦାନ ଅନ୍ୱେଷଣପାଇଁ ବାଇନାରି ସର୍ଚ୍ଚ ବ୍ୟବହାର କରି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
2. ତୁମ ଶ୍ରେଣୀର ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କ ନାମର ତାଲିକା ତୁମକୁ ଦିଆଯାଇଛି । ଅଭିଧାନ ନିୟମକ୍ରମେ ନାମଗୁଡିକ ସଜାଡିବାପାଇଁ ନିଜ ପସନ୍ଦର ଏକ ସର୍ଟିଂ ଆଲଗୋରିଦମ୍ ବ୍ୟବହାର କରି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
3. ତୁମ ନାମର ଅକ୍ଷରଗୁଡ଼ିକୁ ଅଭିଧାନ-ବିପରୀତମୁଖୀ କ୍ରମରେ ସଜେଇବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
4. ବବଲ୍ ସର୍ଟ ପଦ୍ଧତି ବ୍ୟବହାର କରି ସଂଖ୍ୟାର ଏକ ତାଲିକାକୁ ଅବତରଣକ୍ରମେ କ୍ରମବଦ୍ଧପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ । ବର୍ତ୍ତମାନ ଏହି ଆଲଗୋରିଦମରେ ଏପରି କ୍ଷମତା ଅନ୍ତର୍ଭୂକ୍ତ କର ଯେପରିକି ଯଦି ସମସ୍ତ ପର୍ଯ୍ୟାୟ ଶେଷ ହେବା ପୂର୍ବରୁ ତାଲିକା କ୍ରମବଦ୍ଧ ହୋଇଯାଇଥିବ, ତେବେ ଆଲଗୋରିଦମ୍ ବନ୍ଦ ହୋଇଯିବ ।

ପ୍ରାକ୍ଟିକାଲ୍ (PRACTICALS)

ଲ୍ୟାବର ଏକ ଅଂଶ ଭାବରେ ସର୍ଚ୍ଚିଂ ଏବଂ ସର୍ଟିଂପାଇଁ ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମଗୁଡିକ ବ୍ୟତୀତ, ସଂଖ୍ୟାତ୍ମକ ପଦ୍ଧତି(numerical method) ସମାଧାନପାଇଁ ଯେଉଁଥିରେ ସଂଖ୍ୟାତ୍ମକ ଅବକଳନ(numerical differentiation) ଏବଂ ସମାକଳନ (integration) କରିବାପାଇଁ ଲ୍ୟାବ୍ ପ୍ରୋଗ୍ରାମ୍ (lab program) ଅନ୍ତର୍ଭୁକ୍ତ ଅଟେ।

ଏହି ପ୍ରୋଗ୍ରାମଗୁଡିକ ପରବର୍ତ୍ତୀ ଅନୁଚ୍ଛେଦରେ ବର୍ଣ୍ଣନା କରାଯାଇଛି।

1. ସଂଖ୍ୟାତ୍ମକ ଅବକଳନ(numerical differentiation)

ବିଜ୍ଞାନ ଏବଂ ଇଞ୍ଜିନିୟରିଂରେ ଅନେକ ସମସ୍ୟା ବିଭିନ୍ନ ପ୍ରକାରର ଫଙ୍କସନ୍(functions)ର ଅବକଳନ ବ୍ୟବହାର ସହ ଜଡିତ। ଯଦି ଗାଣିତିକ ରୂପରେ ଏକ ଫଙ୍କସନ୍‌କୁ ପ୍ରକାଶ କରାଯାଏ, ତେବେ ଏହାର ବ୍ୟୁତ୍ପତ୍ତି(derivative) ବିଶ୍ଳେଷଣାତ୍ମକ(analytical) ପଦ୍ଧତି ମାଧ୍ୟମରେ ସହଜରେ ପ୍ରାପ୍ତ ହୋଇପାରିବ ।

କିନ୍ତୁ ଅନେକ ପରିସ୍ଥିତିରେ, ସ୍ୱାଧୀନ ଚଳର ମୂଲ୍ୟ x ଏବଂ ନିର୍ଭରଶୀଳ ଚଳର ଅନୁରୂପ ମୂଲ୍ୟ y ସାରଣୀବଦ୍ଧ ରୂପେ ଉପଲବ୍ଧ ।

ଏକ ସ୍ଥିତିର କଳନା କର ଯେଉଁଥିରେ ଜଣେ ଖେଳାଳୀ 100 ମିଟର ଦୌଡ଼ ସମୟରେ, ଅତିକ୍ରମ କରିଥିବା ଦୂରତା, ସମୟର ଏକ ଫଙ୍କସନ୍ ଭାବରେ ନିମ୍ନ ସାରଣୀରେ ଦିଆଯାଇଛି ।

ସମୟ (ସେକେଣ୍ଡ):	0	1	2	3	4	5	6	7	8	9	10
ଦୂରତା (ମିଟର):	0	2.5	10	20	30.5	50	54	65.5	77.2	88.5	10

ଏଠାରେ ଅନେକ ପ୍ରଶ୍ନ ପଚରାଯାଇପାରେ, ଉଦାହରଣ ସ୍ୱରୂପ:

1. 5 ସେକେଣ୍ଡ ପରେ ଖେଳାଳୀଙ୍କ ବେଗ କ'ଣ ଥିଲା ?
2. ଟେପ୍ ନିକଟରେ ପହଞ୍ଚିବା ବେଳକୁ ଖେଳାଳୀଙ୍କ ବେଗ କ'ଣ ଥିଲା ?
3. 88 ସେକେଣ୍ଡ ପରେ ଖେଳାଳୀଙ୍କ ତ୍ୱରଣ କେତେ ଥିଲା ?

ସାଧାରଣତଃ, ବେଗ ଏବଂ ତ୍ୱରଣକୁ ବିଶ୍ଳେଷଣାତ୍ମକ ବିଧିରେ ନିରୂପଣ କରାଯାଏ । ଯଦି ଖେଳାଳୀଙ୍କଦ୍ୱାରା x ସେକେଣ୍ଡରେ ଅତିକ୍ରମ କରିଥିବା ଦୂରତା y ହୁଏ, ତା'ହେଲେ

ବେଗ $= \frac{dy}{dx}$ ତ୍ୱରଣ $= \frac{d^2 y}{dx^2}$

କିନ୍ତୁ ଉପରୋକ୍ତ ଉଦାହରଣରେ y ଏବଂ x ମଧ୍ୟରେ ଗାଣିତିକ ସମ୍ପର୍କ ଜଣାନାହିଁ। ତେଣୁ, ବିଶ୍ଳେଷଣାତ୍ମକ ପଦ୍ଧତି ବ୍ୟବହାର କରିବା ସମ୍ଭବ ନୁହେଁ । ଏଣୁ, $\frac{dy}{dx}$ ଏବଂ $\frac{d^2 y}{dx^2}$ ର ନିକଟବର୍ତ୍ତୀ ମୂଲ୍ୟଗୁଡ଼ିକ ପାଇବାପାଇଁ ଆମେ ସାରଣୀରେ ଦିଆଯାଇଥିବା ତଥ୍ୟ ବ୍ୟବହାର କରିପାରିବା । ଏହିପରି ଗଣନା କରିପାରୁଥିବା କୌଶଳକୁ ସଂଖ୍ୟାତ୍ମକ ଅବକଳନ କୁହାଯାଏ ।

ସାରଣୀବଦ୍ଧ କୌଣସି ଏକ ବିନ୍ଦୁରେ ପ୍ରଥମ ବ୍ୟୁତ୍ପତ୍ତିର ସଂଖ୍ୟାତ୍ମକ ମୂଲ୍ୟ ପାଇବାପାଇଁ ସାଧାରଣ ସୂତ୍ର ହେଉଛି

$$\left.\frac{dy}{dx}\right|_{x=x_k} = \frac{1}{h}\sum_{j=1}^{n-k}\left[(-1)^{j+1}\frac{d_{kj}}{j}\right]$$

ଯେଉଁଠାରେ d_{kj}, Δ^j_k ର ପ୍ରତିନିଧିତ୍ୱ କରେ (ସାରଣୀବଦ୍ଧ ବିନ୍ଦୁ $x = x_k$ ଠାରେ jତମ ଘାତର ଅଗ୍ରାନ୍ତର) ଏବଂ nଟି ବିନ୍ଦୁ ଉପରେ ସାରଣୀବଦ୍ଧ ଫଙ୍କସନ୍ପାଇଁ ଅର୍ଡର $(n-1)\times(n-1)$ର ମ୍ୟାଟ୍ରିକ୍ସ D ଦ୍ୱାରା ଦର୍ଶାହୋଇଥିବା ଅଗ୍ରାନ୍ତର ସାରଣୀର ଏକ ଉପାଦାନ ଅଟେ ।

ଟେବୁଲ୍ 5.1: ଅଗ୍ରାନ୍ତର ସାରଣୀ

i	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
1	x_1	x_1	$\Delta y_1 = y_2 - y_1$	$\Delta^2 y_1 = \Delta y_2 - \Delta y_1$	$\Delta^3 y_1 = \Delta^2 y_2 - \Delta^3 y_1$
2	x_2	x_2	$\Delta y_2 = y_3 - y_2$	$\Delta^2 y_2 = \Delta y_3 - \Delta y_2$	
3	x_3	x_3	$Dy_3 = y_4 - y_3$		
4	x_4	x_4			

ଯେପରିକି ଆମେ ଉପରୋକ୍ତ ସାରଣୀରୁ ଦେଖିପାରିବା ଯେ, ଚାରୋଟି ବିନ୍ଦୁ ଉପରେ ସାରଣୀବଦ୍ଧ ଫଙ୍କସନ୍ପାଇଁ ଅଗ୍ରାନ୍ତର ସାରଣୀ, ସମାନ ଆକାରରେ 3 × 3 ଆକାରର ଏକ ମ୍ୟାଟ୍ରିକ୍ସ D ଦ୍ୱାରା ପଏଣ୍ଟଗୁଡ଼ିକର ବ୍ୟବଧାନ ପ୍ରତିପାଦିତ ହୋଇପାରିବ । ସାଧାରଣତଃ nଟି ସମଦୂରସ୍ଥ ବିନ୍ଦୁ ଉପରେ ସାରଣୀବଦ୍ଧ ଏକ ଫଙ୍କସନ୍ପାଇଁ ଅଗ୍ରାନ୍ତର ସାରଣୀ, ଏକ *(n-1) × (n-1)* ଆକାରର ମ୍ୟାଟ୍ରିକ୍ସଦ୍ୱାରା ଉପସ୍ଥାପିତ ହୋଇପାରିବ, ଯେଉଁଠାରେ *ith* ବିନ୍ଦୁରେ *jth* ଘାତ ଅଗ୍ରାନ୍ତରଟି ମ୍ୟାଟ୍ରିକ୍ସ Dର ଉପାଦାନ dij ଦ୍ୱାରା ଉପସ୍ଥାପିତ ହୋଇପାରିବ । ଧ୍ୟାନ ଦିଅ ଯେ ଆଗ୍ରହର ଉପାଦାନଗୁଡ଼ିକ ହେଲା, କେବଳ ଧାଡି *i = 1, 2, 3, ..., n-1,* ର ସ୍ତମ୍ଭ 1 ରୁ *(n-i)* ପର୍ଯ୍ୟନ୍ତ ।

ଉଦାହରଣ 5.5: ନିମ୍ନଲିଖିତ ଟେବୁଲ୍ ଦିଆଯାଇଛି

x :	0.50	0.75	1.00	0.25	1.50
$x = f(x)$:	0.13	0.42	1.00	1.95	2.35

$f^1(0.75)$ ବାହାର କର

ସମାଧାନ: ଦିଆଯାଇଥିବା ଡାଟାପାଇଁ ଅଗ୍ରାନ୍ତର ସାରଣୀ ହେଉଛି

i	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$
1	0.50	0.13	0.29	0.29	0.08	–1.00
2	0.75	0.42	0.58	0.37	–0.92	
3	1.00	1.00	0.95	–0.55		
4	1.25	1.95	0.40			
5	1.50	2.35				

ଯେହେତୁ $h = 0.25$, ଏବଂ ବ୍ୟୁତ୍ପନ୍ନ $x = 0.75$ (ଅର୍ଥାତ, $i = 2$) ରେ ବାଞ୍ଛନୀୟ ଅଟେ। ତୃତୀୟ ପଦ ପର୍ଯ୍ୟନ୍ତ ଫର୍ମୁଲାକୁ ବିସ୍ତାର କଲେ, ଆମକୁ ମିଳିବ

$$\left.\frac{dy}{dx}\right|_{x=0.75} = \frac{1}{h}\left[d_{21} - \frac{d_{22}}{2} + \frac{d_{23}}{3}\right]$$

$$= \frac{1}{0.25}\left[0.58 - \frac{0.37}{2} + \frac{-0.92}{3}\right] = \frac{1}{0.25}[0.58 - 0.185 - 0.307] = 0.352$$

ତାଲିକା 5.7

```
/*
 *  Program to compute first derivative of the tabulated function
 */
 #include<stdio.h>
 int main()
 {
    float x[10], y[10], d[10][10], a, sum, derivative, h, term;
    int i, j, k, n, sign;
    printf("Enter number of table points n(<=10) : ");
    scanf("%d", &n);
    printf("Enter interval size : ");
    scanf("%f", &h);
    printf("Enter %d pair of values as x,y\n", n);
    for ( i = 1; i <= n; i++ )
    scanf("%f,%f", &x[i], &y[i]);
    printf("Enter tabulated point where to find derivative : ");
    scanf("%f", &a);
  if ( (a < x[1]) || (a > x[n]) )
    {
    printf("\nValue lies outside range\n");
          return 1;
    }
    i = 1;
```

```
    while ( a != x[i] )
       i++;
    k = i;
    for ( j = 1; j <= (n-1); j++ )
    {
   for ( i = 1; i <= (n-j); i++ )
       {
      if ( j == 1 )
         d[i][j] = y[i+1] - y[i];
               else
         d[i][j] = d[i+1][j-1] - d[i][j-1];
                  }
    }
    sum = 0.0;
    sign = 1;
    for ( j = 1; j <= (n-k); j++ )
    {
```

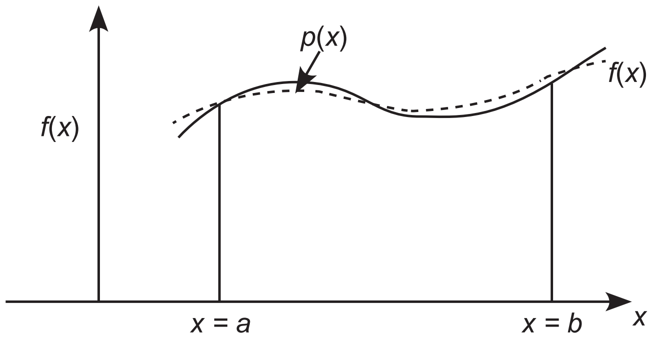
ଟେଷ୍ଟ ରନ୍

```
Enter number of table points n(<=10) : 5
Enter interval size : 0.25
Enter 5 pair of values as x,y
0.5,0.13
0.75,0.42
1.0,1.0
1.25,1.95
1.5,2.35
Enter tabulated point where to find derivative : 0.75
Value of derivative = 0.352
```

2. ସଂଖ୍ୟାତ୍ମକ ସମାକଳନ (Numerical Integration)

ସଂଖ୍ୟାତ୍ମକ ସମାକଳନ ହେଉଛି, ଫଙ୍କସନ୍‌ର ସଂଖ୍ୟାତ୍ମକ ମୂଲ୍ୟର ଏକ ସେଟ୍‌ରୁ ଏକ ସୀମାଯୁକ୍ତ ସମାକଳର (definite integral) ମୂଲ୍ୟ ନିରୂପଣ କରିବାର ପ୍ରକ୍ରିୟା । ଯଦି ଫଙ୍କସନ୍‌କୁ ଏକ ଗାଣିତିକ ଅଭିବ୍ୟକ୍ତି ରୂପେ ବ୍ୟାଖ୍ୟା କରାଯାଏ, ତେବେ ଏହାର ସମାକଳ ସାଧାରଣତଃ କାଲ୍‌କୁଲସ୍ (calculus) କୌଶଳ ବ୍ୟବହାର କରି ନିର୍ଣ୍ଣୟ କରାଯାଏ, ଏହିପରି ସମାଧାନକୁ ଆବଦ୍ଧ ଫର୍ମ (closed form) ସମାଧାନ କୁହାଯାଏ ।

ପ୍ରାୟତଃ, ଟେବୁଲ୍ ଆକାରରେ ଉପଲବ୍ଧ ଥିବା ଫଙ୍କସନ୍ ଏବଂ ଚଳ ମଧ୍ୟରେ କୌଣସି ଗାଣିତିକ ସମ୍ପର୍କ ଜଣା ନାହିଁ । ଏହିପରି ସମସ୍ତ ପରିସ୍ଥିତିରେ, ସମାକଳର ମୂଲ୍ୟ ସଂଖ୍ୟାତ୍ମକ କୌଶଳଦ୍ୱାରା ପ୍ରାପ୍ତ କରାଯାଇପାରିବ ଯାହାର ଲକ୍ଷ୍ୟ ହେଉଛି ସୀମାଯୁକ୍ତ ସମାକଳର ମୂଲ୍ୟାଙ୍କନପାଇଁ ପ୍ରଭାବଶାଳୀ ବିଧି ପ୍ରସ୍ତୁତ କରିବା ।

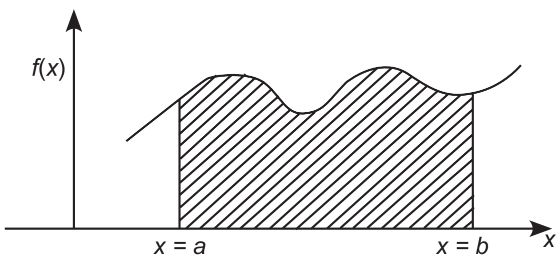


ଚିତ୍ର. 5.5: ଏକ ଫଙ୍କସନ୍ f(x)ପାଇଁ ଆନୁମାନିକ ପଲିନୋମିଆଲ୍ p(x)

ବ୍ୟବହାରପାଇଁ, ଫଙ୍କସନ୍ f(x)ପାଇଁ ମୂଲ୍ୟଗୁଡ଼ିକର ସେଟ୍, ତଥ୍ୟର ନିମ୍ନଲିଖିତ ସାରଣୀ

x :	a	x_1	x_2	x_3	…	b
$f(x)$:	$f(a)$	$f(x_1)$	$f(x_2)$	$f(x_3)$	…	$f(b)$

ସମାକଳ $\int_a^b f(x)dx$ ର ମୂଲ୍ୟ ନିରୂପଣପାଇଁ ବ୍ୟବହୃତ ହୁଏ।



ଚିତ୍ର. 5.6: ଛାୟାଙ୍କିତ କ୍ଷେତ୍ରଦ୍ୱାରା ପ୍ରତିନିଧିତ୍ୱ କରୁଥିବା ସୀମାଯୁକ୍ତ ସମାକଳ

ଏକ ଫଙ୍କସନ୍‌ର ସୀମାଯୁକ୍ତ ସମାକଳ ହେଉଛି x = a ଏବଂ x = b ସୀମା ମଧ୍ୟରେ y = f (x) ବକ୍ରରେଖାର ତଳଭାଗରେ ଥିବା ଆବଦ୍ଧ କ୍ଷେତ୍ରର କ୍ଷେତ୍ରଫଳ, ଏକ ଫଙ୍କସନ୍‌ର ସମାକଳ ନିରୂପଣ କରିବାର ସମସ୍ୟା ଚିତ୍ର 5.6 ରେ ଦେଖାଯାଇଥିବା ପରି ଛାୟାଙ୍କିତ କ୍ଷେତ୍ର ନିରୂପଣ ସମସ୍ୟାକୁ ପରିବର୍ତ୍ତିତ ହୁଏ।

ସଂଖ୍ୟାତ୍ମକ ସମାକଳନ ବାହାର କରିବାପାଇଁ ବ୍ୟବହୃତ ଲୋକପ୍ରିୟ ପଦ୍ଧତିଗୁଡିକ ହେଲା:

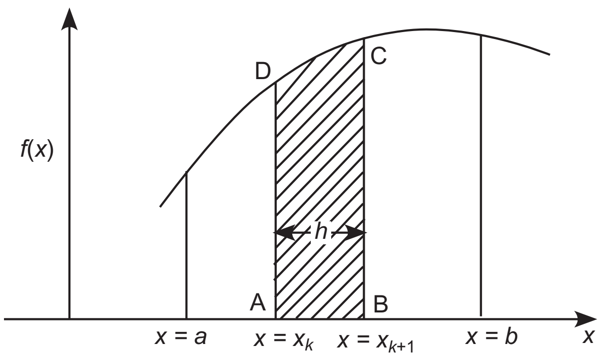
- ଟ୍ରାପିଜଏଡାଲ୍ ନିୟମ (Trapezoidal rule)
- ସିମ୍ପସନଙ୍କ 1/3rd ନିୟମ (Simpson's 1/3rd rule)
- ସିମ୍ପସନଙ୍କ 3/8th ନିୟମ (Simpson's 3/8th Rule)

ଏକ ଫଙ୍କସନ୍‌ର ସଂଖ୍ୟାତ୍ମକ ସମାକଳନ ମୂଲ୍ୟ ନିରୂପଣପାଇଁ ଆମେ ଟ୍ରାପିଜଏଡାଲ୍ (Trapezoidal) ନିୟମକୁ ଏକ ଉଦାହରଣ ଭାବରେ ବିଚାର କରିବା।

ଟ୍ରାପିଜଏଡାଲ୍ (Trapezoidal) ନିୟମ

ଟ୍ରାପିଜଏଡାଲ୍ (Trapezoidal) ନିୟମ ବକ୍ରରେଖାର ତଳଭାଗରେ ଥିବା କ୍ଷେତ୍ରର ଆସନ୍ନ କ୍ଷେତ୍ରଫଳପାଇଁ, ବକ୍ର ଉପରସ୍ଥ ଉତ୍ତରୋତ୍ତର ବିନ୍ଦୁଗୁଡ଼ିକୁ ସଂଯୋଗ କରି କେତେକ ସମାନ ପ୍ରସ୍ଥର ଟ୍ରାପିଜଏଡ ଗଠନ କରେ ଏବଂ ସମସ୍ତ ଟ୍ରାପିଜଏଡ ଅନ୍ତର୍ଗତ କ୍ଷେତ୍ରଫଳଗଡ଼ିକୁ ଯୋଗକରି ଆବଶ୍ୟକ ଆସନ୍ନ କ୍ଷେତ୍ରଫଳ ପାଇଥାଏ ।

ଫଙ୍କସନ୍ f(x) କୁ ବିଚାର କର ଯାହାର ଗ୍ରାଫ୍ x = a ଏବଂ x = b ମଧ୍ୟରେ ଚିତ୍ର. 5.7ରେ ଦର୍ଶାଯାଇଛି । ବକ୍ରରେଖାର ତଳେ ଥିବା କ୍ଷେତ୍ରର କ୍ଷେତ୍ରଫଳର ଏକ ଆସନ୍ନମାନପାଇଁ ଅନ୍ତରାଳ [a, b] କୁ h ପ୍ରସ୍ଥଯୁକ୍ତ nଟି ପଟିରେ ଭାଗ କରା ହୁଏ, ଏବଂ ଛାୟାଙ୍କିତ ଅଞ୍ଚଳଦ୍ୱାରା ଦର୍ଶାଯାଇଥିବା ପ୍ରତ୍ୟେକ ପଟିର କ୍ଷେତ୍ରଫଳକୁ ଆନୁମାନିକ ଏକ ଟ୍ରାପିଜଏଡ୍‌ର କ୍ଷେତ୍ରଫଳ ବୋଲି ଧରାଯାଏ ।



ଚିତ୍ର. 5.7: ଟ୍ରାପିଜୋଇଡାଲ୍ ନିୟମଦ୍ୱାରା କ୍ଷେତ୍ରଫଳର ଆନୁମାନିକତା

ଟ୍ରାପିଜଏଡ୍ ନିୟମପାଇଁ ସୂତ୍ର ହେଉଛି

$$I = \frac{h}{2}\left[y_1 + 2y_2 + 2y_3 + 2y_4 + .. + 2y_n + y_{n+1}\right]$$

ଧରିନେବା ଯେ ଫଙ୍କସନ୍ f(x) ନିମ୍ନଲିଖିତ ଫର୍ମରେ ଦିଆଯାଇଛି ।

x	x_1	$x_1 + h$	$x_1 + 2h$	…	$x_1 + 2h$
$y = f(x)$	$f(a)$	$f(x_1)$	$f(x_2)$	…	y_{n+1}

ଉଦାହରଣ 5.6: ଫଙ୍କସନ୍ f(x) ନିମ୍ନଲିଖିତ ଭାବରେ ଦିଆଯାଇଛି:

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
x	1	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6	2.8	3.0

x = 0 ଏବଂ x =1.0 ମଧ୍ୟରେ f(x) ର ସମାକଳ ନିର୍ଣ୍ଣୟ କର ।

ସମାଧାନ:

ଦିଆଯାଇଛି, h = 0.1 ଏବଂ n = 10

ଟ୍ରାପିଜଏଡାଲ୍ (Trapezoidal) ନିୟମର ସୂତ୍ର ହେଉଛି

$$I = \frac{h}{2}[y_1 + 2(y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 + y_9 + y_{10}) + y_{11}]$$

yi ଏବଂ h ର ମୂଲ୍ୟଗୁଡିକ ପ୍ରତିସ୍ଥାପନ କଲେ, ଆମେ ପାଇବୁ

$$I = \frac{0.1}{2}[1 + 2\times(1.2 + 1.4 + 1.6 + 1.8 + 2.0 + 2.2 + 2.4 + 2.6 + 2.8) + 3.0]$$

ତାଲିକା 5.8

```
/*  Program to implement Trapezoidal rule for tabulated function  */
 #include<stdio.h>
 int main()
 {
    int i, n;
    float x[20], y[20], h, sum, integral;
    printf("Enter number of intervals n(<=20) : ");
    scanf("%d", &n);
    printf("Enter size of interval : ");
    scanf("%f", &h);
    printf("Enter %d pair of values as x,y \n", n+1);
    for ( i = 1; i <= (n+1); i++ )
        scanf("%f,%f", &x[i], &y[i]);
    sum = ( y[1] + y[n+1] )/2;
    for ( i = 2; i <= n; i++ )
        sum = sum + y[i];
    integral = h * sum;
    printf("\nValue of the integral = %7.2f\n", integral);
    return 0;
 }
```

ଟେଷ୍ଟ ରନ୍

```
Enter number of intervals n(<=20) : 10
Enter size of interval : 0.1
Enter 11 pair of values as x,y
0,1
0.1,1.2
0.2,1.4
0.3,1.6
0.4,2.0
0.5,2.2
0.6,2.4
0.7,2.6
0.8,2.8
0.9,3.0
1.0,1.2
Value of the integral = 2.00
```

ଅଧିକ ଜାଣିବାପାଇଁ

ଶିକ୍ଷକ ଉଦାହରଣ ସାହାଯ୍ୟରେ ଏହି ପଦ୍ଧତିଗୁଡ଼ିକୁ ବ୍ୟାଖ୍ୟା କରିବେ ଏବଂ ଛାତ୍ରଛାତ୍ରୀମାନେ ନିଜେ ସମାଧାନ କରିବାକୁ ସମର୍ଥ ହେବାପାଇଁ ସୁବିଧା କରିବେ ବୋଲି ଆଶା କରାଯାଉଛି । ଆଲୋଚନା କରାଯାଇ ବିଭିନ୍ନ ପଦ୍ଧତିକୁ ଛାତ୍ରଛାତ୍ରୀମାନେ ଥରେ ବୁଝିବା ପରେ, ସେମାନେ ନିଜେ ନିଜର ପ୍ରୋଗ୍ରାମ ଲେଖିବାକୁ ପ୍ରୋତ୍ସାହିତ କରାଯିବା ଉଚିତ ।

ପ୍ରୋଗ୍ରାମକୁ ଏକ ସନ୍ଦର୍ଭ ଭାବରେ ବିକଶିତ କରିବାକୁ ଛାତ୍ରଛାତ୍ରୀଙ୍କୁ ମାର୍ଗଦର୍ଶନ କରନ୍ତୁ । ପୁସ୍ତକରେ ପ୍ରୋଗ୍ରାମ ଦେବାର ଉଦ୍ଦେଶ୍ୟ ନୁହେଁ ଯେ 'ପିଲାକୁ ଚାମଚରେ ଖୁଆଇବା', ବରଂ ଉଚ୍ଚତର ପ୍ରୋଗ୍ରାମିଂ ଶୈଳୀର ନିଦର୍ଶନ ଦେବା ।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

6 ପ୍ରକାର୍ଯ୍ୟ (Functions)

ଏହି ଏକକର ବିଶେଷତ୍ୱ

ଏହି ଏକକରେ ପ୍ରକାର୍ଯ୍ୟ ସମ୍ବନ୍ଧୀୟ ବିଷୟବସ୍ତୁ ଉପରେ ଆଲୋଚନା କରାଯାଇଛି । ଜଣେ ପ୍ରୋଗ୍ରାମର୍ ଫଙ୍କସନ୍‌ର ବ୍ୟବହାରଦ୍ୱାରା ଗୋଟିଏ ଜଟିଳ ସମସ୍ୟାର ପୃଥକ ଭାବେ ସମାଧାନ ହେଇପାରୁଥିବା କେତେକ ଛୋଟ-ଛୋଟ ତଥା ସ୍ୱାଧୀନ ଉପ-ସମସ୍ୟାରେ ବିଭକ୍ତ କରିପାରିବ । ଏହି ଉପ-ସମସ୍ୟାବଳିର ସମାଧାନ ଏକାତ୍ର ସଂଶ୍ଳେଷଣଦ୍ୱାରା ଦିଆଯାଇଥିବା ମୂଳ ଜଟିଳ ସମସ୍ୟାର ସମାଧାନ ମିଳିପାରିବ । ଏହି ୟୁନିଟ୍‌ରେ ପ୍ରକାର୍ଯ୍ୟର ବିବିଧ ଦିଗର ବର୍ଣ୍ଣନା ଉପଯୁକ୍ତ ଉଦାହରଣସହ ଉପସ୍ଥାପିତ ହୋଇଛି ।

ଭୂମିକା

ବାସ୍ତବ ଜୀବନର ଅନେକ ପରିସ୍ଥିତିରେ, ଆମକୁ ବୃହତ୍ତ ଓ କଠିନ ସମସ୍ୟାର ମୁକାବିଲା କରିବାକୁ ପଡ଼ିଥାଏ । ଏହିଭଳି ସମସ୍ୟାଗୁଡ଼ିକର ଏକକାଳୀନ ସମାଧାନ ନିର୍ଣ୍ଣୟ ଆମ ପକ୍ଷରେ ଅତ୍ୟନ୍ତ ବିରକ୍ତିକର ତଥା ତ୍ରୁଟି ପୂର୍ଣ୍ଣ ହୋଇପାରେ । ତେଣୁ ସର୍ବୋତ୍ତମ ଉପାୟ ହେଉଛି ପ୍ରଦତ୍ତ ସମସ୍ୟାକୁ ପରସ୍ପରଠାରୁ ସ୍ୱାଧୀନ କେତେଗୁଡ଼ିଏ ଉପ-ସମସ୍ୟାରେ ବିଭକ୍ତ କରିବା, ଯାହାକି ସହଜରେ ଏବଂ ସ୍ୱାଧୀନ ଭାବରେ ସମାଧାନ କରାଯାଇପାରିବ । ପ୍ରତ୍ୟେକ ଉପ-ସମସ୍ୟାର ସମାଧାନ ଥରେ ମିଳିଗଲେ, ସମଗ୍ର ସମସ୍ୟାର ସମାଧାନ ପାଇବାପାଇଁ ସଂଶ୍ଳେଷଣ କରିହେବ ।

ଏହିପରି, ଏକ ଜଟିଳ ପ୍ରୋଗ୍ରାମ୍‌କୁ କେତେଗୁଡ଼ିଏ ସ୍ୱାଧୀନ ଫଙ୍କସନ୍‌ରେ ବିଭକ୍ତ କରାଯାଇପାରିବ । ଏହି ଫଙ୍କସନ୍‌ଗୁଡିକ ପରସ୍ପରଠାରୁ ସ୍ୱାଧୀନଭାବରେ ବିକଶିତ କଲାପରେ, ଏଗୁଡ଼ିକୁ ଏକତ୍ର କରି ଏକ ସମ୍ପୂର୍ଣ୍ଣ ପ୍ରୋଗ୍ରାମ୍ ପ୍ରସ୍ତୁତ ହୋଇପାରିବ ।

ଏହି ୟୁନିଟ୍ ଛାତ୍ରଛାତ୍ରୀଙ୍କୁ ଫଙ୍କସନ୍ ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ଦିଗ ବୁଝିବାରେ ସାହାଯ୍ୟ କରିବ ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- ସର୍ତ୍ତ ଶାଖା (Condition branching)
- ଲୁପ୍ ଏବଂ ନେଷ୍ଟେଡ ଲୁପ୍ (Loops and nested loops)
- ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍ (Arrays and strings)

ୟୁନିଟ ଲବ୍ଧ ଜ୍ଞାନ

ଏହି ୟୁନିଟ ର ସମ୍ପୂର୍ଣ୍ଣ ଅଧ୍ୟୟନ ପରେ, ଛାତ୍ରଛାତ୍ରୀଗଣ ନିମ୍ନଲିଖିତ ବିଷୟରେ ଜ୍ଞାନ ଆହରଣ ସମର୍ଥ ହେବେ ।

U6-O1: ଫଙ୍କସନ୍‌ଗୁଡ଼ିକର ଉପଯୋଗିତାର ବ୍ୟାଖ୍ୟା

U6-O2: ବିଭିନ୍ନ ପ୍ରକାର ଫଙ୍କସନ୍‌ଗୁଡିକର ବ୍ୟାଖ୍ୟା

U6-O3: ସ୍ଥାନୀୟ(local) ତଥ୍ୟ ଏବଂ ଗ୍ଲୋବାଲ୍(global) ତଥ୍ୟର ଧାରଣା ବ୍ୟାଖ୍ୟା

U6-O4: ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟର ଘୋଷଣା(declare), ବ୍ୟାଖ୍ୟା(define) ଏବଂ ଏହାକୁ ଡାକିବା ସମ୍ବନ୍ଧୀୟ ସମସ୍ତ ଦିଗର ବ୍ୟାଖ୍ୟା

U6-O5: ଗୋଟିଏ ଫଙ୍କସନ୍‌କୁ ଚର ପରିବହନର(passing arguments) ବିଭିନ୍ନ ଉପାୟର ବ୍ୟାଖ୍ୟା ।

U6-O6: ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ସମାଧାନପାଇଁ ମଡ୍ୟୁଲାର୍ ପ୍ରୋଗ୍ରାମ୍‌ର (modular programs) ବିକାଶ ।

ୟୁନିଟ୍-6 ୟୁନିଟ ଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ (1 – ଦୁର୍ବଳ ସମ୍ପର୍କ; 2 – ମଧ୍ୟମ ସମ୍ପର୍କ; 3 – ଦୃଢ ସମ୍ପର୍କ)							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U6-O1	1	–	–	–	1	–	–	–
U6-O2	–	–	–	–	1	–	–	–
U6-O3	–	–	–	–	1	–	–	–
U6-O4	–	–	2	–	2	–	–	–
U6-O5	–	–	2	–	2	–	–	–
U6-O6	–	–	–	–	3	–	–	–

6.1 ଉପକ୍ରମ (INTRODUCTION)

ଆମେ ଏପର୍ଯ୍ୟନ୍ତ କେବଳ ସରଳ ଏବଂ ସିଧାସଳଖ ପ୍ରୋଗ୍ରାମଗୁଡ଼ିକ ଆଲୋଚନା କରିଆସିଛେ । ସେଥିରେ ଏପରି ସମସ୍ୟାର ସମାଧାନ ହୋଇଛି ଯାହା ଅଧିକ ପ୍ରୟାସ ବିନା ବୁଝିହେବ । ଆମକୁ କ୍ରମେ ବୃହତ୍ତରୁ ବୃହତ୍ତର ଏବଂ ଅଧିକ ଦୁର୍ବୋଧ ସମସ୍ୟା ଓ ତା'ର ପ୍ରୋଗ୍ରାମକୁ ଯିବାକୁ ହେଉଛି, ତେଣୁ ତୁମେ ଆବିଷ୍କାର କରିବ ଯେ, ଏହାକୁ ଛୋଟଛୋଟ ସରଳ ଅଂଶରେ ପରିଣତ ନ କଲେ ଅନ୍ୟ ସମସ୍ୟାଗୁଡ଼ିକର ସମସ୍ତ ଦିଗ ଏକକାଳୀନ ବୁଝିବା ସମ୍ଭବ ନୁହେଁ ।

ଏହି ୟୁନିଟ ରେ, ଆମେ ଫଙ୍କସନ୍ ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ଦିଗଗୁଡିକ ଶିଖିବା ।

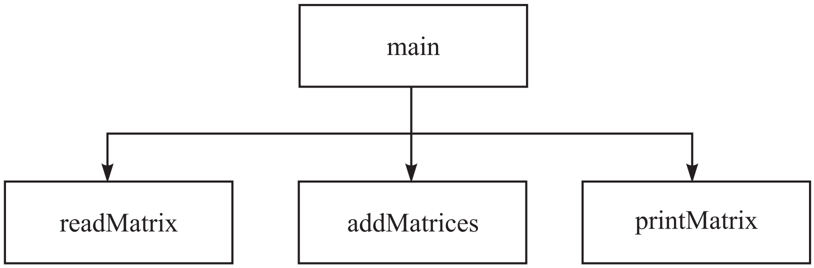
6.2 ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ କ'ଣ ?

ଗୋଟିଏ ଫଙ୍କସନ୍ ହେଉଛି କାର୍ଯ୍ୟନିର୍ବାହର ଏକ ସ୍ୱାଧୀନ ୟୁନିଟ । ଏହା ଏକ ପ୍ରୋଗ୍ରାମର ଫ୍ରେମ୍ ମଧ୍ୟରେ କିଛି ନିର୍ଦ୍ଦିଷ୍ଟ କାର୍ଯ୍ୟ ସମ୍ପାଦନ କରେ ।

ଏହା ଏହି ଅର୍ଥରେ ସ୍ୱାଧୀନ ଯେ, ଫଙ୍କସନ୍‌ଗୁଡିକରେ ସ୍ଥାନୀୟ ଭେରିଏବଲ୍ (local variables) ଘୋଷଣା ଏବଂ ଫଙ୍କସନ୍‌ଦ୍ୱାରା ସମ୍ପନ୍ନ ହେବାକୁ ଥିବା କାର୍ଯ୍ୟର ବ୍ୟାଖ୍ୟା କରୁଥିବା ନିର୍ଦ୍ଦେଶାବଳି ପାଇଁ ଏହାର ନିଜସ୍ୱ ଘୋଷଣାନାମା ଥାଏ ।

ପ୍ରତ୍ୟେକ ସି ପ୍ରୋଗ୍ରାମ୍ (C program) ଏକ ବା ଏକାଧିକ ଫଙ୍କସନ୍‌ର ସମାହାରରେ ଗଠିତ । ଯଦି ଏହା କେବଳ ଗୋଟିଏ ଫଙ୍କସନ୍‌ରେ ନିର୍ମିତ ହୁଏ, ତେବେ ଏହା ହେବ ମୁଖ୍ୟ/ମେନ୍ (main) ଫଙ୍କସନ୍ । ପ୍ରୋଗ୍ରାମ୍‌ର କାର୍ଯ୍ୟକାରୀତା ସର୍ବଦା ମୁଖ୍ୟ ଫଙ୍କସନ୍‌ରୁ ଆରମ୍ଭ ହୁଏ । ପରନ୍ତୁ, ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଫଙ୍କସନ୍‌ର ସଂଖ୍ୟା ଉପରେ କୌଣସି ପ୍ରତିବନ୍ଧକ ନାହିଁ । ପ୍ରୋଗ୍ରାମ୍‌ରେ କେତୋଟି ଫଙ୍କସନ୍ ରହିବ ତାହା ପ୍ରୋଗ୍ରାମ୍‌ଟି ଯେଉଁ ସମସ୍ୟାପାଇଁ ଲେଖାଯାଉଛି ସେହି ସମସ୍ୟାର ଆକାର ଏବଂ ଜଟିଳତା ଉପରେ ନିର୍ଭର କରେ ।

ଯେତେବେଳେ ଆମେ ଏକ ସି ପ୍ରୋଗ୍ରାମ୍ (C program) ନିଷ୍ପାଦନ କରୁ, ଅପରେଟିଂ ସିଷ୍ଟମ୍ (operating system) ମୁଖ୍ୟ ଫଙ୍କସନ୍‌କୁ କଲ୍ (Call) କରୁ, ଏବଂ ତା'ପରେ ପ୍ରୋଗ୍ରାମ୍‌ର ନିୟନ୍ତ୍ରଣ ମୁଖ୍ୟ ଫଙ୍କସନ୍ ପାରିତ ହୁଏ । ମୁଖ୍ୟ ଫଙ୍କସନ୍ ଅନ୍ୟ ଯେକୌଣସି ଫଙ୍କସନ୍‌କୁ କଲ୍ କରିପାରେ, ଆଉ ମୁଖ୍ୟ ଫଙ୍କସନ୍ ବ୍ୟତୀତ ଅନ୍ୟ ଫଙ୍କସନ୍‌ଗୁଡ଼ିକ ମଧ୍ୟ ପରସ୍ପରକୁ କଲ୍ କରିପାରିବେ ।



ଚିତ୍ର . 6.1: ଏକ ବହୁ-ଫଙ୍କ୍ସନ୍ ପ୍ରୋଗ୍ରାମର ପଦାନୁକ୍ରମିକ ସଂଗଠନ

ଯେପରି ଚିତ୍ର 6.1ରେ ଦର୍ଶାଯାଇଛି, ଯେ ପ୍ରୋଗ୍ରାମ୍‌ଟି ମୁଖ୍ୟ ଫଙ୍କ୍ସନ୍ ସହିତ ଚାରୋଟି ଫଙ୍କ୍ସନ୍‌କୁ ନେଇ ଗଠିତ। ଏବଂ ପ୍ରୋଗ୍ରାମ୍‌ର ଉଦ୍ଦେଶ୍ୟ ଦୁଇଟି ମାଟ୍ରିକ୍ସର ଯୋଗଫଳ ନିର୍ଣ୍ଣୟ। ରିଡ୍-ମାଟ୍ରିକ୍ସ ଫଙ୍କ୍ସନ୍‌ଟି ଦତ୍ତ ମାଟ୍ରିକ୍ସର ନିବେଶ (input) ସମ୍ପାଦନ କରିଥାଏ, ଦୁଇଟି ମାଟ୍ରିକ୍ସର ନିବେଶପାଇଁ ଏହାକୁ ମୁଖ୍ୟ ଫଙ୍କ୍ସନ୍‌ଦ୍ୱାରା ଦୁଇଥର ଡକାଯିବ ଯଥା Am×n ଏବଂ Bm×n।

ତା'ପରେ ଆଡ୍-ମାଟ୍ରିକ୍ସ ଫଙ୍କ୍ସନ୍‌କୁ ଡକାଯାଏ ଯାହା Am×n ଏବଂ Bm×n ମାଟ୍ରିକ୍ସକୁ ଯୋଗ କରିବ ଏବଂ ଯୋଗଫଳକୁ ଅନ୍ୟ ଏକ ମାଟ୍ରିକ୍ସ Cm×n ଦ୍ୱାରା, ମୁଖ୍ୟ ଫଙ୍କ୍ସନ୍‌କୁ ଫେରାଇ ଦେବ।

ଶେଷରେ, ମୁଖ୍ୟ ଫଙ୍କ୍ସନ୍ Cm×n ମାଟ୍ରିକ୍ସର ନିର୍ଗମନ (output)ପାଇଁ ପ୍ରିଣ୍ଟ୍-ମାଟ୍ରିକ୍ସ ଫଙ୍କ୍ସନ୍‌କୁ ଡାକିବ।

6.3 ଫଙ୍କ୍ସନ୍ ବ୍ୟବହାରର ଉପଯୋଗିତା

ଫଙ୍କ୍ସନ୍‌ର ବ୍ୟବହାର ସହିତ ଅନେକ ସୁବିଧା ଜଡିତ। କେତେକ ପ୍ରମୁଖ ଉପଯୋଗିତା ହେଉଛି

1. ଫଙ୍କ୍ସନ୍‌ର ବ୍ୟବହାରଦ୍ୱାରା ବଡ଼ବଡ଼ ଓ ଜଟିଳ ସମସ୍ୟାକୁ ଛୋଟଛୋଟ, ସରଳ ଏବଂ ପରିଚାଳନାଯୋଗ୍ୟ ଅଂଶରେ ବିଭକ୍ତ କରିବା।
2. ଫଙ୍କ୍ସନ୍‌ର ବ୍ୟବହାର କୋଡ୍ (code) ପୁନଃବ୍ୟବହାର କରିବାର ଏକ ଉପାୟ ପ୍ରଦାନ କରେ ଯାହା ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଏକାଧିକ ଥର ଆବଶ୍ୟକ।

   ଉଦାହରଣ ସ୍ୱରୂପ, ଧରାଯାଉ, ପ୍ରୋଗ୍ରାମ୍‌ର ପାଞ୍ଚଟି ଭିନ୍ନଭିନ୍ନ ସ୍ଥାନରେ ଅଲଗାଅଲଗା ସଂଖ୍ୟାକ୍ରମର ହାରାହାରି ଗଣନା କରିବା ଦରକାର। ଅର୍ଥାତ ପ୍ରତ୍ୟେକଥର ତଥ୍ୟ(ସଂଖ୍ୟାକ୍ରମ) ଅଲଗା ଅଟେ। ଆମେ ପାଞ୍ଚଥର ହାରାହାରି-ଗଣନାପାଇଁ ଆବଶ୍ୟକ ସ୍ଥାନରେ ପାଞ୍ଚଥର କୋଡ୍ ଲେଖିପାରିବା, କିନ୍ତୁ ଏଥିପାଇଁ ବହୁତ ପରିଶ୍ରମ ଲାଗିବ। କିନ୍ତୁ ହାରାହାରି-ଗଣନାପାଇଁ ଥରେ ଫଙ୍କ୍ସନ୍‌ର କୋଡ୍ ଲେଖିବା ବହୁତ ସହଜ ଏବଂ ତା'ପରେ ବିଭିନ୍ନ ତଥ୍ୟ ଦେଇ ଆବଶ୍ୟକ ସ୍ଥାନରେ ଏହାକୁ ପାଞ୍ଚଥର ଡାକି ବ୍ୟବହାର କରାଯାଇପାରିବ।
3. ଫଙ୍କ୍ସନ୍‌ଗୁଡିକର ବ୍ୟବହାର ତଥ୍ୟ ସୁରକ୍ଷିତ ରଖିପାରେ। ସ୍ଥାନୀୟ ତଥ୍ୟର ଧାରଣା ବ୍ୟବହାର କରି ଏହା କରାଯାଇଥାଏ। ସ୍ଥାନୀୟ ତଥ୍ୟ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟରେ ବ୍ୟବହୃତ ତଥ୍ୟକୁ ନେଇ ଗଠିତ। ଏହି ତଥ୍ୟ କେବଳ ଫଙ୍କ୍ସନ୍‌ର ବ୍ୟବହାରପାଇଁ ଉଦ୍ଦିଷ୍ଟ ଏବଂ ତାହା କେବଳ ଫଙ୍କ୍ସନ୍‌ଟି କାର୍ଯ୍ୟକ୍ଷମ ଥିବା ସମୟରେ ଉପଲବ୍ଧ ହୋଇଥାଏ।
4. ଗୋଟିଏ ପ୍ରୋଗ୍ରାମକୁ ନେଇ ବିଭିନ୍ନ ଫଙ୍କ୍ସନ୍‌ର ବିକାଶ ଓ ପରୀକ୍ଷଣ ସମାନ୍ତରାଳ ଭାବରେ କରାଯାଇପାରେ, ଯଦ୍ୱାରା ପ୍ରୋଗ୍ରାମ୍‌ର ମୋଟ ବିକାଶ ସମୟ ହ୍ରାସ କରିହୁଏ।

6.4 ଫଙ୍କ୍ସନ୍‌ର ପ୍ରକାରଭେଦ

ଫଙ୍କ୍ସନ୍‌ଗୁଡିକୁ ନିମ୍ନଲିଖିତ ଦୁଇଟି ବର୍ଗରେ ବିଭକ୍ତ କରାଯାଇଥାଏ:

- **ଲାଇବ୍ରେରି ଫଙ୍କ୍ସନ୍ (Library functions) –** ଏହି ଫଙ୍କ୍ସନ୍‌ଗୁଡ଼ିକ ହେଉଛି ପୂର୍ବ-ପ୍ରସ୍ତୁତ(built-in) ଓ ସଂକଳିତ।

ଏମାନଙ୍କର ମେସିନ୍ କୋଡ୍ (machine code) ସିଷ୍ଟମ୍ ଲାଇବ୍ରେରି (system library) ଫାଇଲଗୁଡ଼ିକରେ ଉପଲବ୍ଧ । ତୁମ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଦର୍ଶାଯାଇଥିବା ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକର ମେସିନ୍ କୋଡ୍, ଲିଙ୍କ୍ ପ୍ରକ୍ରିୟା ସମୟରେ ଲିଙ୍କର୍(linker)ଦ୍ୱାରା ଯୋଡାଯାଇଥାଏ ।

ଧ୍ୟାନ ଦିଅ ଯେ ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡିକ ସି ଭାଷାର ଅଂଶବିଶେଷ ନୁହେଁ; ସେଗୁଡିକ ବିକ୍ରେତାମାନଙ୍କଦ୍ୱାରା ପ୍ରଦାନ କରାଯାଏ ଯେଉଁମାନେ ବ୍ୟବହାରକାରୀଙ୍କ ସୁବିଧାପାଇଁ C ସଂକଳକ (C compiler) ବିକଶିତ କରିଥାନ୍ତି । ସେଗୁଡିକ ବ୍ୟବହାର କରିବାକୁ ତୁମକୁ ତୁମ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଉପଯୁକ୍ତ ହେଡର୍ ଫାଇଲ୍ ଅନ୍ତର୍ଭୁକ୍ତ କରିବା ଆବଶ୍ୟକ ।

- **ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଫଙ୍କ୍‌ସନ୍ (User-defined functions) –** ଏହି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ପ୍ରୋଗ୍ରାମର୍ ଦ୍ୱାରା ନିଜ ପ୍ରୋଗ୍ରାମ୍‌ର ଆବଶ୍ୟକତା ପୂରଣ କରିବାପାଇଁ ଲେଖାଯାଇଥାଏ । ଯଦି ସେଗୁଡିକ ପୂର୍ବରୁ ବିକଶିତ ହୋଇଛି, ତା'ହେଲେ ସେଗୁଡ଼ିକର ଉତ୍ସ କୋଡ୍ ଆମେ ଆମର ନୂତନ ପ୍ରୋଗ୍ରାମ୍‌ରେ ପୁନଃବ୍ୟବହାର କରିପାରିବା, ଏହାଦ୍ୱାରା ଅନେକ ସମୟ ଏବଂ ପ୍ରୟାସ ସଞ୍ଚୟ କରିପାରିବା ।

ଏହି ୟୁନିଟ୍‌ରେ, ଆମର ଆଲୋଚନା ମୁଖ୍ୟତଃ ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ଉପରେ ପର୍ଯ୍ୟବେଶିତ ।

6.5 ସ୍ଥାନୀୟ ଡାଟା ଏବଂ ଗ୍ଲୋବାଲ୍ ଡାଟା ବିଷୟକ ଧାରଣା (Concept of Local Data & Global Data)

ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟର କାୟା ମଧ୍ୟରେ ଘୋଷିତ ସ୍ୱତନ୍ତ୍ର ଚର (formal arguments) ଏବଂ ଚଳଗୁଡିକ ଫଙ୍କ୍‌ସନ୍‌ର ବାହାରେ ଅଜ୍ଞାତ ଅଟେ । କେଉଁ ଫଙ୍କ୍‌ସନ୍ ଏକ ଚଳକୁ ଚିହ୍ନିଥାଏ ଏବଂ କେଉଁଟି ଚିହ୍ନିନଥାଏ ତାହା ନିର୍ଣ୍ଣୟ କରୁଥିବା କାରକକୁ ଚଳ-ଦୃଶ୍ୟମାନତା(variable visibility) କୁହାଯାଏ । ଯେଉଁ ଫଙ୍କ୍‌ସନ୍‌ରେ ଏକ ସ୍ଥାନୀୟ ଚଳ ଘୋଷିତ ହୋଇଥାଏ, କେବଳ ସେହି ଫଙ୍କ୍‌ସନ୍‌ର କାୟା ଭିତରେ ଚଳଟି ଦୃଶ୍ୟମାନ ହୁଏ, ଅନ୍ୟଥିରେ ନୁହେଁ ।

ଅପରପକ୍ଷରେ, ଆରମ୍ଭରେ ଅର୍ଥାତ, ମୁଖ୍ୟ ଫଙ୍କ୍‌ସନ୍ ପୂର୍ବରୁ ଗ୍ଲୋବାଲ୍ ଚଳଗୁଡ଼ିକ ଘୋଷିତ ହୋଇଥାଏ । ଏହା ସବୁ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ମଧ୍ୟରେ ଦୃଶ୍ୟମାନ ହୁଏ ଏବଂ ପ୍ରୋଗ୍ରାମ୍ ସମାପ୍ତ ହେବା ପର୍ଯ୍ୟନ୍ତ ଏହା ବିଦ୍ୟମାନ ରହେ ।

6.6 ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଫଙ୍କ୍‌ସନ୍ (USER-DEFINED FUNCTIONS)

ଯଦିଓ ଉପଲବ୍ଧ ସମସ୍ତ ସି ସଂକଳକଗୁଡ଼ିକ (C Compilers) ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକର ଏକ ବିଶାଳ ସଂଗ୍ରହ ପ୍ରଦାନ କରେ, ତଥାପି ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ସଫ୍ଟୱେର୍ ସମାଧାନପାଇଁ ପ୍ରସ୍ତୁତ ପ୍ରୋଗ୍ରାମର ଆବଶ୍ୟକତା ଅନୁଯାୟୀ ସ୍ୱତନ୍ତ୍ର ଫଙ୍କ୍‌ସନ୍ ତିଆରିର ପ୍ରୟୋଜନ ଅଛି ।

ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକର ସମସ୍ତ ଦିଗର ମୁକାବିଲାପାଇଁ ସି ଭାଷା ବିଶଦ୍ ବିବରଣ ପ୍ରଦାନ କରେ ଯାହାକୁ ଆମେ ନିମ୍ନରେ ଆଲୋଚନା କରିବା ।

ସି ଭାଷାରେ ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ନିମ୍ନପ୍ରଦତ୍ତ ପାଞ୍ଚଟି ବର୍ଗରେ ବିଭକ୍ତ ହୋଇଥାଏ –

- ଯେଉଁ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ କୌଣସି ଚର ନେଇନଥାନ୍ତି କିମ୍ବା କୌଣସି ମୂଲ୍ୟ ଫେରସ୍ତ କରିନଥାନ୍ତି ।
- ଯେଉଁ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ଚର ନେଇଥାନ୍ତି କିନ୍ତୁ କୌଣସି ମୂଲ୍ୟ ଫେରସ୍ତ କରନ୍ତି ନାହିଁ ।
- ଯେଉଁ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ କୌଣସି ଚର ନେଇନଥାନ୍ତି କିନ୍ତୁ ଏକ ମୂଲ୍ୟ ଫେରସ୍ତ କରନ୍ତି ।
- ଯେଉଁ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ଚର ନେଇଥାନ୍ତି ଆଉ ଏକ ମୂଲ୍ୟ ମଧ୍ୟ ଫେରସ୍ତ କରନ୍ତି ।
- ଯେଉଁ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡ଼ିକ ଏକାଧିକ ମୂଲ୍ୟ ଫେରସ୍ତ କରନ୍ତି ।

6.7 ଫଙ୍କ୍‌ସନ୍‌ଗୁଡିକର ଘୋଷଣା ଏବଂ ବର୍ଣ୍ଣନା

ସି ଭାଷାର ପ୍ରତ୍ୟେକ ଚଳ ପରି, ପ୍ରତ୍ୟେକ ଫଙ୍କସନ୍ ମଧ୍ୟ ଘୋଷଣା ଏବଂ ବର୍ଣ୍ଣନା କରାଯିବା ଦରକାର।

6.7.1 ଗୋଟିଏ ଫଙ୍କସନ୍‌ର ଘୋଷଣା

ଫଙ୍କସନ୍ ଘୋଷଣା କେବଳ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ ହେଡର୍‌କୁ(function header) ନେଇ ଗଠିତ। ଏଥିରେ କୌଣସି କୋଡ୍ ନଥାଏ। ଫଙ୍କସନ୍ ହେଡର୍ ତିନୋଟି ଅଂଶକୁ ନେଇ ଗଠିତ। ଫେରସ୍ତ ପ୍ରକାର(returnType), ଫଙ୍କସନ୍‌ର ନାମ(functionName), ଏବଂ ସ୍ୱତନ୍ତ୍ର ଚର ସୂଚି(formal argument list)। ଏକ ସେମିକୋଲୋନ୍ ଫଙ୍କସନ୍ ହେଡର୍ ଘୋଷଣାର ଶେଷରେ ଦିଆଯାଇଥାଏ।

```
returnType function(formal argument list);
```

ଚିତ୍ର. 6.2: ଫଙ୍କସନ୍ ଘୋଷଣାର ବିଧି (Syntax)

ଫଙ୍କସନ୍ ଘୋଷଣାନାମା ଫଙ୍କସନ୍‌ର ସମ୍ପୂର୍ଣ୍ଣ ଚିତ୍ର ଦେଇଥାଏ ଯାହା ପରେ ବର୍ଣ୍ଣନା କରାଯିବା ଆବଶ୍ୟକ।

ଫଙ୍କସନ୍ ଘୋଷଣାନାମା ମଧ୍ୟ ଫଙ୍କସନ୍ ପ୍ରୋଟୋଟାଇପ୍ (prototype) ଭାବରେ ଜଣାଯାଏ।

ଯଦି ଫଙ୍କସନ୍‌ର କୌଣସି ଚର ନଥାଏ, ତେବେ ଆମେ ବନ୍ଧନି ମଧ୍ୟରେ void ଲେଖିବା। ଯଦି ଫଙ୍କସନ୍‌ର ଦୁଇ କିମ୍ବା ଅଧିକ ଚର ଥାଏ, ତାହେଲେ ପ୍ରତ୍ୟେକଟିକୁ କମାଦ୍ୱାରା ଅଲଗା ହୋଇଥାଏ।

ନିମ୍ନପ୍ରଦତ୍ତ ଫଙ୍କସନ୍ ଘୋଷଣା

```
int larget(int a, int b, int c);
```

ସି କମ୍ପାଇଲର୍‌କୁ କହିଥାଏ ଯେ ଏହି ଫଙ୍କସନ୍ ଏକ int ଫେରସ୍ତ କରିବ, ଫଙ୍କସନ୍‌ର ନାମ ହେଉଛି largest, ଏବଂ ଏହାର ତିନୋଟି ଚର ଅଛି, ଆଉ ସବୁଗୁଡ଼ିକ int ପ୍ରକାରର। ପରିବାହକଗୁଡିକର ନାମ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ନୁହେଁ।

ଏପରିକି ନିମ୍ନଲିଖିତ ଭାବରେ ମଧ୍ୟ ଫଙ୍କସନ୍ ଘୋଷଣା କରାଯାଇପାରିବ

```
int larget(int, int, int);
```

ଏହା ଉପରୋକ୍ତ ଘୋଷଣାର ବିକଳ୍ପ ଏବଂ ଏକା ଉଦ୍ଦେଶ୍ୟ ସାଧନ କରେ।

6.7.2 ଫଙ୍କସନ୍‌ର ବର୍ଣ୍ଣନା

ଫଙ୍କସନ୍‌ର ସଂଜ୍ଞା ଦୁଇଟି ଅଂଶକୁ ନେଇ ଗଠିତ। ଫଙ୍କସନ୍ ହେଡର୍ (Header) ଏବଂ ଫଙ୍କସନ୍ ବଡି (Body)।

Function Header

```
returnType functionName(formal argument list)
```

```
{
   // Local declarations
   . . .
   // statements
   . . .
}
```

Function Body

ଚିତ୍ର. 6.3: ଫଙ୍କସନ୍ ସଂଜ୍ଞା ଲେଖିବାର ବିଧି (Syntax)

ଫଙ୍କସନ୍ ହେଡର୍ (Header)

ଫଙ୍କସନ୍ ହେଡର୍ ତିନୋଟି ଅଂଶକୁ ନେଇ ଗଠିତ: ଫେରସ୍ତ ପ୍ରକାର, ଫଙ୍କସନ୍‌ର ନାମ, ଏବଂ ସ୍ୱତନ୍ତ୍ର ଚର ସୂଚି । ଫଙ୍କସନ୍ ହେଡର୍ ଶେଷରେ ଏକ ସେମିକୋଲୋନ୍ ବ୍ୟବହୃତ ହୁଏ ନାହିଁ ।

ଫଙ୍କସନ୍‌ରୁ କିଛି ଫେରସ୍ତ ହେବାକୁ ନ ଥିଲେ, ଫେରସ୍ତ ପ୍ରକାରପାଇଁ void ବ୍ୟବହୃତ ହୁଏ ।

ସ୍ୱତନ୍ତ୍ର ଚର ସୂଚି, ଚଳଗୁଡ଼ିକର ଘୋଷଣା କରେ ଯାହା କଲିଂ (calling) ଫଙ୍କସନ୍‌ରୁ ତଥ୍ୟ ଗ୍ରହଣ କରେ ଏବଂ ଏଥିରେ ଏପରି ଚଳ ମଧ୍ୟ ଥାଇପାରେ ଯାହା ପୁଣି କଲିଂ ଫଙ୍କସନ୍‌କୁ ଡାଟା ଫେରାଇବାପାଇଁ ବ୍ୟବହୃତ ହୋଇପାରିବ ।

ସେମିକଲୋନ୍ ବ୍ୟତୀତ ଫଙ୍କସନ୍ ପ୍ରୋଟୋଟାଇପ୍ ଏବଂ ଫଙ୍କସନ୍ ହେଡର୍ ନିଶ୍ଚିତ ଭାବରେ ମେଳ ହେବ ।

ଫଙ୍କସନ୍ କାୟା (Body)

ଫଙ୍କସନ୍‌ର କାୟାରେ ସ୍ଥାନୀୟ ଘୋଷଣାନାମା ଏବଂ ଅନୁଦେଶଗୁଡ଼ିକ ରହିଥାଏ । ଏହା ଫଙ୍କସନ୍‌ଦ୍ୱାରା ସମ୍ପନ୍ନ ହେବାକୁ ଥିବା କାର୍ଯ୍ୟର ବ୍ୟାଖ୍ୟା କରେ । ଫଙ୍କସନ୍‌ର କାୟା ସ୍ଥାନୀୟ ଘୋଷଣାନାମାରୁ ଆରମ୍ଭ ହୁଏ । ଏଥିରେ ପ୍ରଥମେ ଫଙ୍କସନ୍‌ପାଇଁ ଆବଶ୍ୟକ ହେଉଥିବା ଚଳଗୁଡ଼ିକୁ ଘୋଷଣା କରାଯାଏ । ତା'ପରେ ଅନୁଦେଶଗୁଡ଼ିକ ଲେଖାଯାଏ ।

ବର୍ତ୍ତମାନ କିଛି ସରଳ କାର୍ଯ୍ୟ କରିବାପାଇଁ ଫଙ୍କସନ୍‌ର ଘୋଷଣା ଏବଂ ସଂଜ୍ଞା ଲେଖିବାର ଉଦାହରଣ ନେବା ।

ଉଦାହରଣ 6.1: maximum ନାମକ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟର ଘୋଷଣା ଓ ବର୍ଣ୍ଣନାକର ଯାହା ତିନୋଟି ଚର ନିଏ, ପ୍ରତ୍ୟେକ int ପ୍ରକାରର ସଂଖ୍ୟା, ଏବଂ ସେଗୁଡ଼ିକ ମଧ୍ୟରୁ ବୃହତ୍ତମ ସଂଖ୍ୟାକୁ ଫେରସ୍ତ କରେ ।

ଘୋଷଣାନାମା (Declaration):

```
int maximum(int x, int y, int z);
```

ସଂଜ୍ଞା (Definition):

```
int maximum(int x, int y, int z)
 {
     if ( ( x > y ) && ( x > z ) )
        return x;
     else if ( y > z )
        return y;
     else
        return z;
 }
```

ଉପରୋକ୍ତ ଫଙ୍କ୍ସନ୍‌ର ତର୍କକୁ (logic) ନିମ୍ନମତେ ମଧ୍ୟ ଲେଖାଯାଇପାରିବ

```
int maximum(int x, int y, int z)
 {
     int big;
     big = x;
     if ( y > big )
        big  = y;
     if ( z > big )
        big  = z;
     return big;
 }
```

ଉଦାହରଣ 6.2: isEven ନାମକ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟର ଘୋଷଣା ଏବଂ ବର୍ଣ୍ଣନା ଲେଖ ଯାହାକି ଏକ int ପ୍ରକାରର ଚର n ନେଇଥାଏ, ଏବଂ ଯଦି n ଏକ ଯୁଗ୍ମ ସଂଖ୍ୟା ହୋଇଥାଏ ତାହେଲେ 1 ଫେରସ୍ତ କରିବ ଅନ୍ୟଥା 0 ଫେରସ୍ତ କରିବ।

ଘୋଷଣାନାମା (Declaration):

```
int isEven(int n)
 {
     if ( n % 2 == 0 )
        return 1;
     else
        return 0;
 }
```

ଉପରୋକ୍ତ ଫଙ୍କ୍ସନ୍‌ର ତର୍କ (logic) ନିମ୍ନମତେ ମଧ୍ୟ ଲେଖାଯାଇପାରିବ

```
int isEven(int n)
 {
    return ( n % 2 ? 1 : 0 );
 }
```

ଉଦାହରଣ 6.3: SumOfDigits ନାମକ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ ଘୋଷଣା ଏବଂ ବ୍ୟାଖ୍ୟା କର ଯାହା ଏକ int ଚର n ନେଇଥିଏ, ଏବଂ n ସଂଖ୍ୟାର ଅଙ୍କଗୁଡିକର ସମଷ୍ଟି ଫେରସ୍ତ କରିପାରୁଥିବ।

ଘୋଷଣାନାମା (Declaration):

```
int sumOfDigits(int n);
```

ସଂଜ୍ଞା (Definition):

```
int sumOfDigits(int n)
 {
     int i, s = 0, d ;
     while ( n > 0 )
     {
        d = n % 10;
        s = s + d;
        n = n / 10;
     }
     return s;
 }
```

6.8 ଫଙ୍କ୍‌ସନ୍ କଲ୍ କରିବା/ଡାକିବା (CALLING A FUNCTION)

ଲାଇବ୍ରେରି ଫଙ୍କ୍‌ସନ୍‌ଗୁଡିକ ପରି, ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଫଙ୍କ୍‌ସନ୍‌ଗୁଡିକ କେବଳ ଏହାର ନାମଦ୍ୱାରା କୌଣସି ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ ମଧ୍ୟରୁ ଡକାଯାଏ । ଯଦି ବାସ୍ତବ ଚର (actual argument) ଥାଏ, ତାହେଲେ ବନ୍ଧନୀ ମଧ୍ୟରେ ଲେଖାଯାଏ ।

ଫଙ୍କ୍‌ସନ୍‌କୁ ପଠାଯିବାକୁ କୌଣସି ବାସ୍ତବ ଚର ନ ଥିଲେ ମଧ ଫଙ୍କ୍‌ସନ୍‌ର ନାମ ପରେ ନିଶ୍ଚିତ ଭାବେ ବନ୍ଧନୀ ଦେବା ଆବଶ୍ୟକ ।

ଯଦି ବାସ୍ତବ ଚର ଥାଏ, ତେବେ ଏହାର ସଂଖ୍ୟା, ପ୍ରକାର, ଏବଂ କ୍ରମ ସ୍ୱତନ୍ତ୍ର ଚର ଅନୁରୂପ ହେବା ଆବଶ୍ୟକ ।

ଯେତେବେଳେ ପ୍ରୋଗ୍ରାମ୍ ନିୟନ୍ତ୍ରଣ ଫଙ୍କ୍‌ସନ୍‌ର ନାମ ପାଖକୁ ଆସେ, ସେତେବେଳେ ଏହି ନିୟନ୍ତ୍ରଣ, କଲ୍ ହୋଇଥିବା ଫଙ୍କ୍‌ସନ୍‌କୁ ସ୍ଥାନାନ୍ତରିତ ହୁଏ । ତା'ପରେ ବାସ୍ତବ ଚର, ସ୍ୱତନ୍ତ୍ର ଚରର ସ୍ଥାନ ନିଏ ଏବଂ ଫଙ୍କ୍‌ସନ୍‌ର କାର୍ଯ୍ୟ ଚାଲୁ ରହେ । ଯେତେବେଳେ ଫେରସ୍ତ ବିବୃତ୍ତି(return statement) କାର୍ଯ୍ୟକାରୀ ହୁଏ କିମ୍ବା ଶେଷ ବିବୃତ୍ତି(last statement) ଏହାର କାର୍ଯ୍ୟ ସାରିଦିଏ, ସେତେବେଳେ ନିୟନ୍ତ୍ରଣ ପୁଣି କଲିଂ ଫଙ୍କ୍‌ସନ୍‌କୁ ଫେରିଯାଏ ।

ଯଦି ଫଙ୍କ୍‌ସନ୍ କୌଣସି ମୂଲ୍ୟ ଫେରସ୍ତ କରୁନାହିଁ, ତେବେ ଫଙ୍କ୍‌ସନ୍ କଲ୍‌ଟି ଏକ ସ୍ୱାଧିନ (standalone) ଅନୁଦେଶ ଭାବେ ପରିଦୃଶ୍ୟ ହେବ ।

ଯଦି ଫଙ୍କ୍‌ସନ୍ ଏକ ମୂଲ୍ୟ ଫେରସ୍ତ କରୁଛି, ତେବେ ଫଙ୍କ୍‌ସନ୍ କଲ୍‌ଟି ନିୟତନ ବିବୃତ୍ତିରେ (assignment statement), ବା ଗାଣିତିକ ଅଭିବ୍ୟକ୍ତିରେ କିମ୍ବା ସର୍ତ୍ତଭିତ୍ତିକ ବିବୃତ୍ତିରେ କିମ୍ବା printf() ଫଙ୍କ୍‌ସନ୍‌ରେ ମଧ ଲେଖାଯାଇପାରେ ।

ଉଦାହରଣ 6.1 ରୁ 6.3 ପର୍ଯ୍ୟନ୍ତ ବ୍ୟାଖ୍ୟା ହୋଇଥିବା ଫଙ୍କ୍‌ସନ୍‌ଗୁଡିକ, ନିମ୍ନପ୍ରଦତ୍ତ ଉପାୟଗୁଡ଼ିକରେ ଡକା କରାଯାଇପାରେ ।

```
big = maximum(a, b, c);
 if ( isEven(n) == 1 )
    printf( "\n%d is an Even number.\n" );
 else
    printf( "\n%d is an Odd number.\n" );
 printf( "\nSum of digits of %d = %d\n", n, sumOfDigits(n) );
```

6.9 ଫେରସ୍ତ ବିବୃତ୍ତି (THE return STATEMENT)

ଫେରସ୍ତ ବିବୃତ୍ତିର ସିଣ୍ଟାକ୍ସ (syntax) ହେଉଛି

```
return;                    or          return (exp);
```

ଯେଉଁଠାରେ exp ଏକ ସ୍ଥିର, ଚଳ କିମ୍ବା ଅଭିବ୍ୟକ୍ତି ହୋଇପାରେ । exp କୁ ବନ୍ଧନୀଯୁକ୍ତାତ୍ମକ କରିବା ଇଚ୍ଛାଧୀନ ଅଟେ । ଫଙ୍କ୍ସନ୍‌ର ଫେରସ୍ତ ପ୍ରକାର void ଥିଲେ ପ୍ରଥମ ଫେରସ୍ତ ପ୍ରକାର ବିବୃତ୍ତି ବ୍ୟବହାର ହୁଏ ଅନ୍ୟଥା ଦ୍ୱିତୀୟ ବିବୃତ୍ତି ବ୍ୟବହାର ହୁଏ ।

ଫେରସ୍ତ ବିବୃତ୍ତି ଦୁଇଟି ଉଦ୍ଦେଶ୍ୟ ପୂରଣ କରେ । ଯଥା –

- ଫେରସ୍ତ ବିବୃତ୍ତିର କାର୍ଯ୍ୟକାରିତା ଫଙ୍କ୍ସନ୍‌ର କାର୍ଯ୍ୟକାରିତାକୁ ସମାପ୍ତ କରେ ଏବଂ ଫଙ୍କ୍ସନ୍‌ରୁ କଲିଂ (calling) ଫଙ୍କ୍ସନ୍‌କୁ ପ୍ରୋଗ୍ରାମ୍ ନିୟନ୍ତ୍ରଣ ସ୍ଥାନାନ୍ତର କରେ ।
- ଫେରସ୍ତ ବିବୃତ୍ତିରୁ *exp*, କଲିଂ (calling) ଫଙ୍କ୍ସନ୍‌କୁ ଏକ ମୂଲ୍ୟ ଭାବରେ ଫେରସ୍ତ ହୁଏ ।
- ଫେରସ୍ତ ବିବୃତ୍ତି ଫଙ୍କ୍ସନ୍‌ର ଶେଷ ବିବୃତ୍ତି ନ ହୋଇପାରେ । ଫଙ୍କ୍ସନ୍‌ର ଯେକୌଣସି ସ୍ଥାନରେ ଏହା ବ୍ୟବହାର କରାଯାଇପାରେ । ଏହା କାର୍ଯ୍ୟକାରୀ ହେବା ମାତ୍ରେ, ନିୟନ୍ତ୍ରଣ କଲିଂ (calling) ଫଙ୍କ୍ସନ୍‌କୁ ଫେରିବ । ଆହୁରି ମଧ୍ୟ, ଗୋଟିଏ ଫଙ୍କ୍ସନ୍‌ରେ ଏକାଧିକ ଫେରସ୍ତ ବିବୃତ୍ତି ରହିପାରେ ।

ଫେରସ୍ତ ବିବୃତ୍ତିର ଏକ ପ୍ରମୁଖ ସୀମିତତା ଅଛି - ଏହା କେବଳ ଗୋଟିଏ ମୂଲ୍ୟହିଁ ଫେରସ୍ତ କରିପାରିବ । ଯଦି ଫଙ୍କ୍ସନ୍‌ରୁ କଲିଂ ଫଙ୍କ୍ସନ୍‌କୁ ଏକାଧିକ ମୂଲ୍ୟ ଫେରସ୍ତ କରିବାକୁ ଚାହୁଁଛ, ତେବେ ତୁମକୁ ଅନ୍ୟ ଉପାୟ ବ୍ୟବହାର କରିବାକୁ ପଡିବ ।

6.10 ଫଙ୍କ୍ସନ୍‌କୁ ଚର ପ୍ରେରଣ (PASSING ARGUMENTS TO A FUNCTION)

ଚର ସୂଚି ମାଧ୍ୟମରେ ଫଙ୍କ୍ସନ୍‌କୁ ତଥ୍ୟ ପ୍ରେରଣ କରାଯାଏ, ଯେଉଁଠାରେ ପ୍ରତ୍ୟେକ ଚରକୁ ବାସ୍ତବ ଚର କୁହାଯାଏ । ଫଙ୍କ୍ସନ୍‌ର ନାମ ପରେ ଏହି ଚରଗୁଡିକ ବନ୍ଧନୀ ମଧ୍ୟରେ ଆବଦ୍ଧ ହୋଇଥାଏ । ଫଙ୍କସନ୍ ସଂଜ୍ଞାରେ ଲେଖାଥିବା ସ୍ୱତନ୍ତ୍ର ଚର ସହିତ ବାସ୍ତବ ଚରର ସଂଖ୍ୟା, ପ୍ରକାର, ଏବଂ କ୍ରମ ଅନୁରୂପ ହେବା ଆବଶ୍ୟକ । ବାସ୍ତବ ଚର, ସ୍ଥିରଚଳ, ଆରେର ନାମ, କିମ୍ବା ଅଭିବ୍ୟକ୍ତି ହୋଇପାରେ ।

ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟକୁ ଚର ପ୍ରେରଣପାଇଁ ଦୁଇଟି ଉପାୟ ଅଛି:

- ମୂଲ୍ୟଦ୍ୱାରା ଡାକିବା (Call by value)
- ରେଫରେନ୍‌ସଦ୍ୱାରା ଡାକିବା (Call by reference)

ଆସ ଏହାକୁ ଗୋଟିଏ ପରେ ଗୋଟିଏ ବର୍ଣ୍ଣନା କରିବା ।

6.10.1 ମୂଲ୍ୟଦ୍ୱାରା ଡାକିବା (Call by value)

ଏହି ପଦ୍ଧତିରେ, ବାସ୍ତବ ଚରଗୁଡିକ ଫଙ୍କ୍ସନ୍‌କୁ ଡାକିବାରେ ବ୍ୟବହୃତ ହୁଏ । ବାସ୍ତବ ଚର, କୌଣସି ଚଳ, ସ୍ଥିର, କିମ୍ବା ଅଭିବ୍ୟକ୍ତି ମଧ୍ୟ ହୋଇପାରେ ।

ଫଙ୍କ୍ସନ୍‌କୁ ଡକାଯିବା ସମୟରେ, ବାସ୍ତବ ଚରର ମୂଲ୍ୟଗୁଡିକ ସ୍ୱତନ୍ତ୍ର ଚରଗୁଡିକର ଅନୁରୂପ ବିକଳ୍ପ ହୋଇଥାଏ, ଏବଂ ତା'ପରେ ନିୟନ୍ତ୍ରଣଟି ଫଙ୍କ୍ସନ୍‌କୁ ସ୍ଥାନାନ୍ତରିତ ହୁଏ ।

ଫଙ୍କ୍ସନ୍‌ରେ ସ୍ଥାନୀୟ ଚଳଗୁଡ଼ିକୁ ତିଆରି ହୁଏ ଏବଂ ଏହା ପରେ ଫଙ୍କ୍ସନ୍‌ର କାର୍ଯ୍ୟ ବ୍ୟାଖ୍ୟା କରୁଥିବା ବିବୃତ୍ତିଗୁଡିକ କାର୍ଯ୍ୟକାରୀ ହୁଏ । ଯଦି ଡକା ହୋଇଥିବା ଫଙ୍କ୍ସନ୍‌ରୁ ଏକ ମୂଲ୍ୟ ଫେରସ୍ତ କରିବାର ଅଛି, ତେବେ ଏହା ଫେରସ୍ତ ବିବୃତ୍ତି ମାଧ୍ୟମରେ ଫେରସ୍ତ ହୋଇଥାଏ ।

ମୂଲ୍ୟଦ୍ୱାରା ଡାକିବା ପଦ୍ଧତି ବ୍ୟବହାର କରି ଚର ପ୍ରେରଣ ବିଷୟରେ ନିମ୍ନପ୍ରଦତ୍ତ ଉଲ୍ଲେଖଯୋଗ୍ୟ ବିନ୍ଦୁଗୁଡ଼ିକ:

1. ବାସ୍ତବ ଚର, ସ୍ଥିରଚଳ, କିମ୍ବା ଅଭିବ୍ୟକ୍ତି ହୋଇପାରେ ।
2. ଯେତେବେଳେ ନିୟନ୍ତ୍ରଣ କଲିଂ ଫଙ୍କସନ୍‌ରୁ କଲ୍‌ଡ ଫଙ୍କସନ୍‌କୁ ସ୍ଥାନାନ୍ତରିତ ହୁଏ, ସ୍ଥାନୀୟ ଚଳପାଇଁ ମେମୋରି (memory) ବଣ୍ଟନ ହୁଏ, ଏବଂ ତା'ପରେ ଫଙ୍କସନ୍ ମଧ୍ୟରେ ଥିବା ବିବୃତ୍ତିଗୁଡିକ କାର୍ଯ୍ୟକାରୀ ହୁଏ ।
3. ଡକା ହୋଇଥିବା ଫଙ୍କସନ୍ ଏହାର କାର୍ଯ୍ୟକାରିତା ସମାପ୍ତ କରିବା ମାତ୍ରେ, ଏହାର ସ୍ଥାନୀୟ ଚଳପାଇଁ ଆବଣ୍ଟିତ ମେମୋରି ବଣ୍ଟନ-ମୁକ୍ତ ହୁଏ, ଏବଂ ଶେଷରେ ନିୟନ୍ତ୍ରଣଟି ଡାକିଥିବା ଫଙ୍କସନ୍‌କୁ ସ୍ଥାନାନ୍ତରିତ ହୋଇଥାଏ ।
4. ସ୍ୱତନ୍ତ୍ର ଚରରେ କରାଯାଇଥିବା କୌଣସି ପରିବର୍ତ୍ତନ ବାସ୍ତବ ଚର ଉପରେ କୌଣସି ପ୍ରଭାବ ପକାଏ ନାହିଁ, କାରଣ ଫଙ୍କସନ୍ କେବଳ ଚରର ସ୍ଥାନୀୟ ପ୍ରତିରୂପ ବ୍ୟବହାର କରିଥାଏ ।

ନିମ୍ନପ୍ରଦତ୍ତ ଫଙ୍କସନ୍‌ର ସଂଜ୍ଞା, ମୂଲ୍ୟଦ୍ୱାରା ଚର ପାରିତ କରିବାର ପ୍ରଣାଳୀ ଦର୍ଶାଇଥାଏ ।

```
void swap( int a, int b )
 {
    int temp;
    temp = a;
    a = b;
    b = temp;
 }
```

ଉପରୋକ୍ତ ଫଙ୍କସନ୍‌କୁ ନିମ୍ନମତେ କଲିଂ(calling) କରାଯିବ

```
swap(x,y);
```

ଯେଉଁଠାରେ x ଏବଂ y କଲିଂ(calling) ଫଙ୍କସନ୍‌ରେ ବାସ୍ତବ ଚର ଅଟେ ।

Listing 6.1

```
/*
     Program to illustrate the passing of arguments by value.
     It calls a function swap() that swaps/interchanges
     values of arguments.
 */
 #include <stdio.h>
 void swap(int a, int b);  /* function prototype */
 int main()
 {
    int x, y;
    printf("\nEnter value for x : ");
    scanf("%d", &x);
    printf("\nEnter value for y : ");
    scanf("%d", &y);
    printf("\nBefore calling swap function\n");
```

```
    printf("\nValue of x = %d, Value of y = %d\n", x, y);
    swap(x,y);  /* function call   */
    printf("\nAfter returning from swap function\n");
    printf("\nValue of x = %d, Value of y = %d\n", x, y);

    return 0;
 }
 void swap( int a, int b )
 {
    int temp;
    printf("\nValues received from the main function\n");
    printf("\nValue of a = %d, Value of b = %d\n", a, b);
    temp  = a;
    a = b;
    b = temp;
    printf("\nValues of local copy after swapping\n");
    printf("\nValue of a = %d, Value of b = %d\n", a, b);

 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter value for x : 10
Enter value for y : 12
Before calling swap function
Value of x = 10, Value of y = 12
Values received from the main function
Value of a = 10, Value of b = 12
Values of local copy after swapping
Value of a = 12, Value of b = 10
After returning from swap function
Value of x = 10, Value of y = 12
```

ତୁମେ ନିଶ୍ଚୟ ଲକ୍ଷ୍ୟ କରିଥିବ ଯେ, ସ୍ୱତନ୍ତ୍ର ଚରରେ ହୋଇଥିବା କୌଣସି ପରିବର୍ତ୍ତନ ବାସ୍ତବ ଚର ଉପରେ ପ୍ରଭାବ ପକାଉ ନାହିଁ।

6.10.2 ରେଫରେନ୍ସଦ୍ୱାରା ଡାକିବା (Call by reference)

ଏହି ପଦ୍ଧତିରେ, ଫଙ୍କ୍ସନ୍ କଲ୍‌ରେ ବାସ୍ତବ ଚରର ଠିକଣା ବ୍ୟବହୃତ ହୁଏ। ବାସ୍ତବ ଚର କେବଳ ଚଳ ହୋଇପାରେ।

ସ୍ୱତନ୍ତ୍ର ଚରକୁ ସୂଚକ(pointers) ଭାବରେ ଘୋଷିତ କରାଯାଏ। ସୂଚକର ପ୍ରକାର ବାସ୍ତବ ଚରର ପ୍ରକାର ସହିତ ମେଳ ଖାଉଥିବା ଆବଶ୍ୟକ।

ଫଙ୍କ୍ସନ୍‌କୁ ଡକାହେଲେ, ବାସ୍ତବ ଚରର ଠିକଣାଗୁଡିକ ଅନୁରୂପ ସ୍ୱତନ୍ତ୍ର ଚରଗୁଡିକୁ ସ୍ଥାନାନ୍ତର ହୋଇଯାଏ, ଯାହାକି ସୂଚକ ଅଟେ, ଏବଂ ତାପରେ ନିୟନ୍ତ୍ରଣ ଫଙ୍କ୍ସନ୍‌କୁ ସ୍ଥାନାନ୍ତରିତ ହୁଏ ।

ଫଙ୍କ୍ସନ୍‌ର ସ୍ଥାନୀୟ ଚଳ ତିଆରି ହୁଏ ଏବଂ ଏହାପରେ ଫଙ୍କ୍ସନ୍‌ର କାର୍ଯ୍ୟକୁ ବ୍ୟାଖ୍ୟା କରୁଥିବା ବିବୃତ୍ତିଗୁଡିକ କାର୍ଯ୍ୟକାରୀ ହୋଇଥାଏ ।

ଡକା ହୋଇଥିବା ଫଙ୍କ୍ସନ୍ ମୂଲ୍ୟ ଫେରସ୍ତ କରିବାର ଥିଲେ, ଏହା ଫେରସ୍ତ ବିବୃତ୍ତି ମାଧ୍ୟମରେ ଫେରସ୍ତ ହୋଇଥାଏ ।

ରେଫରେନ୍ସଦ୍ୱାରା ଡାକିବା ପଦ୍ଧତି ବ୍ୟବହାର କରି ଚର ପ୍ରେରଣ ବିଷୟରେ ନିମ୍ନପ୍ରଦତ୍ତ ଉଲ୍ଲେଖଯୋଗ୍ୟ ବିନ୍ଦୁଗୁଡ଼ିକ:

1. ବାସ୍ତବ ଚର କେବଳ ଚଳ ହୋଇପାରେ ।
2. ଡାକିଥିବା ଫଙ୍କ୍ସନ୍‌ରୁ ଡକାହୋଇଥିବା ଫଙ୍କ୍ସନ୍‌କୁ ନିୟନ୍ତ୍ରଣ ସ୍ଥାନାନ୍ତରିତ ହେବାପରେ, ସ୍ଥାନୀୟ ଚଳପାଇଁ ମେମୋରି (memory) ଆବଣ୍ଟିତ ହୁଏ, ଏବଂ ଫଙ୍କ୍ସନ୍ ମଧ୍ୟରେ ଥିବା ବିବୃତ୍ତିଗୁଡିକ କାର୍ଯ୍ୟକାରୀ ହୁଏ ।
3. ଡକା ହୋଇଥିବା ଫଙ୍କ୍ସନ୍‌ର କାର୍ଯ୍ୟ ସମାପ୍ତ ହେବା ମାତ୍ରେ, ଏହାର ସ୍ଥାନୀୟ ଚଳପାଇଁ ଆବଣ୍ଟିତ ମେମୋରି (memory) ଆବଣ୍ଟନ-ମୁକ୍ତ ହୋଇଥାଏ, ଏବଂ ଶେଷରେ ନିୟନ୍ତ୍ରଣଟି ଡାକିଥିବା ଫଙ୍କ୍ସନ୍‌କୁ ସ୍ଥାନାନ୍ତରଣ ହୋଇଥାଏ ।
4. ସ୍ୱତନ୍ତ୍ର ଚରରେ କରାଯାଇଥିବା ଯେକୌଣସି ପରିବର୍ତ୍ତନ ବାସ୍ତବ ଚର ଉପରେ ତୁରନ୍ତ ପ୍ରଭାବ ପକାଏ, କାରଣ ଫଙ୍କ୍ସନ୍‌ଟି ପ୍ରକୃତରେ ସୂଚକ ମାଧ୍ୟମରେ ବାସ୍ତବ ଚର ଉପରେ ହିଁ କାର୍ଯ୍ୟ କରେ ।

ନିମ୍ନଲିଖିତ ଫଙ୍କ୍ସନ୍‌ଟି, ରେଫରେନ୍ସ (ଠିକଣା) ଦ୍ୱାରା ଚର ପାରିତ କରିବାର ପ୍ରଣାଳୀ ଦର୍ଶାଉଛି ।

```
void swap( int *a, int *b )
 {
    int temp;
    temp  = *a;
    *a = *b;
    *b = temp;
 }
```

ଉପରୋକ୍ତ ଫଙ୍କ୍ସନ୍‌କୁ ଏହି ପ୍ରକାରରେ ଡାକି ହେବ ।

```
swap(&x,&y);
```

ଯେଉଁଠାରେ x ଏବଂ y କଲିଂ ଫଙ୍କ୍ସନ୍‌ରେ ବାସ୍ତବ ଚର ଅଟେ ।

ତାଲିକା (Listing) 6.2

```
/*
     Program to illustrate the passing of arguments by reference.
     It calls a function swap() that interchanges values of arguments.
 */

 #include <stdio.h>
 void swap(int *a, int *b);  /* function prototype */

 int main()
 {
    int x, y;
    printf("\nEnter value for x : ");
    scanf("%d", &x);
    printf("\nEnter value for y : ");
    scanf("%d", &y);
    printf("\nBefore calling swap function\n");
    printf("\nValue of x = %d, Value of y = %d\n", x, y);
    swap(&x,&y);  /* function call   */
    printf("\nAfter returning from swap function\n");
    printf("\nValue of x = %d, Value of y = %d\n", x, y);
    return 0;
 }
 void swap( int *a, int *b )
 {
    int temp;
    printf("\nValues received from the main function\n");
    printf("\nValue of *a = %d, Value of *b = %d\n", *a, *b);
    temp  = *a;
    *a = *b;
    *b = temp;
    printf("\nValues of local copy after swapping\n");
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter value for x : 10
Enter value for y : 12
Before calling swap function
Value of x = 10, Value of y = 12
Values received from the main function
Value of *a = 10, Value of *b = 12
Values of local copy after swapping
Value of *a = 12, Value of *b = 10
After returning from swap function
Value of x = 12, Value of y = 10
```

ତୁମେ ଲକ୍ଷ୍ୟ କରିଥିବ ଯେ ସ୍ୱତନ୍ତ୍ର ଚରରେ କରାଯାଇଥିବା ଯେକୌଣସି ପରିବର୍ତ୍ତନ ବାସ୍ତବ ଚରରେ ପ୍ରତିଫଳିତ ହେଉଛି।

ପରବର୍ତ୍ତୀ ପ୍ରୋଗ୍ରାମ୍ ହେଉଛି ଅନ୍ୟ ଏକ ଉଦାହରଣ ଯେଉଁଥିରେ ରେଫରେନ୍ସଦ୍ୱାରା ଫଙ୍କସନ୍‌କୁ ଡାକିବା ପଦ୍ଧତି ବ୍ୟବହାର କରି ଚରଗୁଡିକ ପାରିତ କରାଯାଇଥାଏ।

ତାଲିକା (Listing) 6.3

```
/* Program that calls function example() to find the average of
    two numbers, largest & smallest of these numbers. The
    average is retuned via return statement while largest and
    smallest numbers are returned via formal arguments
 */
 #include <stdio.h>
 /* function prototype  */
 float example(float x, float y, float *big, float *small);
 int main()
 {
    float x, y, avg, larger, smaller;
    printf("\nEnter value for x : ");
    scanf("%f", &x);
    printf("\nEnter value for y : ");
    scanf("%f", &y);
    avg = example(x,y,&larger,&smaller); /*  function call  */
    printf("\nAverage of %.2f and %.2f is %.2f\n", x, y, avg);
    printf("\nLarger number is %.2f\n", larger);
    printf("\nSmaller number is %.2f\n", smaller);
    return 0;
 }
 /*
    Function to compute the average of two numbers and find the
    largest and smallest of these numbers
 */
 float example(float x, float y, float *big, float *small)
 {
     float average;
     average = ( x + y ) / 2;
     if ( x > y ) {
        *big = x;
        *small = y;
     } else {
        *big = y;
        *small = x;
     }
     return average;
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter value for x : 10.5
Enter value for y : 12.5
Average of 10.50 and 12.50 is 11.50
Larger number is 12.50
Smaller number is 10.50
```

6.10.3 ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ ଏବଂ ରେଫରେନ୍ସ ଦ୍ୱାରା କଲ୍ ମଧ୍ୟରେ ତୁଳନା

ଟେବୁଲ୍ 6.1 ପ୍ରୋଟୋଟାଇପ୍ (prototype), ସଂଜ୍ଞା ଏବଂ ଫଙ୍କ୍ସନ୍ କଲ୍ ସମ୍ବନ୍ଧରେ ପାର୍ଥକ୍ୟର ମୁଖ୍ୟ ବିନ୍ଦୁଗୁଡିକୁ ଦର୍ଶାଉଛି ।

ଟେବୁଲ୍ 6.1: ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ ଏବଂ ରେଫରେନ୍ସଦ୍ୱାରା କଲ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଅଂଶ-1

ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ (Call by Value)	ରେଫରେନ୍ସଦ୍ୱାରା କଲ୍ (Call by Reference)
Function Prototype: `void swap(int a, int b);`	Function Prototype: `void swap(int *a, int *b);`

Function Definition:
`void swap(int a, int b)`
`{`
`int temp;`
`temp = a;`
`a = b;`
`b = temp;`
`}`
	Function Definition:
	`void swap(int *a, int *b)`
	`{`
	`int temp;`
	`temp = *a;`
	`*a = *b;`
	`*b = temp;`
	`}`

Function Call: `swap(x, y);`	Function Call: `swap(&x, &y);`

ଟେବୁଲ୍ 6.2 ବାସ୍ତବ ଚରଗୁଡିକର ପ୍ରକୃତି, ସ୍ୱତନ୍ତ୍ର ଚରଗୁଡିକର ପ୍ରକୃତି ଏବଂ ବାସ୍ତବ ଚର ଉପରେ ସ୍ୱତନ୍ତ୍ର ଚରର ପରିବର୍ତ୍ତନର କୌଣସି ପ୍ରଭାବ ବିଷୟରେ ପାର୍ଥକ୍ୟର ମୁଖ୍ୟ ବିନ୍ଦୁଗୁଡ଼ିକୁ ଦର୍ଶାଉଛି ।

ଟେବୁଲ୍ 6.2: ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ ଏବଂ ରେଫରେନ୍ସଦ୍ୱାରା କଲ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ଅଂଶ-2

ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ (Call by Value)	ରେଫରେନ୍ସଦ୍ୱାରା କଲ୍ (Call by Reference)
ବାସ୍ତବ ଚର ସ୍ଥିର, ଚଳ, କିମ୍ବା ଅଭିବ୍ୟକ୍ତି ହୋଇପାରେ ।	ବାସ୍ତବ ଚର କେବଳ ଚଳ ହୋଇପାରେ ।
ସ୍ୱତନ୍ତ୍ର ଚର ହେଉଛି ସାଧାରଣ ଚଳ ।	ସ୍ୱତନ୍ତ୍ର ଚରଗୁଡ଼ିକ ହେଉଛି ସୂଚକ ଚଳ ।
ବାସ୍ତବ ଚରର ମୂଲ୍ୟଗୁଡ଼ିକ ସ୍ୱତନ୍ତ୍ର ଚରରେ ପ୍ରତିସ୍ଥାପିତ ହୁଏ ।	ବାସ୍ତବ ଚରର ଠିକଣାଗୁଡ଼ିକ ସ୍ୱତନ୍ତ୍ର ଚରରେ ପ୍ରତିସ୍ଥାପିତ ହୁଏ ।
ସ୍ୱତନ୍ତ୍ର ଚରରେ କରାଯାଇଥିବା କୌଣସି ପରିବର୍ତ୍ତନ ବାସ୍ତବ ଚର ଉପରେ କୌଣସି ପ୍ରଭାବ ପକାଇବ ନାହିଁ, କାରଣ ଫଙ୍କ୍ସନ୍‌ରେ ଚରର କେବଳ ସ୍ଥାନୀୟ ପ୍ରତିରୂପ ବ୍ୟବହାର ହେବ ।	ସୂଚକ ଚଳରେ କରାଯାଇଥିବା ଯେକୌଣସି ପରିବର୍ତ୍ତନ ବାସ୍ତବ ଚର ଉପରେ ତୁରନ୍ତ ପ୍ରଭାବ ପକାଇବ, କାରଣ ଫଙ୍କ୍ସନ୍‌ଟି ଠିକଣା ମାଧ୍ୟମରେ ବାସ୍ତବ ଚର ଉପରେ ହିଁ କାର୍ଯ୍ୟ କରିବ ।

6.10.4 ଏକ-ପରିସର ଆରେକ ଚର ଭାବେ ପାରିତ କରିବା (Passing One-Dimensional Array as Argument)

ଧ୍ୟାନ ଦେବାର କଥା ଯେ ଏକ ଆରେର ସମସ୍ତ ଉପାଦାନଗୁଡ଼ିକର ମୂଲ୍ୟଗୁଡ଼ିକୁ ଫଙ୍କ୍ସନ୍‌କୁ ପାରିତ କରିବା ପରିବର୍ତ୍ତେ, କେବଳ ଆରେର ଠିକଣା ପାରିତ ହୋଇଥାଏ । ତୁମେ ଜାଣ ଯେ, C ରେ, ଏକ ଆରେର ନାମ ହେଉଛି ଆଧାର ଠିକଣା, ଅର୍ଥାତ, ଆରେର ପ୍ରଥମ ଉପାଦାନର ଠିକଣା ।

```
/*  main function   */
#include <stdio.h>
void fun( int x[], int m );
void main()
{
   int a[10], n;
   /*  other local declarations */

   func(a, n); /* function call */
   /*  other statements         */
}
```

```
/*  function definition   */
void fun( int x[], int m )
{
   /*  local declarations */
   /*  other statements   */
}
```

ଚିତ୍ର. 6.4: ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟକରେ ଚର ଭାବରେ ଏକ-ପରିସର ଆରେ ପାରିତ କରିବା

କାରଣ ଆରେର ନାମ ବାସ୍ତବରେ ଏହାର ଠିକଣା ଅଟେ, ତେଣୁ, ଆରେ ନାମ ପାରିତ କରିବା କଲିଂ ଫଙ୍କ୍ସନ୍‌ରେ ଆରେକୁ ପୁନର୍ବାର ସୁଚିତ କରିବାକୁ କଲ୍‌ଡ ଫଙ୍କ୍ସନ୍‌କୁ ଅନୁମତି ଦେଇଥାଏ ।

ତାଲିକା (Listing) 6.4

```
/* Program to demonstrate the passing of an array as an argument  */
 #include <stdio.h>
 int largest( int x[], int n );       /*  function prototype */
 int main(void)
 {
    int a[10]={12,15,20,17,25,50,11,10,8,13};
    int i;
    printf( "Largest element of array = %d\n", largest(a,10));
    return 0;
 }
 /* function that returns largest element of the array   */
 int largest( const int x[], int n )
 {
    int big, i;
    big = x[0];
    for ( i = 1; i < n; i++ ) {
       if ( x[i] > big )
          big = x[i];
    }
    return big;
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Largest element of array = 50
```

6.10.5 ଦ୍ୱି-ପରିସର ଆରେକୁ ଚର ଭାବେ ପାରିତ କରିବା (Passing Two-Dimensional Array as Argument)

ଏକ-ପରିସର ଆରେ ଭଳି, ଆମେ ପ୍ରକାର୍ଯ୍ୟ ପାଇଁ ଦ୍ୱି-ପରିସର ଆରେ ପାରିତ କଲାବେଳେ ମଧ୍ୟ ଆରେର ନାମ ବ୍ୟବହାର କରିଥାଉ । କଲ୍ ହୋଇଥିବା ପ୍ରକାର୍ଯ୍ୟ ହେଡରର ଚର ତାଲିକାରେ ଥିବା ସ୍ୱତନ୍ତ୍ର ଚର, ଆରେର ଦୁଇଟି ପରିସର ଥିବା ସୂଚାଉଥିବା ଆବଶ୍ୟକ । ପ୍ରତ୍ୟେକ ପରିସରପାଇଁ ଗୋଟିଏ ଲେଖାଏଁ, ଦୁଇ ଯୋଡ଼ି ବର୍ଗ-ବନ୍ଧନୀ ସଂଯୋଗ କରି ଏହା ଦର୍ଶାଯାଇଥାଏ । ବର୍ଗ-ବନ୍ଧନୀର ପ୍ରଥମ ଯୋଡ଼ି ଖାଲି ହୋଇପାରେ କିନ୍ତୁ ଦ୍ୱିତୀୟ ଯୋଡି ବର୍ଗ-ବନ୍ଧନୀମଧ୍ୟରେ ଆମକୁ ପ୍ରକୃତ ଆରେର ଆକାର ଉଲ୍ଲେଖ କରିବାକୁ ପଡିବ । ପ୍ରକାର୍ଯ୍ୟ କଲ୍ ସମୟରେ, କେବଳ ଆରେର ମୂଳ ଠିକଣା ପାରିତ ହୋଇଥାଏ ।

```
/*  main function   */
#include <stdio.h>
void fun(int x[][4], int m, int n);
void main()
{
   int a[4][4];
   /*  other local declarations  */
   func(a,m,n); /* function call */
   /*  other statements          */
}
```

```
/*  function definition    */
void fun(int x[][4], int m, int n)
{
   /*  local declarations   */
   /*  other statement      */
}
```

ଚିତ୍ର . 6.5: ଗୋଟିଏ କାର୍ଯ୍ୟକାରିତା ଫଙ୍କ୍‌ସନ୍‌ପାଇଁ ଚର ଭାବେ ଦ୍ୱି-ପରିସର ଆରେ ପାରିତ କରିବା

ତାଲିକା (Listing) 6.5

```
/*
   Program to add matrix A(mxn) and matrix B(mxn).
   This program uses function to read, add and display matrices.
*/
#include<stdio.h>
#define ROWS 10
#define COLS 10
/* function declarations */
void readMatrix(int a[][COLS], int m, int n);
void printMatrix(int a[][COLS], int m, int n);
void addMatrices(int a[][COLS], int b[][COLS], int c[][COLS],
                 int m, int n);
int main()
{
    int a[ROWS][COLS], b[ROWS][COLS], c[ROWS][COLS];
    int n, m;
    char ch;
    printf( "Enter size of matriices as mxn: " );
    scanf( "%d%c%d", &m, &ch, &n);
```

```
    readMatrix(a,m,n);        /* input matrix A(mxn) */
    readMatrix(b,m,n);        /* input matrix B(mxn) */
    addMatrices(a,b,c,m,n);  /* add matrix A(mxn) & matrix B(mxn) */
    printf( "\nSum A+B is\n\n" );
    printMatrix(c,m,n);      /* output matrix C(mxn) */
    return 0;
}
/* function that reads a matrix A(mxn)  */
void readMatrix(int a[][COLS], int m, int n)
{
   int i, j;
   printf("\nEnter %d elements of matrix A row-wise\n", m*n );
   for ( i = 0 ; i < m; i++ ) {
      for ( j = 0; j < n; j++ ) {
         scanf( "%d", &a[i][j] );
      }
   }
}
/* function that displays a matrix 'a' of order mxn  */
void printMatrix(int a[][COLS], int m, int n)
{
    int i, j;
    for ( i = 0 ; i < m; i++ ) {
       for ( j = 0; j < n; j++ ) {
           printf( "%4d", a[i][j] );
       }
       printf ( "\n" );
    }
}
/*
   function that adds A(mxn) & b(mxn), and
   stores the sum in matrix c(mxn)
*/
void addMatrices(int a[][COLS], int b[][COLS], int c[][COLS],
                 int m, int n)
{
    int i, j;
    for ( i = 0 ; i < m; i++ )
    {
        for ( j = 0 ; j < n; j++ )
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
}
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter size of matrices as mxn : 3x3
Enter 9 elements of matrix A row-wise
 1 3 2
 4 5 1
 6 5 8
Enter 9 elements of matrix A row-wise
 7 3 5
 2 7 3
 4 2 1
Sum A+B is
 8   6  7
 6  12  4
10   7  9
```

6.10.6 ଷ୍ଟ୍ରିଙ୍ଗ୍‌କୁ ଚର ଭାବେ ପାରିତ (Passing String as Argument)

ଷ୍ଟ୍ରିଙ୍ଗ୍‌କୁ ଏକ ଚର ଭାବରେ ପାରିତ କରିବାପାଇଁ ଆରେର ପାଦାଙ୍କ ସଂକେତନ (subscripted notation) ବ୍ୟବହାର କରିପାରିବା। ଏକ ପ୍ରକାର୍ଯ୍ୟ ସଂଜ୍ଞାରେ, ସ୍ୱତନ୍ତ୍ର ଚରକୁ ଅକ୍ଷରର ଏକ ଆରେ ଭାବେ ଘୋଷିତ କରାଯାଏ। ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍ ଏକ ପ୍ରକାର୍ଯ୍ୟକୁ ଗୋଟିଏ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପାରିତ କରିବା ଦର୍ଶାଉଛି।

ତାଲିକା (Listing) 6.6

```
/* Program to illustrates passing of a string to a function  */
 #include<stdio.h>
 #include<string.h>
 void fun( char temp[] );     /*  function prototype  */
 int main()
 {
    char str[] = "Sample string";
    fun(str);
    return 0;
 }
 void fun( char temp[] )
 {
    printf( "String passed to fun : " );
    puts( temp );
    printf( "and its length is : %d\n", strlen(temp) );
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
String passed to fun : Sample string
and its length is : 13
```

ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ପ୍ରକାର୍ଯ୍ୟର ବ୍ୟାଖ୍ୟା ନିଜେ ପ୍ରୟୋଗ (hands-on) କରିବାପାଇଁ, ଆସ ଆଉ କିଛି ଉଦାହରଣ ନେବା ।

ଉଦାହରଣ 6.4: ଏକ ଯୁକ୍ତାତ୍ମକ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା *n* ର ଫ୍ୟାକ୍ଟୋରିଆଲ୍ (factorial) ବାହାର କରିବାପାଇଁ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ ଯଥା *int factorial (int n)* ଲେଖ । ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ପ୍ରକାର୍ଯ୍ୟର ଏହି ସଂଜ୍ଞା ବ୍ୟବହାର କରି, ନିମ୍ନ ବ୍ୟାଖ୍ୟାନୁସାରେ ବାଇନୋମିଆଲ୍ (Binomial) ଗୁଣାଙ୍କ ଗଣନା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

$$^{n}c_{r} = \frac{n!}{r!(n-r)!}$$

ତାଲିକା (Listing) 6.7

```
/*  Program to compute Binomial coefficient  */
 #include <stdio.h>
 int factorial(int n);  /*  function prototype  */
 int main()
 {
     int ncr, n, r;
     printf( "\nEnter value of n: " );
     scanf( "%d", &n );
     printf( "\nEnter value of r: " );
     scanf( "%d", &r );
     ncr = factorial(n)/(factorial(r)*factorial(n□r));
     printf( "\nValue of ncr = %d\n" , ncr);
     return 0;
 }

/*  Function to compute factorial of a +ve integer number  */
 int factorial(int n)
 {
    int prod = 1, i;
    for ( i = 1; i <= n; i++ )
       prod = prod * i;
    return prod;
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter value of n: 5
Enter value of r: 3
Value of ncr = 10
```

ଏହି ପ୍ରୋଗ୍ରାମ୍‌ଟି ତିନିଥର ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ପ୍ରକାର୍ଯ୍ୟକୁ ଯଥାକ୍ରମେ *n, r, (n–r)* ଚର ସହିତ ଡାକେ ଏବଂ ତା'ପରେ ଏହି ମୂଲ୍ୟଗୁଡ଼ିକ ବ୍ୟବହାର କରି ବାଇନୋମିଆଲ୍ ଗୁଣାଙ୍କ ଗଣନା କରେ ।

ଉଦାହରଣ 6.5: ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ ଲେଖ, ଯେପରିକି *int computeHCF (int m, int n),* ଯାହାକି m ଏବଂ n ର ଗସାଗୁ *(HCF)* ଫେରସ୍ତ କରେ । ଏହି ସଂଜ୍ଞା ବ୍ୟବହାର କରି ଦିଆଯାଇଥିବା ଦୁଇଟି ଯୁକ୍ତାତ୍ମକ ପୂର୍ଣ୍ଣସଂଖ୍ୟା m ଏବଂ n ର ଗସାଗୁ ଗଣନା କରିବାପାଇଁ ଏକ ସମ୍ପୂର୍ଣ୍ଣ କାର୍ଯ୍ୟକ୍ରମ ଲେଖ ।

ତାଲିକା (Listing) 6.8

```
/*
     Program to compute HCF of two given positive integers (m and n)
 */
 #include <stdio.h>
 int computeHCF( int m, int n );   /*  function prototype */
 int main()
 {
     int m, n, hcf;
     printf( "\nEnter value of m: " );
     scanf( "%d", &m );
     printf( "\nEnter value of n: " );
     scanf( "%d", &n );
     hcf = computeHCF(m,n);
     printf( "\nValue of HCF = %d\n" , hcf);
     return 0;
 }

 /* Function to compute HCF of two positive integer numbers */
 int computeHCF( int m, int n )
 {
    int r;
    while ( 1 )
    {
       r = m % n;
       if ( r == 0 )
          return n;
       m = n;
       n = r;
    }
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter value of m: 125
Enter value of n: 35
Value of HCF = 5
```

ଉଦାହରଣ 6.6: ଏକ ପ୍ରକାର୍ଯ୍ୟ *int smallestDigit(int n)* ଲେଖ, ଯାହାକି ସଂଖ୍ୟା *n* ର ସର୍ବନିମ୍ନ ଅଙ୍କଟି ଫେରସ୍ତ କରିବ। ଏହାର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଏହି ଫଙ୍କ୍‌ସନ୍ ବ୍ୟବହାର କର।

ତାଲିକା (Listing) 6.9

```
/*
    Program to find smallest digit in a positive integer number
 */
 #include <stdio.h>
 int smallestDigit(int n);     /* function prototype */

 int main()
 {
     int n;

     printf( "\nEnter positive integer number : " );
     scanf( "%d", &n );
     printf( "\nSmallest digit in %d is %d\n", n, smallestDigit(n));
     return 0;
 }
 /* function that returns smallest digit in a positive integer number */
 int smallestDigit(int n)
 {
     int sd = 9;
     int d;
     while ( n > 0 )
     {
        d = n % 10;
        if ( d < sd )
            sd = d;
        n = n / 10;
     }
     return sd;
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter positive integer number : 23145
Smallest digit in 23145 is 1
```

ଉଦାହରଣ 6.7: ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *int isPrime(int n)* ଲେଖ, ଯଦି *n* ମୌଳିକ ସଂଖ୍ୟା ତେବେ 1 ଫେରସ୍ତ କରିବ ଅନ୍ୟଥା 0 ଫେରସ୍ତ କରିବ । ଦିଆଯାଇଥିବା ଗଣନ ସଂଖ୍ୟାଟି, ମୌଳିକ ସଂଖ୍ୟା କି ନୁହେଁ ତାହା ପରୀକ୍ଷା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଏହି ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାର କର ।

ତାଲିକା (Listing) 6.10

```
/*
    Program to test whether the given natural number is
    prime number or not.
 */
 #include<stdio.h>
 #include<math.h>
 int isPrime(int n);  /*  function prototype  */
 int main()
 {
    int n;
    printf( "\nEnter a positive integer number: " );
    scanf("%d", &n);
    if ( isPrime(n) == 1 )
       printf( "\n%d is a prime number.\n", n );
    else
       printf( "\n%d is not a prime number.\n", n );
    return 0;
 }

 /*
     definition of function that tests whether the given positive
     integer number 'n' is prime number
 */
 int isPrime(int n)
 {
    int k, m;
    if ( ( n > 2 ) && ( ( n % 2 ) == 0 ) ) {
       return 0;
    }
    m = sqrt( n );
    for ( k = 3; k <= m; k += 2 )
    {
        if ( n % k == 0 )
        {
            return 0;
        }
    }
    return 1;
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

First Run

```
Enter a positive integer number: 43
43 is a prime number.
```

Second Run

```
Enter a positive integer number: 92
92 is not a prime number.
```

ୟୁନିଟ୍‌ର ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଶିଖିବାକୁ ପାଇଲୁ ଯେ

- ସଫ୍ଟୱେୟାରର (software) ଆକାର ଏବଂ ଜଟିଳତା ହେତୁ ସୃଷ୍ଟି ହେଉଥିବା ଅନେକ ସମସ୍ୟା ମଡ୍ୟୁଲାର୍ ଡିଜାଇନ୍ (modular design) ଆଭିମୁଖ୍ୟ ବ୍ୟବହାର କରି ଦକ୍ଷତାର ସହିତ ନିୟନ୍ତ୍ରଣ କରାଯାଇପାରିବ।
- ପ୍ରତ୍ୟେକ ପ୍ରକାର୍ଯ୍ୟ ସ୍ୱୟଂ ସମ୍ପୂର୍ଣ୍ଣ ଏବଂ ସ୍ୱାଧୀନ।
- ପ୍ରତ୍ୟେକ ପ୍ରକାର୍ଯ୍ୟ ଜଟିଳ ସମସ୍ୟାର ଗୋଟିଏ ଦିଗ ନିୟନ୍ତ୍ରଣ କରିଥାଏ।
- ଫେରସ୍ତ ବିବୃତ୍ତି ବ୍ୟବହାର କରି ଏକ ପ୍ରକାର୍ଯ୍ୟରୁ ମୂଲ୍ୟ ଫେରସ୍ତ କରାଯାଇପାରିବ।
- ଏକ ପ୍ରକାର୍ଯ୍ୟ ଯେକୌଣସି ପ୍ରକାରର ମୂଲ୍ୟ ଫେରସ୍ତ କରିପାରେ। ଏଥିରେ ସୂଚକ(ପଏଣ୍ଟର୍) ମଧ୍ୟ ଅନ୍ତର୍ଭୁକ୍ତ।
- ମୂଲ୍ୟ ପାରିତ ପଦ୍ଧତି କିମ୍ବା ରେଫରେନ୍ସ ପାରିତ ପଦ୍ଧତି ବ୍ୟବହାର କରି ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟର ଚରଗୁଡ଼ିକ ପାରିତ କରାଯାଇପାରିବ।
- ଚଳ ଘୋଷଣା ପରି ଏକ ପ୍ରକାର୍ଯ୍ୟ ପ୍ରୋଟୋଟାଇପ୍ ହେଉଛି ଏକ ଘୋଷଣା ଯାହା ଫେରସ୍ତ ପ୍ରକାର, ନାମ, ଏବଂ ସ୍ୱତନ୍ତ୍ର ଚରର ପ୍ରକାର ନିର୍ଦ୍ଦିଷ୍ଟ କରେ।

ଅନୁଶୀଳନୀ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. ଏକ ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ଫଙ୍କସନ୍ ଏବଂ ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ କ'ଣ?
2. ପ୍ରକାର୍ଯ୍ୟ କାର୍ଯ୍ୟକାରିତା କେତେବେଳେ ସମାପ୍ତ ହୋଇଥାଏ?
3. ଏକ ପ୍ରକାର୍ଯ୍ୟକୁ କେତେଗୋଟି ଚର ପାରିତ କରାଯାଇପାରିବ?
4. ସ୍ୱତନ୍ତ୍ର ଚର ଏବଂ ବାସ୍ତବ ଚର ମଧ୍ୟରେ ସମ୍ପର୍କ ସମ୍ବନ୍ଧରେ ନିୟମ କ'ଣ?
5. ଏକ ପ୍ରକାର୍ଯ୍ୟକୁ କିପରି ଆହ୍ୱାନ କରାଯାଏ?
6. କେତେଥର ଏକ ପ୍ରକାର୍ଯ୍ୟ ଡକା ଯାଇପାରିବ?
7. ନିମ୍ନଲିଖିତ ପ୍ରକାର୍ଯ୍ୟ ସଂଜ୍ଞାରେ କ'ଣ ଭୁଲ୍ ଅଛି?

```
testFunction( int k ) {
   float temp = 5.25;
   temp = k / 2.0;
   return temp;
}
```

8. ନିମ୍ନଲିଖିତ ପ୍ରକାର୍ଯ୍ୟ ସଂଜ୍ଞାରେ ତ୍ରୁଟି ଥିଲେ ଦର୍ଶାଅ ।

```
void fun( int x, int y ) {
   int z;
   /*  some statement  */
   return z;
}
```

9. ନିମ୍ନଲିଖିତ ପ୍ରକାର୍ଯ୍ୟ ସଂଜ୍ଞାରେ ତ୍ରୁଟି ଥିଲେ ଦର୍ଶାଅ ।

```
void fun( int x, y ) {
   int z;
   /*  some statement  */
   return;
}
```

10. ନିମ୍ନଲିଖିତ ପ୍ରକାର୍ଯ୍ୟ ସଂଜ୍ଞାରେ ତ୍ରୁଟି ଥିଲେ ଦର୍ଶାଅ ।

```
int fun1( int x, int y ) {
   int z;
   /*  some statement  */
   int fun2( int t ) {
      return (t-2);
   }
   /*  some more statements  */
   return z;
}
```

11. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମର ଆଉଟ୍‌ପୁଟ୍ (output) କ'ଣ ହେବ ?

```
void main() {
   float r = 5.0, c;
   float func( float t );        /* function prototype  */
   c = func( r );
   printf( "\nValue returned by function" );
   printf( " = %d\n", c);
}
float func( float t ) {
   float temp;
   temp = t * t / 2;
   return 5.0;
}
```

12. ନିମ୍ନଲିଖିତ ଫଙ୍କ୍ସନ୍ ଘୋଷଣାନାମାରେ ତ୍ରୁଟି ଥିଲେ ଦର୍ଶାଅ ।

(*a*) `int diff(int x, y);`

(*b*) `void fun(void, void);`

(*c*) `float diff(float, float);`

(*d*) `int test(float x, int y)`

ପ୍ରୋଗ୍ରାମିଂ ସମସ୍ୟା (Programming Problems)

1. ଗୋଟିଏ ଯୁକ୍ତାତ୍ମକ ପୂର୍ଣ୍ଣସଂଖ୍ୟାର *(positive integer)* ଅଙ୍କଗୁଡିକର ସମଷ୍ଟି ଫେରସ୍ତ କରୁଥିବା ଏକ ପ୍ରକାର୍ଯ୍ୟ *int sumOfDigits(int number)* ଲେଖ ।
2. ଗୋଟିଏ float ପ୍ରକାରର ସଂଖ୍ୟାର ପରମମାନ *(absolute)* ଫେରସ୍ତ କରିପାରୁଥିବା ଏକ ପ୍ରକାର୍ଯ୍ୟ *float absValue(float value)* ଲେଖ ।
3. float x ର ନିକଟତମ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାର ମୂଲ୍ୟ ଫେରସ୍ତପାଇଁ ଏକ ପ୍ରକାର୍ଯ୍ୟ *int round(float x)* ଲେଖ ।
4. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *int sumOfN(int n, int m)* ଲେଖ, ଯାହା ପୂର୍ଣ୍ଣସଂଖ୍ୟା m ରୁ ଆରମ୍ଭ କରି *n* ସଂଖ୍ୟକ ପୂର୍ଣ୍ଣସଂଖ୍ୟାଗୁଡ଼ିକର ଯୋଗଫଳ ନିର୍ଣ୍ଣୟ କରେ, ଅର୍ଥାତ, *i.e.,* $m + (m + 1) + (m + 2)+ \ldots + (m + n - 1)$. ।
5. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *int isPrime(int n)* ଲେଖ, ଯେଉଁଥିରେ n ଏକ ମୌଳିକ ସଂଖ୍ୟା ହେଲେ ଏହା 1 ଫେରସ୍ତ କରିବ ନଚେତ୍ 0 ଫେରସ୍ତ କରିବ ।
6. ବାମରୁ କିମ୍ବା ଡାହାଣରୁ ପଢିଲେ, ଯଦି ସଂଖ୍ୟାଟି ସମାନ ହୁଏ ତେବେ ଏହି ସଂଖ୍ୟା ହେଉଛି ଏକ ପାଲିଣ୍ଡ୍ରୋମ୍ (Palindrome) । ଉଦାହରଣ ସ୍ୱରୂପ, ସଂଖ୍ୟା 1991, 1001 ଏବଂ 1221 ଗୁଡିକ ପାଲିଣ୍ଡ୍ରୋମ୍ ସଂଖ୍ୟା । ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *int isPalindrome(unsigned int k)* ଲେଖ, ଯଦି ସଂଖ୍ୟା k ପାଲିଣ୍ଡ୍ରୋମ୍ ହୁଏ ତେବେ ଏହା 1 ଫେରସ୍ତ କରିବ ଅନ୍ୟଥା 0 ଫେରସ୍ତ କରିବ ।
7. ଗୋଟିଏ ଯୁକ୍ତାତ୍ମକ ପୂର୍ଣ୍ଣାଙ୍କ ସଂଖ୍ୟା IJK କୁ ସୁସଜ୍ଜିତ ବୋଲି କୁହାଯାଏ ଯଦି $I < J < K$, ଉଦାହରଣ ସ୍ୱରୂପ, ସଂଖ୍ୟା 138 କୁ ସୁସଜ୍ଜିତ ବୋଲି କୁହାଯିଏ କାରଣ ସଂଖ୍ୟାରେ ଥିବା ଅଙ୍କଗୁଡିକ (1, 3, 8) ବାମରୁ ଡାହାଣକୁ ବୃଦ୍ଧି ପାଇଅଛି, ଅର୍ଥାତ, $1 < 3 < 8$ । ସଂଖ୍ୟା 365 ସୁସଜ୍ଜିତ ହୋଇନାହିଁ କାରଣ 6 ଟି 5 ରୁ ବଡ ଅଟେ । ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ ଲେଖ int isWellOrdered(unsigned int k) ଯେଉଁଥିରେ, ଯଦି k ଏକ ସୁସଜ୍ଜିତ ସଂଖ୍ୟା ହୋଇଥାଏ ତେବେ ଏହା 1 ଫେରସ୍ତ କରିବ ଅନ୍ୟଥା 0 ଫେରସ୍ତ କରିବ ।
8. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *int largestDigit(int x)* ଲେଖ, ଯାହା ସଂଖ୍ୟା x ର ବୃହତମ ଅଙ୍କକୁ ଫେରସ୍ତ କରିବ ।
9. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *int unitDigit(int n)* ଲେଖ, ଯାହାକି ଏହି ଚରରେ ଥିବା ସଂଖ୍ୟା *n* ର ୟୁନିଟ ସ୍ଥାନର ଅଙ୍କକୁ ଫେରସ୍ତ କରିବ ।
10. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ int isLeapYear(int year) ଲେଖ, ଯଦି ଚରରେ ଥିବା ସମୟ ଏକ ଅଧିବର୍ଷ ହୁଏ ତାହେଲେ ଫେରସ୍ତ ମୂଲ୍ୟ 1 ହେବ ଅନ୍ୟଥା ଫେରସ୍ତ ମୂଲ୍ୟ 0 ହେବ ।
11. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ int isValidDate(int dd, int mm, int yyyy) ଲେଖ, ଯଦି ତାରିଖଟି ବୈଧ ତେବେ ଫେରସ୍ତ ମୂଲ୍ୟ 1 ହେବ ଅନ୍ୟଥା ଫେରସ୍ତ ମୂଲ୍ୟ 0 ହେବ ।
12. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ int isTrianglePossible(int a, int b, int c) ଲେଖ, ଯଦି a, b, ଏବଂ c ଏକ ତ୍ରିଭୁଜର ପାର୍ଶ୍ୱ ହେବାର ଯୋଗ୍ୟ ତେବେ ଫେରସ୍ତ ମୂଲ୍ୟ 1 ହେବ ଅନ୍ୟଥା ଫେରସ୍ତ ମୂଲ୍ୟ 0 ହେବ ।
13. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ int countDigits(int n) ଲେଖ, ଯାହା ସଂଖ୍ୟା n ରେ ଥିବା ଅଙ୍କର ସଂଖ୍ୟା ଫେରସ୍ତ କରିବ, ଅର୍ଥାତ, ସଂଖ୍ୟା n ର ଆକାର ।

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. ପ୍ରକାର୍ଯ୍ୟଗୁଡ଼ିକ ବିଷୟରେ ନିମ୍ନଲିଖିତ ବିବୃତ୍ତି ମଧ୍ୟରୁ କେଉଁଟି ମିଥ୍ୟା ଅଟେ?
   (a) ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଏକାଧିକ ପ୍ରକାର୍ଯ୍ୟ ରହିପାରିବ।
   (b) ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ ଅନ୍ୟ ଫଙ୍କ୍‌ସନ୍‌କୁ କଲ୍ କରିପାରେ।
   (c) ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ ନିଜକୁ କଲ୍ କରିପାରେ।
   (d) ସ୍ୱତନ୍ତ୍ର ଚର ସୂଚିରେ ସ୍ଥିରାଙ୍କ ରହିପାରିବ।
2. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାର କରିବାର ବୈଧ କାରଣ ନୁହେଁ?
   (a) ସେଗୁଡ଼ିକ ସମାନ କୋଡ୍ ବାରମ୍ବାର ଲେଖିବା ଅପେକ୍ଷା କମ୍ ମେମୋରି ବ୍ୟବହାର କରେ।
   (b) ସେଗୁଡ଼ିକ ପ୍ରୋଗ୍ରାମ୍‌ର ବିଭିନ୍ନ କାର୍ଯ୍ୟକଳାପକୁ ପୃଥକ୍ ରଖେ।
   (c) ସେଗୁଡ଼ିକ ଶୀଘ୍ର ଫଳାଫଳ ପ୍ରଦାନ କରିଥାଏ।
   (d) ସେଗୁଡ଼ିକ ପ୍ରୋଗ୍ରାମର ଅନ୍ୟ ଅଂଶରୁ ଚଳମାନଙ୍କୁ ସୁରକ୍ଷିତ ରଖେ
3. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟର ଡିଫଲ୍ଟ(default) ଫେରସ୍ତ ପ୍ରକାର କ'ଣ?
   (a) void (b) int
   (c) float (d) char
4. କେଉଁଠାରୁ ପ୍ରୋଗ୍ରାମ୍‌ର କାର୍ଯ୍ୟକାରିତା ଆରମ୍ଭ ହୁଏ
   (a) main() ଫଙ୍କ୍‌ସନ୍‌ରୁ
   (b) ପ୍ରଥମେ ବ୍ୟାଖ୍ୟା କରାଯାଇଥିବା ଫଙ୍କ୍‌ସନ୍‌ରୁ
   (c) ଶେଷରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଥିବା ଫଙ୍କ୍‌ସନ୍‌ରୁ
   (d) ସଂକଳକ (compiler) ଉପରେ ନିର୍ଭର କରେ
5. C ଭାଷାରେ ଚର ପାରିତ କରିବାକୁ କେଉଁ ପଦ୍ଧତିଗୁଡ଼ିକ ଯୋଗ୍ୟ
   (a) କେବଳ ମୂଲ୍ୟଦ୍ୱାରା କଲ୍ (b) କେବଳ ରେଫରେନ୍‌ସଦ୍ୱାରା କଲ୍
   (c) ଉଭୟ a & b (d) ସଂକଳକ (compiler) ଉପରେ ନିର୍ଭର କରେ
6. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌କୁ ବିଚାରକୁ ନିଅ

```
swap(int i, int j)
{
   i = i + j;
   j = i - j;
   i = i - j;
}
void main()
{
```

```
        int i = 5, j = 10;
        swap(i, j);
        printf( "\n%d, %d", i, j );
    }
```

ଯେତେବେଳେ ଆମେ ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ୍‌କୁ ସଂକଳନ ଓ କାର୍ଯ୍ୟକାରୀ କରିବାକୁ ଚେଷ୍ଟା କରିବୁ ସେତେବେଳେ କ'ଣ ହେବ?

(a) ପ୍ରୋଗ୍ରାମ୍ ସଂକଳନ କରିବାରେ ବିଫଳ ହେବ କାରଣ ସମାନ ଚଳ ନାମଗୁଡିକ main() ଏବଂ swap() ଫଙ୍କ୍‌ସନ୍‌ରେ ଗୋଟଏ ପ୍ରୋଗ୍ରାମ୍ ଭିତରେ ବ୍ୟବହାର କରାଯାଇପାରିବ ନାହିଁ।

(b) ପ୍ରୋଗ୍ରାମ୍ ସଂକଳନ କରିବାରେ ବିଫଳ ହେବ କାରଣ swap() ଫଙ୍କ୍‌ସନ୍‌ର ଫେରସ୍ତ ପ୍ରକାର ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇନାହିଁ।

(c) ପ୍ରୋଗ୍ରାମ୍‌ଟି ସଂକଳିତ ଏବଂ କାର୍ଯ୍ୟକାରୀ ହେଲାପରେ 5, 10 ଫଳାଫଳ ପ୍ରଦାନ କରିବ।

(d) ପ୍ରୋଗ୍ରାମ୍‌ଟି ସଂକଳିତ ଏବଂ କାର୍ଯ୍ୟକାରୀ ହେଲାପରେ 10, 5 ଫଳାଫଳ ପ୍ରଦାନ କରିବ।

7. ଠିକ୍ ବିବୃତ୍ତି ଚିହ୍ନଟ କର

(a) ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଏକାଧିକ ଥର ଏକ ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟାଖ୍ୟା କରାଯାଇପାରିବ।

(b) ଗୋଟିଏ ଫଙ୍କ୍‌ସନ୍‌କୁ ଅନ୍ୟ ପ୍ରକାର୍ଯ୍ୟର ସଂଜ୍ଞା ମଧ୍ୟରେ ବ୍ୟାଖ୍ୟା କରାଯାଇପାରିବ ନାହିଁ।

(c) ସମସ୍ତ ପ୍ରକାର୍ଯ୍ୟ ସମାନ ଫାଇଲରେ ରହିବା ଆବଶ୍ୟକ।

(d) ପ୍ରକାର୍ଯ୍ୟ, ମୁଖ୍ୟ ପ୍ରକାର୍ଯ୍ୟରେ ଡକାଯାଇଥିବା କ୍ରମରେ ପ୍ରକାର୍ଯ୍ୟଟି ଲେଖାହେବା ଉଚିତ।

8. ନିମ୍ନଲିଖିତ ପ୍ରକାର୍ଯ୍ୟ ସଂଜ୍ଞାକୁ ଦେଖି ଠିକ୍ ବିବୃତ୍ତିଟି ଚିହ୍ନଟ କର

```
myFunction( a + b, c, d, 5 )
int a, b, c, d;
{
    int sum;
    sum = a + b +  c + d + 5;
    return sum;
}
```

(a) ପ୍ରକାର୍ଯ୍ୟ ସଂକଳିତ ହେବାରେ ବିଫଳ ହେବ କାରଣ ଅଭିବ୍ୟକ୍ତି, ସ୍ୱତନ୍ତ୍ର ଚର ଭାବରେ ଅନୁମୋଦିତ ନୁହେଁ।

(b) ପ୍ରକାର୍ଯ୍ୟ ସଂକଳନ କରିବାରେ ବିଫଳ ହେବ କାରଣ ସ୍ଥିରାଙ୍କ, ସ୍ୱତନ୍ତ୍ର ଚର ଭାବରେ ଅନୁମୋଦିତ ନୁହେଁ।

(c) ପ୍ରକାର୍ଯ୍ୟ ସଂକଳନ କରିବାରେ ବିଫଳ ହେବ କାରଣ ପ୍ରକାର୍ଯ୍ୟର ଫେରସ୍ତ ପ୍ରକାର ବାଦ୍ ଦିଆଯାଇଛି।

(d) ଉଭୟ (a) ଏବଂ (b).

9. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ପ୍ରକାର୍ଯ୍ୟ ହେଡରର ଏକ ଅଂଶ ଅଟେ?

(a) ପ୍ରକାର୍ଯ୍ୟ ନାମ (Function name)   (b) ଫେରସ୍ତ ପ୍ରକାର (Return type)

(c) ଚର ତାଲିକା (Argument list)   (d) ଉପରୋକ୍ତ ସମସ୍ତ (All of the above)

10. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ସମ୍ପୂର୍ଣ୍ଣ ପ୍ରକାର୍ଯ୍ୟ ଅଟେ?

(a) int fun();   (b) int fun(int x) { return x+1; }

(c) void fun(int) { print("Hello"); }   (d) void fun(x) { print("hello"); }

ଉତ୍ତର									
1.	(*d*)	**2.**	(*c*)	**3.**	(*b*)	**4.**	(*a*)	**5.**	(*c*)
6.	(*c*)	**7.**	(*b*)	**8.**	(*d*)	**9.**	(*d*)	**10.**	(*b*)

ପ୍ରାକ୍ଟିକାଲ୍ (PRACTICALS)

1. ନିମ୍ନଲିଖିତ ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାର କରି n ($\leq$50) ଉପାଦାନ ବିଶିଷ୍ଟ a ନାମକ ଆରେର ବୃହତ୍ତମ ଉପାଦାନ, କ୍ଷୁଦ୍ରତ୍ତମ ଉପାଦାନ ଏବଂ ହାରାହାରି ଉପାଦାନ ବାହାର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   (a)ଆରେର ବୃହତ୍ତମ ଉପାଦାନପାଇଁ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *findLargest(int a[], int n)* ।

   (b) ଆରେର କ୍ଷୁଦ୍ରତ୍ତମ ଉପାଦାନପାଇଁ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *findSmallest(int a[], int n)* ।

   (c) ଆରେ ଉପାଦାନଗୁଡ଼ିକର ହାରାହାରିପାଇଁ ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ, *float findAverage(int a[], int n)* ।

ତାଲିକା (Listing) 6.11

```
/*
    Program to find the largest element, smallest element, and
    average of elements of array using functions
 */

 #include<stdio.h>

 /*  function prototypes  */
 int findLargest(int a[], int n);
 int findSmallest(int a[], int n);
 float findAverage(int a[], int n);

 int main()
 {
    int a[50];
    int i, n;
    printf("\nEnter size of array n(<=50) : ");
    scanf("%d", &n);
    printf("\nEnter %d elements of array\n\n");
    for ( i = 0; i < n; i++ )
    {
       scanf("%d", &a[i]);
    }
   printf("\nSmallest element    = %d", findSmallest(a,n));
    printf("\nLargest element     = %d", findLargest(a,n));
    printf("\nAverage of elements = %.2f", findAverage(a,n));
    return 0;
 }
```

```
int findLargest(int a[], int n)
 {
    int i, max;
    max = a[0];
    for ( i = 1; i < n; i++ ) {
        if ( a[i] > max )
            max = a[i];
    }
    return max;
 }

 int findSmallest(int a[], int n)
 {
    int i, min;
    min = a[0];
    for ( i = 1; i < n; i++ ) {
        if ( a[i] < min )
            min = a[i];
    }
    return max;
 }

 float findAverage(int a[], int n)
 {
    int i, sum;
    float avg;
    sum = 0;
    for ( i = 0; i < n; i++ ) {
        sum = sum + a[i];
    }
    avg = (float)sum/n;
    return avg;
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter size of array n(<=50) : 10
Enter 10 elements of array
25 20 40 32 10 15 45 50 30 24
Smallest element    = 10
Largest element     = 50
Average of elements = 29.10
```

2. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *int isPrime(int n)* ଲେଖ, ଯଦି *n* ମୌଳିକ ସଂଖ୍ୟା ତେବେ ଏହା 1 ଫେରସ୍ତ କରିବ ଅନ୍ୟଥା 0 ଫେରସ୍ତ କରିବ । ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଏହି ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାର କରି ପ୍ରଥମ *m* ସଂଖ୍ୟକ ମୌଳିକ ସଂଖ୍ୟାଗୁଡ଼ିକ ପ୍ରିଣ୍ଟ କର ।

ତାଲିକା (Listing) 6.12

```
/*
    Program to print first 'm' prime numbers using a function
 */
 #include<stdio.h>
 #include<math.h>
 int isPrime(int n);  /*  function prototype  */
 int main()
 {
    int m, num = 2, count = 0;
    printf( "\nEnter value for m : " );
    scanf("%d", &m);
    printf( "\nFirst %d prime number are\n\n", m );
    while ( count < m )
    {
        if ( isPrime(num) == 1 ) {
            count++;
            printf( "%d  ", num );
        }
        num++;
    }
    printf( "\n" );
    return 0;
 }

 /*
     definition of function that tests whether the given positive
     integer number 'n' is prime number
 */
 int isPrime(int n)
 {
    int k, m;
    if ( ( n > 2 ) && ( ( n % 2 ) == 0 ) )
    {
       return 0;
    }
   m = sqrt( n );
    for ( k = 3; k <= m; k += 2 )
```

```
    {
        if ( n % k == 0 )
        {
            return 0;
        }
    }
    return 1;
}
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter value for m : 10
First 10 prime number are
2  3  5  7  11  13  17  19  23  29
```

3. ଗୋଟିଏ ପ୍ରକାର୍ଯ୍ୟ *int product(int a, int b)* ଲେଖ, ଯାହା ଦୁଇଟି ସଂଖ୍ୟାର ଗୁଣଫଳ ଫେରସ୍ତ କରିବ । ଦିଆଯାଇଥିବା a ଏବଂ b ଦୁଇଟି ସଂଖ୍ୟାର ଗୁଣଫଳ ବାହାର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍‌ରେ ଏହି ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାର କର ।

ତାଲିକା (Listing) 6.13

```
/*
    Program to find Product of two numbers without using function
 */

 #include <stdio.h>

 /*  function prototype  */
 int product(int a, int b);

 int main()
 {
    int m, n, result;
   printf("\nEnter first number : ");
    scanf("%d", &m);
    printf("\nEnter second number : ");
    scanf("%d", &n);
    result = product(m, n);
    printf("\nProduct of %d and %d = %d\n", m, n, result);
    return 0;
 }
/*  definition of functio that returns product of two numbers  */
 int product(int a, int b)
 {
    int temp = 0;
```

```
    while (b != 0)
    {
        temp += a;
        b--;
    }
    return temp;
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter first number : 15
Enter second number : 12
Product of 15 and 12 = 180
```

ଅଧିକ ଜାଣିବାପାଇଁ

ପ୍ରକାର୍ଯ୍ୟଗୁଡିକର ବ୍ୟବହାର, ପ୍ରୋଗ୍ରାମର୍‌ଙ୍କୁ ବଡ଼ ଏବଂ ଜଟିଳ ପ୍ରୋଗ୍ରାମର ସମାଧାନପାଇଁ ପ୍ରୋଗ୍ରାମ୍ ସୃଷ୍ଟି କରିବାକୁ ସମର୍ଥ କରେ ଯାହା କୋଡ୍ (code) ଲେଖିବା ସହଜ, ତ୍ରୁଟିମୁକ୍ତ (debug) ଏବଂ ପରିବର୍ତ୍ତନ କରିବା ସହଜ ହୋଇଥାଏ।

ପ୍ରକାର୍ଯ୍ୟ ବ୍ୟବହାର କରିବାର ଲାଭ ଏବଂ ପ୍ରକାର୍ଯ୍ୟଗୁଡ଼ିକର ବିକାଶ ସହିତ ଜଡିତ ବିଭିନ୍ନ ଦିଗ ବିଷୟରେ ଶିକ୍ଷକ ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କ ମଧ୍ୟରେ ଏକ ବୁଝାମଣା ବିକାଶ କରିବେ ବୋଲି ଆଶା କରାଯାଉଛି।

ଶିକ୍ଷକ, ବାସ୍ତବ ଜୀବନ ପରିସ୍ଥିତିରୁ ଉଦାହରଣ ନେଇ ପ୍ରକାର୍ଯ୍ୟର ବ୍ୟବହାରକୁ ପ୍ରଦର୍ଶନ କରିବା ଆବଶ୍ୟକ, ଏବଂ ଏହାର ସମାଧାନପାଇଁ C ପ୍ରୋଗ୍ରାମ୍ ସୃଷ୍ଟି କରିବା ଆବଶ୍ୟକ।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

7 ରିକର୍ସନ

ଏକକ ବିଶେଷତ୍ୱ

ଏହି ଏକକରେ ରିକର୍ସନ (Recursion) ସମ୍ବନ୍ଧରେ ଆଲୋଚନା କରାଯାଇଛି । ଗଣିତ ଏବଂ କମ୍ପ୍ୟୁଟର ବିଜ୍ଞାନରେ ରିକର୍ସନ ହେଉଛି ଏକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଅଂଶ । ଏହା ଅନେକ ଅନେକ ସମସ୍ୟାର ସମାଧାନପାଇଁ ବହୁଳ ଭାବରେ ବ୍ୟବହୃତ ହୋଇପାରେ । ରିକର୍ସନର ବିଭିନ୍ନ ଦିଗକୁ ବ୍ୟାଖ୍ୟା ସହ ଏବଂ ଉପଯୁକ୍ତ ଉଦାହରଣ ମାଧ୍ୟମରେ ସେସବୁର ଉପଯୋଗ ପ୍ରଦର୍ଶନ କରାଯାଉଛି ।

ଭୂମିକା

ବାସ୍ତବ ଜୀବନର, ଅନେକ ସମସ୍ୟାର ସମାଧାନ ଆଇଟରେଟିଭ (iterative) ପଦ୍ଧତି ସହିତ ରିକର୍ସନ (recursion) ପଦ୍ଧତିରେ ମଧ୍ୟ କରାଯାଇପାରେ । ତେଣୁ କେଉଁ ପଦ୍ଧତିଟିକୁ ଧରିବାକୁ ହେବ ତାହା ଏକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ପ୍ରଶ୍ନ । ଆମେ ରିକର୍ସନ ବ୍ୟବହାର କରିବାର ମୁଖ୍ୟ କାରଣ ହେଉଛି ଏକ ଆଲଗୋରିଦମକୁ ସରଳ କରି ଲେଖିବା ଯେପରି ଏହା ଅଧିକାଂଶଙ୍କଦ୍ୱାରା ସହଜରେ ବୁଝିହେଉଥିବ । ଏଠାରେ ଧ୍ୟାନ ଦେବା ଜରୁରି ଯେ ରିକର୍ସନର ଉଦ୍ଦେଶ୍ୟ ହେଉଛି ଆମର କୋଡ୍‌କୁ ସ୍ୱଚ୍ଛ ଓ ପଠନଯୋଗ୍ୟ କରିବା ଏବଂ ତର୍କ ସାବଲୀଳ କରିବା । ରିକର୍ସନ କୌଶଳର କାର୍ଯ୍ୟଦକ୍ଷତା ବୃଦ୍ଧିପାଇଁ କୋଡ୍ ଅପ୍ଟିମାଇଜ୍(code optimize) ବ୍ୟବହାର କରିପାରିବା ନାହିଁ- ବରଂ; ଆଇଟରେଟିଭ ପଦ୍ଧତିରେ ଲେଖାଯାଇଥିବା ଅନୁରୂପ ଫଙ୍କସନ ତୁଳନାରେ ଏହାର କାର୍ଯ୍ୟଦକ୍ଷତା ଉପରେ ପ୍ରତିକୂଳ ପ୍ରଭାବ ପକାଇପାରେ ।

ମନେରଖିବାପାଇଁ ସଂକ୍ଷେପରେ କହିପାରିବା, "ରିକର୍ସିଭ୍ ଫଙ୍କସନର ବ୍ୟବହାର ପ୍ରୋଗ୍ରାମରପାଇଁ କୋଡ୍ ଲେଖିବାକୁ ସୁବିଧା କରେ । ଆଇଟରେଟିଭ ଫଙ୍କସନର ବ୍ୟବହାର କମ୍ପ୍ୟୁଟରର କାର୍ଯ୍ୟଦକ୍ଷତା ବୃଦ୍ଧିରେ ସାହାଯ୍ୟ କରେ ।"

ଆମେ ବାସ୍ତବରେ ଏକ କୋଡ୍ ଲେଖିବାକୁ ଚେଷ୍ଟା କରୁଥିବାବେଳେ ରିକର୍ସନର ଉପଯୋଗିତା ଅନୁଭବ କରୁ ।

ଏହି ୟୁନିଟରେ ବିଦ୍ୟାର୍ଥୀଙ୍କୁ ରିକର୍ସନ ସମ୍ବନ୍ଧୀୟ ବିବିଧ ଦିଗ ବୁଝିବାରେ ସାହାଯ୍ୟ କରିବ ଏବଂ C ରେ ରିକର୍ସିଭ୍ ଫଙ୍କସନ ଏହା କାର୍ଯ୍ୟକାରିତା ପାଇଁ ସହାୟକ ହେବ ।

ପୂର୍ବ ଆବଶ୍ୟକ ଜ୍ଞାନ

- କଣ୍ଡିସନାଲ୍ ବ୍ରାଞ୍ଚିଂ (Conditional branching) / ସର୍ତ୍ତମୂଳକ ଶାଖାତିଆରି
- ଆରେ (Arrays) / ସାରଣିସମୂହ
- ଉପଭୋକ୍ତା-ନିର୍ଦ୍ଧାରିତ ଫଙ୍କସମୂହ (User-defined functions)

ଏକକଲବ୍ଧ ଜ୍ଞାନ

ଏହି ୟୁନିଟର ସମ୍ପୂର୍ଣ୍ଣ ଅଧ୍ୟୟନ ପରେ, ଛାତ୍ରଛାତ୍ରୀଗଣ ନିମ୍ନଲିଖିତ ବିଷୟରେ ଜ୍ଞାନ ଆହରଣରେ ସମର୍ଥ ହେବେ ।

U7-O1: ରିକର୍ସନ ଧାରଣାର ବ୍ୟାଖ୍ୟା କରିବା –

U7-O2: ମୂଳ ଅବସ୍ଥା (base case) ଏବଂ ରିକର୍ସିଭ ପଦକ୍ଷେପ ସମ୍ବନ୍ଧୀୟ ଧାରଣାର ବ୍ୟାଖ୍ୟା କରିବା –

U7-O3: ରିକର୍ସିଭ ସମସ୍ୟାଗୁଡ଼ିକପାଇଁ ରିକର୍ସିଭ ଫଙ୍କସନ ଲେଖିବା ଏବଂ ବ୍ୟବହାର କରିବା

U7-O4: କ୍ୱିକ୍ ସର୍ଟ (Quick sort) ଆଲଗୋରିଦମର ବ୍ୟାଖ୍ୟା ଏବଂ ପ୍ରୋଗ୍ରାମ ଲେଖିବା

U7-O5: ମର୍ଜ ସର୍ଟ (Merge sort) ଆଲଗୋରିଦମର ବ୍ୟାଖ୍ୟା ଏବଂ ପ୍ରୋଗ୍ରାମ ଲେଖିବା

U7-O6: ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ସମାଧାନପାଇଁ ରିକର୍ସିଭ ଫଙ୍କସନ ବ୍ୟବହାର କରି ମଡ୍ୟୁଲାର୍ ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବା

ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହିତ ୟୁନିଟଲବ୍ଧ ଜ୍ଞାନଗୁଡିକ ମଧ୍ୟରେ ସମ୍ପର୍କ

ୟୁନିଟ-7 ୟୁନିଟଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ (1 – ଦୁର୍ବଳ ସମ୍ପର୍କ; 2 – ମଧ୍ୟମ ସମ୍ପର୍କ; 3 – ଦୃଢ ସମ୍ପର୍କ)						
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7
U7-O1	1	–	–	–	–	–	–
U7-O2	1	–	–	–	–	–	–
U7-O3	–	–	–	–	2	–	–
U7-O4	–	–	–	–	–	–	3
U7-O5	–	–	–	–	–	–	3
U7-O6	–	–	–	–	–	–	3

7.1 ଉପକ୍ରମ (INTRODUCTION)

ପ୍ରୋଗ୍ରାମଗୁଡିକର ବିକାଶରେ ରିକର୍ସନ (recursion) ହେଉଛି ଏକ ଶକ୍ତିଶାଳୀ ଧାରଣା ଯେଉଁଥିରେ ଫଙ୍କସନ ଆପେ ବ୍ୟବହାର କରି ସମସ୍ୟାର ସମାଧାନ କରିପାରେ । ଏହି ନିୟନ୍ତ୍ରଣ କୌଶଳ ରିକର୍ସନଭାବେ ପରିଚିତ ଏବଂ କମ୍ପ୍ୟୁଟର ବିଜ୍ଞାନରେ ଏକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ବିଷୟ । ବିଭିନ୍ନ ସମସ୍ୟାର ସାମାଧାନ ପାଇଁ ଏହା ସୁବିଧାଜନକ । ଏହାର ଅଭାବରେ ହୁଏତ for, while, ଏବଂ do-while ଭଳି ଆଇଟରେଟିଭ ଲୁପ୍ ବ୍ୟବହାର କରି ସମାଧାନ କରିବା କଷ୍ଟକର ହୋଇ ଥାଆନ୍ତା ।

ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ଅନେକ ସମାଧାନପାଇଁ ରିକର୍ସନ ବ୍ୟାପକ ଭାବରେ ବ୍ୟବହୃତ ହୋଇଥାଏ ।

ପ୍ରାୟ ସମସ୍ତ ଆଧୁନିକ ଉଚ୍ଚସ୍ତରୀୟ ପ୍ରୋଗ୍ରାମିଂ ଭାଷା ଯେପରିକି C/C++/C#/Java/Python ରେ ରିକର୍ସିଭ୍ ଫଙ୍କସନଗୁଡିକ ସିଧାସଳଖ କାର୍ଯ୍ୟକାରୀ କରାଯାଇପାରିବ ।

ଏହି ୟୁନିଟରେ ରିକର୍ସିଭ୍ ଫଙ୍କସନଗୁଡିକ ଉପସ୍ଥାପିତ ହୋଇଛି ଏବଂ ବିଭିନ୍ନ ଉଦାହରଣ ମାଧ୍ୟମରେ ସେସବୁର ବ୍ୟବହାର ବର୍ଣ୍ଣନା କରାଯାଇଅଛି ।

7.2 ରିକର୍ସିଭ୍ ଫଙ୍କସନ

ରିକର୍ସିଭ୍ ଫଙ୍କସନ ହେଉଛି ଏକ ଫଙ୍କସନ ଯାହାର ସଂଜ୍ଞା ନିଜ ଉପରେ ଆଧାରିତ ଅର୍ଥାତ ଯାହା ନିଜକୁ ନିଜେ ଡାକିଥାଏ ।

ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କସନର ମୂଳ ଅବସ୍ଥା(base case) ରିକର୍ସିଭ୍ ପଦକ୍ଷେପ(recursive step) ଅନୁଯାୟୀ ବ୍ୟାଖ୍ୟା କରାଯାଇଥାଏ ।

1. **ମୂଳ ଅବସ୍ଥା (Base Case):** ଫଙ୍କସନକୁ ଦିଆଯାଇଥିବା ଇନପୁଟ୍‌ର (inputs) ପରିଣାମକୁ ତତ୍କାଳ ଗଣନା

କରାଯାଏ, ଅର୍ଥାତ୍, ଯେଉଁ ଚର-ମୂଲ୍ୟପାଇଁ ଫଙ୍କ୍ସନ ନିଜକୁ ନିଜେ କଲ୍ କରିବା ଦରକାର ନାହିଁ।

ଅନ୍ୟଭାବେ କହିଲେ, ମୂଳ ଅବସ୍ଥା ହେଉଛି "ସମ୍ଭାବ୍ୟ ସରଳତମ ସମସ୍ୟାର" ସମାଧାନ।

ଉଦାହରଣପାଇଁ ଗୋଟିଏ ସଂଖ୍ୟା ତାଲିକାରେ ଥିବା ସର୍ବାଧିକ ମୂଲ୍ୟର ନିରୂପଣପାଇଁ ବିଚାର କର। ଏହି ତାଲିକାର ସର୍ବାଧିକ ମୂଲ୍ୟ ହେଉଛି ପ୍ରଥମ ସଂଖ୍ୟା ଅଥବା ଅବଶିଷ୍ଟ ସଂଖ୍ୟାଗୁଡ଼ିକ ମଧ୍ୟରେ ବୃହତ୍ତମ ସଂଖ୍ୟା।

ତାଲିକାରେ କେବଳ ଗୋଟିଏ ମାତ୍ର ସଂଖ୍ୟା ଥିବା ହେଲା ଏହି ସମସ୍ୟାର ମୂଳ ଅବସ୍ଥା। ସଂଜ୍ଞା ଅନୁଯାୟୀ, ଯଦି କେବଳ ଗୋଟିଏ ସଂଖ୍ୟା ଥାଏ, ତେବେ ଏହା ହିଁ ସର୍ବବୃହତ।

2. **ରିକର୍ସିଭ୍ ପଦକ୍ଷେପ (Recursive Step):** ଏଥିରେ ଫଙ୍କ୍ସନଟି ନିଜକୁ ଏକ ବା ଏକାଧିକ ଥର ଡାକି (recursive call) ପରିଣାମ ଗଣନା କରିବା, କିନ୍ତୁ ଚରର ଆକାର/ମୂଲ୍ୟ ପ୍ରତ୍ୟେକ ଥର ହ୍ରାସ ପାଇଥାଏ, ଅର୍ଥାତ୍, ମୂଳ ଅବସ୍ଥାର ନିକଟତର ହୋଇଥାଏ।

   ସଂଖ୍ୟାର ଏକ ତାଲିକାରେ ସର୍ବାଧିକ ମୂଲ୍ୟ ଖୋଜିବାର ଉପରୋକ୍ତ ଉଦାହରଣରେ, ରିକର୍ସିଭ୍ ପଦକ୍ଷେପ ହେଉଛି ସଂଖ୍ୟାର ଅବଶିଷ୍ଟ ତାଲିକାରେ ସର୍ବାଧିକ ମୂଲ୍ୟ ଖୋଜିବା, ଅର୍ଥାତ୍ ଆକାର ହ୍ରାସ ପାଇଥିବା ତାଲିକା (ପୂର୍ବ ଆକାରଠାରୁ 1 କମ୍)।

ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍ସନର କିଛି ଜଣାଶୁଣା ଉଦାହରଣ ନେବା।

7.2.1 ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ଫଙ୍କ୍ସନ (Factorial Function)

ଏକ ଧନାତ୍ମକ ସଂଖ୍ୟା n ର ଫ୍ୟାକ୍ଟୋରିଆଲ୍ (factorial), n! ଭାବରେ ଲେଖାଯାଇଥାଏ, ଯାହାକି 1 ରୁ n ପର୍ଯ୍ୟନ୍ତ ଧନାତ୍ମକ ସଂଖ୍ୟା ଗୁଡିକର ଗୁଣନଫଳ:

$n! = 1 \times 2 \times 3 \times \ldots \times (n-2) \times (n-1).n$ *where as* $0! = 1$

ନିମ୍ନମତେ, ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ଫଙ୍କ୍ସନପାଇଁ ଆଇଟରେଟିଭ ସଂସ୍କରଣର ସଂଜ୍ଞା ଲେଖାଯାଇପାରିବ।

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ \prod_{i=1}^{n} i & \text{if } n > 0 \end{cases}$$

ଉପରୋକ୍ତ ସଂଜ୍ଞାରୁ, ଆମେ ପାଇପାରିବା

$$0! = 1$$
$$1! = 1$$
$$2! = 1 \times 2 = 2$$
$$3! = 1 \times 2 \times 3 = 6$$
$$4! = 1 \times 2 \times 3 \times 4 = 24$$
$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$ ଏବଂ ଏହିପରି।

ନିରୀକ୍ଷଣ କର ଯେ

$4! = 4 \times 3! = 4 \times 6 = 24$ and $5! = 5 \times 4! = 5 \times 24 = 120$

ଏବଂ ଏହା ପ୍ରତ୍ୟେକ ଧନାତ୍ମକ ପୂର୍ଣ୍ଣାଙ୍କ ସଂଖ୍ୟା n ପାଇଁ ସତ୍ୟ ଅଟେ

$$n! = n \times (n-1)!$$

ଏହିପରି, ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ଫଙ୍କ୍ସନକୁ ବ୍ୟାଖ୍ୟା କରାଯାଇପାରିବ

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times n-1)! & \text{if } n > 0 \end{cases}$$

ଏହା ନିଜକୁ ନିଜେ ସୂଚିତ କରି ଥିବା ହେତୁ ଏହି ସଂଜ୍ଞାର $n!$ ହେଉଛି ରିକର୍ସନ ଏବଂ ଏହା $(n-1)!$ ବ୍ୟବହାର କରେ ।

Listing 7.1

```
/*
    Program to print the factorials of first 'n' natural numbers
    using recursive function
 */



 #include <stdio.h>
 int factorial( int n );          /*  function prototype   */
 int main()
 {
    int i, n;
    printf( "\nEnter value of n: " );
    scanf( "%d", &n );
    for ( i = 1; i <= n; i++ ) {
        printf( "\n%d! = %d", i, factorial(i) );
    }
    return 0;
 }
 /*
    Recursive function to compute factorial of a number
 */
 int factorial( int n )
 {
   if ( n == 0 )
      return 1;
   else
      return ( n * factorial( n - 1 ) );
 }
```

Test Run

```
Enter value of n: 6
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
```

7.2.2 ଫିବୋନାକି (Fibonacci) ସଂଖ୍ୟାଗୁଡିକ

ଫିବୋନାକି କ୍ରମ ଏକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ କ୍ରମ ଅଟେ, ଯାହାକି ସାଧାରଣତଃ F0, F1, . . ., Fn, ଦ୍ୱାରା ନିମ୍ନମତେ ସୂଚିତ ହୋଇଥାଏ

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

ଯାହାକି ଏବଂ ଏବଂ ପରବର୍ତ୍ତୀ ସଂଖ୍ୟାଗୁଡିକ ପୂର୍ବବର୍ତ୍ତୀ ଦୁଇଟି ସଂଖ୍ୟାର ସମଷ୍ଟି ଅଟେ ।

ଉଦାହରଣ ସ୍ୱରୂପ, ଉପରୋକ୍ତ କ୍ରମରେ ପରବର୍ତ୍ତୀ ସଂଖ୍ୟାଟି ହେଲା

$$8 + 13 = 21$$

ଫିବୋନାକି ସଂଖ୍ୟାପାଇଁ ଏକ ଔପଚାରିକ ରିକର୍ସିଭ୍ ସଂଜ୍ଞା ହେଲା:

$$fib(n) = \begin{cases} n & \text{if } n \le 1 \\ fib(n-1) + fib(n-2) & \text{if } n > 1 \end{cases}$$

ଲକ୍ଷ୍ୟ କର ଯେ ଫିବୋନାକି ସଂଖ୍ୟାଗୁଡ଼ିକର ରିକର୍ସିଭ୍ ସଂଜ୍ଞା ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ଫଙ୍କ୍ସନର ରିକର୍ସନ ସଂଜ୍ଞାଠାରୁ ପୃଥକ୍ ଅଟେ, ଏହି ଅର୍ଥରେ ଯେ ଏହା ନିଜକୁ ନିଜେ ଦୁଇଥରପାଇଁ ଡାକିଥାଏ ।

Listing 7.2

```
/*
    Program to print first 'n' terms of Fibonacci sequence
    using recursive functions
*/

#include <stdio.h>

int fib( int n );       /*  function prototype   */

int main()
{
   int i, m;
   printf( "\nEnter value of m: " );
   scanf( "%d", &m );
   for ( i = 0; i < m; i++ )
   {
      printf( "%d ", fib(i) );
   }
   return 0;
}
```

```
/*
    Recursive function to compute Fibonacci number 'n'
*/

int fib( int n )
{
  if ( n <= 1 )
     return n;
  else
     return ( fib(n - 1) + fib(n - 2) );
}
```

Test Run

```
Enter value of m: 8
 0   1   1   2   3   5   8   13
```

7.2.3 ଆକରମନ୍ (Ackermann) ଫଙ୍କ୍ସନ

m ଏବଂ *n* ର ସମସ୍ତ ଧନାତ୍ମକ ମୂଲ୍ୟପାଇଁ ଆକରମନ୍ ଫଙ୍କ୍ସନ ରିକର୍ସିଭ୍ରୂପେ ନିମ୍ନରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଛି:

$$A(m,n)=\begin{cases} n+1 & \text{if } m=0 \\ A(m-1,1) & \text{if } n=0 \\ A(m-1,A(m,n-1)) & \text{Otherwise} \end{cases}$$

Listing 7.3

```
/*
    Program to compute Ackermann function
*/

#include <stdio.h>

int ackermann(int m, int n);     /*  function prototype   */

int main()
{
   int m, n;
   printf( "\nEnter value of m : " );
   scanf( "%d", &m );
   printf( "\nEnter value of n : " );
   scanf( "%d", &n );
   printf( "\nA(%d,%d) = %d\n", m, n, akermann(m,n) );
   return 0;
}
```

```
/*  function definition to compute ackermann function A(m,n)  */
 int ackermann(int m, int n)
 {
   if ( m == 0 )
       return (n+1);
   else if ( n == 0 )
      return ackermann(m-1, 1);
   else
      return ackermann(m-1, ackermann(m,n-1));
 }
```

Test Run

```
Enter value of m : 1
Enter value of n : 3
A(1,3) = 5
```

7.3 କ୍ୱିକ୍ ସର୍ଟ ଆଲଗୋରିଦମ୍ (QUICK SORT ALGORITHM)

କ୍ୱିକ୍ ସର୍ଟ ହେଉଛି ଏକ ଷ୍ଟାର୍ଟିଂ ଆଲଗୋରିଦମ୍ ଯାହାକି ବିଭାଜନ ଏବଂ ବିଜୟ ଧାରଣାର ବ୍ୟବହାର କରି ଗୋଟିଏ ଆରେର ଉପାଦାନକୁ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟକ୍ରମରେ ସଜାଇଥାଏ । ଏହି ଆଲଗୋରିଦମ୍ ଆରେରୁ ପାଇଭଟ୍ (pivot) ନାମକ ଏକ ଉପାଦାନ ଖୋଜିଥାଏ, ଯାହାକି ଆରେକୁ ଏପରି ଦୁଇ ଭାଗରେ ବିଭକ୍ତ କରେ ଯେପରି କି ବାମ ଉପ-ଆରେରେ ଥିବା ଉପାଦାନଗୁଡ଼ିକ ଏହାଠାରୁ କମ୍ ଏବଂ ଡାହାଣ ଉପ-ଆରେରେ ଥିବା ଉପାଦାନଗୁଡ଼ିକ ଏହାଠାରୁ ଠାରୁ ଅଧିକ ହେଉଥିବ । ତା'ପରେ ଏହି ଦୁଇଟି ଉପ-ଆରେକୁ ପୁଣି ପୃଥକଭାବେ ଉପରୋକ୍ତ ପଦ୍ଧତିରେ ସଜାଯାଏ । ଏହି ପଦ୍ଧତିଟି ଏକ ରିକର୍ସିଭ୍ ପଦ୍ଧତିର ଯାହାର ମୂଳ ଅବସ୍ଥା ହେଲା ଯେତେବେଳେ ଆରେରେ ଉପାଦାନ ସଂଖ୍ୟା ମାତ୍ର 1 ।

ଧରାଯାଉ start ଏବଂ end ଚଳ ଆରେର ପ୍ରଥମ ଏବଂ ଶେଷ ଉପାଦାନର ଇଣ୍ଡେକ୍ସକୁ ସୂଚାଉଛି, ତା'ହେଲେ କ୍ୱିକ୍ ସର୍ଟକୁ ନିମ୍ନମତେ ରିକର୍ସିଭ୍ ପଦ୍ଧତିରେ ବ୍ୟାଖ୍ୟା କରାଯାଇପାରେ:

```
If ( start < end ) then
    Partition the array into two halves
    Quicksort the left half
    Quicksort the right half
 Endif
```

ପାଇଭଟ୍ ଉପାଦାନ ଚୟନ ପ୍ରକ୍ରିୟାନୁସାରେ କ୍ୱିକ୍ ସର୍ଟ ଆଲଗୋରିଦମ୍‌ର କିଛି ଭିନ୍ନତା ନିମ୍ନରେ ଦିଆଯାଇଅଛି:

- ପ୍ରଥମ ଉପାଦାନକୁ ପାଇଭଟ୍ ଭାବରେ
- ଶେଷ ଉପାଦାନକୁ ପାଇଭଟ୍ ଭାବରେ
- ଲକ୍ଷ୍ୟହୀନ ଯେକୌଣସି ଉପାଦାନକୁ ପାଇଭଟ୍ ଭାବରେ
- ମଧ୍ୟବର୍ତ୍ତୀ ଉପାଦାନକୁ ପାଇଭଟ୍ ଭାବରେ

ଆମ ଉଦାହରଣରେ ଆମେ ଶେଷ ଉପାଦାନଟିକୁ ପାଇଭଟ୍ ଭାବରେ ବିବେଚନା କରିବା ।

ଆରମ୍ଭରେ, ଆମେ ନେବା:

```
plndex = start
 pivot = a[end]
```

ବର୍ତ୍ତମାନ, ଆମେ ଇଣ୍ଡେକ୍ସପାଇଁ ଚଳ i କୁ ବ୍ୟବହାର କରି start ରୁ (end-1) ପର୍ଯ୍ୟନ୍ତ ନିମ୍ନ କୋଡ୍‌ଖଣ୍ଡକୁ ପୁନରାବୃତ୍ତି କରିବା:

```
If (aa[i] < pivot) then
swap a[i] and a[plndex]
Increment plandex
Endif
```

ଏବଂ, ଶେଷରେ

```
swap a[plnxdex] and  a [end]
```

ଉପରୋକ୍ତ କୋଡ୍‌ଖଣ୍ଡଗୁଡ଼ିକୁ ଏକତ୍ର କରି ପ୍ରସ୍ତୁତ ନିମ୍ନ ଫଙ୍କ୍‌ସନଟି ବିଭାଜନ କାର୍ଯ୍ୟର ସଂପାଦନ କରେ ।

```
int partition(int a[], int start, int end)
 {
   int i, temp;
   int pIndex = start;
   int pivot  = a[end];
   for( i = start; i < end; i++ )
   {
      if ( a[i] < pivot )
      {
          temp = a[i];
          a[i] = a[pIndex];
          a[pIndex] = temp;
          pIndex++;
      }
   }
   temp = a[end];
   a[end] = a[pIndex];
   a[pIndex] = temp;

   return pIndex;
 }
```

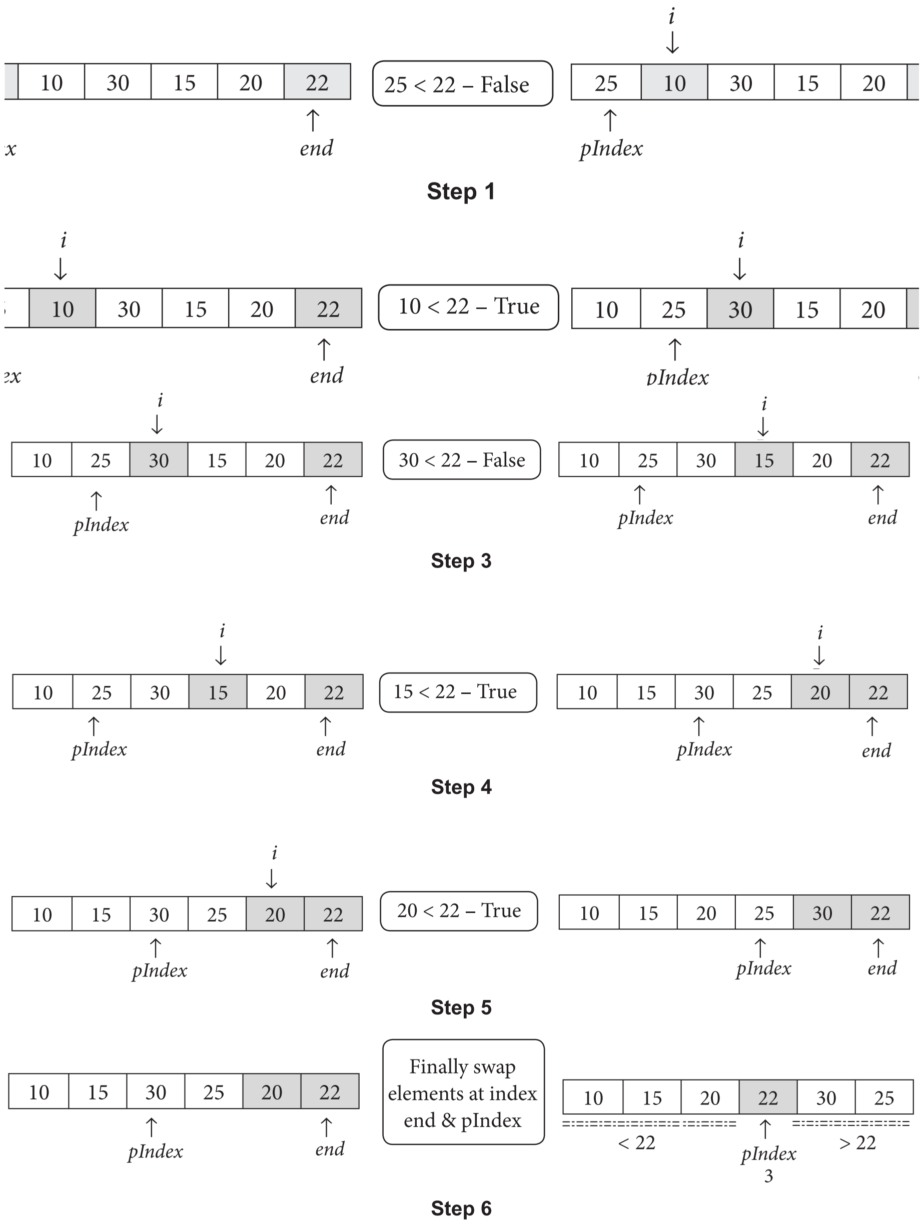
ଉଦାହରଣ 7.1: ବିଭାଜନ ପ୍ରକ୍ରିୟାକୁ ବୁଝିବାପାଇଁ, ନିମ୍ନପ୍ରଦତ୍ତ ଆରେକୁ (array) ବିଚାରକୁ ନିଅ

25 10 30 15 20 22

ସମାଧାନ: ଦିଆଯାଇଥିବା ଆରେରେ, *start = 0, end = 5*

ଶେଷ ଉପାଦାନକୁ (22) ପାଇଭଟ୍ ନିଆଯାଉ

ଆରମ୍ଭ କରିବାକୁ, ଆମେ ନେବା *pIndex = start, i = start*



ଚିତ୍ର . 7.1: ଆରେ ବିଭାଜନର ଉଦାହରଣ

ପାଇଭଟ୍ ଉପାଦାନ 22 ଏହାର ଅନ୍ତିମସ୍ଥିତିରେ ରଖାଯାଇଛି ଏବଂ ଏହା ଆରେକୁ ନିମ୍ନମତେ ଦୁଇଟି ଉପ-ଆରେରେ ବିଭକ୍ତ କରୁଛି।

10 15 20 ଏବଂ 30 25

ଯେପରି ଦେଖୁଛ, ବାମ ଉପ-ଆରେରେ ଥିବା ଉପାଦାନଗୁଡିକ 22 ରୁ ଛୋଟ, ଏବଂ ଡାହାଣ ଉପ-ଆରେରେ ଥିବା ଉପାଦାନଗୁଡିକ 22 ରୁ ଅଧିକ।

ଏହାର ଅର୍ଥ ହେଉଛି ଉପାଦାନ 22 ଏହାର ଅନ୍ତିମ ସ୍ଥିତିରେ ଠିକ୍ ଭାବେ ରଖାଯାଇଛି ଏବଂ ଏହା ଅବଶିଷ୍ଟ ଉପାଦାନଗୁଡ଼ିକୁ ଏପରି ଦୁଇଟି ଉପ-ଆରେରେ ବିଭାଜିତ କରୁଛି ଯେପରି କି ବାମ ଉପ-ଆରେରେ ଉପାଦାନଗୁଡ଼ିକ ଏହାଠାରୁ କମ୍ ଏବଂ ଡାହାଣ ଉପ-ଆରେରେ ଥିବା ଉପାଦାନଗୁଡ଼ିକ ଅଧିକ ହେଉଥିବ।

ଉପ-ଆରେରେ 2 କିମ୍ବା ଅଧିକ ଉପାଦାନ ଥିଲେ ଉପରୋକ୍ତ ବିଭାଜନ ପଦକ୍ଷେପଟି ପ୍ରତ୍ୟେକ ଉପ-ଆରେପାଇଁ ପୁନରାବୃତ୍ତି ହୁଏ। ଯେହେତୁ, ଆମେ ଗୋଟିଏ ସମୟରେ ଗୋଟିଏ ଉପ-ଆରେରେ କାମ କରିପାରିବା, ଆମେ ଦ୍ୱିତୀୟ ଉପ-ଆରେଉପରେ ମଧ୍ୟ ନଜର ରଖିବା ଆବଶ୍ୟକ। ଏହି କାର୍ଯ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ଭାବରେ ଷ୍ଟାକ୍‌ର ବ୍ୟବହାରଦ୍ୱାରା (ଆଇଟରେଟିଭ ପଦ୍ଧତିରେ) କିମ୍ବା ଉହ୍ୟଭାବେ (ରିକର୍ସିଭ୍ ପଦ୍ଧତିରେ) କରାହୁଏ। ନିମ୍ନରେ ଉଭୟ ପ୍ରକାରର କୋଡ୍ ଦିଆଯାଇଛି।

Listing 7.4

```
/*
     Program to sort an array of integers in ascending order using
     Quick sort method
 */
 #include<stdio.h>
 /*  function prototypes  */
 void quickSortRecursive(  int a[], int lb, int ub );
 int partition(int a[], int start, int end);
 int main()
 {
    int i, n, a[20];
    printf( "\nEnter size of array n(<=20) : " );
    scanf( "%d", &n );
    printf( "\nEnter %d integer elements of array\n\n", n );
    for ( i = 0; i < n; i++ )
       scanf( "%d", &a[i] );
    quickSortRecursive( a, 0, n-1 );
    printf( "\n\nSorted list of elements\n\n" );
    for ( i = 0; i < n; i++ )
       printf( "%d ", a[i] );
    printf( "\n");
 } /*---- end of main function ----*/

 void quickSortRecursive( int a[], int lb, int ub )
 {
    int pIndex;
    if ( lb < ub )
    {
       pIndex = partition( a, lb, ub);
       quickSortRecursive( a, lb, pIndex-1 );
       quickSortRecursive( a, pIndex+1, ub );
   }
 }

 int partition(int a[], int start, int end)
 {
    int i, temp;
    int pIndex = start;
    int pivot  = a[end];
```

```
    for(i = start; i < end; i++)
    {
       if (a[i] < pivot)
       {
           temp = a[i];
           a[i] = a[pIndex];
           a[pIndex] = temp;
           pIndex++;
       }
    }
 temp = a[end];
    a[end] = a[pIndex];
    a[pIndex] = temp;
    return pIndex;
 }
```

Test Run

```
Enter size of array n(<=20) : 6
Enter 6 integer elements of array
25 10 30 15 20 28
Sorted list of elements
10 15 20 25 28 30
```

7.4 ମର୍ଜ ସର୍ଟ ଆଲଗୋରିଦମ୍ (MERGE SORT ALGORITHM)

ମର୍ଜ ସର୍ଟ ହେଉଛି ଅନ୍ୟ ଏକ ସର୍ଟିଂ ଆଲଗୋରିଦମ୍ ଯାହା ବିଭାଜନ ଏବଂ ବିଜୟ ଧାରଣାର ବ୍ୟବହାର କରେ । ଏହି ଆଲଗୋରିଦମ୍ ଆରେକୁ ସମାନ ଦୁଇଭାଗରେ ବିଭକ୍ତ କରେ, ସେଗୁଡ଼ିକୁ ପୃଥକ ଭାବରେ ସଜାଏ, ଏବଂ ତା'ପରେ ସେଗୁଡିକୁ ଏକତ୍ର କରେ ।

ଏହା ଏକ ରିକର୍ସିଭ୍ ପଦ୍ଧତି, ଯାହାର ମୂଳ ଅବସ୍ଥା – ଆରେରେ ଏକ ମାତ୍ର ଉପାଦାନ ଅବସ୍ଥିତ ।

ଧରାଯାଉ ଚଳ *beg* ଏବଂ end ଯଥାକ୍ରମେ ଆରେର ପ୍ରଥମ ଏବଂ ଶେଷ ଉପାଦାନର ଇଣ୍ଡେକ୍ସକୁ ସୂଚାଉଛି, ମର୍ଜ ସର୍ଟକୁ ନିମ୍ନମତେ ରିକର୍ସିଭ୍ ଭାବରେ ବ୍ୟାଖ୍ୟା କରାଯାଇପାରିବ ।

```
If ( beg < end ) then
    Divide the list into two halves
    Mergesort the left half
    Mergesort the right half
    Merge the two-sorted halves into one sorted list
 Endif
```

ଉଦାହରଣ 7.2: ମର୍ଜ ସର୍ଟ ପଦ୍ଧତିର କାର୍ଯ୍ୟ ବର୍ଣ୍ଣନା କରିବା ପାଇଁ, 7 ଟି ଉପାଦାନ ସହିତ ନିମ୍ନ ଆରେ a କୁ ବିଚାର କର

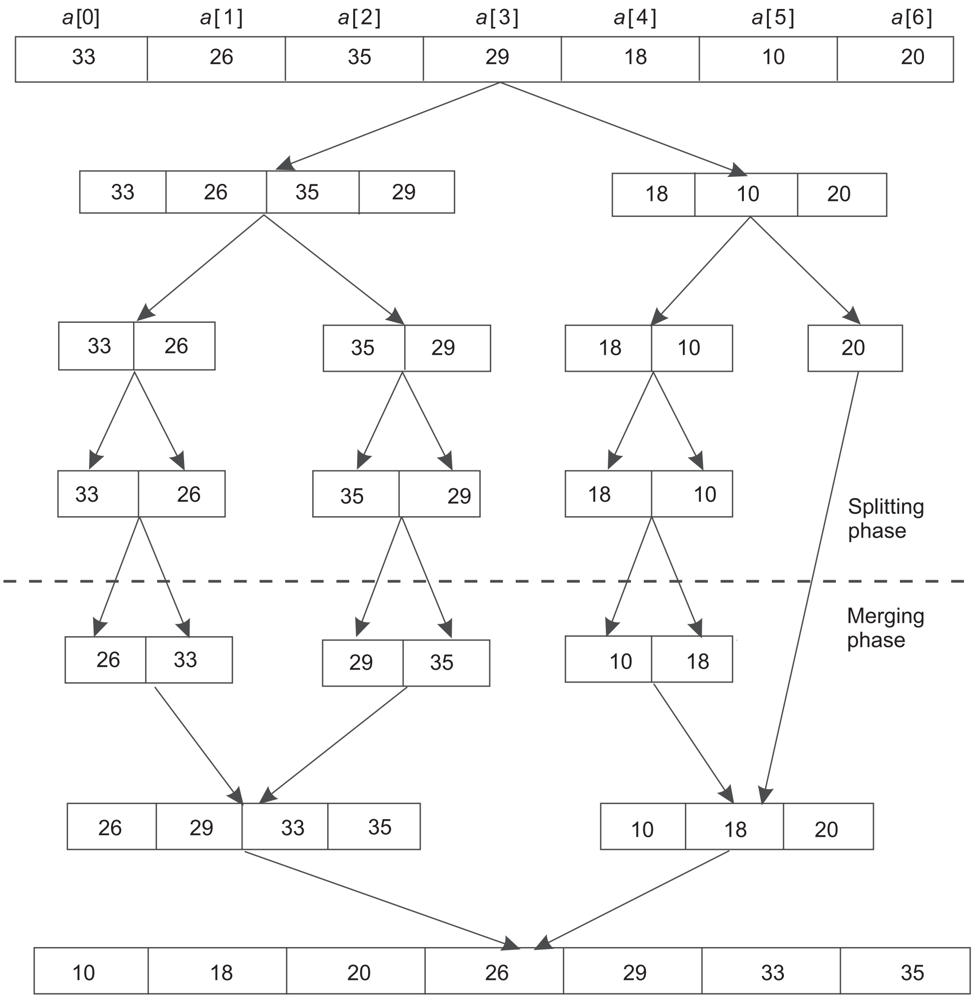
33, 26, 35, 29, 18, 10, 24

ସମାଧାନ: ମର୍ଜ ସର୍ଟର ପ୍ରଥମ ପଦକ୍ଷେପ ହେଉଛି ଆରେକୁ ଦୁଇଟି ସମାନ ଉପ-ଆରେରେ ବିଭକ୍ତ କରିବା । ଏହିପରି ଆମେ ଆରେକୁ ବିଭକ୍ତ କରୁ

33, 26, 35, 29 ଏବଂ 18, 10, 24

ଏବଂ ପ୍ରଥମେ ବାମ ଉପ-ଆରେକୁ ବିଚାର କର । ଏହା ପୁନର୍ବାର ଦୁଇଟି ଉପ-ଆରେରେ ବିଭକ୍ତ ହୋଇଛି

33, 26 ଏବଂ 35, 29



ଚିତ୍ର. 7.2: ମର୍ଜ ସର୍ଟର କ୍ରମାଗତ ପଦକ୍ଷେପଗୁଡ଼ିକର ଚିତ୍ରଣ

ଉପରୋକ୍ତ ପ୍ରତ୍ୟେକ ଉପ-ଆରେ ପାଇଁ, ଆମେ ପ୍ରତ୍ୟେକ ଆରେ ଏକ ଉପାଦାନ ବିଶିଷ୍ଟ ହେଲା ପର୍ଯ୍ୟନ୍ତ ବାରମ୍ବାର ସମାନ ବିଭାଜନ ପଦ୍ଧତି ପ୍ରୟୋଗ କରୁ । ଉପ-ଆରେର ଆକାର ଏକ ହେଲେ, ସେଥିରେ ଆଉ କୌଣସି ସର୍ଟିଂର ଆବଶ୍ୟକତା ନଥାଏ । ଶେଷରେ, ଆମେ ଏକ ସଜାଯାଇଥିବା ଆରେ ପାଇବାପାଇଁ ଉପ-ଆରେଗୁଡ଼ିକର ଏକତ୍ରୀକରଣ ଆରମ୍ଭ କରୁ ।

ଉପ-ଆରେ 33 ଏବଂ 26 ମର୍ଜ କଲେ ସଜ୍ଜିତ ଆରେ 26, 33 ମିଳିବ ଏବଂ ଉପ-ଆରେ 35 ଏବଂ 29 ମର୍ଜ କରି ସଜ୍ଜିତ ଆରେ 29, 35 ମିଳେ ।

ପରବର୍ତ୍ତୀ ପଦକ୍ଷେପରେ, ଆମେ ଆକାର ଦୁଇର ଏହି ଦୁଇଟି ସଜ୍ଜିତ ସବ୍-ଆରେକୁ ପୁନର୍ବାର ମର୍ଜ କରି ଆକାର ଚାରିର ଏକ ସଜ୍ଜିତ ଆରେ ପାଉ ।

26, 29, 33, 35

ବର୍ତ୍ତମାନ ଦିଆଯାଇଥିବା ଆରେର ବାମ ଅଧା ସର୍ଟ ହୋଇଯାଇଛି, ଆମେ ଡାହାଣ ଅଧାରେ ମଧ୍ୟ ସମାନ ପଦକ୍ଷେପ ଗ୍ରହଣ କରୁ । ପ୍ରଥମେ, ଆମେ ଏହାକୁ ଦୁଇଟି ଉପ-ଆରେରେ ବିଭକ୍ତ କରୁ ।

18, 10 ଏବଂ 24

ଏଥିମଧ୍ୟରୁ ପ୍ରଥମଟିକୁ ପୁଣି ଏକ ଉପାଦାନ ବିଶିଷ୍ଟ ଦୁଇଟି ଉପ-ଆରେରେ ବିଭକ୍ତ କରାଯାଏ, ତା'ପରେ ସେହି ଦୁଇଟିକୁ ମର୍ଜକରି ଆରେ 10, 18 ମିଳେ । ଦ୍ୱିତୀୟ ଉପ-ଆରେ, 24, ଆକାର ଏକ, ତେଣୁ କୌଣସି ସର୍ଟିଂର ଆବଶ୍ୟକତା ନାହିଁ । ବର୍ତ୍ତମାନ ଏହି ଉପ-ଆରେ ଗୁଡ଼ିକ ମର୍ଜକଲେ ନିମ୍ନ ସଜ୍ଜିତ ଆରେଟି ମିଳିବ

10, 18, 24

ଶେଷରେ, ଆକାର ଚାରି ଏବଂ ତିନୋଟିର ସଜାଯାଇଥିବା ଉପ-ଆରେଗୁଡିକ ନିମ୍ନମତେ ମର୍ଜକରି ସର୍ଟ କରାଯାଏ ।

10, 18, 24, 26, 29, 33, 35

ଉପରୋକ୍ତ ସର୍ଟିଂ ପ୍ରକ୍ରିୟାକୁ ଚିତ୍ର.7.2 ରେ ଦର୍ଶାଯାଇଥିବା ପରି ମଧ୍ୟ ହୃଦୟଙ୍ଗମ କରିହେବ ।

Listing 7.5

```
/*
   Program to sort an array of integers in ascending order using
   merge sort method
*/
#include<stdio.h>
/*----  function prototype  ----*/
void mergeSortMethod(int a[], int beg, int end );
void mergingSorteosQubArrays(int a[], int lb, int lr, int rb, int
rr);

int main()
{
   int i, n, a[20];
   printf( "\nEnter size of array n(<=20) : " );
   scanf( "%d", &n );
   printf( "\nEnter %d integer elements of array\n\n", n );
   for ( i = 0; i < n; i++ )
      scanf( "%d", &a[i] );
```

```
    mergeSortMethod( a, 0, n-1 );
    printf( "\n\nSorted list of elements\n\n" );
    for ( i = 0; i < n; i++ )
    {
       printf( "%d ", a[i] );
    }
    printf( "\n");
 } /*---- end of main function ----*/
 void mergeSortMethod( int a[], int beg, int end )
 {
    int mid;
    if ( beg < end )
    {
        mid = ( beg + end ) / 2;
          mergeSortMethod( a, beg, mid );
          mergeSortMethod( a, mid+1, end );
          mergingSorteosQubArrays( a, beg, mid, mid+1, end );
    }
 }
 void mergingSorteosQubArrays(int a[], int lb, int lr, int  rb, int
rr)
 {
  int na, nb, nc, k, c[MAX];
na = lb;
nb = rb;
    nc = lb;
    while ( ( na <= lr ) && ( nb <= rr ) )
    {
       if ( a[na] < a[nb] )
            c[nc] = a[na++];
         else
            c[nc] = a[nb++];
         nc++;
    }
    if ( na > lr ) {
       while ( nb <= rr )
          c[nc++] = a[nb++];
    } else {
       while ( na <= lr )
            c[nc++] = a[na++];
    }
    for ( k = lb; k <= rr; k++ )
       a[k] = c[k];
 }
```

Test Run

```
Enter size of array n(<=20) : 7
Enter 7 integer elements of array
33 26 35 29 18 10 20
Sorted list of elements
10 18 20 26 29 33 35
```

7.5 ରିକର୍ସନ, ଆଇଟରେସନ କିମ୍ବା . . .

ଗୋଟିଏ ସମସ୍ୟାର ସମାଧାନ କରିବାର ଅନେକ ଉପାୟ ଥିବାବେଳେ ଏକ ନିଶ୍ଚିତ ପ୍ରଶ୍ନ ହେଉଛି ଯେ କେଉଁ ଗୋଟିକୁ ଚୟନ କରାଯିବା ଉଚିତ୍ ?

ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଉପାୟ ଚୟନ କରିବାକୁ ପୂର୍ବ-ନିର୍ଦ୍ଧାରିତ ନିୟମର କୌଣସି ତାଲିକା ନାହିଁ; ତଥାପି, ଆମେ ନିମ୍ନଲିଖିତ ପ୍ରସଙ୍ଗଗୁଡିକ ଉପରେ ବିଚାର କରିବା ଆବଶ୍ୟକ:

1. ପ୍ରକ୍ରିୟାକରଣପାଇଁ ଆବଶ୍ୟକ ହେଉଥିବା ସମୟ ।
2. ବ୍ୟବହୃତ ହେଉଥିବା କମ୍ପ୍ୟୁଟର ମେମୋରି ।
3. ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବା ପାଇଁ ଆବଶ୍ୟକ ସମୟ ।
4. ପ୍ରୋଗ୍ରାମ୍ ଡିବଗ୍/ତ୍ରୁଟିଶୂନ୍ୟ ନିମିତ୍ତ ଆବଶ୍ୟକ ସମୟ ।
5. ପ୍ରୋଗ୍ରାମ୍‌ର ରକ୍ଷଣାବେକ୍ଷଣପାଇଁ ଆବଶ୍ୟକ ହେଉଥିବା ସମୟ ।

ସାଧାରଣ ଭାବରେ, ରିକର୍ସିଭ୍ ସମାଧାନ ଉପରୋକ୍ତ ପ୍ରସଙ୍ଗ 3, 4, ଏବଂ 5 ପାଇଁ ଭଲ ପ୍ରଦର୍ଶନ ଦେଇଥାଏ । କାରଣ ରିକର୍ସିଭ୍ ସମାଧାନ ସରଳ ଏବଂ ଛୋଟ ହୋଇଥାଏ, ପରେ ସମାନ ସମସ୍ୟାର ପ୍ରୋଗ୍ରାମଗୁଡିକ ପରିବର୍ତ୍ତନ କରିବାକୁ ଆବଶ୍ୟକ ସମୟ, ଅଣ-ରିକର୍ସିଭ୍ ସମାଧାନପାଇଁ ଆବଶ୍ୟକ ସମୟଠାରୁ କମ୍ ହୁଏ, ଯଦି ଏହା ଉପଲବ୍ଧ ହୋଇଥାଏ ।

ଅପରପକ୍ଷରେ, ସାଧାରଣତଃ, ରିକର୍ସିଭ୍ ସମାଧାନଗୁଡିକ ପ୍ରକ୍ରିୟାକରଣ ସମୟର ବ୍ୟବହାର ଏବଂ ସେଗୁଡ଼ିକ ଆବଶ୍ୟକ କରୁଥିବା କମ୍ପ୍ୟୁଟର ମେମୋରିର ପରିମାଣପାଇଁ ଭଲ ପ୍ରଦର୍ଶନ କରେ ନାହିଁ ।

ଟେବୁଲ୍ 7.1 ରିକର୍ସନ ଏବଂ ଆଇଟରେସନର ମୁଖ୍ୟ ତୁଳନାତ୍ମକ ବିନ୍ଦୁଗୁଡ଼ିକ ଆଲୋକପାତ କରୁଛି

ଟେବୁଲ୍ 7.1: ତୁଳନା: ରିକର୍ସନ ବନାମ ଆଇଟରେସନ

ମାନଦଣ୍ଡ	ରିକର୍ସନ	ଆଇଟରେଟିଭ
ପ୍ରୋଗ୍ରାମ ଲେଖା ଶୈଳୀ	ଫଙ୍କସନଟି ନିଜକୁ ନିଜେ ଡାକି	ଲୁପ୍‌ର ବ୍ୟବହାରଦ୍ୱାରା
ସ୍ଥିତି	ଷ୍ଟାକ୍‌ରେ ଗଛିତ ଥିବା ଚର ମୂଲ୍ୟଦ୍ୱାରା ବର୍ଣ୍ଣିତ	ନିୟନ୍ତ୍ରଣ ଚଳର ମୂଲ୍ୟଦ୍ୱାରା ବର୍ଣ୍ଣିତ
ପ୍ରଗତି	ଫଙ୍କସନର ସ୍ଥିତି ମୂଳ ଅବସ୍ଥା ଅଭିମୁଖୀ ହୋଇଥାଏ	ନିୟନ୍ତ୍ରଣ ଚଳର ମୂଲ୍ୟ ଅନ୍ତିମ ମୂଲ୍ୟ ଆଡ଼କୁ ଗତି କରେ
ସମାପ୍ତି	ମୂଳ ଅବସ୍ଥାରେ ପହଞ୍ଚିଲେ	ନିୟନ୍ତ୍ରଣ ଚଳ ସର୍ତ୍ତ ପୂରଣ କରେ

ମାନଦଣ୍ଡ	ରିକର୍ସନ	ଆଇଟରେଟିଭ
କୌଣସି ସମାପ୍ତି ସ୍ଥିତି ନାହିଁ	ମୂଳ ଅବସ୍ଥା ସ୍ଥିରୀକରଣରେ ତ୍ରୁଟି ରହିଲେ ଅନିର୍ଦ୍ଦିଷ୍ଟ କାଳ ପର୍ଯ୍ୟନ୍ତ ରିକର୍ସିଭ୍ କଲ୍ ହୋଇପାରେ, ଫଳସ୍ୱରୂପ ଫଙ୍କସନ ନିଜକୁ ନିଜେ କଲ୍ କରିଥାଏ, ଯାହା ସିଷ୍ଟମକୁ କ୍ରାସ୍ କରିଦେଇପାରେ ।	ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତିରେ, ବୃଦ୍ଧିରେ, କିମ୍ବା ସମାପ୍ତ ଅବସ୍ଥାରେ ଭୁଲ୍ ହେତୁ ଅନିର୍ଦ୍ଦିଷ୍ଟ ଲୁପ୍, ଏବଂ ପ୍ରୋଗ୍ରାମ୍ ଅସୀମ ଭାବରେ କାର୍ଯ୍ୟକାରୀ ହେବ
କୋଡ୍‌ର ଆକାର	ବହୁତ ଛୋଟ ହୁଏ	ବଡ଼ ହୁଏ
କାର୍ଯ୍ୟକାରିତା	କାର୍ଯ୍ୟକାରିତା ମନ୍ଥର ଅଟେ	କାର୍ଯ୍ୟକାରିତା ଦ୍ରୁତ ଅଟେ

ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 7.3: ରିକର୍ସନ ବ୍ୟବହାର କରି ପ୍ରଥମ n ଟି ଗଣନ ସଂଖ୍ୟାର ମିଶାଣଫଳ ବାହାର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ପ୍ରଥମ n ଗଣନ ସଂଖ୍ୟାଗୁଡିକର ମିଶାଣଫଳ ବାହାର କରିବାପାଇଁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କସନ, ଧରାଯାଉ *findSum(n),* ରିକର୍ସିଭ୍ ଭାବେ ବ୍ୟାଖ୍ୟା କରାଯାଇପାରେ

$$\text{findSum}(n) = \begin{cases} 1 & \text{if } n = 1 \\ n + \text{findSum}(n-1) & \text{if } n > 1 \end{cases}$$

Listing 7.6

```
/*
    Program to find the sum of first 'n' natural numbers
    using recursion
 */
 #include<stdio.h>
 int finosQum(int n);   /*  function prototype  */
 int main()
 {
    int n, sum;
    printf("\nEnter value for n : ");
    scanf("%d", &n);
sum = finosQum(n);
    printf("\nSum of first %d natural numbers = %d\n\n", n, sum);
    return (0);
 }
 /*
     definition of recursive function that returns
     sum of first 'n' natural numbers
 */
 int finosQum(int n)
 {
    if ( n == 1 )
       return 1;
    else
       return ( n + finosQum(n-1) );
 }
```

Test Run

```
Enter value for n : 10
Sum of first 10 natural numbers = 55
```

ଉଦାହରଣ 7.4: ରିକର୍ସନ ବ୍ୟବହାର କରି ଏକ ଧନାତ୍ମକ ଘାତାଙ୍କୀ ଘାତ *(x^n)* ଗଣନା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ଏକ ଧନାତ୍ମକ ଘାତାଙ୍କୀ ଘାତ ଫଙ୍କ୍‌ସନକୁ *(x^n)* ରିକର୍ସିଭ୍ ଭାବେ ବ୍ୟାଖ୍ୟା କରାଯାଇପାରେ

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x \times x^{n-1} & \text{if } n > 0 \end{cases}$$

Listing 7.7

```
/*
    Program to find the value of the positive exponential power
    function using recursion
 */
 #include<stdio.h>
 float power(float x, int n);  /*  function prototype  */
 int main()
 {
    int n;
    float x;
    printf("\nEnter value for x : ");
    scanf("%f", &x);
    printf("\nEnter value for n : ");
    scanf("%d", &n);
    printf("\nValue of power function = %.2f\n\n", power(n));
    return (0);
 }
 /*
definition of recursive function that returns value
     positive exponential power function (x^n)
 */
 float power(float x, int n)
 {
    if ( n == 0 )
       return 1;
    else
       return ( x * power(x,n-1) );
 }
```

Test Run

```
Enter value for x : 5.25
Enter value for n : 2
Value of power function = 27.56
```

ଉଦାହରଣ 7.5: ଦୁଇଟି ଗଣନ ସଂଖ୍ୟା m ଏବଂ n ର ଗରିଷ୍ଠ ସାଧାରଣ ଗୁଣନୀୟକ (ଗସାଗୁ) ବାହାର କରିବାପାଇଁ ରିକର୍ସିଭ୍ ଫଙ୍କ୍ସନ ବ୍ୟବହାର କରି ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ଗସାଗୁ (gcd) ଫଙ୍କ୍ସନକୁ ରିକର୍ସିଭ୍ ଭାବରେ ବ୍ୟାଖ୍ୟା କରାଯାଇପାରେ

$$\text{gcd}(m,n) = \begin{cases} n & \text{if } m\%n = 0 \\ \text{gcd}(n, m\%n) & \text{if } m\%n \neq 0 \end{cases}$$

Listing 7.8

```
  /*
     Program to compute GCD of natural numbers m and n
     using recursion
 */

 int gcd( int m, int n );   /*  function prototype */

 void main()
 {
    int m, n;
    printf( "\nEnter value of m : " );
    scanf( "%d", &m );
    printf( "\nEnter value of n : " );
    scanf( "%d", &n );
    printf( "\nValue of GCD(%d,%d) = %d\n", m, n, gcd(mn,n) );
 }

 /*
     definition of recursive function that returns GCD of
     natural numbers 'm' and 'n'
*/
 int gcd( int m, int n )
 {
    if ( m % n == 0 )
       return n;
    else
       return gcd( n, m % n );
 }
```

Test Run

```
Enter value of m: 35
Enter value of n: 125
Value of GCD(35,125) = 5
```

ଉଦାହରଣ 7.6: ରିକର୍ସନ ବ୍ୟବହାର କରି ଦିଆଯାଇଥିବା ଏକ ଦଶମିକ ସଂଖ୍ୟା n କୁ ଏହାର ସମତୁଲ୍ୟ ବାଇନାରି ସଂଖ୍ୟାରେ ରୂପାନ୍ତର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

Listing 7.9

```
/*
   Program to convert decoml number to equivalent binary number
   function using recursion
*/
#include<stdio.h>
void convert(int n);  /*  function prototype  */
int main()
{
   int n;
   printf("\nEnter decimal number : ");
   scanf("%d", &n);
   printf("\nBinary equivalent of %d = ",n);
   convert(n);
   printf("\n");
   return (0);
}
/*
    definition of recursive function that convert
    decimal number to binary
*/
void convert(int n)
{
   if ( n > 0 )
   {
      convert(n/2);
      printf("%d", n%2);
```

Test Run

```
decimal number : 105
Binary equivalent of 105 = 1101001
```

ୟୁନିଟ ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଯାହା ଶିଖିଛୁ

- ପ୍ରୋଗ୍ରାମର ବିକାଶରେ ରିକର୍ସନ ହେଉଛି ଏକ ଶକ୍ତିଶାଳୀ ଧାରଣା ।
- ଯେଉଁ ଫଙ୍କସନ ନିଜକୁ ନିଜେ କଲ୍ କରେ ତାହା ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କସନ ଭାବରେ ଜଣା ।
- C ରେ ରିକର୍ସିଭ୍ ଫଙ୍କସନଗୁଡିକ ସିଧାସଳଖ କାର୍ଯ୍ୟକାରୀ କରାଯାଇଥାଏ ।
- ରିକର୍ସିଭ୍ ଫଙ୍କସନର ଲୋକପ୍ରିୟ ଉଦାହରଣଗୁଡ଼ିକ ମଧ୍ୟରେ ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ଏବଂ ଫିବୋନାକି ସଂଖ୍ୟା ଅନ୍ତର୍ଭୁକ୍ତ ।
- କ୍ୱିକ୍ ସର୍ଟ ହେଉଛି ଏକ ସର୍ଟିଂ ଆଲଗୋରିଦମ୍ ଯାହା ବିଭାଜନ ଏବଂ ବିଜୟ ଧାରଣାର ବ୍ୟବହାର କରେ । ଏହା ଏକ ଉପାଦାନକୁ ପାଇଭଟ୍/ବିଭାଜନକାରୀ ଉପାଦାନ ଭାବରେ ନେଇଥାଏ, ଯାହା ଆରେକୁ ଏପରି ଦୁଇ ଭାଗରେ ବିଭକ୍ତ କରେ ଯେପରିକି ବିଭାଜନକାରୀ ଉପାଦାନ ଠାରୁ ବାମ ଉପ-ଆରେରେ ଥିବା ଉପାଦାନଗୁଡ଼ିକ କମ୍ ଏବଂ ଡାହାଣ ଉପ-ଆରେରେ ଥିବା ଉପାଦାନଗୁଡ଼ିକ ଅଧିକ ହୋଇଥାନ୍ତି । ତା'ପରେ ଏହି ଦୁଇଟି ଉପ-ଆରେକୁ ପୃଥକ ଭାବରେ ସର୍ଟିଂ କରାଯାଇଥାଏ ।
- ମର୍ଜ ସର୍ଟ ହେଉଛି ଅନ୍ୟ ଏକ ସର୍ଟିଂ ଆଲଗୋରିଦମ୍ ଯାହା ବିଭାଜନ ଏବଂ ବିଜୟ ଧାରଣାର ବ୍ୟବହାର କରେ । ଏହି ଆଲଗୋରିଦମ୍ ଆରେକୁ ଦୁଇଭାଗରେ ବିଭକ୍ତ କରେ, ସେମାନଙ୍କୁ ପୃଥକ ଭାବରେ ସର୍ଟିଂ କରାଯାଇଥାଏ, ଏବଂ ତା'ପରେ ସେଗୁଡ଼ିକୁ ମର୍ଜ କରେ ।
- ରିକର୍ସିଭ୍ ସମାଧାନଗୁଡିକ ସାଧାରଣତଃ ପ୍ରୋଗ୍ରାମ୍ ବିକାଶ ଏବଂ ଡିବଗ୍/ତ୍ରୁଟିଶୂନ୍ୟ କରିବାକୁ କମ୍ ସମୟ ନେଇଥାଏ, ଏବଂ ଏହା ଆଇଟରେଟିଭ ସମାଧାନ ଅପେକ୍ଷା ରକ୍ଷଣାବେକ୍ଷଣ କରିବା ସହଜ ଅଟେ ।
- ଅନ୍ୟ ପାର୍ଶ୍ୱରେ, ରିକର୍ସିଭ୍ ସମାଧାନଗୁଡିକ ଅଧିକ ପ୍ରକ୍ରିୟାକରଣ ସମୟ ନେଇଥାଏ ଏବଂ ଆଇଟରେଟିଭ ସମାଧାନ ଅପେକ୍ଷା ଅଧିକ ମେମୋରି ବ୍ୟବହାର କରିଥାଏ ।

ଅନୁଶୀଳନୀ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. ରିକର୍ସନ କ'ଣ?
2. କେତେବେଳେ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କସନ ଭଲ ଭାବରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଛି ବୋଲି କୁହାଯାଏ ?
3. ମୂଳ ଅବସ୍ଥା (base case) ର ଅର୍ଥ ତୁମେ କ'ଣ ବୁଝ ?
4. ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କସନ କିପରି ଆଇଟରେଟିଭ ଫଙ୍କସନ ସହିତ ତୁଳନା କରାଯାଏ?
5. ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କସନକୁ ଆଇଟରେଟିଭ ଫଙ୍କସନରେ ରୂପାନ୍ତର କରିବା ସମ୍ଭବ କି? ଯଦି ହଁ, ଏକ ଉଦାହରଣ ଦିଅ ।
6. ଏକ ଆଇଟରେଟିଭ ଫଙ୍କସନକୁ ରିକର୍ସିଭ୍ ଫଙ୍କସନରେ ରୂପାନ୍ତର କରିବା ସମ୍ଭବ କି? ଯଦି ହଁ, ଏକ ଉଦାହରଣ ଦିଅ ।
7. କ୍ୱିକ୍ ସର୍ଟ (quick sort) ଆଲଗୋରିଦମ୍‌ର କାର୍ଯ୍ୟକୁ ସଂକ୍ଷେପରେ ବର୍ଣ୍ଣନା କର ।
8. ମର୍ଜ ସର୍ଟ (merge sort) ଆଲଗୋରିଦମ୍‌ର କାର୍ଯ୍ୟ ସଂକ୍ଷେପରେ ବର୍ଣ୍ଣନା କର ।
9. ରିକର୍ସନ ସହିତ ଆଇଟରେସନର ତୁଳନା କର ।

ବହୁ ବିକଳ୍ପ ପ୍ରଶ୍ନ

1. ରିକର୍ସନ ବ୍ୟବହାର କରି ନିମ୍ନପ୍ରଦତ୍ତ ମଧ୍ୟରୁ କେଉଁ ସମସ୍ୟାର ସମାଧାନ ହୋଇପାରିବ ନାହିଁ?
   (a) ଏକ ସଂଖ୍ୟାର ଫ୍ୟାକ୍ଟୋରିଆଲ୍ (b) n ତମ ଫିବୋନାକି ସଂଖ୍ୟା
   (c) ଏକ ଷ୍ଟ୍ରିଙ୍ଗର ଦୈର୍ଘ୍ୟ (d) ମୂଳ ଅବସ୍ଥା ବିନା ସମସ୍ୟା
2. ରିକର୍ସନରେ, ଯେଉଁ ଅବସ୍ଥାପାଇଁ ଫଙ୍କସନ ନିଜକୁ କଲ୍ କରିବା ବନ୍ଦ କରିବ ତାହା ହେଉଛି ____
   (a) ସର୍ବୋତ୍ତମ ମାମଲା (b) ସବୁଠାରୁ ଖରାପ ମାମଲା
   (c) ମୂଳ ଅବସ୍ଥା (d) ସେପରି କୌଣସି ଅବସ୍ଥା ନାହିଁ
3. ନିମ୍ନଲିଖିତ ଉକ୍ତି ମଧ୍ୟରୁ କେଉଁ ଉକ୍ତିଟି ସତ୍ୟ?
   (a) ଆଇଟରେସନ ଅପେକ୍ଷା ରିକର୍ସନ ସର୍ବଦା ଭଲ।
   (b) ଆଇଟରେସନ ତୁଳନାରେ ରିକର୍ସନ ଅଧିକ ମେମୋରି ବ୍ୟବହାର କରେ।
   (c) ଆଇଟରେସନ ତୁଳନାରେ ରିକର୍ସନ କମ୍ ମେମୋରି ବ୍ୟବହାର କରେ।
   (d) ଆଇଟରେସନ ସର୍ବଦା ରିକର୍ସନ ଅପେକ୍ଷା ଭଲ ଏବଂ ସରଳ ଅଟେ।
4. ନିମ୍ନ ଲିଖିତ କୋଡ୍ ସ୍ନିପେଟ୍ କାର୍ଯ୍ୟକାରୀ ହେଲେ କ'ଣ ହେବ?

```
void fun()
{
   fun();
}
int main()
{
   fun();
   return 0;

}
```

   (a) କୋଡ୍ ସଫଳତାର ସହ କାର୍ଯ୍ୟକାରୀ ହେବ ଏବଂ କୌଣସି ଆଉଟପୁଟ୍ ସୃଷ୍ଟି ହେବ ନାହିଁ।
   (b) କୋଡ୍ ସଫଳତାର ସହ କାର୍ଯ୍ୟକାରୀ ହେବ ଏବଂ ଅନିୟମିତ ଆଉଟପୁଟ୍ ସୃଷ୍ଟି ହେବ।
   (c) କୋଡ୍ ସଂକଳନ ସମୟରେ ଏକ ତ୍ରୁଟି ଦେଖାଇବ।
   (d) କୋଡ୍ କିଛି ସମୟପାଇଁ ଚାଲିବ ଏବଂ ଷ୍ଟାକ୍ ଓଭରଫ୍ଲୋ ହେବା ପରେ ବନ୍ଦ ହେବ।
5. ନିମ୍ନଲିଖିତ କୋଡ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ?

```
void fun(int n)
{
    if(n == 0)
    return;
    printf("%d ",n);
    fun(n-1);
}
int main()
{
    fun(10);
    return 0;
}
```

   (a) 10 (b) 1
   (c) 1098...10 (d) 1098...1

6. ନିମ୍ନଲିଖିତ କୋଡ୍‌ପାଇଁ ମୂଳ ଅବସ୍ଥା (base case) କ'ଣ ?

```
void fun(int n)
{
     if (n == 0)
     return;
     printf("%d ",n);
     fun(n-1);
}
int main()
{
     fun(10);
     return 0;

}
```

(a) return **(b)** printf("%d ", n)

(c) if(n == 0) **(d)** fun(n-1)

7. ନିମ୍ନଲିଖିତ C କୋଡ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ ?

```
#include<stdio.h>
int main()
{
    printf("Hello");
    main();
    return 0;
}
```

(a) Hello is printed once (b) Hello infinite number of times

(c) Hello is not printed at all (d) 0 is returned

8. ନିମ୍ନଲିଖିତ C କୋଡ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ ?

```
fun(int x)
{
    int b;
    if(x==1)
        return 1;
    else
        b=x*fun(x-1);
        return b;
}
int main()
{
    int n;
    n=fun(4);
    printf("%d",n);
    return 0;
```

(a) 24 (b) 4

(c) 12 (d) 10

9. ନିମ୍ନଲିଖିତ C କୋଡ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
int fun(int x)
{
    if(x==2)
            return 2;
    else
    {
        printf("+");
        fun(x-1);
    }
}
int main()
{
    int n,i;
    n=fun(6);
    printf("%d",n);
    return 0;

}
```

(a) ++++2 (b) +++++2

(c) +++++ (d) 2

10. ନିମ୍ନଲିଖିତ C କୋଡ୍ କାର୍ଯ୍ୟକାରୀ ହେଲେ କେତେ ଥର 'a' ପ୍ରିଣ୍ଟ୍ ହେବ?

```
int fun(int b)
{
    if(b==0)
        return 0;
    else
    {
        printf("a");
        fun(b--);
    }
}
int main()
{
 int a;
 a=fun(10);
  printf("%d",a);
      return 0;
```

(a) 9 ଥର (b) 10 ଥର (c) 0 ଥର d) ଅସୀମ ସଂଖ୍ୟକ ଥର

11. ନିମ୍ନଲିଖିତ C କୋଡ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ?

```
int f(int n)
{
    if(n>0)
        return(n+f(n-2));
}
main()
```

```
{
    int n=10;
    int f(int n);
    printf("%d",f(n));
```

(a) 10 (b) 80 (c) 30 (d) Error

12. ଯଦି x = 4 ଏବଂ y = 5 ହୁଏ ତାହେଲେ ନିମ୍ନଲିଖିତ ଛଦ୍ମ-କୋଡ୍‌ର ଆଉଟପୁଟ୍ ବାହାର କର:

```
int fun(int x, int y)
{
    if(x > 1)
       fun(x - 2, y + 2);
    printf("%d", y);
}
```

(a) 4 5 6 (b) 7 6 5 (c) 9 7 5 (d) 5 7 9

13. n=2 ପାଇଁ ନିମ୍ନଲିଖିତ ଛଦ୍ମ-କୋଡ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ ?

```
int fun ( int n)
{
 if ( n == 4 )
      return n;
   else
      return 2*fun(n+1);
}
```

(a) 2 (b) 16 (c) 8 (d) 4

14. ଇନପୁଟ୍ n =134 ପାଇଁ ନିମ୍ନଲିଖିତ ଛଦ୍ମ-କୋଡ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ ?

```
int fun (int n)
{
   static int a = 0;
   if ( n > 0 ) {
      a = a + 1;
      fun(n/10);
   }
   else
      return a;
}
```

(a) 8 (b) 3 (c) 2 (d) 431

15. ନିମ୍ନଲିଖିତ C ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ କ'ଣ ହେବ ?

```
int f( int x )
{
if ( x <= 0 )
return 1;
return f(x-1) + x;
}
int main()
{
```

```
printf("%d", f(5) );
return 0;
     }
```

(a) 12 (b) 16 (c) 15 (d) 11

16. ନିମ୍ନଲିଖିତ ଫଙ୍କସନ n = 25 ପାଇଁ କ'ଣ ପ୍ରିଣ୍ଟ କରିବ?

```
void fun(int n)
{
    if (n == 0)
   return;
  printf("%d", n%2);
 fun(n/2);
}
```

(a) 11001 (b) 11111 (c) 00000 (d) 10011

17. ଧରାଯାଉ C ରେ ନିମ୍ନଲିଖିତ fun (x, y) ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କସନ। fun (4, 3) ର ମୂଲ୍ୟ କ'ଣ ହେବ?

```
  int fun(int x, int y)
{
   if (x == 0)
       return y;
   return fun(x - 1,  x + y);
}
```

(a) 13 (b) 12 (c) 10 (d) 9

18. ନିମ୍ନଲିଖିତ C ଫଙ୍କସନ n = 25 ପାଇଁ କ'ଣ ପ୍ରିଣ୍ଟ କରିବ?

```
 void fun(int n)
{
    if (n == 0)
       return;
    fun(n/2);
    printf("%d", n%2);
}
```

(a) 11001 (b) 11111 (c) 00000 (d) 10011

19. ଠିକ୍ ଆଉଟପୁଟ୍ ଚୟନ କର।

```
int rec(int num)
{
    return (num) ? num%10 + rec(num/10):0;
}
int main()
{
    printf("%d",rec(4567));
    return 0;
}
```

(a) 4 (b) 12 (c) 22 (d) 21

20. ଠିକ୍ ଆଉଟପୁଟ୍ ଚୟନ କର ।

```
int doSomething(int a, int b)
{
 if (b==1)
    return a;
 else
    return a + doSomething(a,b-1);
}
int main()
{
         int k;
         k = doSomething(2,3);
   printf("%d", k);
   return 0;
}
```

(a) 4 (b) 6 (c) 5 (d) 7

ଉତ୍ତର									
1.	*(d)*	**2.**	*(c)*	**3.**	*(b)*	**4.**	*(d)*	**5.**	*(d)*
6.	*(c)*	**7.**	*(b)*	**8.**	*(a)*	**9.**	*(a)*	**10.**	*(d)*
11.	*(c)*	**12.**	*(c)*	**13.**	*(b)*	**14.**	*(b)*	**15.**	*(b)*
16.	*(d)*	**17.**	*(a)*	**18.**	*(a)*	**19.**	*(c)*	**20.**	*(b)*

ପ୍ରୋଗ୍ରାମିଂ (Programming) ସମସ୍ୟା

1. ପ୍ରଥମ n ଗଣନ ସଂଖ୍ୟାଗୁଡ଼ିକୁ ଓଲଟା କ୍ରମରେ ପ୍ରିଣ୍ଟ କରିବାପାଇଁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍‌ସନ ଲେଖ ।
   ଉଦାହରଣ ସ୍ୱରୂପ, ଯଦି $n = 10$, ଆଉଟପୁଟ୍ *10 9 8 7 6 5 4 3 2 1* ହେବା ଉଚିତ୍ ।
2. ଗୋଟିଏ ଆରେର ସର୍ବବୃହତ ଉପାଦାନ ଖୋଜି ବାହାର କରିବାପାଇଁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍‌ସନ ଲେଖ ।
3. ଗୋଟିଏ ଆରେର ଉପାଦାନଗୁଡ଼ିକର କ୍ରମ ଓଲଟାଇବାପାଇଁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍‌ସନ ଲେଖ ।
4. ଦୁଇଟି ଗଣନ ସଂଖ୍ୟାର ଗରିଷ୍ଠ ସାଧାରଣ ଗୁଣନୀୟକ ବାହାର କରିବାପାଇଁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍‌ସନ ଲେଖ ।
5. ଦୁଇଟି ଗଣନ ସଂଖ୍ୟାର ଗୁଣଫଳ ବାହାର କରିବାପାଇଁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍‌ସନ ଲେଖ ।
6. ଗୋଟିଏ ଗଣନ ସଂଖ୍ୟାର ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ବାହାର କରିବାପାଇଁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍‌ସନ ଲେଖ ।
7. ଗୋଟିଏ ଗଣନ ସଂଖ୍ୟାର ଅଙ୍କଗୁଡ଼ିକର ସମାହାର ରାଶି ବାହାର କରିବାପାଇଁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍‌ସନ ଲେଖ ।
8. ଗୋଟିଏ ଦଶମିକ ସଂଖ୍ୟାକୁ ବାଇନାରି ସଂଖ୍ୟାରେ ରୂପାନ୍ତରିତ କରିବାକୁ ଏକ ରିକର୍ସିଭ୍ ଫଙ୍କ୍‌ସନ ଲେଖ ।

ପ୍ରାକ୍ଟିକାଲ୍ (PRACTICALS)

1. ରିକର୍ସନ ବ୍ୟବହାର କରି ଏକ ଗଣନ ସଂଖ୍ୟା n ର ଫ୍ୟାକ୍ଟୋରିଆଲ୍ ବାହାର କରିବାପାଇଁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   ତାଲିକା 7.1 ର ସହାୟତା ନିଅ
2. ରିକର୍ସନ ବ୍ୟବହାର କରି n ତମ ଫିବୋନାକି ସଂଖ୍ୟା ବାହାର କରିବାପାଇଁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

   ତାଲିକା 7.2 ର ସହାୟତା ନିଅ
3. ରିକର୍ସନ ବ୍ୟବହାର କରି ଏକ ଦଶମିକ ସଂଖ୍ୟାକୁ ଏହାର ସମତୁଲ୍ୟ ବାଇନାରି ସଂଖ୍ୟାରେ ରୂପାନ୍ତର କରିବାପାଇଁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ କର ।

   ତାଲିକା 7.9 ର ସହାୟତା ନିଅ

ଅଧିକ ଜାଣିବାପାଇଁ

ରିକର୍ସନ ହେଉଛି ସମସ୍ୟା ସମାଧାନର ଅନ୍ୟ ଏକ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ବିଷୟ । ଏହା ଅନେକ ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ସମାଧାନପାଇଁ ବ୍ୟାପକ ଭାବରେ ବ୍ୟବହୃତ ହୁଏ ।

ଶିକ୍ଷକ ରିକର୍ସନର ଧାରଣା ବିଷୟରେ ଏକ ବୁଝାମଣା ବିକଶିତ କରିବେ । ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କ ସହିତ ରିକର୍ସିଭ୍ ଫଙ୍କ୍ସନ ବ୍ୟବହାର କରି ସମସ୍ୟାର ସମାଧାନ କରିବେ ବୋଲି ଆଶା କରାଯାଉଛି ।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

୫ ଷ୍ଟ୍ରକ୍ଚର

ଏକକର ବିଶେଷତ୍ୱ

ଏହି ଏକକରେ ଷ୍ଟ୍ରକ୍ଚର ସମ୍ବନ୍ଧୀୟ ବିଷୟବସ୍ତୁଗୁଡ଼ିକ ଉପରେ ଆଲୋଚନା କରାଯାଇଅଛି। ଷ୍ଟ୍ରକ୍ଚରଗୁଡ଼ିକ ଏକ ସତ୍ତା (entity) ସମ୍ବନ୍ଧୀୟ ତଥ୍ୟ ଗଚ୍ଛିତ କରିବାପାଇଁ ବ୍ୟବହୃତ ହୋଇଥାଏ। ସତ୍ତାଟି ଏକ କର୍ମଚାରୀ, ଗ୍ରାହକ, ଯାନବାହାନ କିମ୍ବା ବହି ହୋଇପାରେ। ଏହି ୟୁନିଟରେ ଷ୍ଟ୍ରକ୍ଚରର ବିଭିନ୍ନ ଦିଗ ସମ୍ବନ୍ଧରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଅଛି ଏବଂ ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ସେଗୁଡ଼ିକର ବ୍ୟବହାର ପ୍ରଦର୍ଶିତ ହୋଇଛି।

ଭୂମିକା

ବାସ୍ତବ ଜୀବନରେ ଥିବା ସମସ୍ୟା ପାଇଁ ଆମକୁ କିଛି ତଥ୍ୟ-ସଂଗ୍ରହ ଦରକାର ହୁଏ, ଏହି ନିର୍ଦ୍ଦିଷ୍ଟ ତଥ୍ୟ ଉପାଦାନଗୁଡ଼ିକ (individual data items) ଅଲଗା-ଅଲଗା ପ୍ରକାରର ହୋଇପାରେ। ଉଦାହରଣ ସ୍ୱରୂପ, ଗୋଟିଏ ଶିକ୍ଷାନୁଷ୍ଠାନର ଶିକ୍ଷାର୍ଥୀମାନଙ୍କ ସୂଚନା ସଂରକ୍ଷଣ ନିମନ୍ତେ ରୋଲ୍ ନମ୍ବର, ନାମ, ପିତାଙ୍କ ନାମ, ମାତାଙ୍କ ନାମ, ଜନ୍ମ ତାରିଖ, ଇମେଲ୍-ଆଇଡ଼ି, ମୋବାଇଲ୍ ନମ୍ବର, ଠିକଣା, ପ୍ରତ୍ୟେକ ସେମିଷ୍ଟାରର ପ୍ରତ୍ୟେକ ବିଷୟର ମାର୍କ ବିବରଣୀ, ଶୁଳ୍କ ବିବରଣୀ, ଇତ୍ୟାଦି ନିର୍ଦ୍ଦିଷ୍ଟ ତଥ୍ୟ ଉପାଦାନଗୁଡ଼ିକୁ ଅନ୍ତର୍ଭୁକ୍ତ କରିବାକୁ ହୁଏ। ଏହି ତଥ୍ୟଗୁଡ଼ିକ ବିଭିନ୍ନ ପ୍ରକାରର(data types) ଯେପରିକି ଇଣ୍ଟିଜର୍ (integer), ଲଙ୍ଗ ଇଣ୍ଟିଜର୍ (long integer), ଷ୍ଟ୍ରିଙ୍ଗ୍ (string), ଫ୍ଲୋଟ୍ (float) ଇତ୍ୟାଦି ହୋଇଥାଏ।

ତେଣୁ, ଏହି ପ୍ରକାରର ତଥ୍ୟ ସଂଗ୍ରହକୁ ଏକତ୍ର ରଖିବାପାଇଁ ଆମକୁ ଏକ କୌଶଳ ଆବଶ୍ୟକ, ଏବଂ ଏଥିର ସମାଧାନ ହେଉଛି ଷ୍ଟ୍ରକ୍ଚର।

ଏହି ଏକକରର ବିଦ୍ୟାର୍ଥୀଙ୍କୁ ଷ୍ଟ୍ରକ୍ଚରର ବିଭିନ୍ନ ଦିଗ ବୁଝିବାରେ ସାହାଯ୍ୟ କରିବ ଏବଂ ବାସ୍ତବ ଜୀବନର ସମସ୍ୟା ସମାଧାନପାଇଁ ଷ୍ଟ୍ରକ୍ଚର ବ୍ୟବହାର କରି ପ୍ରୋଗ୍ରାମ୍ ବିକାଶରେ ସହାୟକ ହବ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- ମୌଳିକ ତଥ୍ୟ-ପ୍ରକାର (Basic data types)
- କଣ୍ଡିସନାଲ୍ ବ୍ରାଞ୍ଚିଂ (Conditional branching) / ସର୍ତ୍ତମୂଳକ ଶାଖାତିଆରି
- ଲୁପିଙ୍ଗ୍(Looping)
- ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍ (Arrays and strings) / ସାରଣିସମୂହ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍
- ଫଙ୍କ୍ସନ (Functions)

ଏକକ ଲବ୍ଧ ଜ୍ଞାନ

ଏହି ଏକକର ସମ୍ପୂର୍ଣ୍ଣ ଅଧ୍ୟୟନ ପରେ, ଶିକ୍ଷାର୍ଥୀଗଣ ନିମ୍ନଲିଖିତ ବିଷୟଗୁଡ଼ିକରେ ଜ୍ଞାନ ଆହରଣରେ ସମର୍ଥ ହେବେ ।

U୮-O1: ଷ୍ଟ୍ରକ୍‌ଚରର ଧାରଣା ଉପରେ ବ୍ୟାଖ୍ୟା କରିବା ।

U୮-O2: ଏକ ଷ୍ଟ୍ରକ୍‌ଚରକୁ ବ୍ୟାଖ୍ୟା କରିବାର ବିଭିନ୍ନ ଉପାୟ ବର୍ଣ୍ଣନା କରିବା ।

U୮-O3: ଷ୍ଟ୍ରକ୍‌ଚର ଚଳ ଘୋଷଣା ଏବଂ ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟକରଣ(initialize) ।

U୮-O4: ଷ୍ଟ୍ରକ୍‌ଚରର ଆରେ ଘୋଷଣା ଏବଂ ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟକରଣ ।

U୮-O5: ଷ୍ଟ୍ରକ୍‌ଚର ବ୍ୟବହାର କରି ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାପାଇଁ ପ୍ରୋଗ୍ରାମ୍ ବିକଶିତ କରିବା ।

ୟୁନିଟ୍-୮ ୟୁନିଟଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ *(1 – ଦୁର୍ବଳ ସମ୍ପର୍କ; 2 – ମଧ୍ୟମ ସମ୍ପର୍କ; 3 – ଦୃଢ ସମ୍ପର୍କ)*							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U8-O1	–	–	–	–	–	1	–	–
U8-O2	–	–	–	–	–	2	–	–
U8-O3	–	–	–	–	–	2	–	–
U8-O4	–	–	–	–	–	2	–	–
U8-O5	–	–	–	–	–	3	–	–

8.1 ଉପକ୍ରମ (INTRODUCTION)

ତୁମେ ଏକ ଷ୍ଟ୍ରକ୍‌ଚରକୁ ଦୃଢ଼ଭାବେ ସମ୍ବନ୍ଧିତ ଉପାଦାନଗୁଡ଼ିକର ଏକ ଆରେ ଭାବରେ ଚିନ୍ତା କରିପାରିବ । ଯାହା ହେଉ, ଆରେ ପରି ଏକ ଷ୍ଟ୍ରକ୍‌ଚର ହେଉଛି ସମ୍ବନ୍ଧିତ ତଥ୍ୟ ଉପାଦାନଗୁଡ଼ିକର ଏକ ସମାହାର ଯେଉଁଥିରେ ତଥ୍ୟ ଉପାଦାନର ତଥ୍ୟ-ପ୍ରକାରଗୁଡ଼ିକ ଅଲଗା ଅଲଗା ହୋଇପାରେ । ଆରେ ଭଳି, ଏକ ଷ୍ଟ୍ରକ୍‌ଚର ତିଆରି କରିବା ସମୟରେ ମଧ୍ୟ ତଥ୍ୟ ଉପାଦାନଗୁଡ଼ିକ ସଂଲଗ୍ନ(contiguous) ମେମୋରି ଅବସ୍ଥାନଗୁଡ଼ିକ ଦଖଲ କରେ ।

ଜଟିଳ ତଥ୍ୟକୁ ଅଧିକ ଅର୍ଥପୂର୍ଣ୍ଣ ଉପାୟରେ ସଂଗଠିତ କରିବାପାଇଁ ଷ୍ଟ୍ରକ୍‌ଚରର ବ୍ୟବହାର କରାଯାଇପାରେ ।

ଧରାଯାଉ ଆମେ କୌଣସି ଏକ ସଂଗଠନର କର୍ମଚାରୀଙ୍କ ସୂଚନା ଗଚ୍ଛିତ କରିବାକୁ ଚାହୁଁଛୁ । ଏହି ସୂଚନା ମଧ୍ୟରେ କର୍ମଚାରୀଙ୍କ ନାମ (ଅକ୍ଷର ଆରେ), ବିଭାଗ କୋଡ୍ (ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା), ଦରମା (ଅସ୍ଥାୟୀ-ବିନ୍ଦୁ ସଂଖ୍ୟା), ଏବଂ ଅନ୍ୟାନ୍ୟ ଅନ୍ତର୍ଭୁକ୍ତ ହୋଇପାରେ । ଆମେ ଚାହୁଁ ଯେ ଆମର ପ୍ରୋଗ୍ରାମ୍ ସେମାନଙ୍କୁ ଏକ ଆରେର ଉପାଦାନ ଭାବରେ ବିବେଚନା କରୁ ।

ଗୋଟିଏ ସମ୍ଭାବ୍ୟ ଆଭିମୁଖ୍ୟ ହେଉଛି ଅନେକଗୁଡ଼ିଏ ଆରେ ବ୍ୟବହାର କରିବା ଯେପରି – ନାମପାଇଁ ଅକ୍ଷରର ଏକ ଆରେ, ବିଭାଗୀୟ କୋଡ୍ ପାଇଁ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟାର ଏକ ଆରେ, ଦରମାପାଇଁ ଅସ୍ଥାୟୀ-ବିନ୍ଦୁ ସଂଖ୍ୟାର ଏକ ଆରେ, ଇତ୍ୟାଦି । ଏବଂ ଏକ ସାଧାରଣ ସୂଚକାଙ୍କ ବ୍ୟବହାର କରି ଆମେ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ କର୍ମଚାରୀପାଇଁ ସୂଚନା ପାଇପାରିବା । କିନ୍ତୁ ଏହି ଆଭିମୁଖ୍ୟରୁ ସ୍ପଷ୍ଟ ହେଉଛି ଯେ ଆମେ ଗୋଟିଏ ଏକକ ସତ୍ତା ସହିତ ଜଡ଼ିତ ତଥ୍ୟର ଉପାଦାନଗୁଡ଼ିକ ସହ କାରବାର କରୁଛନ୍ତି । ଏପରି କ୍ଷେତ୍ରରେ ଏହା ଜଣେ କର୍ମଚାରୀ ।

ଏହି ପ୍ରକାରର ସମସ୍ୟାର ସମାଧାନ ପାଇଁ, C ଭାଷା ଏକ ବିଶେଷ ପ୍ରକାର ତଥ୍ୟ ପ୍ରଦାନ କରେ ତାହା ହେଉଛି ଏକ ଷ୍ଟ୍ରକ୍‌ଚର । ସମାବେଶ କରି କରିପାରିବ । ନିମ୍ନରେ ପ୍ରଦତ୍ତ ଷ୍ଟ୍ରକ୍‌ଚରଟିରେ କର୍ମଚାରୀଙ୍କ ନାମ, ବିଭାଗ ନମ୍ବର, ଦରମା, ଏବଂ ଅନ୍ୟାନ୍ୟ ରହିବ ।

ଷ୍ଟ୍ରକ୍‌ଚରର ଆଉ କିଛି ଉଦାହରଣ ନିମ୍ନ ସାରଣୀରେ ଉଲ୍ଲେଖ କରାଯାଇଛି

Name of Structure	Probable Constituent Data Items
Date	day, month, year
Time	hours, minutes, seconds
Book	title, author, publisher, price
Address	name, house number, locality, city, state, pincode
Student	roll number, name, father's name, grade
Customer	name, address, mobile no
Inventory	item code, description, quantity, unit price
Employee	employee id, name, address, mobile no

8.2 ଏକ ଷ୍ଟ୍ରକ୍‌ଚରର ବ୍ୟାଖ୍ୟା

C ଭାଷାରେ ଏକ ଷ୍ଟ୍ରକ୍‌ଚରକୁ ବ୍ୟାଖ୍ୟା କରିବାର ଦୁଇଟି ଉପାୟ ଅଛି: ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର (tagged structure) ଏବଂ ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍‌ଚର (type-defined structure)

ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର (tagged structure)

ଗୋଟିଏ ଷ୍ଟ୍ରକ୍‌ଚରକୁ ବ୍ୟାଖ୍ୟା କରିବାର ପ୍ରଥମ ଉପାୟ ହେଉଛି, ଏକ ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର ବ୍ୟବହାର କରିବା । ଏକ ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର ଘୋଷଣା କରିବାପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
struct STUDENT
{
   . . .
   Field list
   . . .
};
```

(a) ଫର୍ମାଟ୍

```
struct STUDENT
{
   int rollNo;
   char name[20];
   char fname[20];
   char grade;
};
```

(b) ଉଦାହରଣ

ଚିତ୍ର. 8.1: ଏକ ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚରର ବ୍ୟାଖ୍ୟା

ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର କୀୱାର୍ଡରୁ ଆରମ୍ଭ ହୁଏ । ସଂଜ୍ଞାରେ ପରବର୍ତ୍ତୀ ଉପାଦାନ ହେଉଛି ଟ୍ୟାଗ୍ (ଏଠାରେ STUDENT) । ଟ୍ୟାଗ୍ ହେଉଛି ଷ୍ଟ୍ରକ୍‌ଚରର ପରିଚାୟକ, ଏବଂ ଏହାର ବ୍ୟବହାର ଚଳ ଘୋଷଣା, ଫଙ୍କସନର ଚର, ଏବଂ ଫଙ୍କସନର ଫେରସ୍ତ ପ୍ରକାର ଘୋଷଣାପାଇଁ ହୁଏ ।

ଯଦି ଆମେ ଶେଷ କୁଟିଳ ବନ୍ଧନୀ ପରେ ଏକ ସେମିକୋଲୋନ୍ ସହିତ ଷ୍ଟ୍ରକ୍‌ଚର ସଂଜ୍ଞା ସମାପ୍ତ କରୁ, ତେବେ କୌଣସି ଚଳ ଘୋଷଣା ହୁଏ ନାହିଁ । ଏହି କ୍ଷେତ୍ରରେ, ଷ୍ଟ୍ରକ୍‌ଚର କେବଳ ଏକ ପ୍ରକାର ଟେମ୍ପଲେଟ୍ (template) ଯେଉଁଥିରେ କୌଣସି ଗଚ୍ଛିତ ସ୍ଥାନ ସଂପୃକ୍ତ ହୋଇ ନ ଥାଏ ।

ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍‌ଚର (Type-defined Structure)

ଏକ ଷ୍ଟ୍ରକ୍‌ଚରକୁ ବ୍ୟାଖ୍ୟା କରିବାର ଅଧିକ ଶକ୍ତିଶାଳୀ ଉପାୟ ହେଉଛି typedef ବ୍ୟବହାର କରିବା । ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍‌ଚର ଘୋଷଣା କରିବାପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
typedef struct
{
   . . .
   Field list
   . . .
} TYPE;
```

(a) ଫର୍ମାଟ୍

```
typedef struct
{
   int rollNo;
   char name[20];
   char fname[20];
   char grade;
} STUDENT;
```

(b) ଉଦାହରଣ

ଚିତ୍ର. 8.2: ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍‌ଚରର ଘୋଷଣା

ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍‌ଚର ନିମ୍ନଲିଖିତ ଦୃଷ୍ଟିକୋଣରୁ ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର ଠାରୁ ଭିନ୍ନ ହୋଇଥାଏ:

- struct କୀୱାର୍ଡ୍ ପୂର୍ବରୁ typedef କୀୱାର୍ଡ ବ୍ୟବହୃତ ହୁଏ ।
- struct କୀୱାର୍ଡ ପରେ ପରିଚାୟକ ବାଧ୍ୟତାମୂଳକ ନୁହେଁ (ଯେମିତି ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର କ୍ଷେତ୍ରରେ ହୋଇଥାଏ) ।
- ଶେଷ କୁଟିଳ ବନ୍ଧନୀ ପରେ ଏବଂ ସେମିକୋଲୋନ୍ ପୂର୍ବରୁ ପରିଚାୟକ ଆବଶ୍ୟକ ହୋଇଥାଏ ।
- ଆମେ ପରେ ଦେଖିବା ଯେ, ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚରର ଘୋଷଣା ସହିତ ଚଳ ଘୋଷଣା କରାଯାଇପାରିବ କିନ୍ତୁ ଏହା ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍‌ଚର ସହିତ ହୋଇପାରିବ ନାହିଁ ।

ପ୍ରକାର ସଂଜ୍ଞା typedef, ନୂତନଭାବେ ଗଠନଦ୍ୱାରା ଏକ ତଥ୍ୟ-ପ୍ରକାର ନାମ ଦେଇଥାଏ ଯାହା ଯେକୌଣସି ସ୍ଥାନରେ ଏହି ପ୍ରକାରକୁ ବ୍ୟବହାର କରାଯାଇପାରିବ । ପ୍ରକାର ସଂଜ୍ଞାପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
typedef dataType IDENTIFIER;
```

ଯେଉଁଠାରେ dataType ହୁଏତ ପୂର୍ବ-ପ୍ରସ୍ତୁତ (built-in) ତଥ୍ୟ-ପ୍ରକାର କିମ୍ବା ବ୍ୟବହାରକାରୀ-ନିର୍ଦ୍ଧାରିତ ତଥ୍ୟ-ପ୍ରକାର । IDENTIFIER, ତଥ୍ୟ-ପ୍ରକାରପାଇଁ ନୂତନ ନାମ ଯାହା ସାଧାରଣତଃ ବଡ଼ ଅକ୍ଷରରେ ଲେଖାଯାଇଥାଏ । typedef କୀୱାର୍ଡଟି ସଂକଳକକୁ IDENTIFIER କୁ dataType ର ସମାନାର୍ଥକ ଭାବରେ ଚିହ୍ନିବାକୁ ଜଣାଇଥାଏ ।

ଷ୍ଟ୍ରକ୍ଚର ସଂଜ୍ଞାର ଆଉ କିଛି ଉଦାହରଣ ନିମ୍ନରେ ଦିଆଯାଇଅଛି:

```
struct date        struct Time          struct Book
{                  {                    {
int day;           int hours;           char title[25];
int month;         int minutes;         char author[25];
int year;          int seconds;         char
};                 };                   publisher[15];
                                        float price;
                                        };

struct Customer    struct Inventory     struct Address
{                  {                    {
char name[25];     char                 char name[25];
char               item_code[25];       int house_no;
address[80];       char desc[25];       char locality[25];
long mobile_no;    int qty;             char city[15];
};                 float unit_price;    long pincode;
                   };                   };
```

8.3 ଷ୍ଟ୍ରକ୍ଚର ଚଳର ଘୋଷଣା

ଥରେ ଗୋଟିଏ ଷ୍ଟ୍ରକ୍ଚର ବ୍ୟାଖ୍ୟା ହେବା ପରେ, ଆମେ ଏହାର ଚଳ ଘୋଷଣା କରିପାରିବା । ସାଧାରଣତଃ, ସଂଜ୍ଞା ପ୍ରୋଗ୍ରାମର ଗ୍ଲୋବାଲ୍ ପରିସରରେ ରଖାଯାଏ, ଅର୍ଥାତ୍, main() ଫଙ୍କ୍ସନ ପୂର୍ବରୁ, ଯାହା ଫଳରେ ଏହା ପ୍ରୋଗ୍ରାମରେ ଥିବା ସମସ୍ତ ଫଙ୍କ୍ସନକୁ ପରିଦୃଶ୍ୟ କରିପାରିବା । ଅପରପକ୍ଷରେ, ଚଳଗୁଡ଼ିକ ସାଧାରଣତଃ ଫଙ୍କ୍ସନର ସ୍ଥାନୀୟ ପରିସରରେ କିମ୍ବା ଫଙ୍କ୍ସନ ହେଡ଼ର୍ର ଚର ତାଲିକାରେ ଘୋଷଣା କରାଯାଏ । ନିମ୍ନଲିଖିତ ବିବୃତ୍ତିଗୁଡ଼ିକ ଷ୍ଟ୍ରକ୍ଚର ଚଳ ଘୋଷଣା କରିବାର ଉପାୟ ଦର୍ଶାଇଥାଏ ।

ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍ଚର ପାଇଁ, ଚଳକୁ ନିମ୍ନ ଉପାୟରେ ଘୋଷଣା କରାହୁଏ

```
struct STUDENT aStudent;
```

ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍ଚରପାଇଁ, ଚଳକୁ ନିମ୍ନ ଭାବରେ ଘୋଷଣା କରାହୁଏ

```
STUDENT bStudent;
```

ଉଭୟ ଘୋଷଣାକୁ ତୁଳନା କଲେ ତୁମେ ପାଇବ ଯେ ପ୍ରକାର-ନିର୍ଦ୍ଧାରିତ ଷ୍ଟ୍ରକ୍ଚରର ଚଳ ଘୋଷଣା ଅଧିକ ସହଜସାପେକ୍ଷ । କାରଣ ତୁମକୁ struct କୀୱାର୍ଡ୍ ବ୍ୟବହାର କରିବାକୁ ପଡ଼ିନଥାଏ । ଯଦି ପ୍ରୋଗ୍ରାମର ଅନେକ ସ୍ଥାନରେ ଚଳ ଘୋଷଣାର ଆବଶ୍ୟକତା ଥାଏ ତେବେ ଏହାଦ୍ୱାରା ସମୟ ଏବଂ ପ୍ରୟାସ ସଞ୍ଚୟ ହୁଏ ।

ଧ୍ୟାନ ଦିଅ ଯେ, ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର କ୍ଷେତ୍ରରେ, ନିମ୍ନରେ ଦର୍ଶାଯାଇଥିବା ପରି ଚଳ ଘୋଷଣାକୁ ଏହାର ସଂଜ୍ଞା ସହିତ ସଂଯୁକ୍ତ କରାଯାଇପାରିବ ।

```
struct EMPLOYEE
 {
   int   code;
   char  name[25];
   char  department[15];
   float salary;
 } aEmployee, bEmployee;
```

ସାଧାରଣତଃ, ନିମ୍ନଲିଖିତ କାରଣଗୁଡ଼ିକପାଇଁ ଚଳ ଘୋଷଣା କରିବାର ଆଭିମୁଖ୍ୟ ସୁପାରିଶ କରାଯାଏ ନାହିଁ:

1. ଯେହେତୁ ଷ୍ଟ୍ରକ୍‌ଚର ସଂଜ୍ଞା ଗ୍ଲୋବାଲ୍ ପରିସରରେ ରଖାଯାଇଛି, ତେଣୁ ଚଳ ମଧ୍ୟ ଗ୍ଲୋବାଲ୍ ହେବ ।
2. ଏହି ଆଭିମୁଖ୍ୟ ବ୍ୟବହାର କରି ଯଦି ଆମେ ସ୍ଥାନୀୟ ଚଳ ଘୋଷଣା କରିବାକୁ ଚାହୁଁ, ତେବେ ଆମକୁ ଷ୍ଟ୍ରକ୍‌ଚରର ସଂଜ୍ଞାକୁ ସ୍ଥାନୀୟ ପରିସରରେ ରଖିବାକୁ ପଡ଼ିବ, ଏହା ଷ୍ଟ୍ରକ୍‌ଚରକୁ ଅନ୍ୟ ଫଙ୍କସନ ମଧ୍ୟରେ ଅଦୃଶ୍ୟ କରିଥାଏ ।

8.4 ଷ୍ଟ୍ରକ୍‌ଚରର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟକରଣ (INITIALIZING STRUCTURES)

ସାଧାରଣ ଚଳ ଏବଂ ଆରେ ପରି, ଘୋଷଣା ସମୟରେ ଷ୍ଟ୍ରକ୍‌ଚର ଚଳର ମଧ୍ୟ ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇପାରିବ । ଏହାର ଫର୍ମାଟ୍ (format) ଆରେର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟପାଇଁ ବ୍ୟବହୃତ ଫର୍ମାଟ୍ ସହିତ ସମାନ ।

ନିମ୍ନ ବିଭାଗରେ ଏହା ବର୍ଣ୍ଣିତ

```
typedef struct
 {
   int   rollno;
   char  name[25];
   int   age;
   float height;
 } STUDENT;

 STUDENT aStudent = { 1000, "Surbhi", 18, 5.6 };
```

8.5 ଷ୍ଟ୍ରକ୍‌ଚର ଉପାଦାନଗୁଡ଼ିକର ବ୍ୟବହାର (ACCESSING STRUCTURE ELEMENTS)

ଡଟ୍ ଅପରେଟର୍ '.' ବ୍ୟବହାର କରି ଷ୍ଟ୍ରକ୍‌ଚର ଚଳର ଉପାଦାନଗୁଡ଼ିକୁ ବ୍ୟବହାର (access) କରାଯାଇପାରିବ । ଏହି ଡ଼ଟ୍ ଅପରେଟର୍‌କୁ ସଦସ୍ୟ-ଅପରେଟର (membership operator) ମଧ୍ୟ କୁହାଯାଇଥାଏ ।

ଏକ ଷ୍ଟ୍ରକ୍‌ଚର ଉପାଦାନ ବ୍ୟବହାର କରିବାପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
sname.vname
```

ଯେଉଁଠାରେ sname ଷ୍ଟ୍ରକ୍ଚର ଚଳର ନାମ, ଏବଂ vname ଉପାଦାନର ନାମ ଅଟେ।

ନିମ୍ନଲିଖିତ ଉକ୍ତିଗୁଡ଼ିକୁ ଦେଖ

```
struct Student
 {
   int  rollno;
   char name[25];
   char fname[25];
   char grade;
 };

 Student aStudent;
```

ଷ୍ଟ୍ରକ୍ଚର ଚଳ aStudent ର ଉପାଦାନଗୁଡ଼ିକ ନିମ୍ନରେ ଦେଖାଯାଇଥିବା ଅନୁଯାୟୀ ବ୍ୟବହାର କରାଯାଇପାରିବ:

```
aStudent.rollno    // reference to element rollno
aStudent.name      // reference to element name
aStudent.fname     // reference to element fname
aStudent.grade     // reference to element grade
```

8.6 ଷ୍ଟ୍ରକ୍ଚରର ଆବଣ୍ଟନ (ASSIGNING OF STRUCTURES)

ଷ୍ଟ୍ରକ୍ଚର ସାମଗ୍ରିକ ଭାବରେ ବ୍ୟବହାର କରାଯାଇପାରୁଥିବା ଏକ ସତ୍ତା। କେବଳ ଗୋଟିଏ ଅପରେସନ ଅଛି ଯାହାକୁ ସାମଗ୍ରିକ ଭାବରେ ଷ୍ଟ୍ରକ୍ଚରରେ କରାଯାଇପାରିବ ତାହା ହେଉଛି ଏକ ଷ୍ଟ୍ରକ୍ଚରକୁ ଆସାଇନମେଣ୍ଟ ଅପରେଟର ବ୍ୟବହାର କରି ସମାନ ପ୍ରକାରର ଅନ୍ୟ ଏକ ଷ୍ଟ୍ରକ୍ଚରକୁ କପି କରିପାରିବ।

ପୁଣି ନିମ୍ନଲିଖିତ ଉକ୍ତିଗୁଡ଼ିକୁ ଦେଖ

```
typedef struct
 {
   int   rollno;
   char  name[25];
   int   age;
   float height;
 } STUDENT;

 STUDENT aStudent, bStudent;
```

ନିମ୍ନଲିଖିତ ଆସାଇନ୍‌ମେଣ୍ଟ୍ ବିବୃତ୍ତି –

```
sname.vname
```

bStudent ଷ୍ଟ୍ରକ୍ଚରର ବିଷୟବସ୍ତୁକୁ aStudent ଷ୍ଟ୍ରକ୍ଚର ନ୍ୟସ୍ତ କରେ।

ଆରେ ତୁଳନାରେ ଏହା ଷ୍ଟ୍ରକ୍ଚରର ଏକ ଅନନ୍ୟ ବୈଶିଷ୍ଟ୍ୟ, ଯେଉଁଠାରେ ଗୋଟିଏ ଆରେକୁ ସମାନ ପ୍ରକାରର ଅନ୍ୟ ଏକ ଆରେକୁ କପି କରିବାକୁ, ଆମକୁ ଉପାଦାନ-ପରେ-ଉପାଦାନ କପି କରିବାପାଇଁ ଏକ ଲୁପ୍ ବ୍ୟବହାର କରିବାକୁ ପଡ଼ିବ।

ତୁମେ ଚିନ୍ତା କଲେ ଦେଖିବ ଏହା ଏକ ଆଶ୍ଚର୍ଯ୍ୟଜନକ ସାମର୍ଥ୍ୟ। ଯେତେବେଳେ ତୁମେ ଗୋଟିଏ ଷ୍ଟ୍ରକ୍ଚରକୁ ଅନ୍ୟ ଏକ ଷ୍ଟ୍ରକ୍ଚରରେ ନ୍ୟସ୍ତ କର, ଷ୍ଟ୍ରକ୍ଚରର ସମସ୍ତ ମୂଲ୍ୟଗୁଡ଼ିକ ଅନ୍ୟ ଷ୍ଟ୍ରକ୍ଚରର ଅନୁରୂପ ଉପାଦାନଗୁଡ଼ିକୁ ନ୍ୟସ୍ତ ହୋଇଥାଏ। ଏହିପରି ସରଳ ଆସାଇନମେଣ୍ଟ ବିବୃତ୍ତି ଆରେପାଇଁ ବ୍ୟବହୃତ ହୋଇପାରିବ ନାହିଁ। ଅର୍ଥାତ, ଗୋଟିଏ ଆରେ ଅନ୍ୟ ଏକ ଆରେକୁ ନ୍ୟସ୍ତ କରିବା ପାଇଁ ଆମକୁ ଆରେଗୁଡ଼ିକର ପ୍ରତ୍ୟେକ ଉପାଦାନଗୁଡ଼ିକୁ ଗୋଟିଏ ପରେ ଗୋଟିଏ ନ୍ୟସ୍ତ କରିବାକୁ ପଡ଼ିବ।

8.7 ଷ୍ଟ୍ରକ୍ଚରକୁ ପଢ଼ିବା / ଲେଖିବା (READING/WRITING STRUCTURES)

ଆମେ ସାଧାରଣ ଚଳ ଭଳି ସମାନ ଢଙ୍ଗରେ ଏକ ଷ୍ଟ୍ରକ୍ଚରର ଉପାଦାନରୁ ତଥ୍ୟ ପଢ଼ିପାରିବା ଏବଂ ଉପାଦାନକୁ ଲେଖିପାରିବା।

ଉଦାହରଣ ସ୍ୱରୂପ, ଆମେ ନିମ୍ନ ଉକ୍ତିଗୁଡ଼ିକ ବ୍ୟବହାର କରି aStudent ଷ୍ଟ୍ରକ୍ଚରର ଉପାଦାନଗୁଡ଼ିକପାଇଁ ସଂବାଦମୂଳକ ଭାବରେ କୀବୋର୍ଡ଼ରୁ ତଥ୍ୟ ପଢ଼ିପାରିବା ଏବଂ aStudent ଷ୍ଟ୍ରକ୍ଚରରେ ସ୍ଥାନିତ କରିପାରିବା।

```
printf("\nEnter roll number of the student : ");

scanf("%d", &aStudent.rollno);

printf("\nEnter name of the student : ");

scanf("%d", aStudent.name);

printf("\nEnter age of the student : ");

scanf("%d", &aStudent.age);

printf("\nEnter height of the student : ");

scanf("%d", &aStudent.height);
```

ଲକ୍ଷ୍ୟ କର ଯେ scanf() ଫଙ୍କ୍ସନ ମଧ୍ୟର ପରିପ୍ରକାଶ

```
&aStudent.rollno
```

କୁ ସଂକଳକ ବୁଝେ ଯେ

```
&aStudent.rollno
```

କାରଣ, ଆଡ୍ରେସ୍ ଅପରେଟର୍ (&) ଠାରୁ ପ୍ରତ୍ୟକ୍ଷ ଚୟନ ଅପରେଟରର ଉଚ୍ଚତର ଅଗ୍ରାଧିକାର ଉଚ୍ଚରେ ଏବଂ ତାହା ହିଁ ଯଥାର୍ଥ ଭାବରେ ପ୍ରୟୋଜନ।

Listing 8.1

```
/*
    Program to demonstrate input/output of structures
*/
#include<stdio.h>
#include<sting.h>
struct EMPLOYEE
{
   int code;
   char name[25];
   char department[15];
   float salary;
};

int main()
{
   struct EMPLOYEE aEmployee;
   printf("Enter employee’s code: " );
   scanf( "%d", &aEmployee.code );
   fflush(stdin);
   printf("Enter employee’s name: " );
   gets( aEmployee.name );
   printf("Enter employee’s department: " );
   gets( aEmployee.department );
   printf("Enter employee’s salary: " );
   scanf("%f", &aEmployee.salary );
   printf("\n\nParticulars of employee as" );
   printf(" entered by user\n" );
   printf("\nEmployee’s code: %d", aEmployee.code );
   printf("\nEmployee’s name: %s", aEmployee.name );
   printf("\nEmployee’s department: %s",aEmployee.department);
   printf("\nEmployee’s salary: %.2f\n", aEmployee.salary );
   return 0;
}
```

Test Run

```
Enter employee’s code: 826
Enter employee’s name: Rajesh Kumar
Enter employee’s department: Computer Science
Enter employee’s salary: 20000
Particulars of employee as entered by user
Employee’s code: 826
Employee’s name: Rajesh Kumar
Employee’s department: Computer Science
```

8.8 ଷ୍ଟ୍ରକ୍ଚରଗୁଡ଼ିକର ଆରେ (ARRAYS OF STRUCTURES)

ଅନେକ ବ୍ୟବହାରିକ ପରିସ୍ଥିତିରେ, ଆମକୁ ଷ୍ଟ୍ରକ୍ଚରର ଏକ ଆରେ ତିଆରି କରିବା ଆବଶ୍ୟକ। ଉଦାହରଣ ସ୍ୱରୂପ, ଗୋଟିଏ ଶ୍ରେଣୀରେ ଶିକ୍ଷାର୍ଥୀମାନଙ୍କ ଏକ ଗ୍ରୁପ୍‌ପାଇଁ ପ୍ରୋଗ୍ରାମ ଲେଖିବାକୁ ଆମକୁ ଶିକ୍ଷାର୍ଥୀମାନଙ୍କର ଏକ ଆରେ ତିଆରି କରିବା ଆବଶ୍ୟକ। ଶିକ୍ଷାର୍ଥୀମାନଙ୍କ ତଥ୍ୟକୁ ଷ୍ଟ୍ରକ୍ଚରର ଏକ ଆରେରେ ରଖି, ଆମେ ଶିକ୍ଷାର୍ଥୀଙ୍କ ତଥ୍ୟକୁ ଶୀଘ୍ର ଏବଂ ଦକ୍ଷତାର ସହିତ ପ୍ରକ୍ରିୟା କରିପାରିବା।

ଶିକ୍ଷାର୍ଥୀମାନଙ୍କ ତଥ୍ୟ ଗଚ୍ଛିତ କରିବାକୁ ଷ୍ଟ୍ରକ୍ଚରର ଏକ ଆରେ ଏଭଳି ଘୋଷଣା କରାଯିବ

```
STUDENT students[50];
```

8.8.1 ଷ୍ଟ୍ରକ୍ଚର ଆରେର ଉପାଦାନଗୁଡ଼ିକୁ ବ୍ୟବହାର (Accessing Elements of Array of Structures)

ଷ୍ଟ୍ରକ୍ଚରର ଏକ ଆରେରେ ଏକ ଷ୍ଟ୍ରକ୍ଚରର ନିର୍ଦ୍ଦିଷ୍ଟ ଉପାଦାନଗୁଡ଼ିକୁ ବ୍ୟବହାରପାଇଁ ଷ୍ଟ୍ରକ୍ଚର ଚଳର ନାମ, ଆରେର ଇଣ୍ଡେକ୍ସ, ପ୍ରତ୍ୟକ୍ଷ ଚୟନ ଅପରେଟର୍ ଏବଂ ଶେଷରେ ଇପ୍‌ସିତ ଷ୍ଟ୍ରକ୍ଚ ଉପାଦାନ ଉଲ୍ଲେଖ କରି କରାଯାଏ।

ଆମର ଏକ ଶ୍ରେଣୀର ବିଦ୍ୟାର୍ଥୀମାନଙ୍କର ଉଦାହରଣରେ, i ତମ ବିଦ୍ୟାର୍ଥୀଙ୍କ ନାମ ନିମ୍ନପ୍ରଦତ୍ତ ଭାବରେ ବ୍ୟବହାର କରାଯାଇପାରିବ:

```
students[i].name;
```

8.8.2 ଷ୍ଟ୍ରକ୍ଚର ଆରେର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ (Initializing Array of Structures)

ଅନ୍ୟ ଯେକୌଣସି ଆରେ ପରି, ଏକ ଷ୍ଟ୍ରକ୍ଚର ଆରେର ମଧ୍ୟ ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇପାରିବ। ଯେହେତୁ ଷ୍ଟ୍ରକ୍ଚର ଆରେର ପ୍ରତ୍ୟେକ ଉପାଦାନ ମଧ୍ୟ ଗୋଟିଏ ଷ୍ଟ୍ରକ୍ଚର, ସେହେତୁ ଆମକୁ ପ୍ରତ୍ୟେକ ଷ୍ଟ୍ରକ୍ଚରର ଉପାଦାନଗୁଡ଼ିକୁ ପୃଥକ ଭାବେ ଏକ କୁଟିଳ ବନ୍ଧନୀ ମଧ୍ୟରେ ରଖିବା କରିବା ଆବଶ୍ୟକ:

```
STUDENT students[50] = { { 1000, "Monika",{56,76,85,69}, 'A' },
                          { 1001, "Ram", {50,66,70,60}, 'B' },
                                  :
                          { 1049, "Raju", {65,62,68,59}, 'B' },
                        };
```

ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ଟି ଷ୍ଟ୍ରକ୍ଚରର ଏକ ଆରେର ଇନପୁଟ୍, ପ୍ରକ୍ରିୟାକରଣ ଏବଂ ଆଉଟପୁଟ୍ ଦର୍ଶାଉଛି

Listing 8.2

```
/*
    Program to illustrate processing of array of structures by
    storing list of students in memory, and computes their grades
 */
 #include <stdio.h>
 typedef struct {
    int  rollno;
    char sname[25];
    int  marks[4];
    char grade;
 } STUDENT;
 int main()
 {
```

```
STUDENT students[50];
    int totalMarks, i, j, n;
    printf( "\n\nEnter number of students in a class : " );
    scanf( "%d", &n );
    for ( i = 0; i < n; i++ )
    {
       printf("\nParticulars of student #%d\n\n", i+1);
       printf( "Enter students' roll number : " );
       scanf("%d", &students[i].rollno );
       printf( "\nEnter students' name : " );
       gets( students[i].sname );
       printf( "\nEnter students marks in four subjects\n\n" );
       for ( j = 0; j < 4; j++ )
          scanf("%d", &students[i].marks[j] );
    }
    for ( i = 0; i < n; i++ )
    {
       totalMarks = 0;
       for ( j = 0; j < 4; j++ )
       totalMarks += students[i].marks[j];
       if ( totalMarks > 75 )
       students[i].grade = 'A';
       else if ( totalMarks > 60 )
       students[i].grade = 'B';
       else if ( totalMarks > 50 )
       students[i].grade = 'C';
       else if ( totalMarks > 40 )
       students[i].grade = 'D';
       else
       students[i].grade = 'F';
    }
    printf("\nPerformance of students\n\n");
    printf("%-10s", "Roll No.");
    printf("%-30s", "Name");
    printf("%6s\n", "Grade");
    for ( i = 0; i < n; i++ )
    {
       printf("%-10d", students[i].rollno);
       printf("%-30s", students[i].sname);
       printf("%4c\n", students[i].grade);
    )
    return 0;
 }
```

Test Run

```
Enter number of students in a class : 3
Particulars of student #1
Enter students' roll number : 1000
Enter students' name : Ram Kumar
Enter students marks in four subjects
98 99 89 88
Particulars of student #2
Enter students' roll number : 1001
Enter students' name : Mohit Kumar
Enter students marks in four subjects
65 55 45 60
Particulars of student #3
Enter students' roll number : 1002
Enter students' name : Jaspreet Singh
Enter students marks in four subjects
41 44 45 50
Performance of students
SNo.           Student's Name     Grade
  1                      Sonu         A
  2                    bhollu         C
  3                  jaspreet         D
```

8.9 ଏକ ଫଙ୍କ୍ସନକୁ ଷ୍ଟ୍ରକ୍ଚର ପଠାଇବା (PASSING STRUCTURE TO A FUNCTION)

ସମ୍ପୂର୍ଣ୍ଣ ଉପଯୋଗିତା ପାଇଁ, ଆମେ ଷ୍ଟ୍ରକ୍ଚରକୁ ଚର ଭାବରେ ଫଙ୍କ୍ସନ ଭିତରକୁ ପଠାଇବାପାଇଁ ଏବଂ ସେଗୁଡ଼ିକୁ ଫଙ୍କ୍ସନରୁ ଫେରସ୍ତ କରିବାକୁ ସମର୍ଥ ହେବା ପ୍ରୟୋଜନ ।

```
typedef struct {
   int day;
   int month;
   int year;
} DATE;
void printDate (DATE);
void main()
{
   DATE aDate = { 10, 6, 2008 };
   int x;
   printDate(aDate);
   /*  more statements    */
}
```

```
/*  function definition  */
void printDate(DATE tDate )
{

   /*  local declarations */
   /*  other statements   */

}
```

ଚିତ୍ର . 8.3: ଫଙ୍କ୍ସନକୁ ଷ୍ଟ୍ରକ୍ଚର ପ୍ରେରଣ

Listing 8.3

```
 /*
    Program to illustrate the passing of a structure to a function
*/
#include <stdio.h>
typedef struct
{
   int day;
   int month;
   int year;
} DATE;
void printDate( DATE aDate );         /*  function declaration  */
void main()
{
   DATE tDate = { 10, 6, 2008 }; /* initialize a structure  */
   printDate( tDate );                /*  function call          */
}
void printDate( DATE aDate )
{
  printf( "\nDate in format dd/mm/yyyy: %02d/%02d/%2d\n",
           aDate.day, aDate.month, aDate.year );
}
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Date in format dd/mm/yyyy: 10/06/2008
```

8.10 ଫଙ୍କ୍ସନରୁ ଏକ ଷ୍ଟ୍ରକ୍ଚର ଫେରସ୍ତ କରିବା (FUNCTION RETURNING A STRUCTURE)

ନିମ୍ନ ପ୍ରଦର୍ଶିତ ପ୍ରକାରେ ବିବୃତ୍ତି ଏକ ଫଙ୍କ୍ସନ ମଧ୍ୟ return ଉକ୍ତି ଜରିଆରେ ଷ୍ଟ୍ରକ୍ଚର ପ୍ରକାରର ମୂଲ୍ୟ ଫେରସ୍ତ କରିପାରେ ।

```
typedef struct
{
   int day;
   int month;
   int year;
} DATE;
DATE getDate(void);
void main()
{
   DATE aDate;
   int x;
   aDate = getDate();
   /*  more statement   */
}
```

```
/*  function definition   */
DATE getDate(void)
{
   DATE tDate;
   /*  local declaration   */
   /*  more statements    */
   return tDate;
}
```

ଚିତ୍ର . 8.4: ଷ୍ଟ୍ରକ୍ଚର ଫେରସ୍ତ କରୁଥିବା ଏକ ଫଙ୍କ୍ସନ

Listing 8.4

```
 /*
    Program to illustrate function returning a structure
*/
#include <stdio.h>
typedef struct
{
   int day;
   int month;
   int year;
} DATE;
DATE getDate( void );        /*  function declaration   */
void main()
{
   DATE tDate;
   printf( "Enter date in format dd/mm/yyyy : " );
   tDate = getDate();
   printf( "\nDate entered by you: %02d/%02d/%2d\n",
          tDate.day, tDate.month, tDate.year );
}

DATE getDate( void )
{
   DATE xDate;
   scanf( "%d/%d/%d", &xDate.day, &xDate.month, &xDate.year );
   return xDate;
}
```

Test Run

```
Enter date in format dd/mm/yyyy: 10/6/2008
Date entered by you: 10/06/2008
```

ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 8.1: ନିମ୍ନଲିଖିତ ଘୋଷଣାନାମା ଦିଆଯାଇଛି

```
struct STUDENT
 {
   int  rollNo;
   char name[31];
   int   marks;
 };
```

ଜଣେ ଶିକ୍ଷାର୍ଥୀଙ୍କ ସୂଚନା ଇଂଗିତ କରିବାକୁ ।

ଏହି ଷ୍ଟ୍ରକ୍‌ଚର ବ୍ୟବହାର କରି, ଏକ ଫଙ୍କ୍‌ସନ ଲେଖ ଯାହା ଏହାର ଚର ଭାବରେ ଶିକ୍ଷାର୍ଥୀମାନଙ୍କର ଗୋଟିଏ ତାଲିକା ଆରେ ଆକାରରେ ନେଇଥାଏ, ଏବଂ ଶିକ୍ଷାର୍ଥୀମାନଙ୍କ ଦ୍ୱାରା ପ୍ରାପ୍ତ କ୍ରମାନ୍ୱୟେ ସଜାଇଥାଏ ।

ଉପରୋକ୍ତ ଷ୍ଟ୍ରକ୍‌ଚର ଏବଂ ଫଙ୍କ୍‌ସନ ବ୍ୟବହାର କରି, ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା n ବିଦ୍ୟାର୍ଥୀଙ୍କ ପାଇଁ ତଥ୍ୟ ଜଣାଇବ ଏବଂ ବିଦ୍ୟାର୍ଥୀମାନଙ୍କ ଦ୍ୱାରା ପ୍ରାପ୍ତ ମାର୍କଗୁଡ଼ିକୁ କ୍ରମାନ୍ୱୟେ ପ୍ରଦର୍ଶନ କରିପାରୁଥିବ

Listing 8.5

```
 /*
   Program to the given information of students in
   descending order of the marks secured by them
*/
#include <stdio.h>
typedef struct
{
  int  rollno;
  char name[20];
  int  marks;
} STUDENT;
int main()
{
  STUDENT st[50], temp;
  int i, j, n;
  printf( "\nEnter number of students : " );
  scanf( "%d", &n );
  for ( i = 0; i < n; i++ )
  {
     printf("\nParticulars of student #%d\n\n", i+1);
     printf( "Enter students' roll number : " );
     scanf("%d", &st[i].rollno );
     fflush(stdin);
     printf( "Enter students' name : " );
     gets(st[i].name );
     printf( "Enter students' marks : " );
     scanf("%d", &st[i].marks );
  }
  for ( i = 1; i < n; i++ )
  {
      for ( j = 0; j < n-i; j++ )
      {
      if ( st[j].marks < st[j+1].marks )
          {
```

```
temp = st[j];
               st[j] = st[j+1];
               st[j+1] = temp;
                  }
       }
   }
   printf("\nStudent's Information after sorting\n\n");
   printf("%-10s", "Roll No.");
   printf("%-20s", "Name");
   printf("%6s\n\n", "Marks");
  for ( i = 0; i < n; i++ )
   {
      printf("%-10d", st[i].rollno);
      printf("%-20s", st[i].name);
      printf("%4d\n", st[i].marks);
   }
   return 0;
 }
```

Test Run

```
Enter number of students : 5
Particulars of student #1
Enter students' roll number : 1000
Enter students' name : Ravi
Enter students' marks : 80

Particulars of student #2

Enter students' roll number : 1001
Enter students' name : Shankar
Enter students' marks : 79

Particulars of student #3

Enter students' roll number : 1002
Enter students' name : Ankit
Enter students' marks : 65
Particulars of student #4

Enter students' roll number : 1003
Enter students' name : Rupinder
Enter students' marks : 75
```

```
Particulars of student #5

Enter students' roll number : 1004
Enter students' name : Mehak
Enter students' marks : 88

Student's Information after sorting

Roll No.        Name                 Marks
  1004         Mehak                  88
  1000         Ravi                   80
  1001         Shankar                79
  1003         Rupinder               75
  1002         Ankita                 65
```

ଉଦାହରଣ 8.2: ଷ୍ଟ୍ରକ୍ଚର ବ୍ୟବହାର କରି ଇଞ୍ଚ-ଫୁଟ ସିଷ୍ଟମରେ ଦିଆଯାଇଥିବା ଦୁଇଟି ଦୂରତା ଯୋଗ କରିବାକୁ ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।

Listing 8.6

```
/*
   Program to add two distances given in inch-feet system
   using structures and functions
*/

#include <stdio.h>

typedef struct
{
   int feets;
   int inches;
} Distance;

/*  function prototypes  */
Distance getInput(void);
Distance addDistances(Distance d1, Distance d2);
void printDistance(Distance d);

int main()
{
   Distance d1, d2, d3;

   printf("\nEnter the first distance in feets and inches\n");
   d1 = getInput();
   printf("\nEnter the second distance in feets and inches\n");
```

```
   d2 = getInput();
   d3 = addDistances(d1, d2);
   printf("\nSum of given distances\n\n");
   printDistance(d3);
   return 0;
}
Distance getInput()
{
   Distance temp;
   printf("\n\tEnter feets : ");
   scanf("%d", &(temp.feets));
   printf("\n\tEnter inches : ");
   scanf("%d", &(temp.inches));
   return temp;
}
Distance addDistances(Distance d1, Distance d2)
{
   Distance temp;
   temp.feets = d1.feets + d2.feets;
   temp.inches = d1.inches + d2.inches;
   if ( temp.inches >= 12 )
   {
      temp.feets += 1;
      temp.inches -= 12;
   }
   return temp;
}
void printDistance(Distance d)
{
   printf("\n\tFeets = %d", d.feets);
   printf("\n\tInches = %d", d.inches);
}
```

Test Run

```
Enter the first distance in feets and inches
Enter feets : 4
Enter inches : 8
Enter the second distance in feets and inches
Enter feets : 5
Enter inches : 7
Sum of given distances
    Feets = 10
    Inches = 3
```

ଉଦାହରଣ 8.3: କମ୍ପ୍ଲେକ୍ସ ସଂଖ୍ୟାର (complex number) ଏକ ଛାଞ୍ଚ (model) ତିଆରିବାପାଇଁ ଏକ ଷ୍ଟ୍ରକ୍ଚର ବ୍ୟବହାର କର । ଏହି ଷ୍ଟ୍ରକ୍ଚର ବ୍ୟବହାର କରି ନିମ୍ନଲିଖିତ କାର୍ଯ୍ୟଗୁଡ଼ିକ ସମ୍ପୂର୍ଣ୍ଣ କରିବାକୁ ଫଙ୍କ୍ସନଗୁଡ଼ିକ ଲେଖ:

- ଏକ ଫଙ୍କ୍ସନ ଯାହା କୀବୋର୍ଡରୁ ପଢ଼ାହୋଇଥିବା ଷ୍ଟ୍ରକ୍ଚରକୁ ଫେରସ୍ତ କରେ ।
- ଏକ ଫଙ୍କ୍ସନ ଯାହା ଦୁଇଟି ଷ୍ଟ୍ରକ୍ଚରକୁ ମୂଲ୍ୟଦ୍ୱାରା ଡାକିବା ପଦ୍ଧତି ବ୍ୟବହାର କରି ଚର ଭାବରେ ନେଇଥାଏ ଏବଂ ସେଗୁଡ଼ିକ ଯୋଗଫଳ ଫେରସ୍ତ କରେ ।
- ଏକ ଫଙ୍କ୍ସନ ଯାହା ଏକ ଷ୍ଟ୍ରକ୍ଚରକୁ ମୂଲ୍ୟଦ୍ୱାରା ଡାକିବା ପଦ୍ଧତି ବ୍ୟବହାର କରି ଚର ଭାବରେ ନେଇଥାଏ ଏବଂ ଏହାକୁ ପ୍ରିଣ୍ଟ କରେ ।

ସେମାନଙ୍କର କାର୍ଯ୍ୟ ପ୍ରଦର୍ଶନ ନିମିତ୍ତ ପ୍ରୋଗ୍ରାମରେ ଏହି ଫଙ୍କ୍ସନଗୁଡ଼ିକ ବ୍ୟବହାର କର ।

Listing 8.7

```
/*
   Program to demonstrate operations on complex number using
   structure & functions
*/

#include <stdio.h>

typedef struct
{
   float real;
   float imag;
} COMPLEX;

COMPLEX input(void);         /*  function declarations  */
void output(COMPLEX);
COMPLEX add(COMPLEX, COMPLEX);
COMPLEX multiply(COMPLEX, COMPLEX);

int main()
{
   COMPLEX n1, n2, n3, n4;
   printf("\nProgram to add two complex numbers\n\n");
   printf("\nEnter the first complex number\n");
   n1 = input();
   printf("\nEnter the second complex number\n");
   n2 = input();
   n3 = add(n1,n2);
   printf("\nSum of the given complex numbers\n\n");
   output(n3);
   n4 = multiply(n1,n2);
   printf("\nProduct of the given complex numbers\n\n");
   output(n4);
   return 0;
```

```
   }
COMPLEX input(void)
{
   COMPLEX t;
   printf("\nEnter real part : ");
   scanf("%f", &t.real);
   printf("Enter imaginary part : ");
   scanf("%f", &t.imag);
   return t;
}
void output(COMPLEX t)
{
   printf("Real part : %.2f", t.real);
   printf("\nImaginary part : %.2f", t.imag);
}
COMPLEX add(COMPLEX a, COMPLEX b)
{
   COMPLEX t;
   t.real = a.real + b.real;
   t.imag = a.imag + b.imag;
   return t;
}
COMPLEX multiply(COMPLEX a, COMPLEX b)
                 {
   COMPLEX t;
   t.real = a.real * b.real - a.imag * b.imag;
   t.imag = a.real * b.imag + a.imag * b.real;
   return t;
}
```

Test Run

```
Program to add two complex numbers
Enter the first complex number
Enter real part : 1.0
Enter imaginary part : 2.5
Enter the second complex number
Enter real part : 2.5
Enter imaginary part : -1.5
Sum of the given complex numbers
Real part : 3.50
Imaginary part : 1.00
Product of given complex numbers
Real part : 6.25
Imaginary part : 4.75
```

ୟୁନିଟ ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଶିଖିଲୁ

- ଗୋଟିଏ ଷ୍ଟ୍ରକ୍ଚର ହେଉଛି ପରସ୍ପର ସମ୍ବନ୍ଧିତ ଉପାଦାନଗୁଡ଼ିକର ଏକ ସଂଗ୍ରହ, ଷ୍ଟ୍ରକ୍ଚରର ନାମ ଗୋଟିଏ ଥାଇ ଉପାଦାନଗୁଡ଼ିକ ଅଲଗା ପ୍ରକାର ହୋଇପାରେ।
- ଷ୍ଟ୍ରକ୍ଚରର ପ୍ରତ୍ୟେକ ଉପାଦାନକୁ ଏକ କ୍ଷେତ୍ର (field)/ସଦସ୍ୟ(member) କୁହାଯାଏ।
- ଗୋଟିଏ ଷ୍ଟ୍ରକ୍ଚର ଘୋଷଣା କରିବାର ଦୁଇଟି ଉପାୟ ଅଛି: ଟ୍ୟାଗ୍‌ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍ଚର ଏବଂ ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍ଚର।
- ଗୋଟିଏ ଷ୍ଟ୍ରକ୍ଚରର ଘୋଷଣା ଏବଂ ବ୍ୟାଖ୍ୟା କଲାବେଳେ ଏହାର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରାଯାଇପାରିବ। କମା ଦ୍ଵାରା ପୃଥକ ହୋଇଥିବା ମୂଲ୍ୟଗୁଡ଼ିକର ତାଲିକା କୁଟିଳ ବନ୍ଧନୀ ମଧ୍ୟରେ ଆବଦ୍ଧ ହୋଇଥାଏ।
- ଆମେ ପ୍ରତ୍ୟକ୍ଷ ଚୟନ, ଡ଼ଟ୍ ଅପରେଟର୍ (.) ବ୍ୟବହାର କରି ଷ୍ଟ୍ରକ୍ଚରର କ୍ଷେତ୍ରଗୁଡ଼ିକୁ/ସଦସ୍ୟଗୁଡ଼ିକୁ ବ୍ୟବହାର କରିପାରିବା।
- ଏକ ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍ ବ୍ୟବହାର କରି ଅନ୍ୟ ଷ୍ଟ୍ରକ୍ଚରକୁ ଏକ ଷ୍ଟ୍ରକ୍ଚର ନ୍ୟସ୍ତ କରାଯାଇପାରିବ ଯାହା ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗରେ ହୋଇପାରେ ନାହିଁ।
- ଏକ ଷ୍ଟ୍ରକ୍ଚରକୁ ଏକ ଫଙ୍କ୍ସନର ଚର ଭାବରେ ପାରିତ କରାଯାଇପାରିବ।
- ଏକ ଫଙ୍କ୍ସନ ଏକ ଷ୍ଟ୍ରକ୍ଚର ଫେରସ୍ତ କରିପାରେ।

ଅନୁଶୀଳନୀ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. ଷ୍ଟ୍ରକ୍ଚର ବ୍ୟବହାର କରିବାର ମୁଖ୍ୟ କାରଣ କ'ଣ?
2. ଗୋଟିଏ ଷ୍ଟ୍ରକ୍ଚର କିପରି ଏକ ଆରେ ଠାରୁ ଭିନ୍ନ ଅଟେ?
3. ଷ୍ଟ୍ରକ୍ଚର ଟ୍ୟାଗ୍ କ'ଣ ଏବଂ ଏହାର ଉଦ୍ଦେଶ୍ୟ କ'ଣ?
4. ଗୋଟିଏ ଷ୍ଟ୍ରକ୍ଚରକୁ ବ୍ୟାଖ୍ୟା କରିବାର ବିଭିନ୍ନ ଉପାୟ କ'ଣ? ପ୍ରତ୍ୟେକ କ୍ଷେତ୍ରରେ ଉଦାହରଣ ଦିଅ।
5. ନିମ୍ନଲିଖିତ କୋଡ଼ରେ କିଛି ଭୁଲ୍ ଅଛି କି?

```
struct Time
{
   int hrs;
   int mts;
   int secs;
}
time t1;
```

6. ନିମ୍ନଲିଖିତ କୋଡ଼ରେ କ'ଣ ଭୁଲ୍ ଅଛି, କି?

```
struct Time
{
   int hrs =10;
   int mts = 20;
   int secs = 35;
} t1, t2, t3;
```

7. ନିମ୍ନଲିଖିତ କୋଡ଼ରେ ଯଦି କିଛି ତ୍ରୁଟି ଥାଏ ଚିହ୍ନଟ କର ।

```
struct
{
   int hrs;
   int mts;
   int secs;
} time;
time t1, t2, t3;
```

ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍‌ର ଆଉଟପୁଟ୍ ଲେଖ:

```
#include <stdio.h>
struct MyBox {
   int Length, Breadth, Height;
};
void Dimension( MyBox M )
{
   printf("\n%d x %d x %d\n",M.Length,M.Breadth,M.Height);
}
void main()
{
  MyBox B1 = { 12, 20, 8 }, B2, B3;
  ++B1.Height;
  Dimension(B1);
  B3 = B1;
  ++B3.Length;
  B3.Breadth++;
  Dimension(B3);
  B2 = B3;
  B2.Height += 5;
  B2.Length--;
  Dimension(B2);
```

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. ଏକ ଷ୍ଟ୍ରକ୍‌ଚର ହେଉଛି
   (a) Scalar data type (b) Derived data type
   (c) Primitive data type (d) User-defined data type
2. ଷ୍ଟ୍ରକ୍‌ଚର ବିଷୟରେ ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ବିବୃତ୍ତିଟି ସତ୍ୟ?
   (a) ଷ୍ଟ୍ରକ୍‌ଚରର ଉପାଦାନଗୁଡ଼ିକ ବିଭିନ୍ନ ପ୍ରକାରର ହେବା ଆବଶ୍ୟକ।
   (b) ସମସ୍ତ କ୍ଷେତ୍ରରେ ଷ୍ଟ୍ରକ୍‌ଚର ଟ୍ୟାଗ୍ ବାଧ୍ୟତାମୂଳକ ଅଟେ।
   (c) ଷ୍ଟ୍ରକ୍‌ଚର ପ୍ରକାରର ଏକ ଚଳ ଘୋଷଣା କରିବା ସମୟରେ, ତୁମକୁ ଟ୍ୟାଗ୍ ନାମ ପୂର୍ବରୁ struct କୀୱାର୍ଡ୍ ଉପସର୍ଗ ଭାବରେ ବ୍ୟବହାର କରିବା ଆବଶ୍ୟକ।
   (d) ଷ୍ଟାର୍ (*) ଅପରେଟର୍ ବ୍ୟବହାର କରି ଷ୍ଟ୍ରକ୍‌ଚରର ଉପାଦାନଗୁଡ଼ିକ ବ୍ୟବହାର ହୁଏ।
3. ଗୋଟିଏ ଷ୍ଟ୍ରକ୍‌ଚର ହେଉଛି ଏକ ତଥ୍ୟ-ପ୍ରକାର ଯେଉଁଥିରେ
   (a) ପ୍ରତ୍ୟେକ ଉପାଦାନରେ ସମାନ ପ୍ରକାରର ରହିବା ଆବଶ୍ୟକ।
   (b) ଉପାଦାନଗୁଡ଼ିକ ଭିନ୍ନ ପ୍ରକାରର ହୋଇପାରେ।
   (c) ପ୍ରତ୍ୟେକ ଉପାଦାନ ନିଶ୍ଚିତ ଭାବରେ ପଏଣ୍ଟର୍ ପ୍ରକାରର ହେବା ଆବଶ୍ୟକ।
   (d) ଉପରୋକ୍ତ କୌଣସିଟି ନୁହେଁ।
4. ନିମ୍ନ ଘୋଷଣାନାମାକୁ ଦେଖ

```
struct ex {
    char cVvar;
    int iVar;
    long lVar;
};
```

   sizeof ଅପରେଟର ଦ୍ୱାରା ex ର କେଉଁ ମୂଲ୍ୟ ଫେରସ୍ତ ହେବ?
   (a) 4 (b) 7
   (c) 1 (d) 6
5. ଏକ ଷ୍ଟ୍ରକ୍‌ଚର ତଥ୍ୟ-ପ୍ରକାର ତିଆରି କରିବାକୁ ବ୍ୟବହୃତ କୀୱାର୍ଡ୍ ହେଉଛି
   (a) structure (b) structr
   (c) struct (d) struc
6. ଷ୍ଟ୍ରକ୍‌ଚର ସଦସ୍ୟ ହ୍ୟବହାର କରିବାକୁ ବ୍ୟବହୃତ ଅପରେଟର୍ ହେଉଛି
   (a) * (b) .
   (c) & (d) []
7. ଉକ୍ତିଟିକୁ ଦେଖ: "ଏକ struct ର ଆକାର ସର୍ବଦା ଏହାର ସଦସ୍ୟମାନଙ୍କ ସମାହାର ମୂଲ୍ୟ ସହିତ ସମାନ।"
   (a) ବୈଧ (b) କହିହେବ ନାହିଁ
   (c) ଅବୈଧ (d) ସଂକଳକ ନିର୍ଭରଶୀଳ

8. "ଏକ struct ର ଆକାର ସର୍ବଦା ଏହାର ସଦସ୍ୟଗୁଡ଼ିକର ସମାହାର ରାଶି ସହିତ ସମାନ" । ଉକ୍ତିଟିକୁ ବିଚାର କରି ନିମ୍ନ ପ୍ରୋଗ୍ରାମର ଆଉଟ୍‌ପୁଟ କେତେ ହେବ ଦର୍ଶାଅ –

```
#include<stdio.h>
struct st
{
     int x;
     static int y;
};

int main()
{
printf("%d", sizeof(struct st));
return 0;
}
```

(a) 4 (b) 8
(c) ଚାଳନା ସମୟରେ ତ୍ରୁଟି (d) ସଂକଳନ ସମୟରେ ତ୍ରୁଟି

9. ନିମ୍ନୋକ୍ତ କେଉଁ ଅପରେଟରଟି ଷ୍ଟ୍ରକ୍‌ଚର ଚଳ ଉପରେ ପ୍ରୟୋଗ କରାଯାଇପାରିବ ?
(a) == (b) =
(c) > (d) +

10. "s.b = 10" ପରି କୋଡ୍‌ର ଉପସ୍ଥିତି ଏକ _____ କୁ ସୂଚାଉଛି.
(a) ସାଧାରଣ ଚଳର ନାମ (b) double ତଥ୍ୟ-ପ୍ରକାର
(c) ଷ୍ଟ୍ରକ୍‌ଚର (d) ସିନ୍ଟାକ୍ସ ତ୍ରୁଟି

11. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ କ'ଣ ହେବ ?

```
struct Test
{
   char str[20];
};
void main()
{
   struct Test st1, st2;
   strcpy(st1.str, "C Quiz");
   st2 = st1;
   st1.str[0] = 'S';
   cout << st2.str;
}
```

(a) C Quiz (b) S Quiz
(c) ଚାଳନା ସମୟରେ ତ୍ରୁଟି (d) ସଂକଳନ ସମୟରେ ତ୍ରୁଟି

12. ଯଦି ତୁମେ ଏକ ଷ୍ଟ୍ରକ୍ଚର ଚଳକୁ ଏକ ଫଙ୍କ୍ସନକୁ ପ୍ରେରଣ କର ତେବେ ପ୍ରକୃତରେ କ'ଣ ପ୍ରେରଣ ହୋଇଥାଏ ?
    (a) ଷ୍ଟ୍ରକ୍ଚର ଚଳର ରେଫରେନ୍ସ
    (b) ଷ୍ଟ୍ରକ୍ଚର ଚଳର ପ୍ରତିରୂପ
    (c) ଷ୍ଟ୍ରକ୍ଚର ଚଳ ଆରମ୍ଭର ଠିକଣା
    (d) ଷ୍ଟ୍ରକ୍ଚର ଚଳ ଶେଷର ଠିକଣା
13. ନିମ୍ନଲିଖିତ C କୋଡ୍ ର ଆଉଟପୁଟ୍ ଉପରେ ମନ୍ତବ୍ୟ ଦିଅ ।

```
struct temp {
  int a;
  int b;
  int c;
};
main()
{
  struct temp p[] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
}
```

    (a) ଆକାର 3 ବିଶିଷ୍ଟ ଏକ ଷ୍ଟ୍ରକ୍ଚର ଆରେ ତିଆରି କରେ,
    (b) ଆକାର 9 ବିଶିଷ୍ଟ ଏକ ଷ୍ଟ୍ରକ୍ଚର ଆରେ ତିଆରି କରେ,
    (c) ଷ୍ଟ୍ରକ୍ଚରର ସଦସ୍ୟମାନଙ୍କୁ ବେଆଇନ ଆସାଇନମେଣ୍ଟ,
    (d) ଏକ ବହୁପରିସର ଆରେର ବେଆଇନ ଘୋଷଣା ।
14. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଷ୍ଟ୍ରକ୍ଚର ସଦସ୍ୟ ହୋଇପାରିବ ନାହିଁ?
    (a) ଆରେ
    (b) ଷ୍ଟ୍ରକ୍ଚର
    (c) ଷ୍ଟ୍ରିଙ୍ଗ୍
    (d) ଫଙ୍କ୍ସନ
15. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଏକ ସଠିକ୍ ଭାବରେ ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍ଚର ?
    (a) struct {int a;}
    (b) struct a_struct {int a;};
    (c) struct a_struct int a;
    (d) struct a_struct {int a;}

ଉତ୍ତର									
1.	*(d)*	**2.**	*(c)*	**3.**	*(b)*	**4.**	*(b)*	**5.**	*(c)*
6.	*(b)*	**7.**	*(a)*	**8.**	*(d)*	**9.**	*(b)*	**10.**	*(c)*
11.	*(a)*	**12.**	*(b)*	**13.**	*(a)*	**14.**	*(d)*	**15.**	*(b)*

ପ୍ରୋଗ୍ରାମିଂ (Programming) ସମସ୍ୟା

1. ୟାର୍ଡ, ଫିଟ୍, ଏବଂ ଇଞ୍ଚର ଏକ ଦୈର୍ଘ୍ୟ ଗଚ୍ଛିତ କରିବାକୁ distance ନାମରେ ଏକ ଷ୍ଟ୍ରକ୍ଚର ତିଆରି କର । ଏହି ଷ୍ଟ୍ରକ୍ଚରକୁ ବ୍ୟବହାର କରି, ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ବ୍ୟବହାରକାରୀଠାରୁ ଷ୍ଟ୍ରକ୍ଚରଦ୍ୱାରା ପ୍ରତିନିଧିତ୍ୱ ଫର୍ମାଟ୍ରେ ଦୁଇଟି ମାପ ଗ୍ରହଣ କରିବ ଏବଂ ଏହି ଦୁଇଟି ମାପ ମଧ୍ୟରେ ପରମ ପାର୍ଥକ୍ୟ ପ୍ରିଣ୍ଟ କରିବ ।
2. ଶିକ୍ଷାର୍ଥୀଙ୍କ ବିଷୟରେ ତଥ୍ୟ ଗଚ୍ଛିତ କରିବାପାଇଁ Student ନାମକ ଏକ ଷ୍ଟ୍ରକ୍ଚର ତିଆରି କର ଯେଉଁଥିରେ ନିମ୍ନଲିଖିତ ଉପାଦାନଗୁଡ଼ିକ ଥିବ – rollno, int ପ୍ରକାରର, name, ଆକାର 20 ବିଶିଷ୍ଟ ଏକ char ପ୍ରକାରର ଆରେ, college,

ଆକାର 40 ବିଶିଷ୍ଟ ଏକ char ପ୍ରକାରର ଆରେ, ଏବଂ score, float ପ୍ରକାରର । ଅନୁମାନ କର ଯେ ସେଠାରେ 100 ରୁ ଅଧିକ ଶିକ୍ଷାର୍ଥୀ ନାହାଁନ୍ତି । ଶିକ୍ଷାର୍ଥୀମାନଙ୍କ ବିଷୟରେ ତଥ୍ୟ ଇନ୍‌ପୁଟ୍ ଏବଂ ସ୍କୋର୍ ଅନୁଯାୟୀ ଗଚ୍ଛିତ ହୋଇଥିବା ତଥ୍ୟ ଆଉଟ୍‌ପୁଟ୍ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

3. ନିମ୍ନଲିଖିତ ଘୋଷଣାନାମାକୁ ଦେଖ:

```
struct EMPLOYEE
{
   int code;
   char name[31];
   float salary;
};
```

ଦିଆଯାଇଥିବା ଏକ ସଂଗଠନର କର୍ମଚାରୀ ସଂଖ୍ୟା n (≤ 50) । ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହାକି ସର୍ବାଧିକ ଦରମା ଦେଉଥିବା କର୍ମଚାରୀଙ୍କ ବିବରଣୀ ଆଉଟ୍‌ପୁଟ କରିବ ।

4. ତିନୋଟି କ୍ଷେତ୍ର ଘଣ୍ଟା, ମିନିଟ୍ ଏବଂ ସେକେଣ୍ଡ ଥିବା ସମୟକୁ 12 ଘଣ୍ଟା ସମୟ ସିଷ୍ଟମରେ ମଡେଲ୍ କରିବାପାଇଁ Time ନାମକ ଏକ ଷ୍ଟ୍ରକ୍‌ଚର ତିଆରି କର । elapsedTime ନାମକ ଏକ ଫଙ୍କ୍‌ସନ ଲେଖ ଯାହା ଦୁଇଟି ଚର ନେଇଥାଏ, start_time ଏବଂ the end_time । ଏହି ଫଙ୍କ୍‌ସନ ଏକ Time ଷ୍ଟ୍ରକ୍‌ଚର ଫେରସ୍ତ କରିବା ଉଚିତ ଯେଉଁଥିରେ ଦୁଇ ଚର ମଧ୍ୟରେ ଅତିବାହିତ ହୋଇଥିବା ସମୟ ଥିବ । ଉପରୋକ୍ତ ଫଙ୍କ୍‌ସନ ପରୀକ୍ଷା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

5. କାର୍ଟେସିଆନ୍ ସିଷ୍ଟମରେ(Cartesian system) ସମତଳର ଏକ ବିନ୍ଦୁକୁ ଏହାର ସ୍ଥାନାଙ୍କ(coordinates) x ଏବଂ y ଦ୍ୱାରା ଉପସ୍ଥାପିତ କରାଯାଇପାରେ । ନିମ୍ନରେ ଦର୍ଶାଯାଇଥିବା ପରି ସମତଳର ଏକ ବିନ୍ଦୁକୁ ଉପସ୍ଥାପିତ କରିବାପାଇଁ ଆମେ ଏକ ଷ୍ଟ୍ରକ୍‌ଚର ତିଆରି କରିପାରିବା:

```
struct POINT
{
    int x;
    int y;
};
```

ଏକ ଫଙ୍କ୍‌ସନ ଲେଖ ଯାହା ଏକ ବିନ୍ଦୁ ଉପସ୍ଥାପିତ କରୁଥିବା ଷ୍ଟ୍ରକ୍‌ଚର ଗ୍ରହଣ କରେ ଏବଂ ଏକ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା 1, 2, 3 କିମ୍ବା 4 ଫେରସ୍ତ କରେ ଯାହା ବିନ୍ଦୁଟି ଅବସ୍ଥିତ ଥିବା ପାଦକୁ(quadrant) ସୂଚାଇଥାଏ । ଉପରୋକ୍ତ ଫଙ୍କ୍‌ସନ ପରୀକ୍ଷା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

6. ତିନି ସଦସ୍ୟ ବିଶିଷ୍ଟ ଏକ ଷ୍ଟ୍ରକ୍‌ଚରର ବ୍ୟାଖ୍ୟା କର– ଦେଶର ନାମ, ରାଜଧାନୀର ନାମ, ଏବଂ ମୁଣ୍ଡପିଛା ଆୟ । ଏହି ଷ୍ଟ୍ରକ୍‌ଚରକୁ ବ୍ୟବହାର କରି, ଦେଶଗୁଡ଼ିକୁ ସେମାନଙ୍କ ରାଜଧାନୀ ସହିତ ମୁଣ୍ଡପିଛା ଆୟ ହ୍ରାସ କ୍ରମରେ ତାଲିକାଭୁକ୍ତ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ପ୍ରାକ୍ଟିକାଲ୍ (PRACTICALS)

1. ସମୟ ଉପସ୍ଥାପିତ କରିବାପାଇଁ (24 ଘଣ୍ଟା ଫର୍ମାଟରେ) ଏକ ଷ୍ଟ୍ରକ୍ଚର ବ୍ୟବହାର କରି ଏକ ଡିଜିଟାଲ୍ ଘଣ୍ଟାର ନକଲ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

Listing 8.8

```
#include <stdio.h>
/* typedefined structure to represent clock time */
typedef struct {
        int hh;
        int mm;
   int ss;
} CLOCK;
int main()
{
   CLOCK myClock = { 12, 40, 55 };
    int i;
    printf("\nPress any key to stop the clock\n");
    while ( !kbhit() )
         {
       /*  increment clock  */
       (myClock.ss)++;
       if ( myClock.ss == 60 ) {
          myClock.ss = 0;
          (myClock.mm)++;
          if ( myClock.mm == 60 ) {
             myClock.mm = 0;
             (myClock.hh)++;
             if ( myClock.hh == 24 ) {
                 myClock.hh = 0;
             }
          }
              }
              /*  show clock  */
       printf("%02d:%02d:%02d\n",
              myClock.hh, myClock.mm, myClock.ss);
     }
   return 0;
}
```

Test Run

```
Press any key to stop the clock
12:40:56
12:40:57
12:40:58
12:40:59
12:41:00
12:41:01
12:41:02
```

2. ସମୟକୁ ଉପସ୍ଥାପିତ କରୁଥିବା (12 ଘଣ୍ଟା ଫର୍ମାଟରେ) ଏକ ଷ୍ଟ୍ରକ୍‌ଚର ବ୍ୟବହାର କରି ଆରମ୍ଭ ସମୟ ଏବଂ ଶେଷ ସମୟ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ ବାହାର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ତାଲିକା 8.9 (Listing 8.9)

```
#include <stdio.h>
 typedef struct {
   int hh;
   int mm;
   int ss;
 } TIME;

 /*  function prototype  */
 TIME differenceTime( TIME t1, TIME t2);

 int main()
 {
    TIME startTime, stopTime, diffTime;
    printf("Enter the start time in format hh mm ss : ");
    scanf("%d %d %d", &startTime.hh, &startTime.mm, &startTime.ss);
    printf("Enter the stop time in format hh:mm:ss : ");
    scanf("%d %d %d", &stopTime.hh, &stopTime.mm, &stopTime.ss);
    /*  functiona call  */
    diffTime = differenceTime(startTime, stopTime);
    printf("\nTime Difference: %d:%d:%d\n", diffTime.hh,
             diffTime.mm,
             diffTime.ss);
    return 0;
 }
 /* function that return difference between time periods */
 TIME differenceTime( TIME t1, TIME t2)
 {
```

```
  TIME temp;
    if ( t1.ss > t2.ss ) {
 --t2.mm;
       t2.ss += 60;
    }
    temp.ss = t2.ss - t1.ss;
    if ( t1.mm > t2.mm ) {
       --t2.hh;
       t2.mm += 60;
    }
temp.mm = t2.mm - t1.mm;
    temp.hh = t2.hh - t1.hh;
    return temp;
 }
```

ଟେଷ୍ଟ ରନ୍ (Test Run)

```
Enter the start time in format: hh mm ss : 5 20 30
Enter the stop time in format: hh mm ss : 7 15 10
Time Difference: 1:54:40
```

ଅଧିକ ଜାଣିବାପାଇଁ

ଷ୍ଟ୍ରକ୍ଚର ହେଉଛି ଏକ ବ୍ୟବହାରକାରୀ-ନିର୍ଦ୍ଧାରିତ ପ୍ରକାର । ଅନେକ ସମସ୍ୟା ଅଛି ଯେଉଁଥିରେ ମୌଳିକ ପ୍ରକାର (basic types) କିମ୍ବା ଏଥିରୁ ନିଷ୍ପନ୍ନ (derive) ହୋଇଥିବା ଆରେ, ବାସ୍ତବ ଜୀବନର ସମସ୍ୟାକୁ ମଡେଲ୍ କରିବାକୁ ସମର୍ଥ ନୁହେଁ । ସେହି କ୍ଷେତ୍ରରେ ଆମକୁ ବ୍ୟବହାରକାରୀ-ନିର୍ଦ୍ଧାରିତ ପ୍ରକାର ନାମକ ଏକ ନୂତନ ତଥ୍ୟ-ପ୍ରକାର (data type) ତିଆରି କରିବା ଦରକାର, ଯାହା C ରେ ଏକ ଷ୍ଟ୍ରକ୍ଚର ଅଟେ ।

ବାସ୍ତବ ଜୀବନର ସତ୍ତାଗୁଡ଼ିକ ବିଷୟରେ ସୂଚନା ଲିପିବଦ୍ଧର କରିପାରୁଥିବା ବାସ୍ତବ ଜୀବନ ପ୍ରୟୋଗରେ, ଷ୍ଟ୍ରକ୍ଚରର ବ୍ୟବହାର ହେଉଛି ଏକମାତ୍ର ଉପାୟ । ଏହା ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାର ସମାଧାନରେ ଷ୍ଟ୍ରକ୍ଚରର ଗୁରୁତ୍ୱକୁ ସୂଚିତ କରେ ।

ଶିକ୍ଷକ ଷ୍ଟ୍ରକ୍ଚରର ଧାରଣା ବିଷୟରେ ବୁଦ୍ଧିମତ୍ତାର ବିକାଶ କରିବେ ଏବଂ ବାସ୍ତବ ଜୀବନରୁ ଉପଯୁକ୍ତ ଉଦାହରଣ ନେଇ ଷ୍ଟ୍ରକ୍ଚରର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରିବେ ବୋଲି ଆଶା କରାଯାଉଛି ।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

9 ପଏଁଟର

ଏକକର ବିଶେଷତ୍ୱ

ଏହି ଏକକରେ ପଏଁଟର ସମ୍ବନ୍ଧୀୟ ବିଷୟଗୁଡ଼ିକ ଉପରେ ଆଲୋଚନା କରାଯାଇଅଛି । ପଏଁଟରଗୁଡ଼ିକ C ଭାଷାର ଏକ ମହତ୍ୱପୂର୍ଣ୍ଣ ବିଷୟ ଉପସ୍ଥାପିତ କରେ । ଏହାର ବ୍ୟବହାର ଦ୍ୱାରା C ଭାଷାର ବାସ୍ତବ ଶକ୍ତିର ଉପଯୋଗ କରାଯାଇପାରେ । ଏହି ଏକକରେ ପଏଁଟରଗୁଡ଼ିକର ବିଭିନ୍ନ ଦିଗ ବ୍ୟାଖ୍ୟା କରାଯାଇଅଛି ଏବଂ ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ସେଗୁଡ଼ିକର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରାଯାଇଅଛି ।

ଭୂମିକା

ସାଧାରଣ ପରିସ୍ଥିତିରେ, ପ୍ରତ୍ୟେକ C ପ୍ରୋଗ୍ରାମ୍‌କୁ ତଥ୍ୟ ଏବଂ ପ୍ରୋଗ୍ରାମ୍ ନିର୍ଦ୍ଦେଶାବଳିର ସମଷ୍ଟି ପାଇଁ ମେମୋରି ଆବଣ୍ଟନ କରାଯାଇଥାଏ । ଉପରୋକ୍ତ ଉଭୟ କାର୍ଯ୍ୟର ସମଷ୍ଟି ପାଇଁ ଆବଶ୍ୟକ ମେମୋରିର ପରିମାଣ ସଂକଳନ ସମୟରେ ନିର୍ଦ୍ଧାରଣ କରି ଏହାକୁ ସ୍ଥିର ରଖାଯାଏ । ପ୍ରୋଗ୍ରାମ୍ ଚାଲିବା ସମୟରେ ଏହାର ପରିବର୍ତ୍ତନ ହୋଇପାରିବ ନାହିଁ ।

କାର୍ଯ୍ୟତଃ, ତୁମେ ପ୍ରତ୍ୟେକ ଥର ପ୍ରୋଗ୍ରାମ ଚଳାଇଲା ବେଳେ ନିର୍ଦ୍ଦେଶାବଳି ପାଇଁ ଆବଶ୍ୟକ ମେମୋରି ସ୍ଥିର ଥାଏ ତଥ୍ୟର ଆକାର ଗୋଟିଏ ସମସ୍ୟା ସମସ୍ୟାଠାରୁ ଆଉ ଏକ ସମସ୍ୟା ଭିନ୍ନ ହୋଇପାରେ ।

ତଥ୍ୟର ଅଲଗାଅଲଗା ଆକାର, ଅର୍ଥାତ୍ ତଥ୍ୟର ଗତିଶୀଳତା, କେବଳ ମାତ୍ର ପ୍ରୋଗ୍ରାମର ତଥ୍ୟର ଘୋଷଣାରେ ପରିବର୍ତ୍ତନ କରି ପ୍ରୋଗ୍ରାମ୍‌ର ପୁନଃ ନିର୍ମାଣ କରିବାଦ୍ୱାରା ହିଁ ସମାହିତ ହୋଇପାରିବ ।

ଉତ୍ସ କୋଡ଼ ପାଖରେ ତୁମର ପହଁଞ୍ଚ ନଥିଲେ, ଏହି ଉପାୟଟି କାମ କରିବ ନାହିଁ ।

କିନ୍ତୁ ପଏଁଟର ଏହି ବିଷୟ ପରିଚାଳନା କରିବାକୁ ଏକ ଉତ୍ତମ ବିକଳ୍ପ ପ୍ରଦାନ କରେ, ଯେପରିକି ପଏଁଟର ବ୍ୟବହାର କରି, ପ୍ରୋଗ୍ରାମ୍ ଚାଳନ ବେଳେ ତଥ୍ୟର ପ୍ରକୃତ ଆବଶ୍ୟକତା ଅନୁଯାୟୀ ମେମୋରି ଆବଣ୍ଟନ କରାଯାଇପାରିବ ।

ଏଥି ସହିତ, ପଏଁଟର ବ୍ୟବହାର ନିମ୍ନଲିଖିତ ପରିସ୍ଥିତିରେ ଲାଭ ପ୍ରଦାନ କରେ:

- ଦକ୍ଷତାର ସହ ଏକ ଫଙ୍କ୍‌ସନକୁ ଚର ପ୍ରେରଣ ।
- ଚର ମାଧ୍ୟମରେ ଡାକିଥିବା ଫଙ୍କ୍‌ସନକୁ ଏକାଧିକ ମୂଲ୍ୟ ଫେରସ୍ତ କରିପାରେ ।
- ପ୍ରୋଗ୍ରାମ୍‌ଟି ସଂସ୍ଥାପିତ ମେମୋରିର ଯେକୌଣସି ମେମୋରି ଅବସ୍ଥାନକୁ ପହଁଞ୍ଚି ପାରେ । ଏଥିସହିତ ଯେକୌଣସି ଠିକଣା ଥିବା ସଂଯୋଜିତ ଉପକରଣଗୁଡ଼ିକୁ ମଧ୍ୟ ପହଁଞ୍ଚି ପାରେ ।
- ଜଟିଳ, ବାସ୍ତବ ଜୀବନ ସମସ୍ୟାକୁ ମଡେଲ୍ କରିବା ପାଇଁ ଜଟିଳ ଡାଟା-ଷ୍ଟ୍ରକ୍‌ଚରର ନିର୍ମାଣ କରିବା ।

ସଂକ୍ଷେପରେ, ଆମେ କହିପାରିବା ଯେ C ଭାଷାର ସବୁଠାରୁ ଭିନ୍ନ ଏବଂ ରୋମାଞ୍ଚକର ବୈଶିଷ୍ଟ୍ୟ ମଧ୍ୟରୁ ପଏଁଟର ଅନ୍ୟତମ । ଏହା ଭାଷାକୁ ସାମର୍ଥ୍ୟ ଏବଂ ନମନୀୟତା ପ୍ରଦାନ କରେ ।

ଏହି ୟୁନିଟ ଛାତ୍ରଛାତ୍ରୀଙ୍କୁ ପଏଁଟର ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ଦିଗଗୁଡ଼ିକୁ ବୁଝିବାରେ ଏବଂ ବାସ୍ତବ ଜୀବନର ସମସ୍ୟାଗୁଡ଼ିକୁ ଦକ୍ଷତାର ସହ ସମାଧାନ କରିବାରେ ପଏଁଟର ବ୍ୟବହାର କରି ପ୍ରୋଗ୍ରାମ୍ ବିକାଶ କରିବାରେ ସାହାଯ୍ୟ କରିବ ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- ମୌଳିକ ତଥ୍ୟ ପ୍ରକାର
- ଅପରେଟର
- ଆରେ ଏବଂ ଷ୍ଟ୍ରିଙ୍ଗ୍
- ଫଙ୍କ୍ସନ
- ଷ୍ଟ୍ରକ୍ଚରର

ଏକକଲବ୍ଧ ଜ୍ଞାନ

ଏହି ଏକକରେ ସମ୍ପୂର୍ଣ୍ଣ ଅଧ୍ୟୟନ ପରେ, ଶିକ୍ଷାର୍ଥୀଗଣ ନିମ୍ନଲିଖିତ ବିଷୟଗୁଡ଼ିକରେ ଜ୍ଞାନ ଆହରଣରେ ସମର୍ଥ ହେବେ ।

U9-O1: C ପଏଁଟର ଦୁନିଆକୁ ଭଲ ଭାବେ ବୁଝିବା,
U9-O2: ପଏଁଟର ଘୋଷଣା ଏବଂ ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରିବା,
U9-O3: ଗୋଟିଏ ପଏଁଟରକୁ ଡ଼ିରେଫରେନ୍ସ କରି ତଥ୍ୟରେ ପହଁଞ୍ଚିବା,
U9-O4: ପଏଁଟର ଉପରେ ଅନୁମୋଦିତ ଅପରେସନ୍ଗୁଡ଼ିକର ବ୍ୟାଖ୍ୟା କରିବା,
U9-O5: ଗୋଟିଏ ଫଙ୍କ୍ସନକୁ ଚର ପ୍ରେରଣରେ ପଏଣ୍ଟରର ଉପଯୋଗୀତାକୁ ବ୍ୟାଖ୍ୟା କରିବା,
U9-O6: ଗତିଶୀଳ ମେମୋରି ଆବଣ୍ଟନ ଧାରଣାର ବ୍ୟାଖ୍ୟା କରିବା,
U9-O7: ବିଭିନ୍ନ ସମସ୍ୟାର ସମାଧାନ ପାଇଁ ପଏଁଟରର ବ୍ୟବହାରଦ୍ୱାରା ଫଳପ୍ରଦ ପ୍ରୋଗ୍ରାମ୍ ଲେଖିବା,

ୟୁନିଟ-9 ୟୁନିଟଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ *(1 – ଦୁର୍ବଳ ସମ୍ପର୍କ; 2 – ମଧ୍ୟମ ସମ୍ପର୍କ; 3 – ଦୃଢ଼ ସମ୍ପର୍କ)*							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U9-O1	–	–	1	–	–	–	–	–
U9-O2	–	–	1	–	–	–	–	–
U9-O3	–	–	1	–	–	–	–	–
U9-O4	–	–	–	–	–	2	–	–
U9-O5	–	–	–	–	2	–	–	–
U9-O6	–	–	–	–	–	2	–	–
U9-O7	–	–	–	–	–	3	–	–

9.1 ଉପକ୍ରମ (INTRODUCTION)

C ଭାଷାର ସବୁଠାରୁ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ବୈଶିଷ୍ଟ୍ୟ ମଧ୍ୟରୁ ପଏଁଟର ହେଉଛି ଅନ୍ୟତମ। ଅଧିକାଂଶ ଲୋକଙ୍କ ଦ୍ୱାରା ପଏଁଟରଗୁଡ଼ିକ, C ର ସବୁଠାରୁ କଠିନ ବିଷୟ ଭାବରେ ବିବେଚନା କରାଯାଏ। ମୁଁ ତୁମକୁ ସ୍ୱଷ୍ଟ ଭାବରେ କହିବାକୁ ଚାହେଁ ଯେ, ଏହା ବାସ୍ତବତା ଅପେକ୍ଷା ଅଧିକ ମିଥ୍। ମୁଁ କହିବାକୁ ପସନ୍ଦ କରିବି ଯେ ପଏଁଟର C ଭାଷାର ସବୁଠାରୁ ସୂକ୍ଷ୍ମ ଦିଗ ମଧ୍ୟରୁ ଅନ୍ୟତମ। ତୁମେ ଜାଣିଛ; ସୂକ୍ଷ୍ମ ଜିନିଷଗୁଡ଼ିକୁ ସତର୍କତାର ସହିତ ପରିଚାଳନା କରିବା ଆବଶ୍ୟକ। ତେଣୁ, ପଏଁଟର ଯତ୍ନର ସହ ପରିଚାଳନା ଆବଶ୍ୟକ କରେ।

ପଏଁଟର ବ୍ୟବହାର କରିବାର ଅନେକ କାରଣ ଅଛି। ସେଥି ମଧ୍ୟରୁ କେତେକ ନିମ୍ନରେ ଦିଆଯାଇଛି:

- ପଏଁଟରଟିଏ ଏକ ଫଙ୍କ୍ସନ କିମ୍ବା ଏକ ପ୍ରୋଗ୍ରାମକୁ ନିଜର କାୟା/କ୍ଷେତ୍ରର ବାହାରେ ଚଳ ପାଖରେ ପହଁଞ୍ଚିବାକୁ ସୁବିଧା ଦିଏ (ଚଳର ମେମୋରି ଅବସ୍ଥାନରେ)। ଏକ ପଏଁଟର ବ୍ୟବହାର କରି, ତୁମ ପ୍ରୋଗ୍ରାମ୍ କମ୍ପ୍ୟୁଟରର ମେମୋରିରେ ଯେକୌଣସି ମେମୋରି ଅବସ୍ଥାନକୁ ପହଁଞ୍ଚି ପାରିବ।
- return ବିବୃତ୍ତି ବ୍ୟବହାର ମାଧ୍ୟମରେ ଗୋଟିଏ ଫଙ୍କ୍ସନ ଡାକି ଥିବା ଫଙ୍କ୍ସନକୁ କେବଳ ଗୋଟିଏ ମୂଲ୍ୟ ଫେରସ୍ତ କରିପାରେ। ଡାକି ଥିବା ଫଙ୍କ୍ସନ ପହଁଞ୍ଚି ପାରୁଥିବା ମେମୋରି ସ୍ଥାନରେ, ଡକା ହୋଇଥିବା ଫଙ୍କ୍ସନ ଏକାଧିକ ମୂଲ୍ୟ ଲେଖିବାଦ୍ୱାରା ପଏଁଟର ଏକ ଫଙ୍କ୍ସନକୁ ଏକାଧିକ ମୂଲ୍ୟ ଫେରସ୍ତର (ପରୋକ୍ଷ ଭାବେ) ଅନୁମତି ଦେଇଥାଏ।
- ପଏଁଟରର ବ୍ୟବହାରଦ୍ୱାରା, ଆରେ ଏବଂ ଷ୍ଟ୍ରକ୍‌ଚରକୁ ଅଧିକ ଦକ୍ଷ ଉପାୟରେ ପରିଚାଳନା କରାଯାଇପାରିବ।
- ପଏଁଟର ବିନା, ଜଟିଳ ଡାଟା ଷ୍ଟ୍ରକ୍ଚରଗୁଡ଼ିକ ଯେପରିକି ଲିଙ୍କ୍‌ଡ ଲିଷ୍ଟ (linked lists), ଟ୍ରୀ (trees) ଏବଂ ଗ୍ରାଫ୍(graphs) ଇତ୍ୟାଦି ତିଆରି କରିବା ସମ୍ଭବ ହେବ ନାହିଁ।
- ମେମୋରି ବିଷୟରେ ଅପରେଟିଂ ସିଷ୍ଟମ୍ ସହ ଯୋଗାଯୋଗ କରିବାକୁ। ଉଦାହରଣ ସ୍ୱରୂପ, ଅପରେଟର୍ new ଏକ ପଏଁଟର ବ୍ୟବହାର କରି ଅବ୍ୟବହୃତ ମେମୋରି ବ୍ଲକ୍‌ର ଅବସ୍ଥାନ ଫେରସ୍ତ କରେ ଏବଂ ଅପରେଟର୍ delete, ଅପରେଟିଂ ସିଷ୍ଟମକୁ ଏକ ପଏଁଟର ଦ୍ୱାରା ଇଙ୍ଗିତ ମେମୋରି ବ୍ଲକ୍‌କୁ ଫେରସ୍ତ କରେ।

ସମୁଦାୟ ଭାବରେ, ଆମେ କହିପାରିବା ଯେ C ଭାଷାର ପ୍ରକୃତ ସାମର୍ଥ୍ୟ ପଏଁଟରଗୁଡ଼ିକର ବୌଦ୍ଧିକ ବ୍ୟବହାରରେ ଅନ୍ତର୍ନିହିତ। ତେଣୁ, C ର ପ୍ରତ୍ୟେକ ପାଠକ ନିଶ୍ଚିତ ଭାବରେ ପଏଁଟର ବ୍ୟବହାର କରିବାର କଳା ଶିଖିବା ଏବଂ ଆୟତ୍ତ କରିବା ଆବଶ୍ୟକ।

ଏହି ୟୁନିଟରେ, ଆମେ C ଭାଷାରେ ପଏଁଟର ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ଦିଗ ବିଷୟରେ ଜାଣିବା।

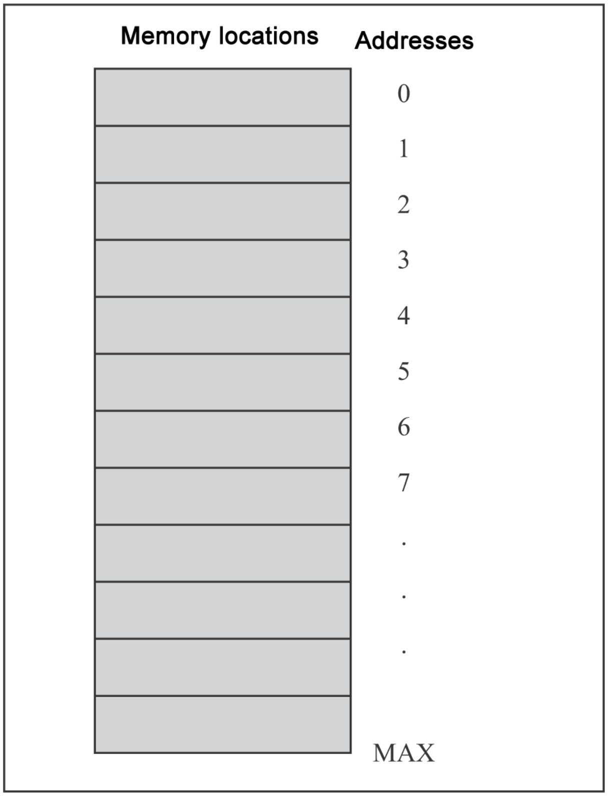
9.2 ପଏଁଟରର ଧାରଣା (IDEA OF POINTERS)

କମ୍ପ୍ୟୁଟରର memory, bytes କିମ୍ବା word ର ଏକ ରୈଖିକ(linear) ସଂଗ୍ରହ ଭାବେ ବ୍ୟବସ୍ଥିତ। ଇଣ୍ଟେଲ୍ ପ୍ରୋସେସର୍ ପାଇଁ, ମେମୋରି ବାଇଟ୍-ବ୍ୟବସ୍ଥିତ ହୋଇଥିବାବେଳେ ଅଧିକାଂଶ ଅଣ-ଇଣ୍ଟେଲ୍ ପ୍ରୋସେସର୍ ପାଇଁ, ଏହା ୱାର୍ଡ଼-ବ୍ୟବସ୍ଥିତ। ବାଇଟ୍-ବ୍ୟବସ୍ଥିତ ମେମୋରିରେ, ପ୍ରତ୍ୟେକ ମେମୋରି ଅବସ୍ଥାନ ଗୋଟିଏ ବାଇଟ୍ ଯେତେବେଳେ ୱାର୍ଡ଼-ବ୍ୟବସ୍ଥିତରେ, ପ୍ରତ୍ୟେକ ମେମୋରି ଅବସ୍ଥାନ ଗୋଟିଏ ୱାର୍ଡ଼ର ଅଟେ। ପ୍ରତ୍ୟେକ ମେମୋରି ଅବସ୍ଥାନକୁ ଏକ ଅନନ୍ୟ ସଂଖ୍ୟା ନ୍ୟସ୍ତ ହୋଇଛି ଯାହାକୁ ଅବସ୍ଥାନ ସଂଖ୍ୟା କିମ୍ବା ଠିକଣା ବୋଲି କୁହାଯାଏ। ଚିତ୍ର ୨.1 ରେ ମେମୋରି ସଂଗଠନର ଯୋଜନା ପ୍ରଦର୍ଶିତ ହୋଇଛି।

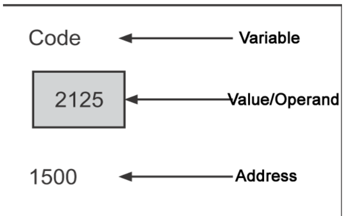
ଆମେ ଜାଣୁ ଯେ ଗୋଟିଏ ବାଇଟ୍ ୮ ଟି ବିଟ୍(bits) ସହିତ ସମାନ, ଯେତେବେଳେ କି ଗୋଟିଏ ୱାର୍ଡ଼ ଦୁଇ କିମ୍ବା ଅଧିକ ବାଇଟ୍‌ରେ ଗଠିତ ହୋଇପାରେ। ଗୋଟିଏ ୱାର୍ଡ଼, ଏକକ ଅପରେସନରେ (ଅର୍ଥାତ, ଯୋଗ, ବିୟୋଗ, ଗୁଣନ, ବିଭାଜିନ, ତୁଳନା ଇତ୍ୟାଦି) ପ୍ରକ୍ରିୟାକରଣ କରିପାରୁଥିବା ଅପେରାଣ୍ଡର(operand) ଆକାର ଅନୁଯାୟୀ ପ୍ରୋସେସରର କ୍ଷମତା ମାପ କରେ। ଯେତେବେଳେ ଆମେ କହୁ ଏକ ପ୍ରୋସେସର୍ ହେଉଛି ଏକ 16-ବିଟ୍, ଏହାର ଅର୍ଥ ୱାର୍ଡ଼ର ଆକାର 16-ବିଟ୍ (2 ବାଇଟ୍) ଏବଂ ଏହା ଏକକ ଅପରେସନରେ 16-ବିଟ୍ ଅପେରାଣ୍ଡର ପ୍ରକ୍ରିୟା କରିପାରିବ। ବଡ଼ ଆକାରର ଅପେରାଣ୍ଡ ପାଇଁ, ଏହା ଅଧିକ ଅପରେସନ୍ ଆବଶ୍ୟକ କରେ।

ଏକ ମେମୋରି କୋଷ ହେଉଛି ଗୋଟିଏ ବାଇଟ୍/ୱାର୍ଡ଼ କିମ୍ବା ଅଧିକ ସଂଲଗ୍ନ ବାଇଟ୍/ୱାର୍ଡ଼। ପ୍ରତ୍ୟେକ କୋଷ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ସମୟରେ ଗୋଟିଏ ମୂଲ୍ୟ ଗଚ୍ଛିତ କରିପାରିବ। ଏକ କୋଷର ଠିକଣା ସର୍ବଦା ପ୍ରଥମ ବାଇଟ୍ / ୱାର୍ଡ଼ର ଠିକଣା ହୋଇଥାଏ।

ପ୍ରୋଗ୍ରାମ୍ ଚଳାଇବା ପାଇଁ ନିର୍ଦ୍ଦେଶ ଦେଲେ, ଅପରେଟିଂ ସିଷ୍ଟମ୍ ପ୍ରଥମେ କମ୍ପ୍ୟୁଟର୍ ମେମୋରିରୁ (ମୁଖ୍ୟ) ଆବଶ୍ୟକ ଆକାରର ଏକ ଅବ୍ୟବହୃତ ବ୍ଲକ୍ ପାଇଥାଏ। ତାପରେ ସେହି ବ୍ଲକ୍ରେ ପ୍ରୋଗ୍ରାମକୁ ଲୋଡ୍‌କରେ। ଏପରିକି ସେହି ବ୍ଲକ୍‌ରେ, ସାଧାରଣତଃ, ପ୍ରୋଗ୍ରାମର କୋଡ୍ ଗୋଟିଏ ଭାଗରେ ଲୋଡ୍ ହୋଇଥାଏ ଏବଂ ବ୍ଲକର ଅନ୍ୟ ଏକ ଅଂଶରେ ପ୍ରୋଗ୍ରାମର ତଥ୍ୟ ଲୋଡ୍ ହୋଇଥାଏ। ପ୍ରତ୍ୟେକ ତଥ୍ୟ ଅପେରାଣ୍ଡ୍ ଏକ କୋଷରେ ଗଚ୍ଛିତ ହୋଇଥାଏ ସିଷ୍ଟମ୍ ଏହି ଠିକଣା ସହିତ ପ୍ରତ୍ୟେକ ଚଳକୁ ସଂଯୁକ୍ତ କରେ। ଏହା ଚିତ୍ର 9.2 ରେ ଦର୍ଶାଯାଇଛି।

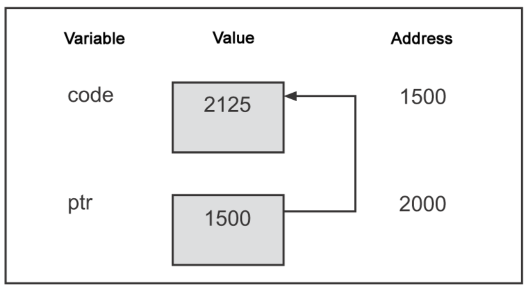


ଚିତ୍ର 9.1 ମେମୋରୀ ସଂଗଠନ



ଚିତ୍ର . 9.2: ମେମୋରୀରେ ଏକ ଚଳର ଉପସ୍ଥାପନା

ତଥ୍ୟ ଅପେରାଣ୍ଡ୍‌ରେ ପହଁଞ୍ଚିବାକୁ ହୁଏତ ଆମେ ଚଳର ନାମ ବ୍ୟବହାର କରୁ କିମ୍ବା ମେମୋରି କୋଷର ଠିକଣା ବ୍ୟବହାର କରୁ। ଯେହେତୁ ଏହି ଠିକଣାଗୁଡ଼ିକ କେବଳ ଧନାତ୍ମକ ପୂର୍ଣ୍ଣସଂଖ୍ୟା, ସେଗୁଡ଼ିକ ଅନ୍ୟ ଚଳ ପରି ମେମୋରିରେ ଗଚ୍ଛିତ କିଛି ଚଳକୁ ମଧ୍ୟ ନ୍ୟସ୍ତ କରିପାରେ। ଠିକଣା ଧାରଣ କରୁଥିବା ଏହିପରି ଚଳଗୁଡ଼ିକୁ ପଏଁଟର ବୋଲି କୁହାଯାଏ। ତେଣୁ ଏକ ପଏଁଟର ହେଉଛି ଏକ ଚଳ, ଯାହା ମେମୋରିରେ ଅନ୍ୟ ଏକ ଚଳର ଠିକଣା ଧାରଣ କରିଥାଏ।



ଚିତ୍ର . 9.3: ପଏଁଟର ଏକ ଚଳ ଭାବରେ

ଯେହେତୁ ପଏଁଟର ଏକ ଚଳ, ଏହାର ମୂଲ୍ୟ ମେମୋରିରେ ଅନ୍ୟ ଏକ ଠିକଣାରେ ମଧ୍ୟ ଗଚ୍ଛିତ ହୋଇଥାଏ। ଧରାଯାଉ ଆମେ ଏକ ଚଳ ptr କୁ ଚଳ code ର ଠିକଣା ନ୍ୟସ୍ତ କଲୁ ତା'ହେଲେ ଚଳ code ଏବଂ ptr ମଧ୍ୟରେ ଲିଙ୍କ୍ (ଚିତ୍ର 9.3 ରେ ଦର୍ଶାଯାଇଥିବା) ପରି କଳ୍ପନା କରାଯାଇପାରିବ।

ଯେହେତୁ ptr ଚଳର ମୂଲ୍ୟ ହେଉଛି code ଚଳର ଠିକଣା, ଆମେ ptr ଚଳର ମୂଲ୍ୟ ବ୍ୟବହାର କରି code ଚଳର ମୂଲ୍ୟରେ

ପହଁଞ୍ଚି ପାରିବା । ତେଣୁ, ଆମେ କହୁ ଯେ ptr ଚଳ code ଚଳକୁ ସୂଚିତ କରେ, ଏବଂ ତେଣୁ ptr ର ନାମ ପଏଁଟର ହୋଇଥାଏ ।

9.3 ଚଳର ଠିକଣାରେ ପହଁଞ୍ଚିବା (ACCESSING ADDRESS OF A VARIABLE)

ଗୋଟିଏ ଚଳର ପ୍ରକୃତ ଠିକଣା ହେଉଛି ଏକ ସିଷ୍ଟମ୍ ନିର୍ଭରଶୀଳ । ସଂକଳନ ଏବଂ ଲିଙ୍କିଂ ସମୟରେ, ଠିକଣାଗୁଡ଼ିକ କିଛି ମୂଳ ଠିକଣା ସହିତ ସମ୍ବନ୍ଧିତ ବୋଲି ଅନୁମାନ କରାଯାଏ । ଏହା ସାଧାରଣତଃ 0 ହୋଇଥାଏ । ଅପରେଟିଂ ସିଷ୍ଟମ୍ ପ୍ରୋଗ୍ରାମକୁ ଏକ ଅବ୍ୟବହୃତ ବ୍ଲକରେ ଲୋଡ୍ କରିଲେ ସମସ୍ତ ଠିକଣା ଅବ୍ୟବହୃତ ବ୍ଲକର ପ୍ରଥମ ମେମୋରି ଠିକଣାରେ ରୂପାନ୍ତରିତ ହୁଏ । ଆମେ ଏକ ଚଳର ଠିକଣା ଜାଣିନାହୁଁ । ତେଣୁ, ଏକ ପ୍ରଶ୍ନ ଉଠେ – ଆମେ କିପରି ଏକ ଚଳର ଠିକଣା ନିର୍ଦ୍ଧାରଣ କରିପାରିବା ? ଏହା ଠିକଣା ଅପରେଟର ସାହାଯ୍ୟରେ କରାଯାଇଥାଏ ।

ନିମ୍ନ ଉକ୍ତିକୁ ବିଚାର କର –

```
ptr = &code;
```

ଏହା ଚଳ ptr କୁ ସଂଖ୍ୟା 1500 (ଚଳ code ର ଠିକଣା) ନ୍ୟସ୍ତ କରିବ ।

ନିମ୍ନ ପ୍ରତିବନ୍ଧକଗୁଡ଼ିକ ପାଳନ କରାଯିବା ଆବଶ୍ୟକ:

1. ଠିକଣା ଅପରେଟର କେବଳ ଚଳ କିମ୍ବା ଏକ ଆରେ ଉପାଦାନ ସହିତ ବ୍ୟବହୃତ ହୋଇପାରିବ । ନିମ୍ନୋକ୍ତିଗୁଡ଼ିକ ଠିକଣା ଅପରେଟରର ଅବୈଧ ବ୍ୟବହାର ଅଟେ:

   (a) &1200 (ଧ୍ରୁବାଙ୍କ ଅଭିବ୍ୟକ୍ତିକୁ ଇସାରା)

   (b) &(a/b) (ଅଭିବ୍ୟକ୍ତିକୁ ଇସାରା)

2. ପଏଁଟର ଚଳର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟପାଇଁ କେବଳ ଠିକଣା ଅପରେଟରର ବ୍ୟବହାର କିମ୍ବା ଅନ୍ୟ ପଏଁଟର ଚଳର କରି କରାଯାଇପାରିବ । ଏକ ପଏଁଟର ଚଳ ptr ର ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ କରିବାର ନିମ୍ନପ୍ରଦତ୍ତ ଉପାୟ ଅବୈଧ ଅଟେ:

   ```
   ptr = 1500;
   ```

   ଯଦିଓ, ସଂଖ୍ୟା 1500 code ଚଳର ଠିକଣା ହୋଇଥାଏ ।

9.4 ପଏଁଟରର ଘୋଷଣା (DECLARING A POINTER)

ପଏଁଟର ଚଳର ଘୋଷଣା, ଏକ ସାଧାରଣ ଚଳର ଘୋଷଣା ସହିତ ସମାନ କିନ୍ତୁ ପଏଁଟର ଚଳର ନାମ * ସଙ୍କେତର ଉପସର୍ଗ ହୋଇଥାଏ ।

ଏକ ପଏଁଟର ଚଳକୁ ଘୋଷଣା କରିବା ପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
datatype *pointerVariableName
```

ଉଦାହରଣ ସ୍ୱରୂପ, ଏକ ବିବୃତ୍ତି

```
int *ptr;
```

ଏକ ପଏଁଟର ଚଳ ptr ର ଘୋଷଣା କରେ ଯାହା ଯେକୌଣସି ଇଣ୍ଟିଜର ଚଳର ଠିକଣା ଗଚ୍ଛିତ ପାଇଁ ବ୍ୟବହାର କରାଯାଇପାରିବ ।

9.5 ପଏଁଟରକୁ ଠିକଣା ନ୍ୟସ୍ତ କରିବା (ASSIGNING ADDRESS TO A POINTER)

ଥରେ ଏକ ପଏଁଟର ଘୋଷଣା ହେବା ପରେ, ଏହାକୁ ଏକ ଚଳର ଠିକଣା ଦିଆଯିବା ଆବଶ୍ୟକ । ସାଧାରଣ ଚଳ ପରି, ଏକ ପଏଁଟର ଚଳ ଘୋଷଣା ହେବା ପରେ ଏହା ଏକ ଅନାବଶ୍ୟକ ମୂଲ୍ୟ ରଖିଥାଏ । ତେଣୁ, ବ୍ୟବହାର ପୂର୍ବରୁ, ଏକ ପଏଁଟର ଚଳକୁ ଅନ୍ୟ କୌଣସି ଏକ ଚଳର ଠିକଣା ନ୍ୟସ୍ତ ହେବା ଆବଶ୍ୟକ ।

ଏକ ପଏଁଟର ଚଳକୁ ଠିକଣା ନ୍ୟସ୍ତ କରିବା ପାଇଁ ସିଣ୍ଟାକ୍ସ

```
pointerVariableName = &variableName;
```

ଉଦାହରଣ ସ୍ୱରୂପ, ଏକ ବିବୃତ୍ତି

```
int a, *ptr;
```

a ଏକ ସାଧାରଣ ଇଣ୍ଟିଜର ଚଳ ଏବଂ *ptr* ଏକ ଇଣ୍ଟିଜର ପଏଁଟର ଚଳ ଭାବରେ ଘୋଷଣା କରାଯାଇଛି ।

ପଏଁଟର ଚଳ *ptr* କୁ ଚଳ *a* ର ଠିକଣା ନ୍ୟସ୍ତ କରିବାକୁ, ଆମେ ନିମ୍ନ ଉକ୍ତି ବ୍ୟବହାର କରୁ

```
ptr = &a;
```

ଏଠାରେ ଆମେ କହୁଛୁ ଯେ ପଏଁଟର ଚଳ *ptr* ଚଳ *a* କୁ ଇସାରା କରୁଛି ।

9.6 ପଏଁଟର ବ୍ୟବହାରଦ୍ୱାରା ଚଳକୁ ପହଁଞ୍ଚିବା (ACCESSING VARIABLE USING a POINTER)

ଏକ ପଏଁଟର ଚଳକୁ ଥରେ ଅନ୍ୟ ଏକ ଚଳର ଠିକଣା ନ୍ୟସ୍ତ ହେବା ପରେ, ପ୍ରଶ୍ନ ହେଉଛି - ପଏଁଟର ଚଳ ବ୍ୟବହାର କରି ଉକ୍ତ ଚଳର ମୂଲ୍ୟରେ କିପରି ପହଁଞ୍ଚି ପାରିବା ? ଏହା * ସଙ୍କେତକୁ ପଏଁଟର ଚଳରେ ଉପସର୍ଗ କରାଯାଇଥାଏ ।

ଏକ ପଏଁଟର ଚଳ ବ୍ୟବହାର କରି ଅନ୍ୟ ଚଳକୁ ପହଁଞ୍ଚିବା ପାଇଁ ସିଣ୍ଟାକ୍ସ

```
*pointerVariableName
```

Listing 9.1

```
/*
    Program to illustrate the use of pointer variable
 */

 #include<stdio.h>

 int main()
 {
    int *ptr, x = 10;
    ptr = &x;
    printf("Value of variable x = %d\n", x );
    printf("Value of variable pointed to by ptr = %d\n",*ptr );
    printf("Address of variable x = %u\n", ptr );
    return 0;
 }
```

Test Run

```
Value of variable x = 10
Value of variable pointed to by ptr = 10
Address of variable x = 65524
```

9.7 ପଏଁଟର ଗଣିତ (Pointer Arithmetic)

C ଭାଷାରେ ପଏଁଟର ଗାଣିତିକ ପଦ୍ଧତି ଅତ୍ୟନ୍ତ ସୁସଙ୍ଗତ । ପଏଁଟର, ଆରେ, ଏବଂ ପଏଁଟର ଗଣିତର ସନ୍ନିବେଶ ହେଉଛି C ଭାଷାର ଅନ୍ୟତମ ଶକ୍ତି । ଏହି ବିଭାଗରେ, ଆମେ ପଏଁଟର ଉପରେ ବିଭିନ୍ନ ଅନୁମୋଦିତ ଅପରେସନଗୁଡ଼ିକୁ ସଂକ୍ଷେପରେ ବର୍ଣ୍ଣନା କରିବା ।

ନିମ୍ନଲିଖିତ ଅପରେସନଗୁଡ଼ିକ ପଏଁଟର ଉପରେ ଅନୁମୋଦିତ ।

1. ପଏଁଟର ଚଳରେ ଏକ ସଂଖ୍ୟା ଯୋଗ କରିବା ।

   ଧରାଯାଉ p ଏକ ଇଣ୍ଟିଜର (integer) ପ୍ରକାରର ଉପାଦାନକୁ ଇଙ୍ଗିତ କରୁଥିବା ଗୋଟିଏ ପଏଁଟର ଚଳ, ତା'ପରେ ବିବୃତ୍ତି

   ```
   p++;      କିମ୍ବା    ++p;
   ```

   p ର ମୂଲ୍ୟ 2 ବୃଦ୍ଧି କରେ, ଯାହାଦ୍ୱାରା ଏହା ଉପସ୍ଥିତ ଅବସ୍ଥାନର ପର ଅବସ୍ଥାନକୁ ଇଙ୍ଗିତ କରେ ଯାହା ଇଣ୍ଟିଜର ପ୍ରକାରର ଅନ୍ୟ ଏକ ମୂଲ୍ୟ ଧାରଣ କରୁଥିବ । ଏହି ବୃଦ୍ଧି କାରକ କ୍ୟାରେକ୍ଟର (character) ପାଇଁ 1 ହେବ, ଲଙ୍ଗ୍ ଇଣ୍ଟିଜର (long integer) ଏବଂ ଫ୍ଲୋଟ (float) ପାଇଁ 4 ଏବଂ ଡବଲ୍ (double) ପାଇଁ 8 ହେବ । ଏକ ବିବୃତ୍ତି

   ```
   p += i;
   ```

   ଯେଉଁଠାରେ i ଏକ ଧନାତ୍ମକ ମୂଲ୍ୟ ଥିବା ଇଣ୍ଟିଜର ସ୍ଥିରାଙ୍କ କିମ୍ବା ଇଣ୍ଟିଜର ଚଳ, p କୁ ଏପରି ବୃଦ୍ଧି କରେ ଯେପରି p ବର୍ତ୍ତମାନ ଇସାରା କରୁଥିବା ଅବସ୍ଥାନର i ତମ ପରବର୍ତ୍ତି ସ୍ଥାନକୁ ଇଙ୍ଗିତ କରିବ ।

2. ପଏଁଟର ଚଳରୁ ଏକ ସଂଖ୍ୟା ବିୟୋଗ କରିବା ।

   ଧରାଯାଉ p ଏକ ଇଣ୍ଟିଜର ପ୍ରକାର ଉପାଦାନକୁ ଇଙ୍ଗିତ କରୁଥିବା ଏକ ପଏଁଟର ଚଳ, ତା'ପରେ ବିବୃତ୍ତି

   ```
   p--;      କିମ୍ବା    --p;
   ```

   p ର ମୂଲ୍ୟରୁ 2 ହ୍ରାସ କରେ, ଯାହା ଦ୍ୱାରା ଏହା ବର୍ତ୍ତମାନ ଅବସ୍ଥାନର ପୂର୍ବ ଅବସ୍ଥାନକୁ ଇଙ୍ଗିତ କରେ । ଏକ ବିବୃତ୍ତି

   ```
   p -= i;
   ```

   ଯେଉଁଠାରେ i ଏକ ଧନାତ୍ମକ ମୂଲ୍ୟ ଥିବା ଇଣ୍ଟିଜର ସ୍ଥିରାଙ୍କ କିମ୍ବା ଇଣ୍ଟିଜର ଚଳ, p କୁ ଏପରି ହ୍ରାସ କରେ ଯେପରି ଏହା ବର୍ତ୍ତମାନ ଇସାରା କରୁଥିବା ଅବସ୍ଥାନର i ତମ ପୂର୍ବବର୍ତ୍ତୀ ସ୍ଥାନକୁ ଇଙ୍ଗିତ କରିବ ।

3. ଗୋଟିଏ ପଏଁଟର ଚଳରୁ ଅନ୍ୟ ଏକ ପଏଁଟର ଚଳର ବିୟୋଗ ।

   ସମାନ ତଥ୍ୟ ଇଙ୍ଗିତ କରୁଥିବା ଗୋଟିଏ ପଏଁଟର ଚଳ ଅନ୍ୟ ଏକ ପଏଁଟର ଚଳରୁ ବିୟୋଗ କରାଯାଇପାରିବ । ଦୁଇଟି ମଧ୍ୟରେ ଥିବା ପୃଥକ ଉପାଦାନଗୁଡ଼ିକୁ ଅଲଗା କରୁଥିବା ସଂଖ୍ୟାକୁ ବାଇଟ୍ ବୋଲି ସୂଚାଏ ।

   ଏହି ଅଙ୍କଗଣିତ କାର୍ଯ୍ୟ ସହିତ, ପଏଁଟର ଚଳଗୁଡ଼ିକୁ ପରସ୍ପର ସହିତ ତୁଳନା କରାଯାଇପାରିବ, ଯଦି ସେଗୁଡ଼ିକ ମେଳ ଯୋଗ୍ୟ, ଅର୍ଥାତ୍, ସମାନ ତଥ୍ୟ ଚଳକୁ ଇଙ୍ଗିତ କରୁଥାନ୍ତି ।

   ପଏଣ୍ଟର ଉପରେ ନିମ୍ନ ଅଙ୍କଗାଣିତିକ ପ୍ରକ୍ରିୟାଗୁଡ଼ିକ ଅନୁମୋଦିତ ନୁହେଁ ।

   - ଦୁଇଟି ପଏଁଟର ଚଳର ଯୋଗ ।

- ପଏଁଟର ଚଳକୁ ଏକ ସଂଖ୍ୟାଦ୍ୱାରା ଗୁଣନ ।
- ପଏଁଟର ଚଳକୁ ଏକ ସଂଖ୍ୟାଦ୍ୱାରା ବିଭାଜନ ।

9.8 ଫଙ୍କ୍ସନର ଚର ଭାବରେ ପଏଁଟର

ଅଧ୍ୟାୟ 6 ରେ, ଉଲ୍ଲେଖ ହେଇଛି ଯେ ଏକ ଫଙ୍କ୍ସନର ଚର ମୂଲ୍ୟ ଦ୍ୱାରା କିମ୍ବା ଠିକଣାଦ୍ୱାରା ଡାକିବା ପଦ୍ଧତି ବ୍ୟବହାର କରି ପାରିତ କରିପାରିବ । ମୂଲ୍ୟଦ୍ୱାରା ଡାକିବା ପଦ୍ଧତି ବ୍ୟବହାର କରାଯାଏ ତେବେ ଫଙ୍କ୍ସନର ସ୍ୱତନ୍ତ୍ର ଚରଗୁଡ଼ିକ ସାଧାରଣ ଚର ଭାବରେ ଘୋଷଣା କରାଯାଏ । ପ୍ରୋଗ୍ରାମ୍ ଚାଳନା ସମୟରେ ବାସ୍ତବ ଚରର ମୂଲ୍ୟଗୁଡ଼ିକ ସ୍ୱତନ୍ତ୍ର ଚରକୁ କପି କରାଯାଏ (ସ୍ୱତନ୍ତ୍ର ଚର ଫଙ୍କ୍ସନ ପାଇଁ ସ୍ଥାନୀୟ), ଏବଂ ଫଙ୍କ୍ସନର ଚାଳନା ଆରମ୍ଭ ହୁଏ ।

ରେଫରେନ୍ସଦ୍ୱାରା ଡାକିବା ପଦ୍ଧତି ବ୍ୟବହାର କରିବା ଦ୍ୱାରା ସ୍ୱତନ୍ତ୍ର ଚରକୁ ପଏଁଟର ଭାବରେ ଘୋଷଣା କରାଯାଏ । ପ୍ରୋଗ୍ରାମ୍ ଚାଳନା ସମୟରେ, ବାସ୍ତବ ଚରର ଠିକଣାଗୁଡ଼ିକ ସ୍ୱତନ୍ତ୍ର ଚରରେ କପି କରାଯାଏ ଏବଂ ଫଙ୍କ୍ସନର ଚାଳନା ଆରମ୍ଭ ହୁଏ ।

Listing 9.2

```
 /*
   Program to illustrate pointers as function arguments
*/

#include<stdio.h>

/* function prototype */
void swap( int *x, int *y )
int main()
{
   int a = 10, b = 20;
   /* function prototype */
   void interchange( int *, int *);
   printf( "\nValue of a = %d and b = %d", a, b );
   printf( " before function call\n"  );
   /* function call  */
   swap ( &a, &b);
   printf( "\nValue of a = %d and b = %d", a, b );
   printf( " after function call\n"  );
   return 0;
}
/* function definition */
void swap( int *x, int *y )
{
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}
```

Test Run

```
Value of a = 10 and b = 20 before function call
Value of a = 20 and b = 10 after function call
```

9.9 ପଏଁଟର ଏବଂ ଷ୍ଟ୍ରକ୍ଟର

ଆମେ ଅନ୍ୟ ତଥ୍ୟ ପ୍ରକାରପାଇଁ ପଏଁଟର ବ୍ୟବହାର କରିବା ହେତୁ ଆମେ ଷ୍ଟ୍ରକ୍ଟର ଚଳପାଇଁ ମଧ୍ୟ ପଏଁଟର ବ୍ୟବହାର କରିପାରିବା। ନିମ୍ନଲିଖିତ ଘୋଷଣାଗୁଡ଼ିକ ବିଚାର କର।

```
 typedef struct
 {
    int code;
    char name[20];
    int  dept_code;
    float  salary;
 } EMPLOYEE;

 EMPLOYEE emp, *ptr;
```

ଏହି ଘୋଷଣାଗୁଡ଼ିକ ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ତଥ୍ୟ ପ୍ରକାର ସୃଷ୍ଟି କରେ ଏବଂ ଏହାକୁ EMPLOYEE ନାମ ଦିଆଯାଇଛି। EMPLOYEE ପ୍ରକାରର *emp* ଏକ ଷ୍ଟ୍ରକ୍ଟର ଚଳ ଭାବରେ ଏବଂ *ptr* ଷ୍ଟ୍ରକ୍ଟରକୁ ଏକ ପଏଁଟର ଚଳ ଭାବରେ ଘୋଷଣା କରାଯାଇଛି।

ନିମ୍ନଲିଖିତ ଏକ ବିବୃତ୍ତି

```
ptr = &emp;
```

ପଏଁଟର ଚଳ ptr କୁ emp ଚଳର ଠିକଣା ନ୍ୟସ୍ତ କରିବାକୁ ବ୍ୟବହାର କରାଯାଇପାରିବ। ବର୍ତ୍ତମାନ ଆମେ ptr ପଏଁଟର ଚଳ ବ୍ୟବହାର କରି ଷ୍ଟ୍ରକ୍ଟର ଚଳ emp ର ସଦସ୍ୟଗୁଡ଼ିକୁ ପହଁଞ୍ଚି ପାରିବା।

ଷ୍ଟ୍ରକ୍ଟରର ଉପାଦାନଗୁଡ଼ିକ ତୀର ଅପରେଟର '–>' ବ୍ୟବହାର କରି ପହଁଞ୍ଚିହୁଏ ଯାହା ଦୁଇଟି ସାଂକେତିକ ଅକ୍ଷର '–' ଏବଂ '>' ଯୋଡ଼ିକରି ହୋଇଅଛି।

ତୀର ଅପରେଟର ବ୍ୟବହାର କରି ଷ୍ଟ୍ରକ୍ଟରର ସଦସ୍ୟଗୁଡ଼ିକୁ ପହଁଞ୍ଚିବା ପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି:

```
ptr->vname
```

ଯେଉଁଠାରେ ptr ହେଉଛି ଏକ ଷ୍ଟ୍ରକ୍ଟରକୁ ଇଙ୍ଗିତ କରୁଥିବା ପଏଁଟର ଚଳ, ଏବଂ vname ଷ୍ଟ୍ରକ୍ଟରର ଏକ ଉପାଦାନ।

ପଏଁଟର ଚଳ ptr ଦ୍ୱାରା ଇସାରା ହୋଇଥିବା କର୍ମଚାରୀ ପ୍ରକାରର ଷ୍ଟ୍ରକ୍ଟର ଉପାଦାନଗୁଡ଼ିକୁ ତୀର ଅପରେଟର ବ୍ୟବହାର କରି ପହଁଞ୍ଚି ହେବ।

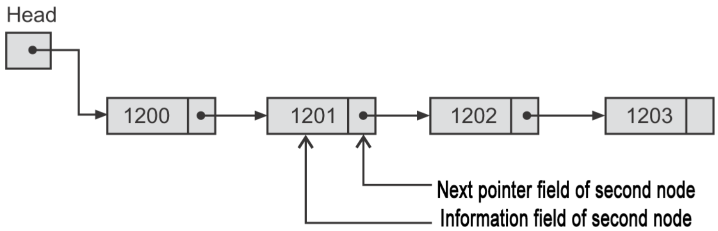
```
ptr->code ptr->name   ptr->dept_code      ptr->salary
```

9.9.1 ସ୍ୱ-ସୂଚିତ ଷ୍ଟ୍ରକ୍ଟର (Self-referential Structures)

ଗୋଟିଏ ସ୍ୱ-ସୂଚିତ ଷ୍ଟ୍ରକ୍ଟର ହେଉଛି ଏକ ଷ୍ଟ୍ରକ୍ଟର ଏହା ଅତି କମରେ ଗୋଟିଏ ନିଜସ୍ୱ ପ୍ରକାରର ଏକ ପଏଁଟରକୁ ସଦସ୍ୟ ଉପାଦାନ ଭାବେ ଅନ୍ତର୍ଭୁକ୍ତ କରେ। ଏହି ଷ୍ଟ୍ରକ୍ଟରଗୁଡ଼ିକ ଜଟିଳ ତଥ୍ୟ ସଂରଚନାପାଇଁ ବ୍ୟବହାର ହୋଇଥାଏ, ଯେପରିକି ଲିଙ୍କ୍‌ଡ ଲିଷ୍ଟ *(linked*

lists), ଟ୍ରୀ *(trees)* ଏବଂ ଗ୍ରାଫ୍ *(graphs)*।

ସ୍ୱ-ସୂଚିତ ଷ୍ଟ୍ରକ୍ଚରର ଏହିପରି ଏକ ପ୍ରୟୋଗ ନିମ୍ନରେ ଦିଆଯାଇଅଛି



ଚିତ୍ର . 9.4: 4 ଟି ନୋଡ୍ ସହିତ ଇଣ୍ଟିଜର୍ ମୂଲ୍ୟର Linear linked list

ମେମୋରିରେ ଉପରୋକ୍ତ ଲିନିୟର ଲିଙ୍କ୍ଡ ଲିଷ୍ଟ *(Linear linked list)* ଉପସ୍ଥାପିତ କରିବା ପାଇଁ ଆମକୁ ନିମ୍ନଲିଖିତ ଘୋଷଣାଗୁଡ଼ିକ ଆବଶ୍ୟକ

```
struct NODE
{
   int info;
   struct NODE *next;
};
struct NODE *head;
```

9.10 ଗତିଶୀଳ ରୂପେ ମେମୋରି ଆବଣ୍ଟନ (DYNAMIC MEMORY ALLOCATION)

ନିର୍ଦ୍ଦେଶାବଳିପାଇଁ ମେମୋରି ଆବଶ୍ୟକତା ସବୁବେଳେ ସ୍ଥିର। କିନ୍ତୁ କିଛି ପରିସ୍ଥିତିରେ ତଥ୍ୟପାଇଁ ଠିକ୍ ମେମୋରି ଆବଶ୍ୟକତା ଆଗୁଆ ଜଣାପଡ଼ି ନ ପାରେ। ତଥ୍ୟ ଆବଶ୍ୟକତା ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ ଚାଳନାଠାରୁ ଅନ୍ୟ ଏକ ପ୍ରୋଗ୍ରାମ ଚାଳନାରେ ଭିନ୍ନ ହୋଇପାରେ, ଅର୍ଥାତ୍, ତଥ୍ୟପାଇଁ ମେମୋରି ଆବଶ୍ୟକତା ଗତିଶୀଳ ହୋଇଥାଏ।

ଏହିପରି କ୍ଷେତ୍ରରେ ଯେତେବେଳେ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ତଥ୍ୟ ଉପାଦାନପାଇଁ ଆମକୁ ମେମୋରି ପରିମାଣ ଜଣାପଡ଼େ ନାହିଁ, ସେତେବେଳେ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନା ସମୟରେ ମେମୋରି ଆବଣ୍ଟନ କରିବା ସର୍ବଦା ଭଲ, ଅର୍ଥାତ୍, ଯେତେବେଳେ ପ୍ରୋଗ୍ରାମ୍ ଚାଲୁଛି। ପ୍ରୋଗ୍ରାମ୍ ଚାଳନା ସମୟରେ ମେମୋରି ଆବଣ୍ଟନକୁ ଗତିଶୀଳ ରୂପେ ମେମୋରି ଆବଣ୍ଟନ କୁହାଯାଏ।

ପ୍ରତ୍ୟେକ ପ୍ରୋଗ୍ରାମ୍‌କୁ ଅଣଆବଣ୍ଟିତ ମେମୋରିର ଏକ ପୁଲ୍ *(pool)* ପ୍ରଦାନ କରାଯାଏ ଯାହା ଏହାର ପ୍ରୋଗ୍ରାମ୍ ଚାଳନା ସମୟରେ ବ୍ୟବହାର ହୋଇପାରିବ। ଅଣଆବଣ୍ଟିତ ମେମୋରିର ଏହି ପୁଲ୍ ଫ୍ରି ଷ୍ଟୋର *(free store)* ଭାବରେ ଜଣା। ତେଣୁ, ପ୍ରୋଗ୍ରାମ୍ ଦ୍ୱାରା ଯେତେବେଳେ ମେମୋରି ପ୍ରୟୋଜନ ହୁଏ, ସେତେବେଳ ଆବଶ୍ୟକୀୟ ପରିମାଣର ମେମୋରି ଫ୍ରି ଷ୍ଟୋରରୁ ନିଆଯାଏ। ଯେତେବେଳେ ପୂର୍ବଆବଣ୍ଟିତ ମେମୋରି ଆଉ ଆବଶ୍ୟକ ହୁଏ ନାହିଁ, ଏହା ଫ୍ରି ଷ୍ଟୋରକୁ ଫେରାଇ ଦିଆଯାଏ।

C ଭାଷା, ମେମୋରି ଆବଣ୍ଟନ *(allocate)* ଓ ଫେରସ୍ତପାଇଁ *(de-allocate)* ଗତିଶୀଳ ମେମୋରି ପରିଚାଳନା ଫଙ୍କସନ *(dynamic memory management functions)* ନାମକ ଲାଇବ୍ରେରି ଫଙ୍କସନ ପ୍ରଦାନ କରେ।

ଟେବୁଲ୍ 9.1: ମେମୋରି ପରିଚାଳନା ଫଙ୍କସନଗୁଡ଼ିକ

Function Name	Description
Malloc	Allocates memory from free store.
Free	Deallocates a previously allocated.

ଆବଣ୍ଟନର ଅର୍ଥ ଏହିକିଯେ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନା ସମୟରେ ମଧ୍ୟ ତୁମେ ପ୍ରୋଗ୍ରାମ୍ ଦ୍ୱାରା ଆବଶ୍ୟକୀୟ ମେମୋରି ପ୍ରାପ୍ତ କରିପାରିବ।

ଡ଼ି-ଆବଣ୍ଟନର(de-allocation) ଅର୍ଥ ହେଉଛି ଯେ ପ୍ରୋଗ୍ରାମ୍ ଚାଳନା ସମୟରେ ଗତିଶୀଳ ଭାବରେ ଅଧିଗ୍ରହଣ କରାଯାଇଥିବା ମେମୋରି ମୁକ୍ତ କରିବା।

ଏହା ଧ୍ୟାନ ଦେବା ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଯେ ଗତିଶୀଳ ଭାବରେ ଆବଣ୍ଟିତ ହୋଇଥିବା ମେମୋରିକୁ, ପ୍ରୋଗ୍ରାମ୍ ଚାଳନା ସମାପ୍ତ ହେବା ପୂର୍ବରୁ ଡ଼ି-ଆବଣ୍ଟିତ/ମୁକ୍ତ କରାଯିବା ଆବଶ୍ୟକ। ଅନ୍ୟଥା, ତୁମ ପ୍ରୋଗ୍ରାମ୍ ସମାପ୍ତ ହେବା ପରେ ମଧ୍ୟ ଗତିଶୀଳ ଭାବରେ ଆବଣ୍ଟିତ ମେମୋରି ସ୍ୱତଃସ୍ଫୁତ ଭାବେ କେବେ ବି ମୁକ୍ତ ହୁଏ ନାହିଁ, ଫଳରେ ଅପରେଟିଂ ସିଷ୍ଟମ୍ ଦୃଷ୍ଟିକୋଣରୁ ମେମୋରି ପ୍ରୋଗ୍ରାମ୍ ସରିଲା ପରେ ମଧ୍ୟ ବ୍ୟବହୃତ ହେଉଛି ବୋଲି ଧରାଯିବ।

ତେଣୁ, ଯଦି ତୁମେ ଏକା ପ୍ରୋଗ୍ରାମ୍‌କୁ ଏକାଧିକ ଥର ଚଳାଉଛ, ଅନେକ ବ୍ୟବହାରକାରୀ ତୁମ ପ୍ରୋଗ୍ରାମ୍ଗୁଡ଼ିକୁ ଏକାସାଙ୍ଗରେ ବ୍ୟବହାର କରୁଥାନ୍ତି, ତାହେଲେ ଅପରେଟିଂ ସିଷ୍ଟମ୍‌ରୁ ମେମୋରି ଶେଷ ହୋଇଯାଇପାରେ।

9.10.1 ମେମୋରି ଆବଣ୍ଟନ

malloc() ଫଙ୍କସନ ଗୋଟିଏ ଚର ନେଇଥାଏ ଯିଏ ମେମୋରି ବ୍ଲକର ଆକାରକୁ ବାଇଟରେ ଉଲ୍ଲେଖ କରେ। ଏହି ଫଙ୍କସନ ଆବଶ୍ୟକ ପରିମାଣର ମେମୋରିକୁ ଗଦାରୁ(heap) ଆବଣ୍ଟନ କରେ, ଆବଣ୍ଟନ ସଫଳ ହେଲେ ଫଙ୍କସନଟି ଆବଣ୍ଟିତ ମେମୋରିର ପ୍ରାରମ୍ଭାବସ୍ଥାନକୁ ଇସାରା କରୁଥିବା ଏକ ପଏଁଟର କିମ୍ବା ବିଫଳ ହେଲେ ଏକ NULL ପଏଁଟର (0) ଫେରସ୍ତ କରେ। ବିଫଳତା କ୍ଷେତ୍ରରେ, ପ୍ରୋଗ୍ରାମ୍‌ଟି ସୁରକ୍ଷିତ ଭାବରେ ପ୍ରସ୍ଥାନ କରିବା ଉଚିତ।

ଫେରସ୍ତ ହୋଇଥିବା ପଏଁଟରଟି ଭଏଡ୍ (void) ପ୍ରକାରର ଅଟେ, ତାହାକୁ ବାଞ୍ଛିତ ପ୍ରକାରର ଏକ ପଏଁଟରରେ ବଦଳାଇବା ଆବଶ୍ୟକ। ଆବଣ୍ଟିତ ମେମୋରି ବ୍ଲକଟି ଅଣ-ପ୍ରାରମ୍ଭିକୃତ(un-initialized) ହୋଇ ରହିଥାଏ, ଅର୍ଥାତ୍ ଏହାର ସମସ୍ତ ଅବସ୍ଥାନରେ ଅଦରକାରି ମୂଲ୍ୟ ଥାଏ।

malloc() ଫଙ୍କସନର ବ୍ୟବହାର ପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
datatype *ptr
ptr = (datatype *) malloc( n * sizeof (datatype) );
```

ଯେଉଁଠାରେ *n* ହେଉଛି ଉପାଦାନଗୁଡ଼ିକର ସଂଖ୍ୟା।

ଉଦାହରଣ ବଶତଃ,

```
int *ptr
 ptr = (int *) malloc( 10 * sizeof(int) );
 if ( ptr == NULL )
 {
    printf("\nMemory allocation failed\n");
    return 1;
 }
```

16-ବିଟ୍ C ସଂକଳକ ପାଇଁ *int* ର ଆକାର 2-ବାଇଟ୍ ଏବଂ 32-ବିଟ୍ C ସଂକଳକ ପାଇଁ *int* ର ଆକାର 4-ବାଇଟ୍, ତେଣୁ, ଏହି ବିବୃତ୍ତି ମେମୋରିର 20/40 ବାଇଟ୍ ବ୍ଲକ୍ ଆବଣ୍ଟନ କରିବ । ଏବଂ, ପଏଁଟର *ptr* ଆବଣ୍ଟିତ ମେମୋରିରେ ପ୍ରଥମ ବାଇଟ୍‌ର ଠିକଣା ଧାରଣ କରେ ।

9.10.2 ମେମୋରି ଡି-ଆବଣ୍ଟନ/ମୁକ୍ତ କରିବା

free() ଫଙ୍କସନ, ମେମୋରିର ଏକ ଗତିଶୀଳ ରୂପେ ଆବଣ୍ଟିତ ମେମୋରି ବ୍ଲକ୍‌କୁ ମୁକ୍ତ କରେ । free() ଫଙ୍କସନର ବ୍ୟବହାର ପାଇଁ ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
free(ptr);
```

ଆବଣ୍ଟିତ ବ୍ଲକକୁ ଇସାରା ଗୋଟିଏରେ ପଏଁଟରକୁ ଉଲ୍ଲେଖ କରେ । ଏହା ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ କେବଳ ଆବଣ୍ଟିତ ବ୍ଲକ୍‌ଟି ମୁକ୍ତ ହୁଏ, ପଏଁଟର ଚଳ ବିଲୋପ ହୁଏନାହିଁ ।

Listing 9.3

```
/*
   Program to illustrate the allocation of memory dynamically
   for one-dimensional array at execution time.

   Program takes an array of arbitrary size as input, find the
   Largest element of the array
*/

#include<stdio.h>
#include<stdlib.h>

int main()
{
   int  *a, n, i, max;
   printf( "\nEnter size of 1D array : "  );
   scanf( "%d", &n );

   /* dynamically allocate memory for array */
   a = (int *) malloc(n*sizeof(int));
   if ( a == NULL )
   {
       printf("\nMemory allocation failed.\n");
       return 1;
   }

   printf( "\nEnter %d elements of array\n", n );
   for ( i = 0 ; i < n ; i++ )
      scanf( "%d", &a[i] );

   max = a[0];
   for ( i = 1; i < n; i++ )
   {
      if ( a[i] > max )
          max = a[i];
   }
   printf( "\nLargest element of array = %d\n", max );
   /* de-allocate memory */
   free(a);
   return 0;
}
```

Test Run

```
Enter size of 1D array : 10
Enter 10 elements of array
10 35 12 9 45 11 50 20 44 32
Largest element of array = 5
```

ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 9.1: ପଏଁଟର ବ୍ୟବହାର କରି ଆରେର ଉପାଦାନଗୁଡ଼ିକ ପ୍ରିଣ୍ଟ କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

Listing 9.4

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int  a[5] = { 1, 2, 3, 4, 5 };
   int i, *ptr;
   ptr = a;
   printf("\nElements of array\n\n");
   for ( i = 0; i < 5; i++ )
       printf("%d  ", *(ptr+i));
   printf("\n\n");
   return 0;
}
```

Test Run

```
Elements of array
1  2  3  4  5
```

ଉଦାହରଣ 9.2: ପଏଁଟର ବ୍ୟବହାର କରି ଏକ ଫଙ୍କ୍ସନକୁ *1D* ଆରେ ପ୍ରେରଣପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

Listing 9.5

```
#include<stdio.h>
#include<stdlib.h>
void fun(int *, int);  /* function prototype  */
int main()
{
   int  a[10] = { 5, 2, 1, 7, 9, 10, 4, 6, 8, 3 };
   /*  call to function  */
   fun(a,10);
   return 0;
}
```

```
/* function definition  */
 void fun(int *p, int n)
 {
    int i;
    printf("\nElements of array\n\n”);
    for ( i = 0; i < n; i++ )
        printf("%d  ", p[i]);
    printf("\n\n");
 }
```

Test Run

```
Elements of array
5 2 1 7 9 10 4 6 8 3
```

ୟୁନିଟର ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଯାହା ଶିଖିଛୁ

- ପଏଁଟର ହେଉଛି ଏକ ଚଳ ଯାହା ଏକ ଅପେରାଣ୍ଡର (operand) ଠିକଣା ଧାରଣ କରେ ।
- ପଏଁଟର ଅନେକ ସୁବିଧା ପ୍ରଦାନ କରେ ଯେପରିକି ଏକ ଫଙ୍କ୍ସନରୁ ଏକାଧିକ ମୂଲ୍ୟ ଫେରସ୍ତ କରିବା, ଜଟିଳ ତଥ୍ୟ ଜଡ଼ିତ ଷ୍ଟ୍ରକ୍ଚରର ନିର୍ମାଣ କରିବା, ହାର୍ଡ଼ୱେୟାର ସହିତ ଯୋଗାଯୋଗ କରିବା ଇତ୍ୟାଦି ।
- ପଏଁଟରରେ ଯୋଗ ଅପରେସନ ଅନୁମୋଦିତ । ପ୍ରତିବନ୍ଧିତ ଅର୍ଥରେ ଏକ ବିୟୋଗ ଅପରେସନ ଅନୁମୋଦିତ ଯାହାକି ଏହା ଧନାତ୍ମକ ପାର୍ଥକ୍ୟ ଦେବା ଆବଶ୍ୟକ । ଗୁଣନ ଏବଂ ବିଭାଜନ ଅପରେସନ ଅନୁମୋଦିତ ନୁହେଁ ।
- ଠିକଣା ମାଧ୍ୟମରେ ଏକ ଫଙ୍କ୍ସନକୁ ଚର ପ୍ରେରଣପାଇଁ ପଏଁଟର ବ୍ୟବହାର କରାଯାଇଥାଏ ଯାହା ରେଫରେନ୍ସ ଦ୍ୱାରା କଲ୍ ପଦ୍ଧତି ଭାବରେ ଜଣା ଅଟେ ।
- ଏକ ସ୍ୱ-ସୂଚିତ ଷ୍ଟ୍ରକ୍ଚର ହେଉଛି ଏକ ଷ୍ଟ୍ରକ୍ଚର ଯାହା ଅତି କମ୍‌ରେ ନିଜ ପାଇଁ ଗୋଟିଏ ଉପାଦାନ ଅନ୍ତର୍ଭୁକ୍ତ କରେ ।
- ମେମୋରି ଆବଣ୍ଟନ ଦୁଇଟି ଉପାୟରେ କରାଯାଇପାରିବ - ନିଶ୍ଚଳ ଭାବରେ ଏବଂ ଗତିଶୀଳ ଭାବରେ ।
- ସଂକଳନ ସମୟରେ କରାଯାଇଥିବା ମେମୋରି ଆବଣ୍ଟନକୁ ନିଶ୍ଚଳ ମେମୋରି ଆବଣ୍ଟନ କୁହାଯାଏ ।
- ପ୍ରୋଗ୍ରାମ ଚାଳନ ସମୟରେ କରାଯାଇଥିବା ମେମୋରି ଆବଣ୍ଟନକୁ ଗତିଶୀଳ ମେମୋରି ଆବଣ୍ଟନ କୁହାଯାଏ ।
- ପ୍ରତ୍ୟେକ ପ୍ରୋଗ୍ରାମ୍‌କୁ ଆବଣ୍ଟନଯୋଗ୍ୟ ମେମୋରିର ଏକ ପୁଲ୍ ପ୍ରଦାନ କରାଯାଇଥାଏ ଯାହାକି ଏହାକୁ ପ୍ରୋଗ୍ରାମ ଚାଳନ ସମୟରେ ବ୍ୟବହାର କରିପାରିବ । ଆବଣ୍ଟିତ ମେମୋରିର ଏହି ପୁଲକୁ ହିପ୍ କିମ୍ବା free store କୁହାଯାଏ ।
- ପ୍ରୋଗ୍ରାମ ଚାଳନ ସମୟରେ ମେମୋରି ପରିଚାଳନା କରିବାର ପ୍ରକ୍ରିୟାଟି ଗତିଶୀଳ ମେମୋରି ପରିଚାଳନା ଭାବରେ ଜଣା ।
- malloc() ଫଙ୍କ୍ସନ ଗତିଶୀଳ ଭାବରେ ମେମୋରି ଆବଣ୍ଟନ କରିବାକୁ ବ୍ୟବହୃତ ହୁଏ ।
- ପୂର୍ବରୁ malloc() ଫଙ୍କ୍ସନ ଦ୍ୱାରା ଆବଣ୍ଟିତ ମେମୋରିକୁ ମୁକ୍ତ କରିବାକୁ free() ଫଙ୍କ୍ସନ ବ୍ୟବହୃତ ହୁଏ ।

- ପ୍ରାଥମିକ ମୂଲ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ହୋଇ ନ ଥିବା ପଏଁଟରକୁ ୱାଇଲ୍ଡ(wild) ପଏଁଟର କୁହାଯାଏ, ଯେହେତୁ ଏହା ମେମୋରିର ଯେକୌଣସି ସ୍ଥାନକୁ ଇଙ୍ଗିତ କରିପାରେ ।
- ମୁକ୍ତ ହୋଇଯାଇଥିବା ମେମୋରି ଅବସ୍ଥାନକୁ ଇସାରା କରୁଥିବା ପଏଁଟରକୁ ଡ୍ୟାଙ୍ଗେଲିଙ୍ଗ (dangling) ପଏଁଟର କୁହାଯାଏ ।
- ଯେତେବେଳେ ବି ଠିକଣା ଶୂନରେ/ନଲ୍ ପଏଁଟରରେ (Null pointer) ଲେଖିବାକୁ ଚେଷ୍ଟା କରାଯାଏ, ପ୍ରୋଗ୍ରାମ ସମାପ୍ତ ହେବା ପରେ ସିଷ୍ଟମ୍ "Null pointer assignment" ନାମକ ଏକ ବାର୍ତ୍ତା ଦେଖାଇଥାଏ ।
- ମେମୋରି ଲିକ୍ (Memory leak) ହେଉଛି ଏକ ପ୍ରକାର ପରିସ୍ଥିତି ଯେତେବେଳେ ଏକ ଡକା ହୋଇଥିବା ଫଙ୍କ୍ସନ ମଧ୍ୟରେ ମେମୋରି ଆବଣ୍ଟନ ହୋଇଥାଏ କିନ୍ତୁ ଏହାର ବ୍ୟବହାର ସରିବା ପରେ ସେହି ଫଙ୍କ୍ସନରେ ମୁକ୍ତ କରାହୋଇ ନ ଥାଏ ଫଳସ୍ୱରୂପ, ଯେତେବେଳେ ଡକା ହୋଇଥିବା ଫଙ୍କ୍ସନଟିର ଚାଳନା ସମାପ୍ତ ହୁଏ, ପଏଁଟର ଚଳଟି ମଧ୍ୟ ବିଲୋପ ହୋଇଯାଏ କିନ୍ତୁ ସେହି ମୁକ୍ତ ହୋଇ ନ ଥିବା ସେହି ଆବଣ୍ଟିତ ମେମୋରି ବ୍ଲକରେ ପହଞ୍ଚିବାର ଆଉ କୌଣସି ମାଧ୍ୟମ ନଥାଏ ।
- ଆବଣ୍ଟନ ବିଫଳତା ହେଉଛି ଏକ ପରିସ୍ଥିତି ଯେତେବେଳେ ସିଷ୍ଟମ୍ଟି ଅବ୍ୟବହୃତ ମେମୋରିର ଏକ ଆବଶ୍ୟକ ବ୍ଲକ୍ ଖୋଜିପାଇବାରେ ସକ୍ଷମ ହୁଏ ନାହିଁ । ଏକ ନଲ୍ (NULL) ପଏଁଟର ଫେରସ୍ତ କରି ସିଷ୍ଟମ୍ ଏହି ପରିସ୍ଥିତିକୁ ଇଙ୍ଗିତ କରିଥାଏ ।

ଅନୁଶୀଳନୀ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. ଆମେ କିପରି ଏକ ଚଳର ଠିକଣା ପାଇପାରିବା ?
2. ଏକ ପଏଁଟର ଚଳ କ'ଣ ?
3. ଏକ ପଏଁଟର ଚଳ କିପରି ଘୋଷିତ ହୋଇଥାଏ ?
4. ଏକ ପଏଁଟର ଚଳଦ୍ୱାରା ଇଙ୍ଗିତ ମୂଲ୍ୟଗୁଡ଼ିକ କିପରି ପାଇପାରିବ ?
5. ବିଭିନ୍ନ ପ୍ରକାରର ପଏଁଟରଗୁଡ଼ିକ କ'ଣ ?
6. ପଏଁଟର ଉପରେ ଅନୁମୋଦିତ ଅପରେସନଗୁଡ଼ିକ କ'ଣ ?
7. ପଏଁଟର ଉପରେ ଅନୁମୋଦିତ ନ ଥିବା ଅପରେସନଗୁଡ଼ିକ କ'ଣ ?

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. _____ ଅପରେଟର ବ୍ୟବହାର କରି ଏକ ଚଳର ଠିକଣା ପ୍ରାପ୍ତ କରାଯାଇପାରିବ ।

   (a) * (b) & (c) ? (d) ->
2. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଅପରେଟର ଇନ୍ଡାଇରେକ୍ସନ(indirection) ବା ଡି-ରେଫରେନ୍ସ (dereference) ଅପରେଟର ଭାବରେ ଜଣା ?

   (a) & (b) << (c) ^ (d) *
3. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଅପରେଟର ଠିକଣା ଅପରେଟର (address of operator) ଭାବରେ ଜଣା ?

   (a) & (b) * (c) -> (d) **
4. ପଏଁଟର ଚଳ **ptr1, ptr2** ର ସଠିକ୍ ଘୋଷଣାନାମାଟି ଚିହ୍ନଟ କର

   (a) int ptr1, *ptr2; (b) int *ptr1, ptr2;

   (c) int ptr1, ptr2; (d) int *ptr1, *ptr2;

5. କେବଳ ପଏଁଟର ସହିତ ବ୍ୟବହୃତ ଅପରେଟର୍ ଗୁଡ଼ିକ ହେଉଛି

(a) * ଏବଂ / (b) & ଏବଂ * (c) & ଏବଂ ^ (d) * ଏବଂ +

6. ଠିକଣା ଅପରେଟର (address of operator) ଅପେରାଣ୍ଡ ଏକ _____________ ହୋଇପାରେ

(a) ସ୍ଥିରାଙ୍କ (constant) (b) ଆରେ ଉପାଦାନ (array element)

(c) ଅଭିବ୍ୟକ୍ତି (expression) (d) କିଛି ନୁହେଁ (None)

7. ପଏଁଟର ଚଳ ଗୁଡ଼ିକ _____________ କୁ ନ୍ୟସ୍ତ ହୋଇପାରେ

(a) ହେକ୍ସାଡେସିମାଲ୍ ନୋଟେସନରେ ଠିକଣାର ମୂଲ୍ୟ ଉପସ୍ଥାପିତ କରେ,

(b) ଅକ୍ଟାଲ୍ ନୋଟେସନରେ ଠିକଣାର ମୂଲ୍ୟ ଉପସ୍ଥାପିତ କରେ,

(c) ଅନ୍ୟ ଏକ ଚଳର ଠିକଣା,

(d) ବାଇନାରୀ ନୋଟେସନରେ ଠିକଣାର ମୂଲ୍ୟ ଉପସ୍ଥାପିତ କରେ,

8. ଇନ୍‌ଡାଇରେକ୍ସନ(indirection) ବା ଡି-ରେଫରେନ୍ସ (dereference) ଅପରେଟରର ଅପେରାଣ୍ଡ ହେଉଛି

(a) ପଏଁଟର ଚଳ (b) ସାଧାରଣ ଚଳ

(c) ଇଣ୍ଟିଜର ସ୍ଥିରାଙ୍କ (d) ଉପରୋକ୍ତ କୌଣସିଟି ନୁହେଁ

9. ଅପରେଟର୍ ଚିହ୍ନଟ କର ପଏଁଟର ସହ ବ୍ୟବହୃତ ହେଉନଥିବା ଅପରେଟର୍ ଚିହ୍ନଟ କର

(a) -> (b) & (c) * (d) >>

10. ଠିକଣା ଅପରେଟର୍ (&) ଏକ ଚଳରେ ଉପସର୍ଗ ହେଲେ, ଏହା କ'ଣ ଦିଏ ?

(a) ଚଳର ମୂଲ୍ୟ (b) ଚଳର ପ୍ରକାର

(c) ଚଳର ଠିକଣା (d) ଉପରୋକ୍ତ କୌଣସିଟି ନୁହେଁ

11. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଅପରେସନ୍ ପଏଁଟରରେ ଅନୁମୋଦିତ ନୁହେଁ?

(a) ଏକ ପଏଁଟର ଚଳ ମୂଲ୍ୟକୁ ବୃଦ୍ଧି କରିବା (b) ପଏଁଟର ଚଳରେ ଏକ ସଂଖ୍ୟା ଯୋଗ କରିବା

(c) ଏକ ସଂଖ୍ୟା ଦ୍ୱାରା ପଏଁଟର ଚଳର ବିଭାଜନ (d) ଦୁଇଟି ପଏଁଟର ଚଳର ପାର୍ଥକ୍ୟ

12. ନିମ୍ନଲିଖିତ କୋଡ଼ କାର୍ଯ୍ୟକାରୀ କରିବା ପରେ x ର ମୂଲ୍ୟ କ'ଣ ?

```
int x = 5, *p;
p = &x;
*p = 7;
```

(a) 5 (b) 7 (c) ଅପରିଭାଷିତ (d) କିଛି ନୁହେଁ

13. ନିମ୍ନଲିଖିତ କୋଡ଼ର ଫଳାଫଳ କ'ଣ ହେବ?

```
int a[5] = {1,2,3,4};
int *p = a;
printf("%d",*(p+2));
```

(a) 4 (b) 0 (c) 3 (d) 2

14. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ କୋଡ଼କୁ ବିଚାର କର:

```
int *p, a[] = { 1, 2, 3, 4, 5 };
p = &a[2];
printf("%d", p[-1]);
```

ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ୍ କୋଡ଼ ବିଷୟରେ ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଠିକ୍?

(a) ସଂକଳନ ସମୟରେ ତ୍ରୁଟି କାରଣ ଏଠାରେ p ଏକ ଆରେ ନୁହେଁ

(b) ଆଉଟପୁଟ୍ 2 ହେବ

(c) ସଂକଳନ ସମୟରେ ତ୍ରୁଟି କାରଣ ଆରେ ଇଣ୍ଡେକ୍ସ ଋଣାତ୍ମକ ହୋଇପାରିବ ନାହିଁ

(d) ଉପରୋକ୍ତ କୌଣସିଟି ନୁହେଁ

15. ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ୍ କୋଡ଼ ବିଚାର କର:

```
const int i = 5;
int *p;
i = 20;
p = &i;
printf("\n%d, %d\n", *p, i);
```

ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ୍ କୋଡ଼ ବିଷୟରେ ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଠିକ୍?

(a) ଆଉଟପୁଟ୍ 20, 20 ହେବ

(b) ଆଉଟପୁଟ୍ 5, 20 ହେବ

(c) ଆଉଟପୁଟ୍ 20, 5 ହେବ

(d) ସଂକଳନ ସମୟରେ ତ୍ରୁଟି କାରଣ ଏଠାରେ ଚଳ i ର ମୂଲ୍ୟ ପରିବର୍ତ୍ତନ କରାଯାଇପାରିବ ନାହିଁ।

ଉତ୍ତର									
1.	*(b)*	**2.**	*(d)*	**3.**	*(a)*	**4.**	*(d)*	**5.**	*(b)*
6.	*(b)*	**7.**	*(c)*	**8.**	*(a)*	**9.**	*(d)*	**10.**	*(c)*
11.	*(c)*	**12.**	*(b)*	**13.**	*(c)*	**14.**	*(b)*	**15.**	*(d)*

ପ୍ରୋଗ୍ରାମିଂ (Programming) ସମସ୍ୟା

1. ପଏଁଟର ବ୍ୟବହାର କରି ଗୋଟିଏ ଆରେକୁ ଅନ୍ୟ ଆରେରେ କପି କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
2. ପଏଁଟର ବ୍ୟବହାର କରି ଏକ ଆରେର କ୍ରମକୁ ଓଲଟା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
3. ପଏଁଟର ବ୍ୟବହାର କରି ଷ୍ଟ୍ରିଙ୍ଗର ଦୈର୍ଘ୍ୟ ବାହାର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
4. ପଏଁଟର ବ୍ୟବହାର କରି ଏକ ଷ୍ଟ୍ରିଙ୍ଗର ଅକ୍ଷରଗୁଡ଼ିକୁ ଓଲଟା କ୍ରମରେ ରଖିବା ପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
5. ପଏଁଟର ବ୍ୟବହାର କରି ଦୁଇଟି ଷ୍ଟ୍ରିଙ୍ଗର ତୁଳନା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ।
6. ପଏଁଟର ବ୍ୟବହାର କରି ଏକ ଫଙ୍କସନ ଲେଖ ଯାହାକି ଦିଆଯାଇଥିବା ଆରେର କ୍ଷୁଦ୍ରତମ ଉପାଦାନ, ବୃହତ୍ତମ ଉପାଦାନ, ଏବଂ ହାରାହାରି ଉପାଦାନ ଫେରସ୍ତ କରେ। ଏହି ଫଙ୍କସନର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କର।

7. ପଏଁଟର ବ୍ୟବହାର କରି ଏକ ଫଙ୍କ୍ସେନ ଲେଖ ଯାହାକି ଏକ ଫ୍ଲୋଟିଂ ପଏଁଟ୍ (floating point) ସଂଖ୍ୟା ଗ୍ରହଣ କରିପାରୁଥିବା ପୂର୍ଣାଂଶ ଭାଗ ଏବଂ ଭଗ୍ନାଂଶ ଭାଗ ଫେରସ୍ତ କରେ । ଏହି ଫଙ୍କ୍ସନର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କର ।
8. ପଏଁଟର ବ୍ୟବହାର କରି ଏକ ଆରେରେ ନକଲ ଉପାଦାନ ଅଛି କି ନାହିଁ ତାହା ବାହାର କରିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
9. ପଏଁଟର ବ୍ୟବହାର କରି ଏକ ଆରେରୁ ପୁନରାବୃତ୍ତ ଉପାଦାନଗୁଡ଼ିକ ଅପସାରଣ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
10. ଏକ ପଏଁଟର ବ୍ୟବହାର କରି ଏକ ଷ୍ଟ୍ରକ୍ଚରର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।
11. ଏକ ଫଙ୍କ୍ସନ replace() ଲେଖ ଯାହାର ପ୍ରୋଟୋଟାଇପ୍ ନିମ୍ନରେ ଦିଆଯାଇଛି

```
int replace(char *str, char c1, char c2);
```

ଏହା କ୍ୟାରେକ୍ଟର(character) c1 ର ପ୍ରତ୍ୟେକ ଉପସ୍ଥିତିକୁ କ୍ୟାରେକ୍ଟର c2 ସହିତ ବଦଳାଇଥାଏ, ଏବଂ ହୋଇଥିବା ମୋଟ ବଦଳ ସଂଖ୍ୟା ଫେରସ୍ତ କରେ । ଉପରୋକ୍ତ ଫଙ୍କ୍ସନ ପରୀକ୍ଷା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

12. ନିମ୍ନଲିଖିତ ଷ୍ଟ୍ରକ୍ଚରର ସଂଜ୍ଞା ବିଚାର କର:

```
struct BOX
{
   char make[25];
   float length, breadth, height;
};
```

ଏକ ଫଙ୍କ୍ସନ ଲେଖ ଯାହା BOX ଷ୍ଟ୍ରକ୍ଚରର ଠିକଣା ଗ୍ରହଣ କରିଥାଏ ଏବଂ ଏହାର ଘନଫଳ ଫେରସ୍ତ କରେ ।

13. ନିମ୍ନଲିଖିତ ଘୋଷଣାନାମାକୁ ବିଚାର କର:

```
struct EMPLOYEE
{
   int code;
   char name[31];
   float salary;
};
```

ଏକ ସଂଗଠନରେ କର୍ମଚାରୀ ସଂଖ୍ୟା $n(\leq 50)$ ବୋଲି ଦିଆଯାଇଅଛି । ପଏଁଟର ସଙ୍କେତନ ବ୍ୟବହାର କରି ଜଣେ କର୍ମଚାରୀଙ୍କ ତଥ୍ୟ ପାଇବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ଯାହା ସର୍ବାଧିକ ଦରମା ପାଇ ଥିବା କର୍ମଚାରୀଙ୍କ ବିବରଣୀ ପ୍ରଦାନ କରିବ ।

14. ନିମ୍ନଲିଖିତ ଘୋଷଣାନାମା ବିଚାର କର:

```
char *names[]={ "Vimal","Amit","Anuj","Rohit","Abhijit" };
```

ପଏଁଟର ବ୍ୟବହାର କରି ନିମ୍ନଲିଖିତ ଆଉଟପୁଟ୍ ଦେବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

```
char *names[]={ "tijihbA","tihoR","junA","timA","lamiV" };
```

15. ଏକ ଫଙ୍କ୍ସନ substr() ଲେଖ ଯାହାର ପ୍ରୋଟୋଟାଇପ୍ ନିମ୍ନରେ ଦିଆଯାଇଛି

```
char *substr(char *str1, char *str2);
```

ଏହା ଦ୍ୱିତୀୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ର ଉପସ୍ଥିତି ଖୋଜିବା ପାଇଁ ପ୍ରଥମ ଷ୍ଟ୍ରିଙ୍ଗ୍କୁ ପଢ଼େ, ଏବଂ ପ୍ରଥମ ଷ୍ଟ୍ରିଙ୍ଗରେ ଯେଉଁଠାରୁ ଦ୍ୱିତୀୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଆରମ୍ଭ

ହୋଇଛି ସେହି ଉପାଦାନକୁ ଇସାରା କରୁଥିବା ଏକ ପଏଁଟର ଫେରସ୍ତ କରେ । ଯଦି ଦ୍ଵିତୀୟ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପ୍ରଥମ ଷ୍ଟ୍ରିଙ୍ଗରେ ନ ଥାଏ, ତେବେ ଫଙ୍କ୍ସନ NULL ଫେରସ୍ତ କରିବା ଉଚିତ । ଉପରୋକ୍ତ ଫଙ୍କ୍ସନ ପରୀକ୍ଷା କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

ପ୍ରାକ୍ଟିକାଲ୍ (PRACTICALS)

1. ସୀମିତ ସଂଖ୍ୟକ ଅପରେସନ ସହିତ ଲିନିୟର ଲିଙ୍କ୍ଡ ଲିଷ୍ଟ (linear linked lists) କାର୍ଯ୍ୟକାରୀ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ୍ ଲେଖ ।

Listing 9.6

```
/*
   Program to demonstrate the use of pointers and structures
   by implementing a linear linked list, where the insertion and
   deletions operations are limited to beginning of the l
   inked list,for simplicity.

   Program used separate function to perform each operation.
*/

#include <stdio.h>
#include <stdlib.h>

typedef struct nodeType
{
int info;
    struct nodeType *next;
} NODE;

/*-------   function prototypes   -------*/
void traverse(NODE *);
void search(NODE *, int);
NODE *insert(NODE * int);
NODE *delete(NODE *);

int main()
{
    NODE *head = NULL;
    int choice, element, after;
    while ( 1 )
    {
         printf( "\n\n         Options available \n" );
        printf( "+++++++++++++++++++++++++++++++++ \n\n" );
        printf( "  1.  Insert\n" );
        printf( "  2.  Delete\n" );
        printf( "  3.  Search\n" );
        printf( "  4.  Traverse\n" );
```

```
    printf( "  5.  Exit\n\n" );
printf( "Enter your choice ( 1-5 ) : " );
         scanf( "%d", &choice );
         switch ( choice )
         {
            case 1 : printf( "\nEnter element : " );
                                    scanf( "%d", &element );
                                    head = insert( head, element
);
                                    break;
            case 2 : head = delete(head);
                     break;
            case 3 : printf( "\nEnter element to search : " );
                                    scanf( "%d", &element );
                                    search( element );
                                    break;
case 4 : if ( head == NULL )
                                      printf( "\nList is empty ...
" );
                                    else
                                       traverse(head);
                                    printf("\n\nPress any key to
continue ... ");
                                    getch();
                                    break;
case 5 : printf("\nProgram terminated on success\n");
                     return 0;
        }
     }
 }  /****** end of main function ******/


 /*
    function to traverse and print elements of the linked list
 */
 void traverse(NODE *head)
 {
     NODE *ptr = head;
     printf("\n\nLinked list\n\n");
     printf("\thead" );
     while ( ptr != NULL )
     {
         printf( " -> %d", ptr->info );
         ptr = ptr->next;
     }
 }
 /*
```

```
 function to search a given element in the linked list
 */
 void search(NODE *head, int item)
 {
     NODE *ptr = head;
     while ( ptr != NULL )
                  {
        if ( item == ptr->info )
        {
        printf("\nElement %d found in linked list\n", item);
            return;
                       }
        ptr = ptr->next;
                  }
    printf("\nElement %d not found in linked list\n", item);
}
 /*
    function to insert node in the beginning of the linked list
 */
 NODE *insert(NODE *head, int item )
 {
    NODE *ptr;
    ptr = ( NODE * ) malloc( sizeof( NODE ) );
    ptr->info = item;
    ptr->next =  head;
    head = ptr;
    return ptr;
 }
 /*
    function to delete node from the beginning of the linked list
 */
 NODE *delete(NODE *head)
 {
     NODE *ptr;
     if ( head == NULL ) {
         printf("\nList is empty ... ");
         return;
                  }
     printf("\nElement %d deleted from list\n", head->info);
     ptr = head;
     head = head->next;
     free(ptr);
     return head;
}
```

Test Run

```
Options available
 +++++++++++++++++++++++++++++++++

   1.  Insert
   2.  Delete
   3.  Search
   4.  Traverse
   5.  Exit

 Enter your choice ( 1-5 ) : 1

 Enter element : 1
         Options available
 +++++++++++++++++++++++++++++++++

   1.  Insert
   2.  Delete
   3.  Search
   4.  Traverse
   5.  Exit
Enter your choice ( 1-5 ) : 1

 Enter element : 15

         Options available
 +++++++++++++++++++++++++++++++++

   1.  Insert
   2.  Delete
   3.  Search
   4.  Traverse
   5.  Exit

 Enter your choice ( 1-5 ) : 1

 Enter element : 10
Options available
 +++++++++++++++++++++++++++++++++

   1.  Insert
   2.  Delete
   3.  Search
   4.  Traverse
   5.  Exit
```

```
Enter your choice ( 1-5 ) : 1

 Enter element : 25

         Options available
 +++++++++++++++++++++++++++++++++

   1.  Insert
   2.  Delete
   3.  Search
   4.  Traverse
   5.  Exit

 Enter your choice ( 1-5 ) : 4



 Linked list
head -> 10 -> 15 -> 1

 Press any key to continue ...

         Options available
 +++++++++++++++++++++++++++++++++

   1.  Insert
   2.  Delete
   3.  Search
   4.  Traverse
   5.  Exit

 Enter your choice ( 1-5 ) : 5

 Program terminated on success\n");
```

10 ଫାଇଲ୍ ପରିଚାଳନା

ଏକକ ବିଶେଷତ୍ୱ

ଏହି ୟୁନିଟରେ ଫାଇଲ୍ ସମ୍ବନ୍ଧୀୟ ବିଷୟଗୁଡିକ ଉପରେ ଆଲୋଚନା ହୋଇଛି । ତଥ୍ୟର ଦୀର୍ଘକାଳୀନ ଭଣ୍ଡାରଣପାଇଁ ଫାଇଲ୍ ଏକ ମାଧ୍ୟମ ପ୍ରଦାନ କରେ । କିଛି କମାଣ୍ଡ ବ୍ୟବହାର କରି ଫାଇଲରୁ ତଥ୍ୟ ପଢ଼ାଯାଇପାରିବ କିମ୍ବା ଫାଇଲରେ ଲେଖାଯାଇପାରିବ । ଏହି ୟୁନିଟରେ ଫାଇଲଗୁଡିକର ବିଭିନ୍ନ ଦିଗ ବ୍ୟାଖ୍ୟା କରାଯାଇଅଛି ଏବଂ ଉପଯୁକ୍ତ ଉଦାହରଣ ସହିତ ସେଗୁଡ଼ିର ବ୍ୟବହାର ପ୍ରଦର୍ଶନ କରାଯାଇଅଛି ।

ଭୂମିକା

ବାସ୍ତବ ଜୀବନର ସମସ୍ୟାରେ, ଆମକୁ ନିମ୍ନଲିଖିତ ପରିସ୍ଥିତି ମଧ୍ୟରୁ ଯେକୌଣସି ସହିତ ମୁକାବିଲା କରିବାକୁ ପଡିବ:

- ଇନପୁଟ୍ ତଥ୍ୟର ପରିମାଣ ଅତ୍ୟଧିକ ହୋଇପାରେ
- ସମାନ ତଥ୍ୟ ଏକାଧିକ ଥର ପ୍ରକ୍ରିୟାକରଣ କରିବାକୁ ଆବଶ୍ୟକ ହୋଇପାରେ
- ଗୋଟିଏ ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ ଅନ୍ୟ ପ୍ରୋଗ୍ରାମ ପାଇଁ ଇନପୁଟ୍ ଭାବରେ ବ୍ୟବହାର କରିବାକୁ ପଡିପାରେ
- ମେସିନ୍- ଉତ୍ପନ୍ନ ତଥ୍ୟ ହୋଇପାରେ, ଯାହା ମାନବ-ପଠନଯୋଗ୍ୟ ନୁହେଁ

ଏହି ସମସ୍ତ ପରିସ୍ଥିତିରେ, କୀବୋର୍ଡରୁ ତଥ୍ୟ ଭର୍ତ୍ତି କରିବା ଏବଂ ପ୍ରୋଗ୍ରାମ ଚାଳନ ସମୟରେ କମ୍ପ୍ୟୁଟର ସ୍କ୍ରିନରେ ଫଳାଫଳ ଦର୍ଶାଇବାର ପାରମ୍ପାରିକ ଉପାୟ କାର୍ଯ୍ୟତଃ ଅସମ୍ଭବ ।

ଉତ୍ତମ ସମାଧାନ ହେଉଛି ଇନପୁଟ୍ ତଥ୍ୟକୁ ଏକ ଫାଇଲରେ ଗଚ୍ଛିତ କରାଯାଇପାରେ, ଯାହାକୁ ଏକ ଡାଟା ଫାଇଲ୍ କୁହାଯାଏ, ଏବଂ ତା'ପରେ ସେହି ଫାଇଲରୁ ତଥ୍ୟ ପଢିବାପାଇଁ ପ୍ରୋଗ୍ରାମ ଦ୍ୱାରା ନିର୍ଦ୍ଦେଶ ଦିଆଯାଇପାରିବ । ସେହିଭଳି, ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ ଏକ ଡାଟା ଫାଇଲ୍‌ରେ ଲେଖିବା ପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମକୁ ମଧ୍ୟ ନିର୍ଦ୍ଦେଶ ଦିଆଯାଇପାରେ ।

ଫାଇଲଗୁଡିକ ସହିତ କାରବାର କରୁଥିବା ଏହି ଅପରେସନ୍‌ଗୁଡିକୁ ଫାଇଲ୍ ପରିଚାଳନା କୁହାଯାଏ ।

ଏହି ୟୁନିଟ ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କୁ ଫାଇଲର ବିଭିନ୍ନ ଦିଗଗୁଡ଼ିକୁ ବୁଝିବାରେ ଏବଂ ବାସ୍ତବ ଜୀବନର ସମସ୍ୟାର ସମାଧାନ ପାଇଁ ଫାଇଲ୍ ବ୍ୟବହାର କରି ପ୍ରୋଗ୍ରାମ ବିକାଶ କରିବାରେ ସାହାଯ୍ୟ କରିବ ।

ପୂର୍ବାବଶ୍ୟକ ଜ୍ଞାନ

- ମାନକ ଇନପୁଟ୍ / ଆଉଟପୁଟ୍ (Standard Input / Output)
- ଅଣଫର୍ମାଟେଡ୍ ଏବଂ ଫର୍ମାଟେଡ୍ ଇନପୁଟ୍ / ଆଉଟପୁଟ୍ (Unformatted and formatted input / output)
- ପୂର୍ବନିର୍ଦ୍ଧାରିତ ଫାଇଲ୍ / ଷ୍ଟ୍ରିମ୍ (Predefined files / steams)

ୟୁନିଟଲବ୍ଧ ଜ୍ଞାନ

ଏହି ୟୁନିଟର ସମ୍ପୂର୍ଣ୍ଣ ଅଧ୍ୟୟନ ପରେ, ଛାତ୍ରଛାତ୍ରୀଗଣ ନିମ୍ନଲିଖିତ ବିଷୟଗୁଡ଼ିକରେ ଜ୍ଞାନ ଆହରଣରେ ସକ୍ଷମ ହେବେ ।

U10-O1: ଡାଟା ଫାଇଲ୍‌ର ଧାରଣା ଏବଂ ଏହାର ଉପଯୋଗିତା ବ୍ୟାଖ୍ୟା

U10-O2: ଫାଇଲଗୁଡିକର ଖୋଲିବା ଏବଂ ବନ୍ଦ କରିବାର କୌଶଳ ପ୍ରଦର୍ଶନ

U10-O3: ଫାଇଲଗୁଡିକରେ ପଢିବା ଏବଂ ଲେଖିବା ସଂକ୍ରିୟା କରିବା

U10-O4: ଫାଇଲ୍ ପରିଚାଳନା ପାଇଁ ଦକ୍ଷ ପ୍ରୋଗ୍ରାମ ଲେଖିବା

ୟୁନିଟ -10 ୟୁନିଟଲବ୍ଧ ଜ୍ଞାନ	ବିଷୟଲବ୍ଧ ଜ୍ଞାନ ସହ ଅପେକ୍ଷିତ ସମ୍ବନ୍ଧ *(1 – ଦୁର୍ବଳ ସମ୍ପର୍କ; 2 – ମଧ୍ୟମ ସମ୍ପର୍କ; 3 – ଦୃଢ ସମ୍ପର୍କ)*							
	CO-1	CO-2	CO-3	CO-4	CO-5	CO-6	CO-7	CO-8
U10-O1	1	–	–	–	–	–	–	–
U10-O2	–	–	1	–	–	–	–	–
U10-O3	–	–	1	–	–	–	–	–
U10-O4	–	–	–	–	–	2	–	–

10.1 ଉପକ୍ରମ (INTRODUCTION)

ଗୋଟିଏ ଫାଇଲ୍ ହେଉଛି ସମ୍ବନ୍ଧୀୟ ତଥ୍ୟଗୁଡ଼ିକର ଏକ ସଂଗ୍ରହ । ଫାଇଲ୍‌ର ପ୍ରାଥମିକ ଉଦ୍ଦେଶ୍ୟ ହେଉଛି ତଥ୍ୟର ଏକ ରେକର୍ଡ ରଖିବା । ଯେହେତୁ କମ୍ପ୍ୟୁଟର ମେମୋରି ଉଦ୍‌ବାୟୀ ଅଟେ (ଅର୍ଥାତ, କମ୍ପ୍ୟୁଟର୍ ବନ୍ଦ ହେବା ପରେ ମେମୋରି ବିଷୟବସ୍ତୁ ହଜିଯାଏ), ତେଣୁ ଆମକୁ ପରବର୍ତ୍ତି ସମୟରେ ବ୍ୟବହାର ପାଇଁ ତଥ୍ୟ ଗଚ୍ଛିତ ରଖିବାକୁ ହେବ । ଏହା ବ୍ୟତୀତ, ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ସମୟରେ ମେମୋରିରେ ସମ୍ପୂର୍ଣ୍ଣ ଭାବରେ ରହିବା ପାଇଁ ତଥ୍ୟର ପରିମାଣ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ହୋଇପାରେ । ତେଣୁ, ଆମର ପ୍ରୋଗ୍ରାମଗୁଡିକ ତଥ୍ୟର ଏକ ଅଂଶକୁ ପଢିବା ଏବଂ ଲେଖିବାର କ୍ଷମତା ରହିବା ଆବଶ୍ୟକ ଯେତେବେଳେ ବାକି ଡାଟା ଫାଇଲ୍‌ରେ ରହିଥାଏ ।

ପ୍ରୋଗ୍ରାମଗୁଡିକ ଦ୍ୱାରା ବ୍ୟବହୃତ ହେବାକୁ ଥିବା ଫାଇଲଗୁଡିକ ସାଧାରଣତଃ ହାର୍ଡ ଡିସ୍କରେ ଗଚ୍ଛିତ କରାଯାଏ । ଯେତେବେଳେ ପ୍ରୋଗ୍ରାମ ପଢେ, ଫାଇଲରୁ ତଥ୍ୟ ମେମୋରିକୁ ସ୍ଥାନାନ୍ତର ହୁଏ; ଲେଖିବା ସମୟରେ ଏହା ମେମୋରିରୁ ଫାଇଲ୍‌କୁ ସ୍ଥାନାନ୍ତର ହୁଏ । ଏହି ସ୍ଥାନାନ୍ତରିତ ତଥ୍ୟ ନିର୍ଦ୍ଦିଷ୍ଟ ଏକ କାର୍ଯ୍ୟ କ୍ଷେତ୍ର ବ୍ୟବହାର କରେ ଯାହାକୁ ବଫର୍ କୁହାଯାଏ, ଅର୍ଥାତ, ଏକ ବଫର୍ ହେଉଛି ଏକ ଅସ୍ଥାୟୀ ଭଣ୍ଡାରଣ କ୍ଷେତ୍ର ଯାହା ମେମୋରି କୁ/ଠାରୁ ଯିବା ସମୟରେ ତଥ୍ୟ ଧାରଣ କରେ ।

C ଭାଷାରେ ଏକ ପ୍ରୋଗ୍ରାମ ବିଭିନ୍ନ ଉପାୟରେ ଫାଇଲ୍ ପଢିପାରେ ଏବଂ ଲେଖିପାରେ । ଏହି ୟୁନିଟରେ ପ୍ରତ୍ୟେକ ଉପାୟର ସିଧାସଳଖ ବର୍ଣ୍ଣନା ହୋଇଛି ଏବଂ ଭଲ ଭାବେ ଡିଜାଇନ୍ ହୋଇଥିବା ପ୍ରୋଗ୍ରାମିଂ ଉଦାହରଣ ସହିତ ଦର୍ଶାଯାଇଛି ।

10.2 ଫାଇଲ୍ ପ୍ରକାର

C ଭାଷାରେ ଦୁଇ ଟାଇପ୍‌ର ଫାଇଲ୍ ଅଛି - ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଏବଂ ବାଇନାରି ଫାଇଲ୍ ।

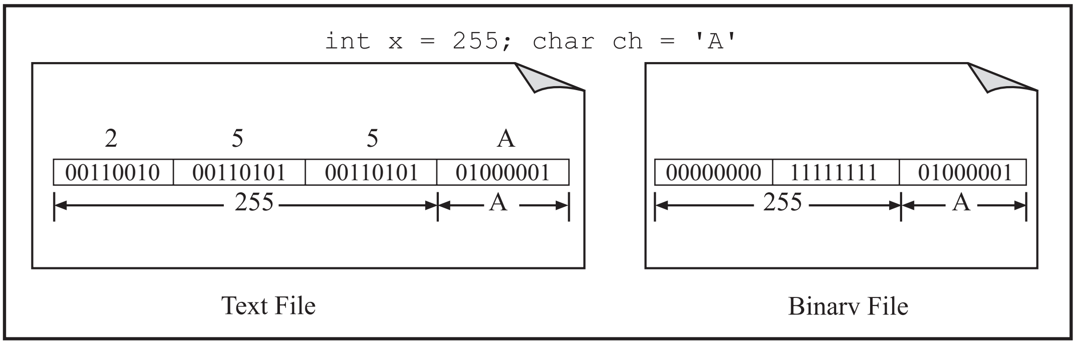
- **ଟେକ୍ସଟ୍ ଫାଇଲ୍ (Text File) -** ଗୋଟିଏ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ବର୍ଣମାଳା, ସଂଖ୍ୟା ଏବଂ ଅନ୍ୟାନ୍ୟ ବିଶେଷ ସଙ୍କେତଗୁଡିକର ତଥ୍ୟ ସେଗୁଡ଼ିର ASCII ମୂଲ୍ୟ ଦ୍ୱାରା ସଂରକ୍ଷଣ କରେ । ଏହା ମଣିଷ ପଠନ ଉପଯୋଗୀ ରୂପରେ ଥାଏ । ଏହି ଫାଇଲ୍‌ଗୁଡିକ ଖୋଲିବା ବେଳେ, ଫାଇଲ୍ ମଧ୍ୟରେ ଥିବା ସମସ୍ତ ବିଷୟବସ୍ତୁ ସାଧାରଣ ଟେକ୍ସଟ୍ ଭାବରେ ଦେଖି ହୁଏ । ଯେ କେହି ସହଜରେ ବିଷୟବସ୍ତୁ ସମ୍ପାଦନ କିମ୍ବା ବିଲୋପ କରିପାରିବ । ଏହି ଫାଇଲଗୁଡିକ ରକ୍ଷଣାବେକ୍ଷଣ କରିବାକୁ

ସର୍ବନିମ୍ନ ପ୍ରୟାସ ଦରକାର କରିଥାଏ । ସହଜରେ ପଠନଯୋଗ୍ୟ ହୋଇଥାଏ ସର୍ବନିମ୍ନ ସୁରକ୍ଷା ପ୍ରଦାନ କରିଥାଏ, ଏବଂ ଗଚ୍ଛିତ କରିବାପାଇଁ ଅନେକ ସ୍ଥାନ ନେଇଥାଏ ।

- **ବାଇନାରି ଫାଇଲ୍ (Binary File) -** ଏକ ବାଇନାରି ଫାଇଲ୍ ତଥ୍ୟକୁ ବାଇଟ୍‌ର ଏକ ଅନୁକ୍ରମ ଭାବେ ସଂରକ୍ଷଣ କରେ ଯାହା ମାନବ-ପଠନଯୋଗ୍ୟ ସଂରୂପରେ ନଥାଏ । ସେଗୁଡିକ କେବଳ ପ୍ରୋଗ୍ରାମର ବ୍ୟବହାରଦ୍ୱାରା ତିଆରି କରାଯାଇପାରିବ । ସେମାନେ ବହୁ ପରିମାଣର ତଥ୍ୟ ଧାରଣ କରିପାରିବ, ସହଜରେ ପଠନଯୋଗ୍ୟ ନୁହଁନ୍ତି, ଏବଂ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଅପେକ୍ଷା ଉତ୍ତମ ସୁରକ୍ଷା ପ୍ରଦାନ କରିଥାନ୍ତି ।

ଟେବୁଲ୍ 10.1: ଟେକ୍ସଟ୍ ଫାଇଲ୍ ବନାମ ବାଇନାରି ଫାଇଲ୍

ଟେକ୍ସଟ୍ ଫାଇଲ୍	ବାଇନାରି ଫାଇଲ୍
ବର୍ଣ୍ଣଗୁଡିକ ବ୍ୟବହାର କରି ମାନବ-ପଠନଯୋଗ୍ୟ ତଥ୍ୟ ଗଚ୍ଛିତ ହୋଇଥାଏ ।	ଏହା ମେମୋରିରେ ଗଚ୍ଛିତ ତଥ୍ୟ ଭଳି ସମାନ ସଂରୁପରେ ଗଚ୍ଛିତ ହୋଇଥାଏ ।
ତଥ୍ୟର ପ୍ରତ୍ୟେକ ଧାଡ଼ି ଏକ ନୂତନ-ଧାଡ଼ି(Newline) ବର୍ଣ୍ଣ ସହ ଶେଷ ହୁଏ ।	ନୂତନ-ଧାଡ଼ି ବର୍ଣ୍ଣ ନ ଥାଏ ।
ଫାଇଲ୍ ଶେଷରେ ଏକ ଅନନ୍ୟ ବର୍ଣ୍ଣ ଥାଏ ଯାହାକୁ ଏଣ୍ଡ-ଅଫ୍-ଫାଇଲ୍ (EOF) କୁହାଯାଏ । ।	ଫାଇଲ୍ ଶେଷରେ ଏଣ୍ଡ-ଅଫ୍-ଫାଇଲ୍ ନାମକ ଏକ ମାର୍କର୍ ଥାଏ ।



ଚିତ୍ର - 10.1: ଫାଇଲଗୁଡିକରେ ତଥ୍ୟ ସଂରକ୍ଷଣର ଚିତ୍ରଣ

10.3 ଫାଇଲ୍ ପ୍ରକ୍ରିୟାକରଣର ପଦକ୍ଷେପ

ଏକ ଫାଇଲ୍ ପ୍ରକ୍ରିୟାକରଣରେ ନିମ୍ନଲିଖିତ ପଦକ୍ଷେପଗୁଡିକ ସଂଶ୍ଳିଷ୍ଟ ଅଟେ:

1. **ଫାଇଲ୍ ଖୋଲିବା :** ଫାଇଲ୍ ପ୍ରକ୍ରିୟାକରଣର ପ୍ରଥମ ପଦକ୍ଷେପ ହେଉଛି ଏକ ଉପଯୁକ୍ତ ମୋଡରେ ଫାଇଲ୍ ଖୋଲିବା । ଯଦି ତୁମେ ତୁମର ପ୍ରୋଗ୍ରାମରେ ଏକ ଫାଇଲରୁ ପଢିବାକୁ ଚାହୁଁଛ, ତେବେ ଫାଇଲ୍ ପଠନ/ଇନପୁଟ୍ ମୋଡରେ ଖୋଲିବା ଆବଶ୍ୟକ । ସେହିଭଳି, ଯଦି ତୁମେ ଏକ ଫାଇଲ୍ ଭିତରେ ତୁମର ପ୍ରୋଗ୍ରାମର ଆଉଟପୁଟ୍ ଲେଖିବାକୁ ଚାହୁଁଛ, ତେବେ ଫାଇଲ୍‌କୁ ଲିଖନ/ଆଉଟପୁଟ୍ ମୋଡରେ ଖୋଲିବା ଆବଶ୍ୟକ । ଯାହାହେଉ, ଯେତେବେଳେ ତୁମେ ଫାଇଲକୁ ଏକକାଳୀନ ଉଭୟ ପଢିବାକୁ ଏବଂ ଲେଖିବାକୁ ଚାହୁଁଛ; ତେବେ, ଫାଇଲ୍‌କୁ ପଠନ-ଲିଖନ ମୋଡରେ ଖୋଲିବା ଆବଶ୍ୟକ ।
2. **ଫାଇଲରେ ଲେଖାପଢ଼ା କରିବା :** ଥରେ ଏକ ଫାଇଲ୍ ସଫଳତାର ସହ ଖୋଲିବା ପରେ, ଫାଇଲରୁ ତଥ୍ୟ ପଢାଯାଇପାରିବ କିମ୍ବା ବିଭିନ୍ନ ଉପାୟରେ ଫାଇଲ୍‌କୁ ଲେଖାଯାଇପାରିବ ।

3. **ଫାଇଲ୍ ବନ୍ଦ କରିବା :** ଫାଇଲ୍ ପ୍ରକ୍ରିୟାକରଣରେ ଏହା ହେଉଛି ଅନ୍ତିମ ପଦକ୍ଷେପ । ଥରେ ଆମେ ଫାଇଲଗୁଡିକର ପଢିବା ଏବଂ ଲେଖିବା କାର୍ଯ୍ୟ ସମାପ୍ତ କରିବା ପରେ, ପ୍ରତ୍ୟେକ ଫାଇଲ୍‌କୁ ବନ୍ଦ କରିବା ଆବଶ୍ୟକ ଅଟେ ।

10.3.1 ଫାଇଲ୍ ଖୋଲିବା

ପ୍ରୋଗ୍ରାମଟି ଏକ ଫାଇଲ୍‌ରେ ଲେଖିପଢା କରିବା ପୂର୍ବରୁ, ନିଶ୍ଚିତ ଭାବରେ ଏହାକୁ ଖୋଲିବ । ଫାଇଲ୍ ଖୋଲା ହେବା ସମୟରେ ପ୍ରୋଗ୍ରାମ ଏବଂ ଅପରେଟିଂ ସିଷ୍ଟମ୍ ମଧ୍ୟରେ ଏକ ବୁଝାମଣା ହୁଏ । ପ୍ରୋଗ୍ରାମଟି ଫାଇଲ୍ ନାମ ଏବଂ ମୋଡ୍ ଯେଉଁଥିରେ ଫାଇଲ୍ ଖୋଲିବାକୁ ହେବ ଅପରେଟିଂ ସିଷ୍ଟମ୍‌କୁ ପ୍ରଦାନ କରେ, ଅର୍ଥାତ, ପଢିବା, ଲେଖିବା, କିମ୍ବା ଯୋଡିବା ପାଇଁ ହେଉ ।

ଫାଇଲ୍ ଏବଂ ପ୍ରୋଗ୍ରାମ ମଧ୍ୟରେ ଯୋଗାଯୋଗ କ୍ଷେତ୍ରଗୁଡିକ ପ୍ରତିଷ୍ଠା ହୋଇଛି । ଏହି କ୍ଷେତ୍ରଗୁଡ଼ିକ ମଧ୍ୟରୁ ଗୋଟିଏ ହେଉଛି FILE ଟାଇପ୍‌ର ଏକ ଷ୍ଟ୍ରକ୍‌ଚର୍ ଯାହା stdio.h ହେଡର୍ ଫାଇଲ୍‌ରେ ଘୋଷିତ ଏବଂ ଏହା ଫାଇଲ୍ ସମ୍ବନ୍ଧୀୟ ସୂଚନାଗୁଡ଼ିକ ଧାରଣ କରେ ।

ନିମ୍ନଲିଖିତ ଘୋଷଣା ବ୍ୟବହାର କରି ପ୍ରତ୍ୟେକ ଫାଇଲ୍‌ପାଇଁ ଆମକୁ ଗୋଟିଏ ଫାଇଲ୍ ପଏଁଟର୍ ଘୋଷଣା କରିବା ଆବଶ୍ୟକ

```
FILE *fp;
```

ପରବର୍ତ୍ତୀ କାଳରେ, ଆମେ fopen() ଫଙ୍କ୍‌ସନ୍ ବ୍ୟବହାର କରି ଫାଇଲ୍ ଖୋଲିପାରିବା । ଯେପରି

```
fp = fopen( "filename", "mode" );
```

filename ନାମକ ଏକ ଫାଇଲ୍ ଦିଆଯାଇଥିବା ମୋଡ୍‌ରେ ଖୋଲିବାକୁ fopen() ଫଙ୍କ୍‌ସନ୍ ଅପରେଟିଂ ସିଷ୍ଟମକୁ ଅନୁରୋଧ କରିଥାଏ । ଯଦି ଅନୁରୋଧ ମଞ୍ଜୁର ହୁଏ ତେବେ ଏହା FILE ଷ୍ଟ୍ରକ୍‌ଚର୍‌କୁ ଏକ ପଏଁଟର୍ ଫେରସ୍ତ କରେ; ଅନ୍ୟଥା ଏକ NULL ପଏଁଟର୍ ଫେରସ୍ତ କରେ ।

ତେଣୁ, fopen() ଫଙ୍କ୍‌ସନ୍ ଦ୍ୱାରା ଫେରସ୍ତ ହୋଇଥିବା ମୂଲ୍ୟକୁ ଏକ ପ୍ରୋଗ୍ରାମ ଯାଞ୍ଚ କରି ଫାଇଲ୍ ସଫଳତାର ସହ ଖୋଲାଯାଇଛି କି ନାହିଁ ତାହା ନିର୍ଦ୍ଧାରଣ କରିପାରିବ । ଏକ ଫାଇଲ୍ ଖୋଲିବା ପାଇଁ ବ୍ୟବହୃତ ଏକ ପ୍ରୋଗ୍ରାମ ଖଣ୍ଡ ଲେଖିବା ।

```
fp = fopen ("filename", "mode" );
if ( fp == NULL )
{
   printf( "\nUnable to open file.\n" );
   return 1;
}
```

ଖୋଲାଯାଇଥିବା ପ୍ରତ୍ୟେକ ଫାଇଲର ନିଜସ୍ୱ FILE ଷ୍ଟ୍ରକ୍‌ଚର୍ ଏହାର ଏକ ପଏଁଟର୍ ସହିତ ରହିଥାଏ । ଯେକୌଣସି ପ୍ରୋଗ୍ରାମ ଯେକୌଣସି ସଂଖ୍ୟକ ଫାଇଲ୍ ଖୋଲିପାରେ ।

ଟେବୁଲ୍ 10.2: ଫାଇଲ୍ ଖୋଲିବା ମୋଡ୍

ମୋଡ୍ (Mode)	ବର୍ଣ୍ଣନା (Description)
R	ପଢିବା ପାଇଁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଖୋଲ । ଫାଇଲ୍ ପୂର୍ବରୁ ବିଦ୍ୟମାନ ଥିବା ଆବଶ୍ୟକ ।
w	ଲେଖିବା ପାଇଁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଖୋଲ । ଯଦି ଫାଇଲ୍ ପୂର୍ବରୁ ବିଦ୍ୟମାନ ଅଛି, ଏହାର ବିଷୟବସ୍ତୁ ଅଧିଲିଖିତ ହୁଏ । ଯଦି ଏହା ବିଦ୍ୟମାନ ନାହିଁ, ତେବେ ଏହା ସୃଷ୍ଟି ହେବ ।

a	ଯୋଡିବା ପାଇଁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଖୋଲ । ଫାଇଲ୍ ଶେଷରେ ତଥ୍ୟ ଯୋଡାଯିବ । ଯଦି ଫାଇଲ୍ ବିଦ୍ୟମାନ ନାହିଁ, ଏହା ସୃଷ୍ଟି ହେବ ।
r+	ଉଭୟ ପଠନ ଏବଂ ଲେଖନ ପାଇଁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଖୋଲ । ଫାଇଲ୍ ପୂର୍ବରୁ ବିଦ୍ୟମାନ ହେବା ଆବଶ୍ୟକ ।
w+	ଉଭୟ ପଠନ ଏବଂ ଲେଖନ ପାଇଁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଖୋଲ । ଯଦି ଫାଇଲ୍ ବିଦ୍ୟମାନ ଅଛି, ଏହାର ବିଷୟବସ୍ତୁ ଅଧିଲିଖିତ ହୁଏ । ଯଦି ଏହା ବିଦ୍ୟମାନ ନାହିଁ, ତେବେ ଏହା ସୃଷ୍ଟି ହେବ ।
a+	ଉଭୟ ପଠନ ଏବଂ ଯୋଡିବା ପାଇଁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଖୋଲ । ଯଦି ଫାଇଲ୍ ବିଦ୍ୟମାନ ନାହିଁ, ଏହା ସୃଷ୍ଟି ହେବ ।
rb	ପଢିବା ପାଇଁ ଏକ ବାଇନାରି ଫାଇଲ୍ ଖୋଲ । ଫାଇଲ୍ ପୂର୍ବରୁ ବିଦ୍ୟମାନ ହେବା ଆବଶ୍ୟକ ।
wb	ଲେଖିବା ପାଇଁ ଏକ ବାଇନାରି ଫାଇଲ୍ ଖୋଲ । ଯଦି ଫାଇଲ୍ ପୂର୍ବରୁ ବିଦ୍ୟମାନ ଅଛି, ଏହାର ବିଷୟବସ୍ତୁ ଅଧିଲିଖିତ ହୁଏ । ଯଦି ଏହା ବିଦ୍ୟମାନ ନାହିଁ, ତେବେ ଏହା ସୃଷ୍ଟି ହେବ ।
ab	ଯୋଡିବା ପାଇଁ ଏକ ବାଇନାରି ଫାଇଲ୍ ଖୋଲ । ଫାଇଲ୍ ଶେଷରେ ତଥ୍ୟ ଯୋଡାଯିବ । ଯଦି ଫାଇଲ୍ ବିଦ୍ୟମାନ ନାହିଁ, ତେବେ ଏହା ସୃଷ୍ଟି ହେବ ।
rb+	ଉଭୟ ପଠନ ଏବଂ ଲେଖନ ପାଇଁ ଏକ ବାଇନାରି ଫାଇଲ୍ ଖୋଲ । ଫାଇଲ୍ ପୂର୍ବରୁ ବିଦ୍ୟମାନ ହେବା ଆବଶ୍ୟକ ।
wb+	ଉଭୟ ପଠନ ଏବଂ ଲେଖନ ପାଇଁ ଏକ ବାଇନାରି ଫାଇଲ୍ ଖୋଲ । ଯଦି ଫାଇଲ୍ ବିଦ୍ୟମାନ ଅଛି, ଏହାର ବିଷୟବସ୍ତୁ ଅଧିଲିଖିତ ହୁଏ । ଯଦି ଏହା ବିଦ୍ୟମାନ ନାହିଁ, ତେବେ ଏହା ସୃଷ୍ଟି ହେବ ।
ab+	ଉଭୟ ପଠନ ଏବଂ ଯୋଡିବା ପାଇଁ ଏକ ବାଇନାରି ଫାଇଲ୍ ଖୋଲ । ଯଦି ଫାଇଲ୍ ବିଦ୍ୟମାନ ନାହିଁ, ତେବେ ଏହା ସୃଷ୍ଟି ହେବ ।

10.3.2 ଫାଇଲ୍ ବନ୍ଦ କରିବା

ଯେତେବେଳେ ପ୍ରୋଗ୍ରାମରେ ଏକ ଫାଇଲ୍ ସହିତ ପଠନ/ଲେଖନ ସମାପ୍ତ ହୋଇଥାଏ, ତା'ପରେ ଏହା ବନ୍ଦ ହେବା ଆବଶ୍ୟକ । ଏହି କାର୍ଯ୍ୟଟି fclose() ଲାଇବ୍ରେରି ଫଙ୍କସନ୍‌ଦ୍ୱାରା ହୋଇଥାଏ, ଯାହାର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
FILE *fp;
```

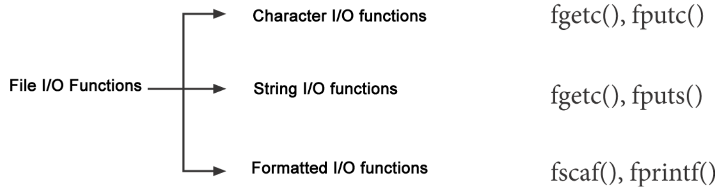
ଯେଉଁଠାରେ fp ଏକ ଫାଇଲ୍ ସହ ଜଡିତ ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଯାହାକୁ ବନ୍ଦ କରିବାକୁ ହେବ ।

ଏକ ଫାଇଲ୍ ସ୍ପଷ୍ଟ ଭାବରେ ବନ୍ଦ କରିବା ଦ୍ୱାରା ଦୁଇଟି ପ୍ରଭାବ ଅଛି । ଏଗୁଡ଼ିକ ହେଉଛି

- ବଫର୍‌ର ଅବଶିଷ୍ଟ ଥିବା ଡାଟା ଫାଇଲ୍‌ରେ ଲେଖାଯାଇଥାଏ । ଏହା ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ ବ୍ୟବହୃତ ବଫର୍ ପ୍ରୋଗ୍ରାମର୍ ପାଇଁ ଅଦୃଶ୍ୟ ।
- ନିର୍ଦ୍ଦିଷ୍ଟ ଫାଇଲ୍‌ଦ୍ୱାରା ବ୍ୟବହୃତ ଯୋଗାଯୋଗ କ୍ଷେତ୍ରଗୁଡିକ ମୁକ୍ତ କରାଯାଏ ଯାହା ଦ୍ୱାରା ସେଗୁଡିକ ଅନ୍ୟ ଫାଇଲ୍‌ପାଇଁ ଉପଲବ୍ଧ ହେବେ । ଏହି କ୍ଷେତ୍ରଗୁଡିକରେ FILE ଷ୍ଟ୍ରକ୍‌ଚର୍ ଏବଂ ବଫର୍ ଅନ୍ତର୍ଭୁକ୍ତ ।

10.3.3 ଟେକ୍ସଟ୍ ଫାଇଲ୍ ପଠନ ଏବଂ ଲେଖନ

ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍‌ରୁ ତଥ୍ୟ ପଢ଼ିବା ଏବଂ ଲେଖିବାର ବିଭିନ୍ନ ଉପାୟ ଅଛି । ଏଗୁଡ଼ିକ ହେଉଛି



10.3.3.1 ଅକ୍ଷର ଇନପୁଟ୍ / ଆଉଟପୁଟ୍ ଫଙ୍କସନ୍ ବ୍ୟବହାର କରି ପଠନ ଏବଂ ଲେଖନ

ଅକ୍ଷରର ଇନପୁଟ୍ / ଆଉଟପୁଟ୍ (I/O) ଫଙ୍କସନ୍‌ଗୁଡିକ ବ୍ୟବହାର କରି, ଏକ ସମୟରେ ତଥ୍ୟର ଗୋଟିଏ ଅକ୍ଷର ପଢ଼ିପାରିବ କିମ୍ବା ଲେଖାଯାଇପାରିବ । ଏହା କାର୍ଯ୍ୟଗୁଡ଼ିକ putchar() ଏବଂ getchar() ର ବ୍ୟବହାରଦ୍ୱାରା ସ୍କ୍ରିନରେ ତଥ୍ୟ ଲେଖିବା ଏବଂ କୀବୋର୍ଡ୍‌ରୁ ତଥ୍ୟ ପଢ଼ିବା ଶୈଳୀ ସହିତ ସମାନ ।

ଫାଇଲ୍‌କୁ ଲେଖିବା

ଥରେ ପ୍ରୋଗ୍ରାମ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଫାଇଲ୍ ଖୋଲି ଯୋଗାଯୋଗ ପ୍ରତିଷ୍ଠା କରିବା ପରେ, ଏଥିରେ ଲେଖିପାରେ । ଏକ ସମୟରେ ଗୋଟିଏ ଅକ୍ଷର ଲେଖି ହେଉଥିବା ଫଙ୍କସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
fputc( ch, fp );
```

ଯେଉଁଠାରେ *ch* ଏକ ଅକ୍ଷର ଚଳ କିମ୍ବା ଧ୍ରୁବକ, ଏବଂ *fp* ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ ।

ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମର ଉଦାହରଣଟି କୀବୋର୍ଡ୍‌ରୁ ଥରକେ ଗୋଟିଏ ଅକ୍ଷର ନେଇ *file1.dat* ନାମକ ଏକ ଡିସ୍କ ଫାଇଲ୍‌ରେ ଲେଖେ, ଏବଂ ସମୁଦାୟ ଗୋଟିଏ ଧାଡ଼ି ଅକ୍ଷର ଗ୍ରହଣ କରେ ।

Listing 10.1

```
/*
   Program to demonstrate writing of one character at a time to file
*/
#include<stdio.h>
int main()
{
   FILE *fp;
   char ch;
   fp = fopen( "file1.dat", "w" );
   if ( fp == NULL ) {
      printf( "\nUnable to open file1.dat\n" );
      return 1;
   }
   printf( "\nType a line of text, when finished" );
   printf( ", when finished hit Enter key\n" );
   while ( ( ch = getche() ) != '\r' )
      fputc ( ch, fp );
   fclose( fp );
   return 0;
}
```

ଯେତେବେଳେ ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ ଚାଳନ ହୁଏ, ଏହା ବ୍ୟବହାରକାରୀଙ୍କୁ ଟେକ୍ସଟ୍‌ର ଏକ ଧାଡ଼ି ଟାଇପ୍ କରିବାକୁ ପ୍ରେରିତ କରେ। ତୁମେ ଲେଖି ସାରିଲା ପରେ, ପ୍ରୋଗ୍ରାମ ସମାପ୍ତ କରିବାକୁ Enter (↵)କୀ କୁ ଦବାଅ।

Test Run

```
C is a powerful procedural language. It provides both low-level as well
as high-level features. ↵
```

ଫାଇଲରୁ ପଢ଼ିବା

ଯଦି ପ୍ରୋଗ୍ରାମ ଏକ ଫାଇଲ୍‌କୁ ଲେଖିପାରେ, ଏହା ମଧ୍ୟ ଏକ ଫାଇଲରୁ ପଢ଼ିବାକୁ ସମର୍ଥ ହେବା ଉଚିତ। ଏକ ସମୟରେ ଗୋଟିଏ ଅକ୍ଷର ପଠନ ଏବଂ ଫେରସ୍ତ କରୁଥିବା ଫଙ୍କସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
ch = fgetc( fp );
```

ଯେଉଁଠାରେ *ch* ଏକ ଅକ୍ଷର ଚଳ, ଏବଂ *fp* ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ।

fgetc() ଫଙ୍କସନ୍ ଫାଇଲ୍‌ରୁ ପଢିଥିବା ଅକ୍ଷରଟି ଫେରସ୍ତ କରେ କିମ୍ବା ଯଦି ଏହା ଫାଇଲ୍ ଶେଷରେ ପହଞ୍ଚି ଯାଇଛି ତେବେ ଫାଇଲ୍ ଶେଷ (EOF) ଅକ୍ଷର ଫେରସ୍ତ କରେ।

ନିମ୍ନଲିଖିତ ତାଲିକା ଏକ ପ୍ରୋଗ୍ରାମର ଉଦାହରଣ ଯାହା ଏକ ଫାଇଲରୁ ଗୋଟିଏ ଅକ୍ଷର ପଢେ ଏବଂ ସ୍କ୍ରିନରେ ଲେଖେ।

Listing 10.2

```
/*
     Program to demonstrate reading of character at a time from a file
 */
 #include<stdio.h>
 int main()
 {
    FILE *fp;
    char ch;
    fp = fopen( "file1.dat", "r" );
    if ( fp == NULL ) {
       printf( "\nUnable to open file1.dat\n" );
       return 1;
    }
    printf( "\nContens of file are:\n\n" );
    while ( ( ch = fgetc (fp) ) != EOF )
       putchar( ch );
    fclose( fp );
    return 0;
 }
```

ଯେତେବେଳେ ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ ଚାଳନ ହୁଏ, ଏହା file1.dat ର ବିଷୟବସ୍ତୁରୁ EOF ବର୍ଣ୍ଣର ସମ୍ମୁଖୀନ ନହେବା ପର୍ଯ୍ୟନ୍ତ ଥରକେ ଗୋଟିଏ ଅକ୍ଷର ପଢ଼ି, ସ୍କ୍ରିନରେ ଲେଖେ।

Test Run

```
Contents of the file are:
C is a powerful procedural language. It provides both low-level as well
as high-level features.
```

10.3.3.2 ଷ୍ଟ୍ରିଙ୍ଗ୍ I/O ଫଙ୍କ୍ସନ୍ ବ୍ୟବହାରଦ୍ୱାରା ପଠନ ଏବଂ ଲେଖନ

ଷ୍ଟ୍ରିଙ୍ଗ୍ I/O ଫଙ୍କ୍ସନ୍‌ଗୁଡିକ ବ୍ୟବହାର କରି, ଅକ୍ଷରଗୁଡ଼ିକର ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ରୂପେ ପଢିହେବ କିମ୍ବା ଲେଖିହେବ । ଗୋଟିଏ ଅକ୍ଷରର ପଢ଼ା/ଲେଖା କଲାଭଳି ଷ୍ଟ୍ରିଙ୍ଗର ଲେଖାପଢ଼ା ମଧ୍ୟ ଅତି ସହଜ ।

ଫାଇଲ୍‌ରେ ଲେଖିବା

ଏକାଥରେ ଅକ୍ଷରର ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଲେଖିବାପାଇଁ ଫଙ୍କ୍ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
fputs( str, fp );
```

ଯେଉଁଠାରେ *str* ଅକ୍ଷରଗୁଡିକର ଏକ ଆରେ କିମ୍ବା ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଧ୍ରୁବକ, ଏବଂ *fp* ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ ।

କୀବୋର୍ଡରୁ ଷ୍ଟ୍ରିଙ୍ଗର ଏକ ଶ୍ରେଣୀ ଗ୍ରହଣ କରିବା ଏବଂ ସେଗୁଡିକ ଏକ ଡିସ୍କ ଫାଇଲରେ ଲେଖିବାପାଇଁ ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ ଉଦାହରଣ ଦେଖ

Listing 10.3

```
/*
    Program to demonstrate writing of strings to a file
 */
 #include<stdio.h>
 #include<string.h>
 int main()
 {
    FILE *fp;
    char str[81];
    fptr = fopen( "file2.dat", "w" );
    if ( fp == NULL ) {
       printf( "\nUnable to open file2.dat\n" );
       return 1;
    }
    printf( "\nEnter a set of strings, press just" );
    printf( " Enter key to finish\n" );
    while ( strlen( gets( str ) ) > 0 )
    {
       fputs( str, fp );
       fputs( "\n", fp );
    }
    fclose( fp );
    return 0;
 }
```

Test Run

```
C is a powerful procedural language. ↵
It is a middle-level language. ↵
All Programs are tested using Turbo C Compiler. ↵
↵
```

ଯେତେବେଳେ ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମର ଚାଳନ ହୁଏ, ଏହା Enter କୀ ଦବାଇ ସମାପ୍ତ କରା ହୋଇଥିବା ଷ୍ଟ୍ରିଙ୍ଗଗୁଡିକର ଏକ ସେଟ୍ ଗ୍ରହଣ କରେ । ସରିଲା ପରେ, ପ୍ରୋଗ୍ରାମ ସମାପ୍ତ କରିବାକୁ କୌଣସି ଷ୍ଟ୍ରିଙ୍ଗ ଇନ୍‌ପୁଟ୍ ନ କରି କେବଳ Enter କୀ ଦବାଅ, ଯାହା 0 ଦୈର୍ଘ୍ୟ ବିଶିଷ୍ଟ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ନେବ, ଅର୍ଥାତ, *null* ଷ୍ଟ୍ରିଙ୍ଗ୍ ।

ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମରେ, ଆମେ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଗଚ୍ଛିତ କରିବାକୁ ଅକ୍ଷରଗୁଡିକର ଏକ ଆରେର ବ୍ୟବସ୍ଥା କରିଛୁ । ତାପରେ fputs() ଫଙ୍କ୍‌ସନ୍ ଆରେରେ ଥିବା ବିଷୟବସ୍ତୁକୁ ଡିସ୍କରେ ଲେଖେ । ଯେହେତୁ fputs() ଫଙ୍କ୍‌ସନ୍ ସ୍ୱତଃସ୍ଫୂର୍ତ ଭାବରେ ଷ୍ଟ୍ରିଙ୍ଗର ଶେଷରେ ଏକ ନୂତନ-ଧାଡ଼ି ଅକ୍ଷର ଯୋଡି ନ ଥାଏ, ତେଣୁ ଫାଇଲରୁ ଷ୍ଟ୍ରିଙ୍ଗକୁ ପଢିବାପାଇଁ ଆମକୁ ପ୍ରତ୍ୟକ୍ଷ ଭାବରେ ସରଳ କରିବାକୁ ପଡିବ ।

ଫାଇଲରୁ ପଢିବା

ଏକ ଫାଇଲରୁ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପଠନ କରୁଥିବା ଫଙ୍କ୍‌ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
fgets( str, n, fp );
```

ଯେଉଁଠାରେ *str* ଅକ୍ଷରଗୁଡ଼ିକର ଏକ ଆରେ ଆଉ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଭଣ୍ଡାରମ ହୋଇଥିବା ଠିକଣାକୁ ବୁଝାଏ, *n* ହେଉଛି ଇନପୁଟ୍ ଷ୍ଟ୍ରିଙ୍ଗର ସର୍ବାଧିକ ଦୈର୍ଘ୍ୟ, ଏବଂ *fp* ହେଉଛି ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ।

fgets() ଫଙ୍କ୍‌ସନ୍ ଏକ *NULL* ମୂଲ୍ୟ ଫେରସ୍ତ କରେ ଯେତେବେଳେ ଏହା ଫାଇଲର ଶେଷ, ଅର୍ଥାତ *EOF* ଅକ୍ଷର ପଠନ କରେ ।

ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମ ତାଲିକା ଏକ ଫାଇଲରୁ ଏକାଥରରେ ଗୋଟିଏ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପଢିବା ଏବଂ ସ୍କ୍ରିନରେ ଲେଖିବାର ଏକ ଉଦାହରଣକୁ ଦର୍ଶାଉଛି ।

Listing 10.4

```
/*
    Program to demonstrate reading of strings from a file
 */
 #include<stdio.h>
 #include<string.h>
 int main()
 {
    FILE *fp;
    char str[81];
    fp = fopen( "file2.dat", "r" );
    if ( fp == NULL ) {
       printf( "\nUnable to open file2.dat\n" );
       return 1;
    }
    printf( "\nContens of file are:\n\n" );
    while ( fgets( str, 80, fp ) != NULL ) {
       puts( str );
    }
    fclose( fp );
    return 0;
 }
```

ଯେତେବେଳେ ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ ଚାଳନ ହୁଏ, ଏହା EOF ର ସମ୍ମୁଖୀନ ନହେବା ପର୍ଯ୍ୟନ୍ତ file3.dat ର ବିଷୟବସ୍ତୁରୁ ଥରକେ ଗୋଟିଏ ଷ୍ଟ୍ରିଙ୍ଗ୍ ପଠନ କରେ ଏବଂ ସେଗୁଡିକ ସ୍କ୍ରିନରେ ଲେଖେ ।

Test Run

```
Contents of the file are:
C is a powerful procedural language.
It is a middle-level language.
All Programs are tested using Turbo C Compiler.
```

10.3.3.3 ସଂରୂପିତ I/O ଫଙ୍କ୍ସନ୍ ବ୍ୟବହାରଦ୍ୱାରା ପଠନ ଏବଂ ଲେଖନ

ଏପର୍ଯ୍ୟନ୍ତ, ଆମେ ଅକ୍ଷର, ଷ୍ଟ୍ରିଙ୍ଗ୍, ଏବଂ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ପଢିବା ଏବଂ ଲେଖିବା ବିଷୟରେ ବିଚାର କରିଛୁ । ବାସ୍ତବ ସଂଖ୍ୟା ବିଷୟରେ କ'ଣ? ଏବଂ ମିଶ୍ରିତ ଟାଇପ୍‌ର ତଥ୍ୟ ବିଷୟରେ କ'ଣ? ଉଦାହରଣ ସ୍ୱରୂପ, ଧରାଯାଉ ଯେ ଆମେ ଜଣେ ଏଜେଣ୍ଟଙ୍କ ବିଷୟରେ ସୂଚନା ଗଚ୍ଛିତ କରିବାକୁ ଚାହୁଁଛୁ ଯେଉଁଥିରେ ଏଜେଣ୍ଟଙ୍କ ନାମ (ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍), କୋଡ୍ ନମ୍ବର (ଏକ ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା), ଏବଂ ଉଚ୍ଚତା (ଏକ ବାସ୍ତବ ସଂଖ୍ୟା) କୁ ନେଇ ଗଠିତ । ଆମେ ଏଜେଣ୍ଟମାନଙ୍କର ତାଲିକା ପାଇଁ ଏକ ଡାଟା ଫାଇଲ୍ ସୃଷ୍ଟି କରିବାକୁ ଚାହୁଁଛୁ । ଏହା ସଂରୂପିତ I/O ଫଙ୍କ୍ସନ୍ ବ୍ୟବହାରକରି କରାଯାଇପାରିବ ।

ଫାଇଲରେ ଲେଖିବା

ଏକ ଫାଇଲ୍‌କୁ ସଂରୂପିତ ତଥ୍ୟ ଲେଖୁଥିବା ଫଙ୍କ୍ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
fprintf( fp, "format-string" , ditems );
```

ଯେଉଁଠାରେ *fp* ଏକ ଫାଇଲ୍ ପଏଁଟର୍, ଏବଂ *ditems* ଏକ ଫାଇଲ୍‌କୁ ଲେଖାଯିବାକୁ ଥିବା ଚଳଗୁଡ଼ିକର ଏକ ତାଲିକା । *fprintf()* ଫଙ୍କ୍ସନ୍ *printf()* ଫଙ୍କ୍ସନ୍ ସହିତ ସମାନ; ଏକମାତ୍ର ପାର୍ଥକ୍ୟ ହେଉଛି *printf()* ଫଙ୍କ୍ସନ୍‌ଟି ସ୍କ୍ରିନରେ ସଂରୂପିତ ତଥ୍ୟ ଲେଖେ ।

Listing 10.5

```
/*
Program to demonstrate writing of formatted data to a file
*/
#include<stdio.h>
 int main()
 {
    FILE *fp;
    char yes_no;
    char name[41];
    int code;
    float height;
    fp = fopen( "file3.dat", "w" );
    if ( fp == NULL ) {
       printf( "\nUnable to open file3.dat\n" );
       return 1;
    }
    while(1)
    {
       printf( "\nEnter name, code number, height" );
```

```
        scanf( "%s,%d,%f", name, &code, &height );
        fprintf( fp,"%s,%d,%f", name, code, height );
        printf( "Any more input y/n?: " );
        yes_no = tolower( getche() );
        fflush ( stdin );
        if ( yes_no == 'n' )
           break;
    }
    fclose( fp );
return 0;
}
```

Test Run

```
Enter name, code number, height
Geetu, 10, 160.25
Any more input y/n?: y
Enter name, code number, height
Shivani, 11, 158.5
Any more input y/n?: y
Enter name, code number, height
Vijay, 12, 165.5
Any more input y/n?: y
Enter name, code number, height
Gurpreet, 13, 166.25
Any more input y/n?: n
```

ଉପରୋକ୍ତ ସୂଚନା ବର୍ତ୍ତମାନ *file3.dat* ନାମକ ଏକ ଫାଇଲରେ ଅଛି । ଯଦି ଆମେ TYPE କମାଣ୍ଡ ବ୍ୟବହାର କରି ଏହାକୁ ଦେଖିବାକୁ ଚେଷ୍ଟା କରୁ, ତାହେଲେ ଡାଟାରେ କୌଣସି ନୂଆ ଲାଇନ୍ ନ ଥିବାରୁ ସମଗ୍ର ଆଉଟପୁଟ୍ ସମାନ ଲାଇନରେ ରହିବ । ଅଧିକ ସୁବିଧାଜନକ ଭାବରେ ଆଉଟପୁଟ୍‌କୁ ସଂରୂପ କରିବାକୁ, ଆମେ ସଂରୂପିତ ଇନପୁଟ୍ ବ୍ୟବହାର କରି ଉପରୋକ୍ତ ଫାଇଲ୍ ପଢିବା ପାଇଁ ବିଶେଷ ଭାବରେ ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖିପାରିବା ।

ଫାଇଲରୁ ପଢ଼ିବା

ଏକ ଫାଇଲରୁ ସଂରୂପିତ ତଥ୍ୟ ପଠନ କରୁଥିବା ଫଙ୍କ୍‌ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
fscanf( fp, "format-string" , ditems );
```

ଯେଉଁଠାରେ *fp* ଏକ ଫାଇଲ୍ ପଏଁଟର୍, ଏବଂ *ditems* ଠିକଣାଗୁଡିକର ଏକ ତାଲିକା ଯେଉଁଠାରେ ଏକ ଫାଇଲରୁ ପଢିଥିବା ମୂଲ୍ୟଗୁଡିକ ଗଚ୍ଛିତ କରିବାକୁ ହେବ ।

Listing 10.6

```
/*
    Program to demonstrate writing of formatted from a file
 */

 #include<stdio.h>

 int main()
 {
    FILE *fp;
    char yes_no, char name[41];
    int code;
    float height;
    fp = fopen( "file3.dat", "r" );
    if ( fp == NULL ) {
       printf( "\nUnable to open file3.dat\n" );
       return 1;
    }
    printf( "\nContens of file are:\n\n" );
    while( fscanf(fp,"%s,%d,%f", name, &code, &height) != EOF ) {
       printf( "%s %d %f\n", name, code, height );
    }
    fclose( fp );
    return 0;
 }
```

Test Run

```
Contents of file are:
Geetu 10 160.25
Shivani 11 158.5
Vijay 12 165.5
Gurpreet 13 166.25
```

10.3.4 ବାଇନାରି ଫାଇଲର ପଠନ ଏବଂ ଲେଖନ

C ଭାଷା ରେକର୍ଡ଼ I/O ପ୍ରଦାନ କରିଥାଏ, ଏହାକୁ ବେଳେବେଳେ ବ୍ଲକ୍ I/O ମଧ୍ୟ କୁହାଯାଏ, ଫଙ୍କ୍ସନ୍‌ଗୁଡିକ ବାଇନାରି ଫାଇଲଗୁଡିକର ତଥ୍ୟ ପଢିବା ଏବଂ ଲେଖିବାପାଇଁ କାର୍ଯ୍ୟ କରେ ।

ରେକର୍ଡ଼ I/O ସଂଖ୍ୟାଗୁଡ଼ିକୁ ବାଇନାରି ସଂରୂପରେ ଡିସ୍କ ଫାଇଲରେ ଲେଖେ ଯେପରିକି ଇଣ୍ଟିଜର୍ ଦୁଇଟି ବାଇଟ୍‌ରେ ଗଚ୍ଛିତ ହେବ; ଲଙ୍ଗ ଇଣ୍ଟିଜର୍ ଚାରୋଟି ବାଇଟ୍, ଚାରୋଟି ବାଇଟ୍‌ରେ ଏକକ-ସଠିକତା ଫ୍ଲୋଟିଂ-ପଏଣ୍ଟ ନମ୍ବର, ଏବଂ ଆଠଟି ବାଇଟ୍‌ରେ ଡବଲ୍-ପ୍ରିସିସନ୍ ଫ୍ଲୋଟିଂ-ପଏଣ୍ଟ ନମ୍ବରରେ ଗଚ୍ଛିତ ହୋଇଛି – ମେମୋରିରେ ସଂଖ୍ୟାତ୍ମକ ତଥ୍ୟ ଗଚ୍ଛିତ କରିବାକୁ ସମାନ ସଂରୂପ ବ୍ୟବହୃତ ହୁଏ ।

ରେକର୍ଡ I/O ଏକକାଳୀନ ତଥ୍ୟ ପଢିବା ଏବଂ ଲେଖିବାପାଇଁ ମଧ୍ୟ ଅନୁମତି ଦେଇଥାଏ; ଏହି ପ୍ରକ୍ରିୟା କେବଳ ଗୋଟିଏ ଅକ୍ଷର କିମ୍ବା ଷ୍ଟ୍ରିଙ୍ଗ୍ କିମ୍ବା କିଛି ମୂଲ୍ୟରେ ସୀମିତ ନୁହେଁ । ଆରେ, ଷ୍ଟ୍ରକ୍‌ଚର୍, ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଆରେ ମଧ୍ୟ ଏକ ୟୁନିଟ୍ ଭାବରେ ପଢା ଏବଂ ଲେଖାଯାଇପାରିବ ।

ଫାଇଲ୍‌ରେ ଲେଖିବା

ଏକାଥରକେ ଡାଟାର ଏକ ବ୍ଲକ୍ ଲେଖୁଥିବା ଫଙ୍କସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
fwrite( ptr, m, n, fp );
```

ଯେଉଁଠାରେ *ptr* ହେଉଛି ଲେଖାଯିବାକୁ ଥିବା ଏକ ଆରେ ବା ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଠିକଣା, *m* ହେଉଛି ଆରେ ବା ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଆକାର, *n* ହେଉଛି କେତୋଟି ଆରେ କିମ୍ବା ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଲେଖାଯିବ, ଏବଂ *fp* ହେଉଛି ବାଇନାରି ମୋଡରେ ଲେଖିବା ପାଇଁ ଖୋଲାଯାଇଥିବା ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ ।

ବ୍ଲକଟି ଲେଖିବା ପରେ, *fwrite()* ଫଙ୍କସନ୍ ଲିଖିତ ଡାଟା ଉପାଦାନର ପ୍ରକୃତ ସଂଖ୍ୟା ଫେରସ୍ତ କରେ । ଯଦି ଲିଖିତ ଉପାଦାନର ସଂଖ୍ୟା ଅନୁରୋଧ କରା ଯାଇଥିବା ସଂଖ୍ୟାଠାରୁ କମ୍, ଏହାର ଅର୍ଥ କିଛି ତ୍ରୁଟି ଘଟିଛି ।

ଆରେ ଲେଖିବା

ଧରାଯାଉ ଆମେ file4.dat ନାମକ ଏକ ଫାଇଲ୍‌ରେ 10 ଟି ଉପାଦାନ ଥିବା ଏକ ଇଣ୍ଟିଜର୍ ଆରେ ଗଚ୍ଛିତ କରିବାକୁ ଚାହୁଁଛୁ ।

Listing 10.7

```
/*
    Program to demonstrate writing of an entire array to a file
 */

 # include <stdio.h>
 int main()
 {
    FILE *fp;
    int i, a[10];
    fp = fopen( "file4.dat", "wb" );
    if ( fp == NULL ) {
       printf( "\nUnable to open file4.dat\n" );
       return 1;
    }
    printf( "\nEnter ten values\n" );
    for ( i = 0; i <= 10; i++ )
        scanf( "%d", &a[i] );
    fwrite(a, sizeof(a), 1, fp); /* write entire array to file */
    fclose( fp );
    return 0;
 }
```

ଏହି ପ୍ରୋଗ୍ରାମଟି ଚାଳନ ହେବା ପରେ, ବ୍ୟବହାରକାରୀଙ୍କୁ ଦଶଟି ଇଣ୍ଟିଜର୍ ମୂଲ୍ୟ ଇନ୍‌ପୁଟ୍ କରିବାକୁ ପ୍ରେରିତ କରିବ ଏବଂ ତା'ପରେ *file4.dat* ନାମକ ଡିସ୍କ ଫାଇଲ୍‌ରେ ପୁରା ଆରେଟି ଲେଖାହେବ ।

ଷ୍ଟ୍ରକ୍ଚର୍ ଲେଖିବା

ଧରାଯାଉ agent ନାମକ ଏକ ଷ୍ଟ୍ରକ୍ଚର୍‌କୁ ଯାହାର ଉପାଦାନଗୁଡ଼ିକ - ନାମ (ସର୍ବାଧିକ 40 ଟି ଅକ୍ଷର), କୋଡ୍ (ସର୍ବାଧିକ 5 ଟି ଅକ୍ଷର), ଏବଂ ଉଚ୍ଚତା (ବାସ୍ତବ ସଂଖ୍ୟା) ଆମେ *file5.dat* ନାମକ ଏକ ଡିସ୍କ ଫାଇଲ୍‌ରେ ଲେଖିବାକୁ ଚାହୁଁଛୁ ।

Listing 10.8

```
/*
    Program to demonstrate writing of an entire structure to a file
 */
 #include<stdio.h>

 struct
 {
    char name[41];
    char code[6];
    float height;
 } agent;

 int main()
 {
    FILE *fp;
    char yes_no, numstr[40];
    fp = fopen( "file5.dat", "wb" );
    if ( fp == NULL ) {
       printf( "\nUnable to open file5.dat\n" );
       return 1;
    }
    while ( 1 ) {
       printf( "\nEnter name : " );
       gets( agent.name );
       printf( "\nEnter code number : " );
       gets( agent.code );
       printf( "\nEnter height : " );
       gets( numstr );
       agent.height = atof ( numstr );
       fwrite( &agent, sizeof(agent), 1, fp );
       printf( "Any more input y/n?: " );
       yes_no = tolower( getche() );
       fflush( stdin );
       if ( yes_no == 'n' )
           break;
    }
    fclose( fp );
    return 0;
 }
```

ଏହି ପ୍ରୋଗ୍ରାମଟି ଚାଳନ ହେବା ପରେ, ବ୍ୟବହାରକାରୀଙ୍କୁ ଏକ *agent* ର ବିବରଣୀ ଇନ୍‌ପୁଟ୍ କରିବାକୁ ପ୍ରେରିତ କରିବ, ସେଗୁଡ଼ିକୁ ଏକ ଷ୍ଟ୍ରକ୍‌ଚର୍‌ରେ ଗଚ୍ଛିତ କରିବ, ଏବଂ ତା'ପରେ ଗୋଟିଏ *write* ଅପରେସନ୍‌ଦ୍ୱାରା *file5.dat* ନାମକ ଡିସ୍କ ଫାଇଲ୍‌ରେ ଏହି ଷ୍ଟ୍ରକ୍‌ଚର୍‌କୁ ଲେଖେ।

ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଏକ ଚଳ ପରିବର୍ତ୍ତେ, ଏକ ଏଜେଣ୍ଟ ହେଉଛି *n* ଉପାଦାନ ବିଶିଷ୍ଟ ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଏକ ଆରେ, ଏବଂ ଆମେ ଗୋଟିଏ write ଅପରେସନ୍‌ରେ ସମଗ୍ର ଆରେ ଲେଖିବାକୁ ଚାହୁଁଛୁ।

ଏହି କାର୍ଯ୍ୟ *fwrite()* ଫଙ୍କସନ୍‌ଦ୍ୱାରା ସମ୍ପାଦନ କରାଯାଇପାରିବ ଯେପରି

```
fwrite( agent, sizeof(agent[0]), n, fp );
```

ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମ ସଂଖ୍ୟାତ୍ମକ ଡାଟା ଇନପୁଟ୍ କରିବାର ଅନ୍ୟ ଏକ ଉପାୟ ପ୍ରଦର୍ଶନ କରେ। ସଂଖ୍ୟାତ୍ମକ ଡାଟାକୁ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ଭାବରେ ମଧ୍ୟ ପଢ଼ାଯାଇପାରେ ଏବଂ ପରେ ଉପଯୁକ୍ତ ମୂଲ୍ୟରେ ରୂପାନ୍ତରିତ କରା ଯାଇପାରେ। ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ atoi() ଏକ ଇଣ୍ଟିଜର୍ ଷ୍ଟ୍ରିଙ୍ଗ୍‌କୁ ଏକ ଇଣ୍ଟିଜର୍ ମୂଲ୍ୟରେ ରୂପାନ୍ତରିତ କରେ, ଯେତେବେଳେ କି ଫଙ୍କସନ୍ atof() ଏକ ବାସ୍ତବ ସଂଖ୍ୟା ଷ୍ଟ୍ରିଙ୍ଗକୁ ଏକ ବାସ୍ତବ ସଂଖ୍ୟାରେ ରୂପାନ୍ତରିତ କରେ। ସେମାନଙ୍କର ପ୍ରତିପକ୍ଷଗୁଡ଼ିକ ହେଉଛି itoa() ଏବଂ fcvt(), ଯାହା ଏକ ଇଣ୍ଟିଜର୍ ମୂଲ୍ୟ ଏବଂ ବାସ୍ତବ ମୂଲ୍ୟକୁ ଉପଯୁକ୍ତ ଷ୍ଟ୍ରିଙ୍ଗରେ ରୂପାନ୍ତର କରେ। ଏହି ଫଙ୍କସନ୍‌ଗୁଡ଼ିକ stdlib.h ହେଡର୍ ଫାଇଲଗୁଡ଼ିକରେ ବ୍ୟାଖ୍ୟା କରାଯାଇଛି। ଏଠାରେ ପ୍ରଶ୍ନ ଉଠିପାରେ ¾ ଏହି ରୂପାନ୍ତରଣଗୁଡ଼ିକ କାହିଁକି ଆବଶ୍ୟକ ?

ଏହାର ଦୁଇଟି କାରଣ ଅଛି :

- ବେଳେବେଳେ ଆମକୁ ଏକ ଷ୍ଟ୍ରିଙ୍ଗ୍ ସହିତ ଏକ ସଂଖ୍ୟାତ୍ମକ ମୂଲ୍ୟକୁ ସଂଯୁକ୍ତ କରିବାକୁ ପଡ଼ିପାରେ, ଏବଂ ଏହା କେବଳ ସମ୍ଭବ ହେବ ଯଦି ସଂଖ୍ୟାତ୍ମକ ମୂଲ୍ୟଟି ଏକ ଷ୍ଟ୍ରିଙ୍ଗରେ ରୂପାନ୍ତରିତ ହୁଏ।
- C ସିଷ୍ଟମରେ *scanf()* ଫଙ୍କସନ୍ ଦ୍ୱାରା ବାସ୍ତବ ସଂଖ୍ୟାର ଇନପୁଟ୍ ଅତ୍ୟନ୍ତ ଅସଙ୍ଗତ। ବେଳେବେଳେ ଏହା ଏକ ତ୍ରୁଟି ସନ୍ଦେଶ ଦେଇଥାଏ – *floating point format not linked.*

ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମରେ ବ୍ୟବହୃତ ଅନ୍ୟ ଏକ ଫଙ୍କସନ୍ ହେଉଛି *fflush()* ଫଙ୍କସନ୍। କୀବୋର୍ଡ ବଫର୍‌ରେ ଛାଡ଼ି ହୋଇଯାଇଥିବା ଅବାଞ୍ଛିତ ଡାଟାକୁ ଫ୍ଲସ୍ କରିବାପାଇଁ ଏହି ଫଙ୍କସନ୍ ବ୍ୟବହୃତ ହୁଏ। ତୁମେ ହୁଏତ ଦେଖିଥିବ ଯେ ବେଳେବେଳେ କିଛି ଇନପୁଟ୍ ଫଙ୍କସନ୍ କଲ୍‌ରେ ସିଷ୍ଟମ୍ ଏକ ନିର୍ଦ୍ଦିଷ୍ଟ ଇନପୁଟ୍‌ପାଇଁ ଅଟକି ନ ଥାଏ। ଏହାର କାରଣ କିଛି ଅବାଞ୍ଛିତ ଡାଟା କୀବୋର୍ଡ ବଫର୍‌ରେ ରହିଥାଏ ଯାହାକୁ ସେହି ଇନପୁଟ୍‌ପାଇଁ ଆପେଆପେ ନେଇଯାଏ। ତେଣୁ ଆମେ *fflush()* ଫଙ୍କସନ୍ ବ୍ୟବହାର କରି ପରବର୍ତ୍ତୀ ଇନପୁଟ୍ ନେବା ପୂର୍ବରୁ ଏହାକୁ ଫ୍ଲସ୍ କରିପାରିବା।

ଫାଇଲରୁ ପଢ଼ିବା

ଏକ ଫାଇଲରୁ ଏକ ବ୍ଲକ୍ ପଠନ କରୁଥିବା ଫଙ୍କସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
fread( ptr, m, n, fp );
```

ଯେଉଁଠାରେ *ptr* ଏକ ଆରେ ବା ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଠିକଣା ଯେଉଁଠି ପଢ଼ିବା ପରେ ବ୍ଲକ୍‌ଟି ଗଚ୍ଛିତ ହେବ, *m* ହେଉଛି ଆରେ ବା ଷ୍ଟ୍ରକ୍‌ଚର୍‌ର ଆକାର, *n* ହେଉଛି କେତୋଟି ଆରେ ବା ଷ୍ଟ୍ରକ୍‌ଚର୍‌କୁ ପଢ଼ାହେବ, ଏବଂ *fp* ହେଉଛି ପଢ଼ିବା ପାଇଁ ବାଇନାରି ମୋଡରେ ଖୋଲାଯାଇଥିବା ଫାଇଲ୍‌ର ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ।

fread() ଫଙ୍କସନ୍‌ଟି ପଢାଯାଇଥିବା ଡାଟା ଉପାଦାନର ପ୍ରକୃତ ସଂଖ୍ୟା ଫେରସ୍ତ କରେ। ଏହି ଫଙ୍କସନ୍‌ର ବ୍ୟବହାର ନିମ୍ନଲିଖିତ ପ୍ରୋଗ୍ରାମରେ ଦର୍ଶାଯାଇଛି।

Listing 10.9

```
/*
    Program to demonstrate reading of entire structure from a file
 */

 #include<stdio.h>

 struct
 {
    char name[41];
    char code[6];
    float height;
 } agent;

 int main()
 {
    FILE *fp;
    int record_no = 0;
    fp = fopen( "file5.dat", "rb" );
    if ( fp == NULL )
    {
       printf( "\nUnable to open file5.dat\n" );
       return 1;
    }

    while ( fread(&agent,sizeof(agent),1,fp) > 0 )
    {
       printf( "\nRecord #%d\n", record_no );
       printf( "\nName : %s", agent.name );
       printf( "\nCode number : %s", agent.code );
       printf( "\nHeight : %.2f", agent.height );
       record_no++;
       printf("\n\nPress any key to see next record..." );
       getch();
    }
    fclose( fp );
    return 0;
 }
```

ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମଟି 10.8 ରେ ତାଲିକାଭୁକ୍ତ ପ୍ରୋଗ୍ରାମରେ ତିଆରି ହୋଇଥିବା ଡାଟା ଫାଇଲ୍‌କୁ ପଢ଼େ। ଏହା ଏକାଥରରେ ଗୋଟିଏ ଷ୍ଟ୍ରକ୍‌ଚର୍ ପଢେ ଏବଂ ଏହାକୁ ସ୍କ୍ରିନରେ ପ୍ରଦର୍ଶନ କରେ। ଯେତେବେଳେ fread() ଫଙ୍କ୍‌ସନ୍‌ଟି ମୂଲ୍ୟ 0 ଫେରସ୍ତ କରେ ସେତେବେଳେ while ଲୁପ୍ ସମାପ୍ତ ହୁଏ, ଯାହାର ଅର୍ଥ ଏହା ଏକ ବ୍ଲକ୍ ପଢିପାରିବ ନାହିଁ। ଏହା ଫାଇଲ୍‌ର ଶେଷକୁ ସୂଚିତ କରେ।

10.4 ଫାଇଲ୍ ସ୍ଥିତି ଫଙ୍କ୍ସନ୍(FILE POSITIONING FUNCTIONS)

ଫାଇଲ୍ ସ୍ଥିତି ଅପରେସନ୍‌ଗୁଡ଼ିକର ପରିଚାଳନାପାଇଁ C ଭାଷା ନିମ୍ନଲିଖିତ ଫଙ୍କ୍ସନ୍‌ଗୁଡିକ ଯୋଗାଇଥାଏ:

- *rewind()* ଫଙ୍କ୍ସନ୍, ଫାଇଲ୍ ଆରମ୍ଭରେ ଫାଇଲ୍ ପଏଁଟର୍‌କୁ ସେଟ୍ କରିବାକୁ ।
- *ftell()* ଫଙ୍କ୍ସନ୍, ଫାଇଲ୍‌ରେ ଫାଇଲ୍ ପଏଁଟର୍‌ର ସମ୍ପ୍ରତି ସ୍ଥିତି ଜାଣିବା ପାଇଁ ।
- *seek()* ଫଙ୍କ୍ସନ୍, ଫାଇଲ୍‌ରେ ଫାଇଲ୍ ପଏଁଟର୍‌ର ସ୍ଥିତି ପରିବର୍ତ୍ତନ କରିବାକୁ ।

10.4.1 ରିୱାଇଣ୍ଡ ଫାଇଲ୍: rewind() ଫଙ୍କ୍ସନ୍ (Rewind File: rewind() Function)

ଫାଇଲ୍ ପଏଁଟର୍‌କୁ ଫାଇଲ୍ ଆରମ୍ଭରେ ରଖିବାର ଗୋଟିଏ ଉପାୟ ହେଉଛି ଫାଇଲ୍‌କୁ ବନ୍ଦକରି ପୁଣିଥରେ ଏହାକୁ ଖୋଲିବା । ତାହା ସତ୍ତ୍ୱେ, ଆମେ rewind() ଫଙ୍କ୍ସନ୍ ବ୍ୟବହାର କରି ଫାଇଲ୍ ବନ୍ଦ ନକରି ମଧ୍ୟ ସମାନ କାର୍ଯ୍ୟ କରିପାରିବା । ଏହି ଫଙ୍କ୍ସନ୍ ଫାଇଲ୍ ଆରମ୍ଭରେ ଫାଇଲ୍ ପଏଁଟର୍‌କୁ ରଖିପାରେ । ଏହି ଫଙ୍କ୍ସନ୍ ଆରମ୍ଭରୁ କ୍ୟାସେଟ୍ ଶୁଣିବା କିମ୍ବା ଦେଖିବା ପାଇଁ ଅଡିଓ କିମ୍ବା ଭିଡିଓ କ୍ୟାସେଟ୍ ରିୱାଇଣ୍ଡ କରିବା ପରି ଲାଗେ ।

rewind() ଫଙ୍କ୍ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
rewind(fp);
```

ଯେଉଁଠାରେ *fp* ହେଉଛି ସମ୍ପ୍ରତି ଖୋଲାଯାଇଥିବା ଫାଇଲ୍‌ପାଇଁ ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ ।

10.4.2 ସାମ୍ପ୍ରତିକ ଅବସ୍ଥାନ: ftell() ଫଙ୍କ୍ସନ୍ (Current Location: ftell() Function)

କିଛି ପରିସ୍ଥିତିରେ, ଫାଇଲ୍ ମଧ୍ୟରେ ଫାଇଲ୍ ପଏଁଟର୍‌ର ସାମ୍ପ୍ରତିକ ଅବସ୍ଥାନ ଜାଣିବାର ଆବଶ୍ୟକ ହୋଇପାରେ । *ftell()* ଫଙ୍କ୍ସନ୍ ଆମକୁ ଫାଇଲ୍ ପଏଣ୍ଟରର ସାମ୍ପ୍ରତିକ ସ୍ଥିତି ଜାଣିବାକୁ ଦିଏ ।

ftell() ଫଙ୍କ୍ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
long k = ftell(fp);
```

ଯେଉଁଠାରେ *fp* ହେଉଛି ସମ୍ପ୍ରତି ଖୋଲାଯାଇଥିବା ଫାଇଲ୍‌ପାଇଁ ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ, ଉଲ୍ଲେଖଯୋଗ୍ୟ ଯେ *ftell()* ଫଙ୍କ୍ସନ୍ ଏକ ଲଙ୍ଗ ଇଣ୍ଟିଜର୍ ଫେରସ୍ତ କରେ । ଏହା ଆବଶ୍ୟକ କାରଣ ଅନେକ ଫାଇଲ୍‌ରେ 32767 ରୁ ଅଧିକ ବାଇଟ୍ ଡାଟା ଥାଇପାରେ ।

ମନେ ପକାଅ ଯେ C I/O ସିଷ୍ଟମ୍ ଫାଇଲଗୁଡିକୁ ଡାଟା ବାଇଟ୍‌ଗଡ଼ିକର ଏକ ଷ୍ଟ୍ରିମ୍ ଭାବରେ ବିବେଚନା କରେ । ଏହା ଶୂନ୍‌ରୁ ଆରମ୍ଭକରି ବାଇଟ୍ ସଂଖ୍ୟାଦ୍ୱାରା ଫାଇଲ୍‌ରେ ସ୍ଥିତି ମାପିଥାଏ, ଅର୍ଥାତ, ଫାଇଲ୍ ଆରମ୍ଭରେ ବାଇଟ୍ ସଂଖ୍ୟା ହେଉଛି ଶୂନ୍ । ଫାଇଲ୍ ପଏଁଟର୍ ଫାଇଲ୍ ଆରମ୍ଭରେ ଥିବା ବେଳେ, *ftell()* ଫଙ୍କ୍ସନ୍ 0 ଫେରସ୍ତ କରେ । ଯଦି ଫାଇଲ୍ ପଏଁଟର୍ ଦ୍ୱିତୀୟ ବାଇଟ୍‌ରେ ଥାଏ, ତେବେ *ftell()* ଫଙ୍କ୍ସନ୍ ମୂଲ୍ୟ 1 ଫେରସ୍ତ କରେ ।

10.4.3 ନିର୍ଦ୍ଦିଷ୍ଟ ସ୍ଥାନରେ ରଖିବା: fseek() ଫଙ୍କ୍ସନ୍ (Set Position: fseek() Function)

ଏକ ଫାଇଲ୍‌ରେ ଯେକୌଣସି ସ୍ଥାନରୁ ଏକ ଡାଟା ଉପାଦାନ ପଢିବାକୁ, ଆମକୁ ଫାଇଲ୍ ପଏଁଟର୍‌କୁ ସେହି ଡାଟା ଉପାଦାନର ଆରମ୍ଭକୁ ଘୁଞ୍ଚାଇବାକୁ ପଡିବ । ଏହି କାର୍ଯ୍ୟ ପାଇଁ, ଆମେ *fseek()* ଫଙ୍କ୍ସନ୍ ବ୍ୟବହାର କରିପାରିବା ।

fseek () ଫଙ୍କ୍ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
fseek(fp, offset, wherefrom );
```

fseek () ଫଙ୍କସନ୍ ତିନୋଟି ଚର ନେଇଥାଏ, ଯେଉଁଠାରେ ପ୍ରଥମ ଚର *fp* ହେଉଛି ଏକ ଫାଇଲ୍ ପଏଁଟର୍, ଦ୍ୱିତୀୟ ଚର offset ହେଉଛି ଏକ ଲଙ୍ଗ ଇଣ୍ଟିଜର୍ ପ୍ରକାରର ଚଳ, ଏହା କେତେ ବାଇଟ୍ ଫାଇଲ୍ ପଏଁଟର୍‌କୁ ଘୁଞ୍ଚାଯିବ ସେହି ସଂଖ୍ୟା ନିର୍ଦ୍ଦିଷ୍ଟ କରେ । ତୃତୀୟ ଚର *wherefrom* କେଉଁ ସ୍ଥିତିରୁ *offset* ମପାଯିବ ତାହା ନିର୍ଦ୍ଦିଷ୍ଟ କରେ ।

wherefrom ଚର ପାଇଁ ବିଭିନ୍ନ ମୂଲ୍ୟଗୁଡିକ ଟେବୁଲ୍ 10.3 ରେ ତାଲିକାଭୁକ୍ତ ହୋଇଛି ।

ଟେବୁଲ୍ 10.3: *fseek()* ଫଙ୍କସନ୍ ପାଇଁ *wherefrom*ର ବିଭିନ୍ନ ମୂଲ୍ୟ

ମୋଡ୍	ଅଫସେଟ୍ ରୁ ମାପ କରାଯାଏ
SEEK_BET	ଫାଇଲ୍‌ର ଆରମ୍ଭ
SEEK_CUR	ଫାଇଲ୍‌ର ସମ୍ପ୍ରତି ସ୍ଥିତି
SEEK_END	ଫାଇଲ୍‌ର ଶେଷ

ଟେବୁଲ୍ 10.4: *fseek ()* ଫଙ୍କସନ୍‌ର ବ୍ୟବହାରକୁ ବର୍ଣ୍ଣନା କରୁଥିବା କିଛି ଉଦାହରଣ ।

କଲ୍ ଖୋଜନ୍ତୁ	କାର୍ଯ୍ୟ ସମ୍ପାଦିତ
`fseek(fp, 0, EEK_BET);`	ଫାଇଲ୍ ପଏଁଟର୍ fptr କୁ ଫାଇଲ୍ ଆରମ୍ଭ ପର୍ଯ୍ୟନ୍ତ ଘୁଞ୍ଚାଏ । ଯଦି ଫାଇଲ୍ ପଏଁଟର୍ ବର୍ତ୍ତମାନ ଫାଇଲ୍‌ର ଆରମ୍ଭରେ ଅଛି, ତେବେ ଏହା କୌଣସି କାର୍ଯ୍ୟ କରେ ନାହିଁ ।
`fseek(fp, n, SEEK_BET);`	ଫାଇଲ୍ ପଏଁଟର୍ fptr ଦ୍ୱାରା ଆଗକୁ n ବାଇଟ୍ ଘୁଞ୍ଚାଏ, ଅର୍ଥାତ, ଫାଇଲ୍ ପଏଁଟର୍‌କୁ ଫାଇଲ୍‌ରେ (n+1) ବାଇଟ୍ ଘୁଞ୍ଚାଏ ।
`fseek(fp, -n, SEEK_CUR);`	ସାମ୍ପ୍ରତିକ ସ୍ଥିତିରୁ n ବାଇଟ୍ ପର୍ଯ୍ୟନ୍ତ ଫାଇଲ୍ ପଏଁଟର୍ fptr କୁ ପଛକୁ ଘୁଞ୍ଚାଏ ।
`fseek(fp, 0, SEEK_END);`	ଫାଇଲ୍ ପଏଁଟର୍ fptr କୁ ଫାଇଲ୍ ଶେଷ ପର୍ଯ୍ୟନ୍ତ ଘୁଞ୍ଚାଏ । ଯଦି ଫାଇଲ୍ ପଏଁଟର୍ ବର୍ତ୍ତମାନ ଫାଇଲ୍ ଶେଷରେ ଅଛି, ତେବେ ଏହା କୌଣସି କାର୍ଯ୍ୟ କରେ ନାହିଁ ।।
`fseek(fp, 0, SEEK_END);`	ସାମ୍ପ୍ରତିକ ସ୍ଥିତିରୁ n ବାଇଟ୍ ପର୍ଯ୍ୟନ୍ତ ଫାଇଲ୍ ପଏଣ୍ଟର୍ fptr କୁ ଆଗକୁ ଘୁଞ୍ଚାଏ ।
`fseek(fp, n, SEEK_CUR);`	ଫାଇଲ୍ ପଏଁଟର୍ fptr କୁ ପରବର୍ତ୍ତୀ ବାଇଟ୍‌କୁ ଘୁଞ୍ଚାଏ ।
`fseek(fp, 1, SEEK_CUR);`	ଫାଇଲ୍ ପଏଁଟର୍ fptrକୁ ପୂର୍ବ ବାଇଟକୁ ଘୁଞ୍ଚାଏ ।
`fseek(fp, -1, SEEK_CUR);`	ଫାଇଲ୍ ପଏଁଟର୍ fptr କୁ ଫାଇଲ୍‌ର ଶେଷରୁ m ବାଇଟ୍ ପଛକୁ ଘୁଞ୍ଚାଏ ।
`fseek(fp, m, SEEK_END);`	iQkby ikWbaVj fptr dks iQkby osQ var ls m ckbV~l }kjk ihNs dh vksj ys tkrk gSA

10.5 ଫାଇଲ୍ ଅବସ୍ଥା ଫଙ୍କସନ୍(FILE STATUS FUNCTIONS)

ଫାଇଲ ଅବସ୍ଥା ଅନୁସନ୍ଧାନଗୁଡ଼ିକର ପରିଚାଳନାପାଇଁ C ଭାଷା ନିମ୍ନଲିଖିତ ଫଙ୍କସନ୍‌ଗୁଡ଼ିକ ଯୋଗାଇଥାଏ:

- *feof()* ଫଙ୍କସନ୍, ଫାଇଲ୍‌ର ଶେଷ ପରୀକ୍ଷା କରିବାକୁ ବ୍ୟବହାର ହୋଇଥାଏ ।

- *ferror()* ଫଙ୍କ୍‌ସନ୍, ଫାଇଲ୍‌ର ତ୍ରୁଟି ପରୀକ୍ଷଣପାଇଁ ବ୍ୟବହାର ହୋଇଥାଏ ।
- *clearerr()* ଫଙ୍କ୍‌ସନ୍, ଫାଇଲ୍‌କୁ ତ୍ରୁଟିମୁକ୍ତ କରିବାପାଇଁ ବ୍ୟବହାର ହୋଇଥାଏ ।

10.5.1 ଫାଇଲ୍‌ର ଶେଷ ପରୀକ୍ଷା କରିବା: feof() ଫଙ୍କ୍‌ସନ୍ (Test End of File: feof() Function)

ଫାଇଲ୍ ପଏଁଟର୍ ଫାଇଲ୍‌ର ଶେଷରେ ପହଞ୍ଚିଛି କି ନାହିଁ ଯାଞ୍ଚ କରିବାକୁ feof() ଫଙ୍କ୍‌ସନ୍ ବ୍ୟବହାର କରାଯାଏ । ଯଦି ଫାଇଲ୍ ପଏଁଟର୍ ଶେଷରେ ଅଛି, ଅର୍ଥାତ, ସମସ୍ତ ଡାଟା ପଢିସାରିଛି, ତେବେ ଉକ୍ତ ଫଙ୍କ୍‌ସନ୍ 1 ଫେରସ୍ତ କରେ । ଯଦି ଫାଇଲ୍‌ର ଶେଷରେ ପହଞ୍ଚି ନଥାଏ, ତେବେ ଏହା ମୂଲ୍ୟ 0 ଫେରସ୍ତ କରେ ।

feof() ଫଙ୍କ୍‌ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି:

```
feof(fp);
```

ଯେଉଁଠାରେ *fp* ହେଉଛି ସମ୍ପ୍ରତି ଖୋଲାଥିବା ଫାଇଲ୍‌ପାଇଁ ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ।

10.5.2 ଫାଇଲ୍‌ର ତ୍ରୁଟି ପରୀକ୍ଷଣ: ferror() ଫଙ୍କ୍‌ସନ୍ (Test Error: ferror() Function)

ଫାଇଲ୍‌ର ତ୍ରୁଟିର ଅବସ୍ଥିତି ପରୀକ୍ଷା କରିବାକୁ ferror() ଫଙ୍କ୍‌ସନ୍ ବ୍ୟବହୃତ ହୁଏ । ଯେପରି କି ପୂର୍ବରୁ ଉଲ୍ଲେଖ ହୋଇଛି, ଖରାପ ମାଧ୍ୟମ (ଡିସ୍କ, ସିଡି ଇତ୍ୟାଦି) ଠାରୁ ଆରମ୍ଭ କରି ଅବୈଧ କାର୍ଯ୍ୟ ଯେପରିକି ଏକ ଫାଇଲ୍‌ରେ ଲେଖିବାବେଳେ ସେଥିରୁ ପଢିବା ପର୍ଯ୍ୟନ୍ତ ଅନେକ କାରଣରୁ ତ୍ରୁଟି ସୃଷ୍ଟି ହୋଇପାରେ ।

ଧରାଯାଉ ଯଦି ଏକ ଫାଇଲ୍ ଅପରେସନ୍ ପରେ ଏକ ତ୍ରୁଟି ଘଟିଛି ତେବେ ferror() ଫଙ୍କ୍‌ସନ୍ 1 ଫେରସ୍ତ କରେ । ଯଦି କୌଣସି ତ୍ରୁଟି ହୋଇନାହିଁ, ferror() ମୂଲ୍ୟ 0 ଫେରସ୍ତ କରେ ।

ferror() ଫଙ୍କ୍‌ସନ୍‌ର ସିଣ୍ଟାକ୍ସ ହେଉଛି

```
ferror(fp);
```

ଯେଉଁଠାରେ *fp* ବର୍ତ୍ତମାନ ଖୋଲାଯାଇଥିବା ଫାଇଲ୍ ପାଇଁ ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ ।

ଏହା ଉଲ୍ଲେଖ କରିବା ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଯେ ଏକ ତ୍ରୁଟି ପରୀକ୍ଷଣ ତ୍ରୁଟି ଅବସ୍ଥାକୁ ସୁଧାରେ ନାହିଁ । ଥରେ ଏକ ତ୍ରୁଟି ଘଟିବା ପରେ, ପରବର୍ତ୍ତୀ ବର୍ଣ୍ଣିତ *clearer()* ଫଙ୍କ୍‌ସନ୍ ବ୍ୟବହାର କରି ତ୍ରୁଟି ସ୍ଥିତିକୁ ଖାଲି କରିବା ପରେ ଏହା ଏକ ସାଧାରଣ ପଠନ କିମ୍ବା ଲିଖନ ଅବସ୍ଥାକୁ ଫେରିପାରିବ ।

13.5.3 ତ୍ରୁଟିମୁକ୍ତ କରିବା: clearerr() ଫଙ୍କ୍‌ସନ୍ (Clear Error: clearerr() Function)

ଯେତେବେଳେ ଏକ ତ୍ରୁଟି ଘଟେ, ଫାଇଲ୍‌ର ତ୍ରୁଟି ସ୍ଥିତି ରିସେଟ୍ ନହେବା ପର୍ଯ୍ୟନ୍ତ clearerr() ଫଙ୍କ୍‌ସନ୍‌କୁ ପରବର୍ତ୍ତୀ କଲ୍ ଗୁଡିକ 1 ଫେରସ୍ତ କରେ । ଏହି ଉଦ୍ଦେଶ୍ୟରେ clearerr() ଫଙ୍କ୍‌ସନ୍ ବ୍ୟବହୃତ ହୁଏ ।

clearerr() ଫଙ୍କ୍‌ସନ୍ ସିଣ୍ଟାକ୍ସ ହେଉଛି:

```
clearerr(fp);
```

ଯେଉଁଠାରେ *fp* ବର୍ତ୍ତମାନ ଖୋଲାଯାଇଥିବା ଫାଇଲ୍‌ପାଇଁ ଏକ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ ।

ଏଠାରେ ଉଲ୍ଲେଖ କରିବା ଜରୁରୀ ଯେ ଯଦିଓ ଆମେ ତ୍ରୁଟି ସଫା କରିଛୁ ତଥାପି ଆମେ ସମସ୍ୟାକୁ ସମାଧାନ କରିନାହୁଁ । ଆମେ ପୁଣି ପରବର୍ତ୍ତୀ ପଠନ କିମ୍ବା ଲିଖନ ଅପରେସନ୍‌ରେ ତ୍ରୁଟି ଅବସ୍ଥାକୁ ଫେରିପାରୁ ।

ଦୃଷ୍ଟାନ୍ତମୂଳକ ଉଦାହରଣ

ଉଦାହରଣ 10.1: ଏକ ଦିଆଯାଇଥିବା ଫାଇଲ୍‌ର ଆକାର ବାହାର କରିବା ପାଇଁ ପ୍ରୋଗ୍ରାମ, ଯେଉଁଠାରେ ବ୍ୟବହାରକାରୀ ଫାଇଲ୍‌ର ନାମ ପ୍ରଦାନ କରଥାନ୍ତି ।

ନିର୍ଦ୍ଦିଷ୍ଟ ଫାଇଲ୍‌ର ଆକାର ଜାଣିବା ପାଇଁ, *fseek()* ଫଙ୍କସନ୍ ବ୍ୟବହାର କରି ଫାଇଲ୍ ପଏଁଟର୍‌କୁ ଫାଇଲ୍ ଶେଷରେ ରଖ ଏବଂ *ftell()* ଫଙ୍କସନ୍ ବ୍ୟବହାର କରି ଫାଇଲ୍ ପଏଁଟର୍‌ର ମୂଲ୍ୟ ଆହରଣ କର ଯାହା ଫାଇଲ୍‌ର ଆକାର ଦେବ ।

Listing 10.10

```
/*
    Program to find the size of a given file
 */
 #include <stdio.h>
 int main()
 {
    FILE *fptr;
    char fname[30];
    printf("\nEnter name of file : ");
    gets(fname);
    fptr = fopen(fname, "r");
    if (!fptr) {
       printf("\nFile %s does not exist\n", fname);
       return 1;
    }
    fseek(fptr, 0L, 2);
    printf("\nSize of file = %ld bytes.\n", ftell(fptr));
    fclose(fptr);
    return 0;
 }
```

Test Run

```
Enter name of file : fsize.c
Size of file = 443 bytes.
```

ଉପରୋକ୍ତ ପରୀକ୍ଷଣ ରନ୍ ଦର୍ଶାଏ ଯେ ତାଲିକା 10.7 ରେ ସୃଷ୍ଟି ହୋଇଥିବା ପ୍ରୋଗ୍ରାମ ଫାଇଲ୍ (ଉତ୍ସ କୋଡ୍)ର ଆକାର ହେଉଛି 443 ବାଇଟ୍ ।

ଉଦାହରଣ 10.2: ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ, ଯାହା କୀବୋର୍ଡରୁ କିଛି ଟେକ୍ସଟ୍ ପଠନ କରେ ଏବଂ ଏହାକୁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍‌ରେ ଲେଖେ । ତା'ପରେ ପ୍ରୋଗ୍ରାମଟି ଏହି ଫାଇଲ୍‌କୁ ପଢ଼େ ଏବଂ ସ୍କ୍ରିନରେ ଏହାର ବିଷୟବସ୍ତୁକୁ ଦେଖାଏ ।

Listing 10.11

```
/*
    A program that creates a file and then reads its contents
   and display them on the computer screen
 */
 #include <stdio.h>
 int main()
 {
    char ch;
    FILE *fp1;
    fp1 = fopen("TEXT", "w");    /*  open file for writing  */
    if ( fp1 == NULL ) {
       printf("\nUnable to open file TEXT for writing\n");
       return 1;
    }
    printf("\nType some text and terminate");
    printf(" the input by Enter key\n\n");
    while ( ( ch = getche() ) != '\r' )
       fputc(ch, fp1);
    fclose(fp1);   /*  close file  */
    fp1 = fopen("TEXT", "r");    /*  open file for reading  */
    if ( fp1 == NULL ) {
       printf("\nUnable to open file TEXT for reading\n");
       return;
    }
    printf("\n\nContents of file TEXT are\n\n");
    while ( !feof(fp1) ) {
       ch = fgetc(fp1);
       putchar(ch);
    }
    fclose(fp1);
    return 0;
 }
```

Test Run

```
Type some text and terminate the input by Entering key
Don't do anything with others that you wish others should not
do with you. ↵
Contents of file TEXT are
Don't do anything with others that you wish others should not
do with you.
```

ଉଦାହରଣ 10.3: ଗୋଟିଏ ଫାଇଲ୍‌ର ବିଷୟବସ୍ତୁକୁ ଅନ୍ୟ ଏକ ଫାଇଲ୍‌ରେ ବାଇଟ୍-ପରେ-ବାଇଟ୍ କପି କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ । ବ୍ୟବହାରକାରୀ ଫାଇଲଗୁଡ଼ିକର ନାମ ପ୍ରଦାନ କରନ୍ତି ।

Listing 10.12

```
/*
     Program to copy the contents of a given file to another file.
 */
 #include <stdio.h>
 int main()
 {
    char ch;
    FILE *fp1, *fp2;
    char file1[30], file2[30];

    printf("\nEnter name of source file : ");
    gets(file1);
    printf("\nEnter name of destination file : ");
    gets(file2);
    fp1 = fopen(file1, "r");
        if ( fp1 == NULL ) {
        printf("\nUnable to open file: %s for reading\n", file1);
        return 1;
    }
    fp2 = fopen(file2, "w");
    if ( fp2 == NULL ) {
        printf("\nUnable to open file: %s for writing\n", file2);
        return 1;
    }
    while ( !feof(fp1) ) {
        ch = fgetc(fp1);
        fputc(ch, fp2);
    }
    printf("\nFile copied successfully...\n");
    fclose(fp1);
    fclose(fp2);
    return 0;
 }
```

Test Run

```
Enter the name of the source file: file_copy.c
Enter the name of destination file: temp
File copied successfully...
```

ତୁମେ ଫାଇଲ୍ *temp*ର ବିଷୟବସ୍ତୁ ଯାଞ୍ଚ କରିପାରିବ ଯାହାକି *file_copy.c* ର ବିଷୟବସ୍ତୁ ସହିତ ସମାନ ହେବ ।

ୟୁନିଟର ସାରାଂଶ

ଏହି ଅଧ୍ୟାୟରେ, ଆମେ ଯାହା ଶିଖିଛୁ

- ଯଦି ଇନପୁଟ୍ ତଥ୍ୟର ପରିମାଣ ବିଶାଳ, ତେବେ ଡାଟା ଫାଇଲ୍ ବ୍ୟବହାର କରି ଏହାକୁ ସର୍ବୋତ୍ତମ ଭାବରେ ପରିଚାଳନା କରାଯାଇପାରିବ ।
- ଯଦି କିଛି ଉପକରଣ ଦ୍ୱାରା ଉତ୍ପନ୍ନ ତଥ୍ୟ ମେସିନ୍-ପଠନଯୋଗ୍ୟ ଫର୍ମରେ (ବାଇନାରି ଫର୍ମ) ଅଛି, ତେବେ ଏହି ତଥ୍ୟକୁ ପ୍ରୋଗ୍ରାମକୁ ପ୍ରଦାନ କରିବାକୁ ତୁମକୁ ଏକ ଡାଟା ଫାଇଲ୍ ବ୍ୟବହାର କରିବାକୁ ପଡିବ ।
- ଯଦି ଆଉଟପୁଟ୍ ପରିମାଣ ବିଶାଳ ଅଟେ ଏବଂ ସ୍କ୍ରିନରେ ସଠିକ୍ ଭାବରେ ଦେଖାଯାଇପାରିବ ନାହିଁ, ତେବେ ଏକ ଡାଟା ଫାଇଲ୍‌ରେ ଆଉଟପୁଟ୍ ଲେଖିବା ଭଲ, ଯାହାକୁ ତୁମେ ଯେକୌଣସି ସମୟରେ ତୁମର ପ୍ରୋଗ୍ରାମ ପୁନଃ-କାର୍ଯ୍ୟକାରୀ ନ କରି ମଧ୍ୟ ରେଫର୍ କରିପାରିବ ।
- ଯଦି ଗୋଟିଏ ପ୍ରୋଗ୍ରାମ ଦ୍ୱାରା ଉତ୍ପନ୍ନ ଆଉଟପୁଟ୍ ଅନ୍ୟ ପ୍ରୋଗ୍ରାମ ପାଇଁ ଇନପୁଟ୍ ଭାବରେ ବ୍ୟବହୃତ ହେବ, ତା'ହେଲେ, ଡାଟା ଫାଇଲ୍‌ଗୁଡିକର ବ୍ୟବହାର ଲାଭଦାୟକ ହେବ ।
- ଏକ ଡାଟା ଫାଇଲ୍ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ କିମ୍ବା ଏକ ବାଇନାରି ଫାଇଲ୍ ହୋଇପାରେ ।

ଅନୁଶୀଳନୀ

ବିଷୟଗତ ପ୍ରଶ୍ନ

1. C ରେ ଫାଇଲ୍ I/O ପ୍ରଦର୍ଶନ ପାଇଁ ବିଭିନ୍ନ ସିଷ୍ଟମ୍‌ର ନାମ କୁହ ।
2. ଲେଖିବା ପାଇଁ ପ୍ରତ୍ୟକ୍ଷ ଭାବରେ ଖୋଲାଯାଇଥିବା ଏକ ଫାଇଲ୍‌କୁ ବନ୍ଦ କରିବା ପରାମର୍ଶଦାୟକ କି?
3. ଏକ ଫାଇଲ୍ ବନ୍ଦ କରିବାକୁ ଫଙ୍କସନ୍‌ର ନାମ କୁହ ।
4. ଏକ ଫାଇଲ୍ ଖୋଲିବା ଦ୍ୱାରା କେଉଁ ସୂଚନା ସିଷ୍ଟମ୍ ପାଇଥାଏ?
5. ଫାଇଲ୍ ପଏଁଟର୍ କହିଲେ ତୁମେ କ'ଣ ବୁଝୁଛ?
6. fopen() ଫଙ୍କସନ୍‌ଦ୍ୱାରା ଏକ ଫାଇଲ୍ କିପରି ଖୋଲାହୁଏ ଏକ ପ୍ରୋଗ୍ରାମରେ ଉଲ୍ଲେଖ କର?
7. ଧରାଯାଉ ଏକ ପ୍ରୋଗ୍ରାମ ଗୋଟିଏ ଫାଇଲ୍‌ର ପ୍ରତ୍ୟେକ ବାଇଟ୍ ପରୀକ୍ଷା କରିବାକୁ ଚାହୁଁଛି; କେଉଁ ମୋଡ୍ ଅଧିକ ଉପଯୁକ୍ତ ହେବ?
8. ଏକ ଫାଇଲ୍ ସନ୍ଦର୍ଭରେ, offset ଶବ୍ଦର ଅର୍ଥ ତୁମେ କ'ଣ ବୁଝ?
9. fseek() ଫଙ୍କସନ୍ ଦ୍ୱାରା କେଉଁ କାର୍ଯ୍ୟ କରାଯାଇପାରିବ?
10. ftell() ଫଙ୍କସନ୍ ଦ୍ୱାରା କେଉଁ କାର୍ଯ୍ୟ କରାଯାଇପାରିବ?
11. ଯଦି ଏକ ଫାଇଲ୍ ଖୋଲିବାରେ କିଛି ତ୍ରୁଟି ହୁଏ ତେବେ fopen() ଦ୍ୱାରା କେଉଁ ମୂଲ୍ୟ ଫେରସ୍ତ ହୁଏ?
12. w+ ଏବଂ r+ ମୋଡ୍ ମଧ୍ୟରେ ପାର୍ଥକ୍ୟ କ'ଣ?
13. rewind() ଫଙ୍କସନ୍ କ'ଣ କରେ?

ବହୁ ବୈକଳ୍ପିକ ପ୍ରଶ୍ନ

1. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଏକ ଡିଫଲ୍ଟ ଫାଇଲ୍ ପଏଁଟର୍ ଅଟେ?
   (a) stdin (b) stdout
   (c) stderr (d) All of the above
2. *fopen ()* ଫଙ୍କସନ୍ ଫାଇଲ୍ ଖୋଲିବାରେ ବିଫଳ ହେଲେ, କେଉଁ ମୂଲ୍ୟ ଫେରସ୍ତ କରେ
   (a) NULL (b) -1
   (c) Null (d) void
3. *flose()* ଫଙ୍କସନ୍ ସାଧାରଣତଃ, ବ୍ୟବହାର ହୁଏ
   (a) ଏକ ଫାଇଲ୍ ବିଲୋପ କରିବାକୁ (b) ପ୍ରୋଗ୍ରାମରୁ ଫାଇଲ୍ ବିଚ୍ଛିନ୍ନ କରିବାକୁ
   (c) ଏକ ଫାଇଲରୁ ତଥ୍ୟ ପଢିବାକୁ (d) ଏକ ଫାଇଲ୍‌ରେ ତଥ୍ୟ ଲେଖିବାକୁ
4. seek(fptr, offset, from) ଫଙ୍କସନ୍‌ରୁ ନିମ୍ନଲିଖିତ ମୂଲ୍ୟଗୁଡ଼ିକ ମଧ୍ୟରୁ କେଉଁଟି ମିଳେ ନାହିଁ ?
   (a) 2 (b) 0
   (c) 1 (d) EOF
5. fseek(fptr, 0, 0) ଫଙ୍କସନ୍ ଦ୍ୱାରା ହେଉଥିବା କାର୍ଯ୍ୟ କ'ଣ?
   (a) fclose (fptr) (b) ftell (fptr)
   (c) search (fptr) (d) rewind (fptr)
6. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଏକ ବୈଧ ଫାଇଲ୍ ଖୋଲିବା ମୋଡ୍ ନୁହେଁ?
   (a) *r+* (b) +r
   (c) r (d) rb
7. ଯଦି ଫାଇଲ୍ ଖୋଲିବା ମୋଡ୍ ସହିତ b ଅକ୍ଷର ଯୋଡ଼ା ହୁଏ, ତେବେ ଏହା କ'ଣ ଜଣାଇବ?
   (a) ଫାଇଲ୍ କେବଳ ପଢିବା ପାଇଁ ଖୋଲାଯିବ (b) ଫାଇଲ୍ କେବଳ ଲେଖିବା ପାଇଁ ଖୋଲାଯିବ
   (c) ଉଭୟ ପଠନ-ଲେଖନ ପାଇଁ ଫାଇଲ୍ ଖୋଲାଯିବ (d) ଫାଇଲ୍ ହେଉଛି ଏକ ବାଇନାରି ଫାଇଲ୍
8. ନିମ୍ନଲିଖିତ କୋଡ୍ ଖଣ୍ଡକୁ ବିଚାର କରନ୍ତୁ

```
FILE *fp;
fp =  fopen("inventory.dat", "rb");
```

   "rb" ରେ b ଅକ୍ଷରର ଭୂମିକା କ'ଣ?
   (a) inventory.dat ଫାଇଲ୍ ପଢିବା ପାଇଁ ବାଇନାରି ମୋଡରେ ଖୋଲ।
   (b) inventory.dat ଫାଇଲ୍ ପଠନ ଏବଂ ଲିଖନ ମୋଡରେ ଖୋଲ।
   (c) ଏକ ନୂତନ ଫାଇଲ୍ inventory.dat ଲେଖିବା ପାଇଁ ସୃଷ୍ଟି କର।
   (d) inventory.dat ଫାଇଲ୍ ପଠନ ଏବଂ ଲିଖନ ପାଇଁ ବାଇନାରି ମୋଡରେ ଖୋଲ।
9. ନିମ୍ନଲିଖିତ କୋଡ୍ ଖଣ୍ଡକୁ ବିଚାର କରନ୍ତୁ

```
FILE *fp;
fp =  fopen("notes.txt", "r+");
```

notes.txt ଫାଇଲରେ ନିମ୍ନଲିଖିତ ଅପରେସନ୍‌ଗୁଡ଼ିକ ମଧ୍ୟରୁ କେଉଁଟି ସଂପାଦିତ ହୋଇପାରିବ?

(a) ପଢିବା (b) ଲେଖିବା

(c) ଯୋଡିବା (d) ଉପରୋକ୍ତ ସମସ୍ତ

10. FILE ହେଉଛି ____

(a) *int* ପ୍ରକାର (b) *struct* ପ୍ରକାର

(c) *string* ପ୍ରକାର (d) *structure* ପ୍ରକାର

11. ଯଦି ଏକ ଫାଇଲ୍ ଖୋଲିବା ସମୟରେ କୌଣସି ତ୍ରୁଟି ଅଛି, ତେବେ fopen _____ ଫେରସ୍ତ କରେ ।

```
FILE *fp;
```

(a) null (b) NULL

(c) Null (d) EOF

12. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଲାଇବ୍ରେରି ଫଙ୍କସନ୍ ଫାଇଲ୍ ଶେଷ ଚିହ୍ନଟ କରିବାକୁ ବ୍ୟବହାର କରାଯାଇପାରିବ?

(a) eof() (b) isend()

(c) feof() (d) end()

13. ଏକ ବିଦ୍ୟମାନ ଫାଇଲ୍‌କୁ ଉଭୟ ପଠନ ଏବଂ ଲିଖନ ପାଇଁ ଖୋଲିବାକୁ ବ୍ୟବହୃତ ଏକ ମୋଡ୍ ହେଉଛି ___ ।

(a) +r (b) r+

(c) w+ (d) w

14. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁ ଫଙ୍କସନ୍ ବନ୍ଦ ଏବଂ ପୁନଃ-ଖୋଲିବା ବିନା ଏକ ଫାଇଲ୍ ପୁନଃ-ପଢିବା ପାଇଁ ବ୍ୟବହାର କରାଯାଇପାରିବ?

(a) fseek() (b) rewind()

(c) ftell() (d) Both (a) and (b)

15. ନିମ୍ନଲିଖିତ ମଧ୍ୟରୁ କେଉଁଟି ଫାଇଲ୍ ଅବସ୍ଥା ଫଙ୍କସନ୍ ନୁହେଁ?

(a) ferror() (b) ftell()

(c) clearerr() (d) feof()

ଉତ୍ତର									
1.	*(d)*	**2.**	*(a)*	**3.**	*(b)*	**4.**	*(d)*	**5.**	*(d)*
6.	*(b)*	**7.**	*(d)*	**8.**	*(d)*	**9.**	*(d)*	**10.**	*(b)*
11.	*(b)*	**12.**	*(c)*	**13.**	*(b)*	**14.**	*(d)*	**15.**	*(b)*

ପ୍ରୋଗ୍ରାମିଂ ସମସ୍ୟା

1. କିଛି ଲେଖାଥିବା ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଦିଆଯାଇଛି । ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ଯାହା ବ୍ୟବହାରକାରୀଙ୍କୁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍‌ର ନାମ ଇନପୁଟ୍ କରିବାକୁ କହିଥାଏ, ଏହି ଫାଇଲ୍‌କୁ ପଢ଼ିଥାଏ, ଏବଂ ଲେଖାରେ ଥିବା ସ୍ୱରବର୍ଣ୍ଣ ଏବଂ ଶବ୍ଦର

ସଂଖ୍ୟା ଆଉଟପୁଟ୍ ଭାବରେ ଉପସ୍ଥାପିତ କରେ ।

2. କିଛି ଲେଖାଥିବା ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍ ଦିଆଯାଇଛି । ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ଯାହା ବ୍ୟବହାରକାରୀଙ୍କୁ ଏକ ଟେକ୍ସଟ୍ ଫାଇଲ୍‌ର ନାମ ଇନପୁଟ୍ କରିବାକୁ କହିଥାଏ, ଏହି ଫାଇଲ୍ ପଢ଼ିଥାଏ, ଏବଂ ପ୍ରତ୍ୟେକ ଅକ୍ଷର 'x' କୁ ବଡ଼ ଅକ୍ଷର 'X' ସହିତ ବଦଳାଇ ଦିଏ ।
3. C ଭାଷାର ଜଣେ ଶିକ୍ଷାନବୀଶ, ସମଗ୍ର ପ୍ରୋଗ୍ରାମକୁ ବଡ ଅକ୍ଷରରେ ଟାଇପ୍ କରିଛନ୍ତି; ତୁମେ ଜାଣ ଯେ, C ଭାଷାରେ ସମ୍ମତ ପ୍ରଥା ହେଉଛି କେବଳ ମାକ୍ରୋ ଏବଂ ପରିଭାଷିତ ଧ୍ରୁବକ ବ୍ୟତୀତ ଅନ୍ୟ ସମସ୍ତପାଇଁ ଛୋଟ ଅକ୍ଷର ବ୍ୟବହାର କରି ଉତ୍ସକୋଡ୍ ଟାଇପ୍ କରିବା । ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ଯାହା ଉତ୍ସକୋଡ୍‌କୁ ଛୋଟ ଅକ୍ଷରରେ ରୂପାନ୍ତର କରିବ। ଟିପ୍ପଣୀଗୁଡିକ ଅପରିବର୍ତ୍ତିତ ରହିବା ଉଚିତ ।
4. ଫାଇଲରୁ ସଂଖ୍ୟାଗୁଡିକ ପଢି, ଯୁଗ୍ମ, ଅଯୁଗ୍ମ ଏବଂ ମୌଳିକ ସଂଖ୍ୟାଗୁଡିକ ପୃଥକ ଫାଇଲ୍‌ରେ ଲେଖିବାପାଇଁ ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ।
5. ଗୋଟିଏ ଫାଇଲ୍ ଶେଷରେ ଆଉ ଗୋଟିଏ ଫାଇଲକୁ ଯୋଡ଼ି ଏକ ତୃତୀୟ ଫାଇଲରେ ଲେଖିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ।
6. ସ୍କ୍ରିନରେ ଆଉଟପୁଟ୍ ଭାବେ ନିଜର ଉତ୍ସ କୋଡ୍ ପ୍ରିଣ୍ଟ କରିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ।
7. ଏକ ଟେକ୍ସଟ୍ ଫାଇଲରେ ଅକ୍ଷର, ଶବ୍ଦ ଏବଂ ଧାଡ଼ି ସଂଖ୍ୟା ଗଣିବାକୁ ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ।

ପ୍ରାକ୍ଟିକାଲ୍ (PRACTICALS)

1. ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ଯାହା କୀବୋର୍ଡରୁ କିଛି ଲେଖା ପଢ଼େ ଏବଂ sample.txt ନାମକ ଏକ ଫାଇଲରେ ଲେଖେ । ପ୍ରୋଗ୍ରାମ ତା'ପରେ ଏହି ଫାଇଲ୍‌କୁ ବନ୍ଦ ନ କରି ମୂଳରୁ ପଢ଼େ ଏବଂ ସ୍କ୍ରିନରେ ଏହାର ବିଷୟବସ୍ତୁ ପ୍ରଦର୍ଶନ କରେ ।

   ଫାଇଲ୍ " w+ " ମୋଡରେ ଖୋଲାଯାଇଛି, ଏବଂ ଥରେ ଫାଇଲ୍ ସୃଷ୍ଟି ହେବା ପରେ, ଫାଇଲ୍ ଆରମ୍ଭରେ ଫାଇଲ୍ ପଏଁଟର୍ ପୁନଃ ସ୍ଥାପନ କରିବାକୁ ଆମେ ଏହାକୁ rewind() ବ୍ୟବହାର କରିଥାଉ ।

Listing 10.13

```
/*
    The program that creates a file and then reads its contents
without
closing and re-opening, and display them on the computer screen
 */
 #include <stdio.h>
 int main()
 {
 char ch;
    FILE *fp;
    fp = fopen("sample.txt", "w+");
    if ( fp == NULL ) {
       printf("\nUnable to open file sample.txt\n");
       return 1;
```

```
    }
    printf("\nType some text...\n\n");
    while ( ( ch = getche() ) != '\r' )
       fputc(ch, fp);
    rewind(fp);
    printf("\n\n\nContents of file...\n\n");
    while ( !feof(fp) ) {
       ch = fgetc(fp);
       putchar(ch);
    }
    fclose(fp);
    return 0;
 }
```

Test Run

```
Type some text...
This is a sample file.
Contents of file...
This is a sample file.
```

2. ଏକ ଫାଇଲ୍‌ରେ ପ୍ରଥମ *n* ଅକ୍ଷରଗୁଡ଼ିକୁ ଓଲଟା କରିବା ପାଇଁ ଏକ C ପ୍ରୋଗ୍ରାମ ଲେଖ । ବ୍ୟବହାରକାରୀ ଫାଇଲ୍‌ର ନାମ ଏବଂ *n*ର ମୂଲ୍ୟ ପ୍ରଦାନ କରନ୍ତି ।

Listing 10.14

```
/*
     Program to reverse the first n characters in a file
 */
 #include <stdio.h>
 int main()
 {
    char str[80], filename[30], ch;
    int i, n;
    FILE *fp;
    printf("\nEnter file name : ");
    gets(filename);
    printf("\nEnter value for n : ");
    scanf("%d", &n);
    fp = fopen(filename, "r+");
    if ( fp == NULL ) {
   printf("\nUnable to open file: %s for reading\n", filename);
   return 1;
    }
   i = 0;
```

```
    while ( ( !feof(fp) ) && ( i < n ) )
    {
       str[i] = fgetc(fp);
       i++;
    }
   rewind(fp);   /* reposition the file pointer to the beginning */
    i = n-1;
    while ( i >= 0 )
    {
       fputc( str[i], fp);
       i--;
    }
    fclose(fp);
    return 0;
 }
```

Test Run

```
Enter file name: testfile.txt
Enter the value for n: 10
```

ଫାଇଲର ମୂଳ ବିଷୟବସ୍ତୁ ଥିଲା –

```
1234567890abcdefghijklmnopqrstuvwxyz
```

ପ୍ରୋଗ୍ରାମ ଚାଳନ ପରେ, ଫାଇଲ୍ ବିଷୟବସ୍ତୁଗୁଡିକ ହେଉଛି

```
0987654321abcdefghijklmnopqrstuvwxyz
```

ତୁମେ କମାଣ୍ଡ ପ୍ରମ୍ପ୍ଟରେ type କମାଣ୍ଡ ବ୍ୟବହାର କରି ପ୍ରୋଗ୍ରାମ ଚାଳନପରେ ଫାଇଲ୍‌ର ବିଷୟବସ୍ତୁ ଯାଞ୍ଚ କରିପାରିବ।

ଉପରୋକ୍ତ ପ୍ରୋଗ୍ରାମରେ, ପ୍ରଥମେ, ଦିଆଯାଇଥିବା ଫାଇଲ୍ ପଠନ/ଲିଖନ ମୋଡରେ ଖୋଲାଯାଏ। ତା'ପରେ, ଫାଇଲର ପ୍ରଥମ n ବାଇଟ୍‌କୁ ଗତିଶୀଳ ଭାବରେ ଆବଣ୍ଟିତ ବର୍ଣ୍ଣ ଆରେରେ କପି କରାଯାଇଥାଏ। ପରବର୍ତ୍ତୀ ସମୟରେ, ଆମେ ଦିଆଯାଇଥିବା ଫାଇଲ୍‌କୁ ରିୱାଇଣ୍ଡ କରିବା, ଅର୍ଥାତ, ଫାଇଲ୍ ପ୍ରାରମ୍ଭରେ ଫାଇଲ୍ ପଏଁଟର୍‌କୁ ପୁନଃଅବସ୍ଥାପିତ କରିବା।

ଶେଷରେ, ଆମେ ଫାଇଲ୍‌ରେ ଓଲଟା କ୍ରମରେ ଅକ୍ଷର ଆରେର ବିଷୟବସ୍ତୁ ଲେଖିବା। ଏହିପରି, ଆମେ ଚାହୁଁଥିବା କାର୍ଯ୍ୟକୁ ପୂରଣ କରିପାରିବା।

3. ଏକ ପ୍ରୋଗ୍ରାମ ଲେଖ ଯାହା କି-ବୋର୍ଡରୁ n ସଂଖ୍ୟକ ଛାତ୍ରଙ୍କ ନାମ ଏବଂ ମାର୍କ ଗୁଡିକ ପଢ଼ି ଏକ ଫାଇଲ୍‌ରେ ଗଚ୍ଛିତ କରେ। ଯଦି ଫାଇଲ୍ ପୂର୍ବରୁ ବିଦ୍ୟମାନ ଅଛି, ତେବେ ଫାଇଲ୍‌ରେ ସାମ୍ପ୍ରତିକ ସୂଚନା ଯୋଡ଼େ।

Listing 10.9

```
/*
     Program to reverse first n characters in a file
 */
 #include <stdio.h>
 int main()
 {
    FILE *fp;
    char name[50], filename[30];
    int i, n, marks;
    printf("Enter name of file : ");
    gets(filename);
    printf("Enter number of students to add : ");
    scanf("%d", &n);
    fp = fopen(filename, "a");
    if(fp == NULL) {
       printf("\nUnable to open file: %s for reading\n", filename);
          return 1;
    }
    for(i = 0; i < n; i++)
    {
       printf("\nEnter data for student %d\n", i+1);
       printf("\nEnter name : ");
       scanf("%s", name);
       printf("Enter marks: ");
       scanf("%d", &marks);
       fprintf(fp,"\nName: %s \nMarks=%d \n", name, marks);
   }
   fclose(fp);
   return 0;
 }
```

ଅଧିକ ଜାଣିବାପାଇଁ

ଶିକ୍ଷକ ଡାଟା ଫାଇଲ୍‌ର ଧାରଣା ଏବଂ C ଭାଷାରେ ଫାଇଲ୍ ପରିଚାଳନା ସମ୍ବନ୍ଧୀୟ ବିଭିନ୍ନ ଦିଗକୁ ବୁଝିବେ ବୋଲି ଆଶା କରାଯାଉଛି।

ଶିକ୍ଷକ ମଧ୍ୟ ଉପଯୁକ୍ତ ଉଦାହରଣ ନେଇ ଏବଂ ଛାତ୍ରଛାତ୍ରୀମାନଙ୍କ ଅଂଶଗ୍ରହଣ ସହିତ ଫାଇଲଗୁଡ଼ିକୁ ପ୍ରକ୍ରିୟାକରଣ ପାଇଁ C ପ୍ରୋଗ୍ରାମ ଲେଖିବା ଉଚିତ।

ସହାୟକ ଏବଂ ପ୍ରସ୍ତାବିତ ପଠନସୂଚି

1. R. S. Salaria, Problem Solving & Programming in C, Khanna Book Publishing Co(P) Ltd., New Delhi.
2. E. Balagurusamy, Programming in ANSI C, Tata McGraw Hill, New Delhi.
3. Yashavant Kanetkar, Let Us C, BPB Publications, New Delhi.
4. Byron Gottfried, Programming with C, Schaum's Outlines.
5. https://onlinecourses.nptel.ac.in/noc21_cs01/preview
6. https://ocw.mit.edu/courses/intro-programming/
7. https://www.programiz.com/c-programming
8. https://www.javatpoint.com/c-programming-language-tutorial

ଅଧିକ ଶିକ୍ଷା ପାଇଁ ସହାୟକ

1. Brain W. Kernighan and Deniss M. Ritchie, The C Programming Language, 2nd Edition, Prentice Hall.
2. Herbert Schildt, C: The Complete Reference, 4th Edition, McGraw Hill.
3. R. S. Salaria, Test Your Skills in C, Khanna Book Publishing Co(P) Ltd.
4. R. S. Salaria, Cracking IT Interviews, Khanna Book Publishing Co(P) Ltd.
5. Stephen Prata, C Primer Plus, Addison-Wesley Professional.
6. David Griffiths and Dawn Griffiths, Head First C, Shroff Publishers & Distributors Pvt. Ltd.
7. Antti Laaksonen, Guide to Competitive Programming, 2nd Edition, Springer.
8. https://www.greatlearning.in/academy/learn-for-free/courses/c-programming
9. https://www.udemy.com/topic/c-programming/
10. https://www.edx.org/learn/c-programming
11. https://www.coursera.org/courses?query=c%20programming
12. https://www.learnconline.com/
13. https://www.codecademy.com/

CO ଏବଂ PO ପ୍ରାପ୍ତିପାଇଁ ଟେବୁଲ

ପାଠ୍ୟକ୍ରମ ସମାପ୍ତ ହେବା ପରେ ଏହି ପାଠ୍ୟକ୍ରମ ପାଇଁ ପାଠ୍ୟକ୍ରମ ଫଳାଫଳ (CO) କାର୍ଯ୍ୟକ୍ରମ ଫଳାଫଳ (PO) ସହିତ ମ୍ୟାପ୍ କରାଯାଇପାରିବ ଏବଂ ବ୍ୟବଧାନକୁ ବିଶ୍ଳେଷଣ କରିବା ପାଇଁ PO ହାସଲ ଉଦ୍ଦେଶ୍ୟରେ ଏକ ପାରସ୍ପରିକ ସମ୍ପର୍କ କରାଯାଇପାରିବ। ବ୍ୟବଧାନର ଉପଯୁକ୍ତ ବିଶ୍ଳେଷଣ ପରେ ବ୍ୟବଧାନକୁ ଦୂର କରିବାପାଇଁ PO ହାସଲରେ ଆବଶ୍ୟକ ପଦକ୍ଷେପ ନିଆଯାଇପାରିବ ।

CO ଏବଂ PO ପ୍ରାପ୍ତିପାଇଁ ଟେବୁଲ

ପାଠ୍ୟକ୍ରମ ଫଳାଫଳ (COs)	**କାର୍ଯ୍ୟକ୍ରମ ଫଳାଫଳ ହାସଲ କରିବା** *(1- ଦୁର୍ବଳ ସହବନ୍ଧନ; 2- ମଧ୍ୟମ ସମ୍ପର୍କ; 3- ଦୃଢ ସହବନ୍ଧନ)*											
	PO-1	PO-2	PO-3	PO-4	PO-5	PO-6	PO-7	PO-8	PO-9	PO-10	PO-11	PO-12
CO-1												
CO-2												
CO-3												
CO-4												
CO-5												
CO-6												

ଉପରୋକ୍ତ ଟେବୁଲରେ ପୂରଣ ହୋଇଥିବା ତଥ୍ୟ, ବ୍ୟବଧାନ ବିଶ୍ଳେଷଣପାଇଁ ବ୍ୟବହୃତ ହୋଇପାରିବ ।

ସୂଚକାଙ୍କ

214	ପରମ ଅଶୁଦ୍ଧି	Absolute Error
323	ଠିକଣାରେ ପହଞ୍ଚିବା	Accessing Address
266, 267	ଆକ୍କର୍ମନ୍ ଫଙ୍କସନ୍	Ackermann Function
326	ବାସ୍ତବ ଚର	Actual Arguments
49	ଠିକଣା ଅପରେଟର	Address of Operator
14	ଆଲଗୋରିଦମ	Algorithm
219	ବିଶ୍ଳେଷଣାତ୍ମକ	Analytical
5	ଗଣିତ ଏବଂ ଲଜିକ ୟୁନିଟ	Arithmetic and Logic Unit
54	ଗାଣିତିକ ପରିପ୍ରକାଶ	Arithmetic Expressions
49	ଗାଣିତିକ ଅପରେଟର	Arithmetic Operators
29	ଆର୍ମଷ୍ଟ୍ରଙ୍ଗ ନମ୍ବର	Armstrong Number
47, 151, 153	ଆରେ	Array
10	ଆସେମ୍ବଲର୍	Assembler
77	ଆସାଇନମେଣ୍ଟ ଅପରେଟର୍	Assignment operators
3,22,323	ଠିକଣା ନ୍ୟସ୍ତ କରିବା	Assigning Address
80, 83	ସହଯୋଗୀତା	Associativity
118, 229	ହାରାହାରି	Average
262, 263, 267	ମୂଳ ଅବସ୍ଥା	Base Case
344, 345, 365	ବାଇନାରି ଫାଇଲ୍	Binary File
69	ବାଇନାରି ଅପରେଟର୍	Binary Operators
1,96,198	ବାଇନାରୀ ସର୍ଚ୍ଚ	Binary Search
246	ବାଇନୋମିଆଲ୍ ଗୁଣାଙ୍କ	Binomial Coefficient
19,52,13,214	ଦ୍ୱିବିଭାଜନ ପଦ୍ଧତ	Bisection Method
49	ବିଟ୍ ୱାଇଜ୍ ଅପରେଟର୍	Bit-Wise Operators
83	ବରହଗୁମିଫେ	Bodmas
65	ବଡି ମାସ୍ ଇଣ୍ଡେକ୍ସ	Body Mass Index
13	ବୁଟ୍ ଷ୍ଟ୍ରା ପ୍	Bootstrap
97	ବ୍ରାଂଚିଙ୍ଗ୍, ଶାଖା	Branching
200, 201	ବବଲ୍ ସର୍ଟ	Bubble Sort

344, 347, 357	ବଫର୍	Buffer
43	ବଗ୍	Bugs
45	ମୌଳିକ ଗଢ଼ଣ	Building Blocks
49	ପୂର୍ବ -ପ୍ରସ୍ତୁତ ଡାଟା ଟାଇପ୍	Built-in Data Types
235	ରେଫରେନ୍ସ ଦ୍ୱାରା ଡାକିବା	Call by Reference
235	ମୂଲ୍ୟ ଦ୍‌ବାରା ଡାକିବା	Call by Value
105	ମାମଲା	case
82	କାଷ୍ଟିଂ	Casting
3	କେନ୍ଦ୍ରୀୟ ପ୍ରକ୍ରିୟାକରଣ ୟୁନିଟ	Central Processing Unit
134, 142, 151	ଅକ୍ଷର	Character
14	ଶୀତଳ ବୁଟିଂ	Cold Booting
370	କମାଣ୍ଡ ପ୍ରମ୍ପ୍ଟ	Command Prompt
40	ସଂକଳନ ପ୍ରକ୍ରିୟା	Compilation
10, 14	କମ୍ପାଇଲର୍,	Compiler
10	ସଂକଳକ	Compiler
198	ଜଟିଳତା	Complexity
198	ଜଟିଳତା ବିଶ୍ଳେଷଣ	Complexity Analysis
98,161	ସର୍ତ୍ତମୂଳକ ଶାଖା	Conditional Branching
77,	ସର୍ତ୍ତମୂଳକ ପରିପ୍ରକାଶ	Conditional Expressions
54	କନସୋଲ୍	Console
98	ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତି	Control Statements
6	ନିୟନ୍ତ୍ରଣ ୟୁନିଟ	Control Unit
47, 59,	ସିପିୟୁ	CPU
49,	ଡାଟା ଟାଇପ୍	Data Type
11, 247	ତ୍ରୁଟିମୁକ୍ତ	Debugging
35, 232	ଘୋଷଣାନାମା	Declaration
357	ପଏଁଟରର ଘୋଷଣା	Declaring A Pointer
6	ଡିକୋଡ	Decode
233	ସଂଜ୍ଞା	Definition
14	ନିର୍ଦ୍ଦିଷ୍ଟତା	Definiteness
169	ସୀମାଧାର୍ଯ୍ୟ	Delimited

82	ଅବନତ	Demote
219	ବ୍ୟୁପୃତ୍ତି	Derivative
49	ବ୍ୟୁ୍ପୃନ୍ନ ଡାଟା ଟାଇପ୍	Derived Data Type
213	ପ୍ରତ୍ୟକ୍ଷ ପଦ୍ଧତି	Direct Methods
108,	ବିଭେଦକ	Discriminant
267, 271, 280	ବିଭାଜନ ଏବଂ ବିଜୟ	Divide and Conquer
86	ଡ୍ରାଇ ରନ୍	Dry Run
328	ଗତିଶୀଳ ରୂପେ ମେମୋରୀ ଆବଣ୍ଟନ	Dynamic Memory Allocation
14	ପ୍ରଭାବଶାଳୀତା	Effectiveness
349	ଫାଇଲ୍ ଶେଷ	End-of-File
214	ଇପସିଲନ୍	Epsilon
48,58	ଏସ୍କେପ ସିକ୍ୱେନ୍ସ	Escape Sequences
39	ଚାଳନ କୋଡ୍	Executable Code
12	ଚାଳନ ଫାଇଲ୍	Executable File
6	ଚାଳନ	Execute
6	ଚାଳନ ଚକ୍ର	Execution Cycle
82	ଟାଇପ୍ ର ପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ	Explicit Type Conversion
78	ପରିପ୍ରକାଶ	Expression
11	ସମ୍ପ୍ରସାରଣ	Extension
246	ଫ୍ୟାକ୍ଟୋରିଆଲ ପ୍ରକାର୍ଯ୍ୟ	Factorial Function
6	ଫେଚ୍	Fetch
265	ଫିବୋନାକି ସିକ୍ୱେନ୍ସ	Fibonacci Sequence
309	କ୍ଷେତ୍ର	Fields
370,	ଫାଇଲ୍ ପଏଣ୍ଟର	File Pointer
14	ସୀମିତତା	Finiteness
15	ଫ୍ଲୋ 'ଚାର୍ଟ	Flowchart
98	ପ୍ରବାହ ନିୟନ୍ତ୍ରଣ ବିବୃତ୍ତି	Flow Control Statements
230, 231,	ସ୍ୱତନ୍ତ୍ର ଚର	Formal Arguments
39	ଫର୍ମାଟ୍ ଷ୍ଟ୍ରିଙ୍ଗ୍	Format String
328	ଫ୍ରି ଷ୍ଟୋର	Free Store
227	ଫଙ୍କସନ୍	Function

231	ଫଙ୍କ୍ସନ୍ ବଡି	Function Body
231	ଫଙ୍କ୍ସନ୍ ହେଡର୍	Function Header
231	ଫଙ୍କ୍ସନ୍ ପ୍ରୋଟୋଟାଇପ	Function Prototype
329, 323	ଅନିର୍ଦ୍ଦିଷ୍ଟ /ଅଦରକାରି/ଅନାବଶ୍ୟକ ମୂଲ୍ୟ	Garbage Value
31, 32, 129, 130	ଗରିଷ୍ଠ ସାଧାରଣ ଗୁଣନୀୟକ	Gcd
45	ଗ୍ରାଫିକାଲ୍ ଉପଭୋକ୍ତା ଇଣ୍ଟରଫେସ୍	Graphical User Interfaces
31, 32, 129, 130	ଗରିଷ୍ଠ ସାଧାରଣ ଗୁଣନୀୟକ	Greatest Common Divisor
14	କଠିନ ବୁଟିଂ	Hard Booting
3, 7	ହାର୍ଡୱେୟାର	Hardware
31, 32	ହାଇଏଷ୍ଟ କମନ୍ ଫ୍ୟାକ୍ଟର୍	Hcf
217, 329	ସ୍ତୂପ	Heap
229	ପଦାନୁକ୍ରମିକ ସଂଗଠନ	Hierarchical Organization
31, 32	ସର୍ବୋଚ୍ଚ ସାଧାରଣ କାରକ	Highest Common Factor
44	ଉଚ୍ଚସ୍ତରୀୟ ଭାଷା	High-Level Languages
46	ଅଭିଜ୍ଞାପକ	Identifier
81	ଟାଇପ୍ ର ଅପ୍ରତ୍ୟକ୍ଷ ରୂପାନ୍ତରଣ	Implicit Type Conversion
74	ବୃଦ୍ଧି ଏବଂ ହ୍ରାସ ଅପରେଟର୍	Increment & Decrement Operators
76	ପରୋକ୍ଷ ଅପରେଟର୍	Indirection Operator
3	ଇନପୁଟ୍-ପ୍ରୋସେସ-ଆଉଟପୁଟ୍ ଚକ୍ର	Input-Process-Output Cycle
4	ଇନପୁଟ୍ ୟୁନିଟ	Input Unit
200, 210	ଇନ୍ ସର୍ ସନ ସର୍ଟ	Insertion Sort
6	ନିର୍ଦ୍ଦେଶ ଚକ୍ର	Instruction Cycle
7	ନିର୍ଦ୍ଦେଶ ସେଟ୍	Instruction Set
70	ପୂର୍ଣ୍ଣ ସଂଖ୍ୟା ଗଣିତ	Integer Arithmetic
8	ପାରସ୍ପରିକ କ୍ରିୟା	Interact
8	ଇଣ୍ଟରଫେସ୍	Interface
11	ଇଣ୍ଟର୍ ପ୍ରିଟର	Interpreter
131	ପୁନରାବୃତ୍ତି	Iterate
16, 123	ପୁନରାବର୍ତ୍ତନ	Iteration
19	ଆଇଟରେଟିଭ୍ ଲଜିକ୍	Iterative Logic
213	ଆଇଟରେଟିଭ୍ ପଦ୍ଧତି	Iterative Methods

98, 122	ଜମ୍ପିଙ୍ଗ	Jumping
46	କୀୱାର୍ଡ	Keyword
10	ଭାଷା ପ୍ରୋସେସର	Language Processor
10	ଭାଷାନୁବାଦକ	Language Translators
65, 102	ଅଧିବର୍ଷ	Leap Year
229	ଲାଇବ୍ରେରୀ ଫଙ୍କ୍ ସନ୍	Library Functions
153	ରୈଖିକ ଆରେ	Linear Array
196	ଲିନିୟର ସର୍ଚ୍ଚ	Linear Search
327	ଲିଙ୍କ୍ଡ ଲିଷ୍ଟ	Linked Lists
46,	ଧ୍ରୁବକ ମୂଲ୍ୟ	Literal
321	ଅବସ୍ଥାନ ସଂଖ୍ୟା	Location Number
43	ତାର୍କିକ ତ୍ରୁଟି	Logical Errors
54	ତାର୍କିକ ପରିପ୍ରକାଶ	Logical Expressions
72	ଲଜିକାଲ ଅପରେଟର	Logical Operators
289, 97,98, 113	ଲୁପିଙ୍ଗ୍	Looping
43	ନିମ୍ନସ୍ତରୀୟ ଭାଷା	Low-Level Languages
45	ନିମ୍ନସ୍ତରୀୟ ପ୍ରୋଗ୍ରାମିଂ	Low-Level Programming
10, 230	ଯାନ୍ତ୍ରିକ	Machine
6	ମେସିନ୍ ସାଇକେଲ୍	Machine Cycle
44	ମେସିନ୍ ଦକ୍ଷତା	Machine Efficiency
322	ମେମୋରି କୋଷ	Memory Cell
6	ମେମୋରୀ ୟୁନିଟ	Memory Unit
271	ମର୍ଜ ସର୍ଟ	Merge Sort
214	କ୍ରମାନ୍ୱୟ ଆସନ୍ନ ରାଶି ପଦ୍ଧତି	Method of Successive Approximations
7	ମାଇକ୍ରୋପ୍ରୋସେସର	Microprocessor
44	ମଧ୍ୟମ ସ୍ତରର ଭାଷା	Middle-Level Language
71	ମିଶ୍ରିତ ମୋଡ ଗଣିତ	Mixed-Mode Arithmetic
345, 346	ମୋଡ	Mode
3, 4, 54	ମନିଟର୍	Monitor
3	ମଦରବୋର୍ଡ	Motherboard
229	ବହୁ-ଫଙ୍କ୍ସନ୍ ପ୍ରୋଗ୍ରାମ	Multifunction Program

151, 227	ନୀଡ଼ନ ଲୁପ୍	Nested Loop
214	ନ୍ୟୁଟନ୍-ରାଫସନ୍ ପଦ୍ଧତି	Newton-Raphson Method
48, 169	ନଲ୍ ବର୍ଣ୍ଣ	Null Character
351	ନଲ୍ ଷ୍ଟ୍ରିଙ୍ଗ୍	Null String
219	ସଂଖ୍ୟାତ୍ମକ ଅବକଳନ	Numerical Differentiation
219, 222	ସଂଖ୍ୟାତ୍ମକ ସମାକଳନ	Numerical Integration
10	ଅବଜେକ୍ଟ କୋଡ୍	Object Code
360	ଅଫସେଟ୍	Offset
153, 154, 155	ଏକ-ପରିସରଯୁକ୍ତ	One Dimentional
8, 228	ଅପରେଟିଂ ସିଷ୍ଟମ୍	Operating System
8, 228	ଅପରେଟିଂ ସିଷ୍ଟମ୍	Os
28, 127	ପାଲିଣ୍ଡ୍ରୋମ୍	Palindrome
16	ପ୍ରୋଗ୍ରାମ ଡିଜାଇନ୍ ଭାଷା	Pdl
267	ପାଇଭଟ୍	Pivot
319, 320, 321	ପଏଁଟର	Pointer
328	ପୁଲ୍	Pool
13	ପାୱାର ଅନ୍ ସେଲ୍ଫ ଟେଷ୍ଟ	Power on Self Test
83, 84	ପ୍ରାଥମିକତା	Precedence
35	ପ୍ରିପ୍ରୋସେସର୍	Preprocessor
35	ପ୍ରିପ୍ରୋସେସର୍ ନିର୍ଦ୍ଦେଶାଗୁଡ଼ିକୁ	Preprocessor Directives
45	ଉପସ୍ଥାପନା ଗ୍ରାଫିକ୍ସ	Presentation Graphics
249	ମୌଳିକ ସଂଖ୍ୟା	Prime Number
6	ପ୍ରୋସେସିଙ୍ଗ ସାଇକେଲ୍	Processing Cycle
9	ପ୍ରୋସେସର	Processor
16	ପ୍ରୋଗ୍ରାମ ଡିଜାଇନ୍ ଭାଷା	Program Design Language
44, 45	ପ୍ରୋଗ୍ରାମିଂ ଦକ୍ଷତା	Programming Efficiency
11	ପ୍ରୋଗ୍ରାମିଂ ପରିବେଶ	Programming Environment
82	ଉନ୍ନତ	Promote 62
82	ଉନ୍ନତ	Promotion
16	ସ୍ୟୁଡୋ	Pseudo
16	ଛଦ୍ମକୋଡ୍	Pseudocode

46	ବିରାମ ଚିହ୍ନ	Punctuators
15	ଦ୍ୱିଘାତୀ ସମୀକରଣ	Quadratic Equation
70	ବାସ୍ତବ ସଂଖ୍ୟା ଗଣିତ	Real Arithmetic
262	ରିକର୍ସିଭ୍ ପଦକ୍ଷେପ	Recursive Step
214	ରେଗୁଲା-ଫାଲସି	Regula-Falsi
214	ରେଗୁଲା-ଫାଲସି ପଦ୍ଧତି	Regula-Falsi Method
54	ସମ୍ବନ୍ଧିତ ପରିପ୍ରକାଶ	Relational Expressions
71	ସମନ୍ଧୀୟ ଅପରେଟର	Relational Operators
37	ଫେରସ୍ତ ପ୍ରକାର	Return Type
234	ଫେରସ୍ତ ବିବୃତ୍ତି	Return Statement
214	ମୂଳ ଆନୁମାନିକତା	Root Approximation
213	ରାଉଣ୍ଡ -ଅଫ୍ ତ୍ରୁଟି	Round-Off Errors
12	ଚାଳନ ଫାଇଲ୍	Run File
13, 38, 47, 52	ଚାଳନ ସମୟ	Run Time
45	ବିଜ୍ଞାନ ସମ୍ବନ୍ଧୀୟ ପରିଦୃଶ୍ୟତା	Scientific Visualization
196	ସର୍ଚ୍ଚିଂ	Searching
214	ସେକାଣ୍ଟ ପଦ୍ଧତ	Secant Method
6	ମାଧ୍ୟମିକ ଭଣ୍ଡାରଣ ୟୁନିଟ	Secondary Storage Unit
17	ଚୟନ ତର୍କ	Selection Logic
206	ସିଲେକ୍ ସନ ସର୍ଟ	Selection Sort
327	ସ୍ୱ-ସୂଚିତ ଷ୍ଟ୍ରକ୍ଚର	Self-Referential Structure
16	ସିକ୍ୱେନ୍ସ ଲଜିକ୍	Sequence Logic
196	ସିକ୍ୱେନସିୟାଲ ସର୍ଚ୍ଚ	Sequential Search
78	ସଂକ୍ଷିପ୍ତ ଆସାଇନମେଣ୍ଟ ଅପରେଟର	Shorthand Assignment Operators
223	ସିମ୍ପସନଙ୍କ 1/3rd ନୟି ମ	Simpson’s 1/3rd Rule
14	ନରମ ବୁଟିଂ	Soft Booting
1,3,7	ସଫ୍ଟୱେୟାର	Software
1,96,200	ସର୍ଟିଂ	Sorting
40	ଉତ୍ସ କୋଡ୍	Source Code
40	ଉତ୍ସ ଫାଇଲ୍	Source File
49,69, 74	ସ୍ୱତନ୍ତ୍ର ଅପରେଟର୍	Special Operators
291	ଟ୍ୟାଗ୍ ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍ଚର	Structure Tag

45	ସଂରଚନା ପ୍ରୋଗ୍ରାମିଂ	Structuring Programming
152	ସବସ୍କ୍ରିପ୍ଟ	Subscript
152	ସବ୍ ସ୍କ୍ରିପ୍ଟ ହୋଇଥିବା ଚଳ	Subscripted Variable
8	ସିଷ୍ଟିମ୍ ସଫ୍ଟୱେୟାର	System Software
52	ସାଙ୍କେତିକ ଧ୍ରୁବକ	Symbolic Constant
40	ସିଣ୍ଟାକ୍ସ ତ୍ରୁଟି	Syntax Error
291	ଟ୍ୟାଗ୍ ଯୁକ୍ତ ଷ୍ଟ୍ରକ୍ଚର	Tagged Structure
77	ଟର୍ନାରୀ ଅପରେଟର୍	Ternary Operator
43	ଟେଷ୍ଟ ଡାଟା	Test Data
12	ଟେକ୍ସ୍ଟ ଏଡିଟର	Text Editor
12, 344,345	ଟେକ୍ସଟ୍ ଫାଇଲ୍	Text File
40	ଅନୁବାଦ ୟୁନିଟ	Translation Unit
40	ଅନୁବାଦ	Translator
1,64,165	ଟ୍ରାନ୍ସପୋଜ୍	Transpose
224	ଟ୍ରାପିଜଏଡ୍	Trapezoid
224	ଟ୍ରାପିଜଏଡାଲ୍ ନୟିମ	Trapezoidal Rule
213	ପରୀକ୍ଷାମୂଳକ ପଦ୍ଧତି	Trial and Error Methods
161, 162	ଦୁଇ-ପରିସରଯୁକ୍ତ ଆରେ	Two-Dimensional Array
81	ପ୍ରକାର ରୂପାନ୍ତରଣ	Type Conversion
292	ଟାଇପଡେଫ୍	Typedef
292	ପ୍ରକାର-ପରିଭାଷିତ ଷ୍ଟ୍ରକ୍ ଚର	Type-Defined Structure
69,74	ୟୁନାରି ଅପରେଟର୍	Unary Operators
327, 50	ବ୍ୟବହାରକାରୀ-ବର୍ଣ୍ଣିତ ତଥ୍ୟ ଟାଇପ୍	User-Defined Data Type
2,30,261	ବ୍ୟବହାର କାରୀ -ବର୍ଣ୍ଣିତ ଫଙ୍କ୍ ସନ୍	User-Defined Functions
4	ଭିଜୁଆଲ୍ ଡିସ୍ ପ୍ଲେ ୟୁନିଟ	Vdu
7	ଭେରି ଲାର୍ଜସ୍କେଲ ଇଣ୍ଟିଗ୍ରେସନ	Very Large Scale Integration
230	ଚଳ-ଦୃଶ୍ୟମାନତା	Visibility of Variable
4	ଭିଜୁଆଲ୍ ଡିସ୍ ପ୍ଲେ ୟୁନିଟ	Visual Display Unit
344	ଉଦବାୟୀ	Volatile
5	ଉଦବାୟୀ ମେମୋରି	Volatile Memory
14	ଉଷ୍ମ ବୁଟିଂ	Warm Booting
6	ପୁନଃଲିଖନ	Write-Back

ପରିଭାଷା

Accessibility-ସୁଗମ୍ୟତା
Access-ସୁଗମ
Ad-blocking-ବିଜ୍ଞାପନ ଅବରୋଧ
Address bar-ଠିକଣା ଦଣ୍ଡିକା
Advance search-ଉନ୍ନତ ସନ୍ଧାନ
Alias-ଉପନାମ, ଛଦ୍ମନାମ
Align-ପଙ୍କ୍ତିବଦ୍ଧ
Application-ଅନୁପ୍ରୟୋଗ
Artificial Intelligence-କୃତ୍ରିମ ବୁଦ୍ଧିମତା
Authentication-ପ୍ରାମାଣିକରଣ
Background-ପ୍ରଚ୍ଛଦପଟ, ପୃଷ୍ଠପଟ
Cashless-ନଗଦବିହୀନ
Character-ବର୍ଣ୍ଣ
Clean installation-ବିଶୁଦ୍ଧ ସଂସ୍ଥାପନ
Combinational-ସଂଯୋଜନ
Compatibility-ସଂଗତତା
Confidentiality- ଗୋପନୀୟତା
Control circuit-ନିୟନ୍ତ୍ରଣ ସର୍କିଟ୍
Control unit-ନିୟନ୍ତ୍ରଣ ୟୁନିଟ୍
Copy-ପ୍ରତୀରୂପ
Crystal-ସ୍ଫଟିକ
Database-ତଥ୍ୟସଂଗ୍ରହାଳୟ, ତଥ୍ୟ ଭଣ୍ଡାର
Delete-ବିଲୋପ
Display-ପ୍ରଦର୍ଶନ
Distribution-ବିତରଣ
Embedded-ସନ୍ନିହିତ
Encapsulate-ଆଚ୍ଛାଦ
Extension-ସମ୍ପ୍ରସାରଣ
Features-ବୈଶିଷ୍ଟ୍ୟ
Handheld-ହାତଧରା
Header-ପ୍ରବେଶିକା
Host-ପୃଷ୍ଠପୋଷକତା
Inbuilt-ପୂର୍ବ ଗଠିତ
Index-ସୂଚକ
Input-ନିବେଶ
Instructions-ନିର୍ଦ୍ଦେଶାବଳୀ
Install-ସଂସ୍ଥାପନ
Integrity-ଅଖଣ୍ଡତା
Interaction-ବାର୍ତାଳାପ
Interface-ମାଧ୍ୟମ
Interpretation-ବ୍ୟାଖ୍ୟା
Manipulation-ପ୍ରକଳନ
Matching-ମେଳକ
Memory-ସ୍ମୃତି,
Nesting-ନୀଡନ
Non-volatile-ଅଣ ଉଦବାୟୀ
Open source-ଖୋଲା ଉତ୍ସ
Operation-ସଂକ୍ରିୟା
Optional-ବୈକଳ୍ପିକ
Ordered list-କ୍ରମିତ ତାଲିକା
Output-ନିର୍ଗମ
Parse-ପଦବ୍ୟାଖ୍ୟା
Partition-ବିଭାଜନ
Peripheral device-ପ୍ରାନ୍ତସ୍ଥ ଉପକରଣ
Polarisation-ଧ୍ରୁବୀକରଣ
Preview-ପୂର୍ବପ୍ରଦର୍ଶନ, ପୂର୍ବାବଲୋକନ
Processed-ପ୍ରକ୍ରିୟାକୃତ
Property-ଗୁଣଧର୍ମ

Real - time- ବାସ୍ତବ-ସମୟ
Remove-ଅପସାରଣ
Resource sharing- ସମ୍ବଳ ଅଂଶୀଦାର
Resource-ସମ୍ବଳ
Search engine-ସନ୍ଧାନ ଇଞ୍ଜିନ
Search query-ସନ୍ଧାନ ପ୍ରଶ୍ନ
Search symbol-ସନ୍ଧାନ ପ୍ରତୀକ
Security principle-ସୁରକ୍ଷା ସିଦ୍ଧାନ୍ତ
Sequential-ଅନୁକ୍ରମିକ
Share-ଅଂଶୀଦାର
Source-ଉତ୍ସ
Specification-ବିଶିଷ୍ଟତା
Storage-ଭଣ୍ଡାରଣ
Style-ଶୈଳୀ
Technology-ପ୍ରଯୁକ୍ତିବିଦ୍ୟା
Title-ଶୀର୍ଷକ
Unordered list-ଅଣକ୍ରମିତ ତାଲିକା
Update- ଅଧୁନାତନ
User friendly-ବ୍ୟବହାରକାରୀ-ଅନୁକୂଳ
Utility-ଉପଯୋଗୀ
Version-ସଂସ୍କରଣ
Volatile-ଉଦବାୟୀ
Wireless- ବେତାର
Fraudulent- ପ୍ରତାରଣାକାରୀ

□□□

ସମସ୍ୟା ସମାଧାନପାଇଁ
ପ୍ରୋଗ୍ରାମିଂ
ପ୍ରୟୋଗଶାଳା ନିୟମାବଳି ସହିତ

ପ୍ରଫେସର ସତ୍ୟ ନରାୟଣ ଦାସ ବର୍ତ୍ତମାନ ଜିଆଇଇଟି ବିଶ୍ୱବିଦ୍ୟାଳୟରେ କମ୍ପ୍ୟୁଟର ସାଇନ୍ସ ଏବଂ ଇଞ୍ଜିନିୟରିଂ ବିଭାଗରେ ପ୍ରାଧ୍ୟାପକ ଭାବରେ କାର୍ଯ୍ୟ କରୁଛନ୍ତି । ବିଜୁ ପଟ୍ଟନାୟକ ଟେକ୍ନୋଲୋଜି ବିଶ୍ୱବିଦ୍ୟାଳୟରୁ ଏମ ସି ଏ ମାଷ୍ଟର ଡିଗ୍ରୀ ହାସଲ କରିବା ପରେ ଶିକ୍ଷା ଓ ଗବେଷଣା ପ୍ରତି ଆଗ୍ରହ ରହିଥିବା ହେତୁ ସେ ପ୍ରଧ୍ୟାପକ ଭାବେ କାର୍ଯ୍ୟ କରୁଛନ୍ତି । ୧୮ ବର୍ଷରୁ ଅଧିକ ସମୟର ଶିକ୍ଷାଦାନ ଅଭିଜ୍ଞତା ସହିତ ସେ ବିଭିନ୍ନ ଆନ୍ତର୍ଜାତୀୟ ସଂଗଠନର ସଦସ୍ୟ ଅଟନ୍ତି । ସେ ମଧ୍ୟ ବହୁ ପୁସ୍ତକର ରଚୟିତା।

ଡକ୍ଟର ଶଚିକାନ୍ତ ଦାଶ ବର୍ତ୍ତମାନ ଜିଆଇଇଟି ବିଶ୍ୱବିଦ୍ୟାଳୟରେ କମ୍ପ୍ୟୁଟର ସାଇନ୍ସ ଏବଂ ଇଞ୍ଜିନିୟରିଂ ବିଭାଗରେ ସହଯୋଗୀ ପ୍ରଫେସର ଭାବରେ କାର୍ଯ୍ୟରତ । ବ୍ରହ୍ମପୁର ବିଶ୍ୱବିଦ୍ୟାଳୟର କମ୍ପ୍ୟୁଟର ସାଇନ୍ସର ପିଜି ବିଭାଗରେ **"ପ୍ରତିଛବି ପ୍ରକ୍ରିୟାକରଣ ଏବଂ ମେସିନ୍ ଲର୍ଣିଂ"** କ୍ଷେତ୍ରରେ ପିଏଚଡି ଡିଗ୍ରୀ ହାସଲ କରିଛନ୍ତି । ସେଥିରେ ସେ **"ଓଡିଆ ଅକ୍ଷରର ଚରିତ୍ର ଚିହ୍ନଟ ପାଇଁ ଆଲଗୋରିଦମ ଅଧ୍ୟୟନ"** ବିଷୟ ଉପରେ ସେ ଅନୁସନ୍ଧାନ କରିଥିଲେ । ଜାତୀୟ ତଥା ଆନ୍ତର୍ଜାତୀୟ ସ୍ତରୀୟ ୧୫ରୁ ଅଧିକ ପତ୍ରପତ୍ରିକାରେ ତାଙ୍କର ଗବେଷଣା ନିବନ୍ଧ ପ୍ରକାଶଲାଭ କରିଛି। ସେ ବିଭିନ୍ନ ଆନ୍ତର୍ଜାତୀୟ ସଂଗଠନର ସଦସ୍ୟ ଅଟନ୍ତି ।

ପ୍ରଫେସର ଅଜିତ କୁମାର ନାୟକ 'ଶିକ୍ଷା ଓ ଅନୁସନ୍ଧାନ' ବିଶ୍ୱବିଦ୍ୟାଳୟର କମ୍ପ୍ୟୁଟର ବିଜ୍ଞାନ ଏବଂ ସୂଚନା ପ୍ରଯୁକ୍ତି ବିଭାଗର ପ୍ରଫେସର ତଥା ବିଭାଗୀୟ ମୁଖ୍ୟ । ତାଙ୍କର ଗବେଷଣା ମଧ୍ୟରେ କମ୍ପ୍ୟୁଟର ନେଟୱାର୍କିଂ, ଆଡହକ୍ ଓ ସେନସର୍ ନେଟୱାର୍କ, ମେସିନ୍ ଲର୍ନିଂ, ପ୍ରାକୃତିକ ଭାଷା କମ୍ପ୍ୟୁଟିଂ, କଥା ଓ ଚିତ୍ର ପ୍ରକ୍ରିୟାକରଣ ଇତ୍ୟାଦି ଅନ୍ତର୍ଭୁକ୍ତ । ସେ ବିଭିନ୍ନ ଜର୍ଣ୍ଣାଲ ଏବଂ ସମ୍ମିଳନୀରେ ପ୍ରାୟ 70 ଟି ଗବେଷଣାତ୍ମକ ନିବନ୍ଧ ପ୍ରକାଶ କରିଛନ୍ତି । ସେ ସିଆର୍ସି ପ୍ରେସଦ୍ୱାରା ପ୍ରକାଶିତ "କମ୍ପ୍ୟୁଟର ନେଟୱାର୍କ ସିମୁଲେସନ୍ ୟୁଜିଙ୍ଗ ଏନଏସ2' ନାମକ ଏକ ପୁସ୍ତକର ମଧ୍ୟ ମୁଖ୍ୟ-ଲେଖକ । ତାଙ୍କ ତତ୍ତ୍ୱାବଧାନରେ ଆଠ ଜଣ ଛାତ୍ରଛାତ୍ରୀ ପିଏଚଡି ଉପାଧି ଲାଭ କରିସାରିଛନ୍ତି । ସେ ଆନ୍ତର୍ଜାତୀୟ ଏବଂ ଜାତୀୟ ସ୍ତରରେ ଅନେକ ସଂଗଠନ, ସମ୍ମିଳନୀ ଓ କର୍ମଶାଳାର ସଦସ୍ୟ ଭାବେ କାର୍ଯ୍ୟରତ।

ଡକ୍ଟର ନଟବର ଶତପଥୀ ରେଭେନ୍ସା ବିଶ୍ୱବିଦ୍ୟାଳୟ ସ୍ନାତକୋତ୍ତର ଓଡିଆ ବିଭାଗର ପ୍ରାକ୍ତନ ବିଭାଗୀୟ ମୁଖ୍ୟ ଭାବରେ ଅବସର ନେବା ପରେ ଓଡିଆ ଅଧ୍ୟୟନ ଓ ଗବେଷଣା ସଂସ୍ଥା ଉପାଧ୍ୟକ୍ଷ ଭାବରେ କାର୍ଯ୍ୟରତ। ୪୦ବର୍ଷର ଅଧ୍ୟାପନା ଓ ଗବେଷଣା କାର୍ଯ୍ୟରେ ନିୟୋଜିତ ଥାଇ ୭ଟି ପୁସ୍ତକ ତଥା ୧୫୦ରୁ ଉର୍ଦ୍ଧ୍ୱ ଗବେଷଣା ନିବନ୍ଧ ଜାତୀୟ ଓ ଆନ୍ତର୍ଜାତୀୟ ସ୍ତରର ପତ୍ରପତ୍ରିକାରେ ପ୍ରକାଶିତ। ଏହାଙ୍କଦ୍ୱାରା ପ୍ରସ୍ତୁତ **ଶବ୍ଦସିନ୍ଧୁ** ଓଡିଆ ଅଭିଧାନ ବର୍ତ୍ତମାନ ପ୍ରଚଳିତ ସର୍ବବୃହତ ଓଡିଆ ଶବ୍ଦକୋଷ ଭାବରେ ପରିଚିତ। ଓଡିଆ ଭାଷାର ଶାସ୍ତ୍ରୀୟତାପାଇଁ ଏହାଙ୍କ ଉଦ୍ୟମ ଅତୁଳନୀୟ। ଓଡିଆ ବିଶ୍ୱବିଦ୍ୟାଳୟ ସ୍ଥାପନା, ଓଡିଆରେ ସରକାରି କାମ, ଓଡିଆରେ ଶିକ୍ଷା ଓ ଓଡିଶାରେ ନିଯୁକ୍ତି ଭଳି ବହୁ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ କାର୍ଯ୍ୟକ୍ରମ ସହିତ ସେ ପ୍ରତ୍ୟକ୍ଷ ଭାବରେ ସଂପୃକ୍ତ।

ଡକ୍ଟର ସୁବ୍ରତ କୁମାର ପୃଷ୍ଟି ସଦସ୍ୟ ସଚିବ, ଓଡିଆ ଅଧ୍ୟୟନ ଓ ଗବେଷଣା ସଂସ୍ଥା, ଭୁବନେଶ୍ୱରରେ କାର୍ଯ୍ୟରତ। ଓଡିଆ ଭାଷାର ଏକନିଷ୍ଠ ଗବେଷକ ଭାବରେ ସ୍ୱତନ୍ତ୍ର ପରିଚୟ ସୃଷ୍ଟି କରିଛନ୍ତି। ଓଡିଆ ଭାଷା ୫୦୦ନୁହେଁ, ବରଂ ୫୦୦୦ବର୍ଷ ତଳର ଭାଷା ବୋଲି ସେ ପ୍ରମାଣ କରିବା ସହିତ ଶାସ୍ତ୍ରୀୟତାପାଇଁ ନିଜସ୍ୱ ଉଦ୍ୟମରେ ଗବେଷଣା ସନ୍ଦର୍ଭ ପ୍ରସ୍ତୁତ କରି ଉଭୟ ସରକାରି ଓ ବେସରକାରି କମିଟିକୁ ପ୍ରଦାନ କରିଥିଲେ। ଓଡିଆ ଭାଷାର ଶାସ୍ତ୍ରୀୟ ମାନ୍ୟତା, ଓଡିଆ ବିଶ୍ୱବିଦ୍ୟାଳୟ ସ୍ଥାପନା, ଓଡିଆରେ ସରକାରି କାମ, ଓଡିଆରେ ଶିକ୍ଷା ଓ ଓଡିଶାରେ ନିଯୁକ୍ତି ଭଳି ବହୁ ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ କାର୍ଯ୍ୟ ହେଉଛି ଏହାଙ୍କ ମାନସ ସନ୍ତାନ। ତାଙ୍କର ପ୍ରାୟ ୩୦ରୁ ଉର୍ଦ୍ଧ୍ୱ ପୁସ୍ତକ ତଥା ୫୦ରୁ ଉର୍ଦ୍ଧ୍ୱ ଗବେଷଣା ନିବନ୍ଧ ଜାତୀୟ ଓ ଆନ୍ତର୍ଜାତୀୟ ସ୍ତରର ପତ୍ରପତ୍ରିକାରେ ପ୍ରକାଶିତ। ପ୍ରତିବର୍ଷ ଜାତୀୟ ଭାଷା ସମ୍ମିଳନୀ ଆୟୋଜନ କରି ଆନ୍ତର୍ଜାତିକ ସ୍ତରରେ ଓଡିଆ ଭାଷାର ନୂଆ ଦିଗ ଉପରେ ଆଲୋକପାତ କରିବାରେ ତାଙ୍କର ଗୁରୁତ୍ୱପୂର୍ଣ୍ଣ ଭୂମିକା ରହିଛି। ନିଜର ଗବେଷଣା କାର୍ଯ୍ୟପାଇଁ ଯୁବ ଭାଷା ଗବେଷକ ଭାବରେ ସେ ଭାରତର ମହାମହିମ ରାଷ୍ଟ୍ରପତିଙ୍କଠାରୁ ସମ୍ମାନଜନକ **ମହର୍ଷି ବାଦରାୟଣ ବ୍ୟାସ ସମ୍ମାନ** ଗ୍ରହଣ କରିବାପାଇଁ ମନୋନୀତ ହୋଇଛନ୍ତି।

ମୁଖ୍ୟ ବିଶେଷତ୍ୱ :

Also available as an ebook

- ବିଦ୍ୟାର୍ଥୀଙ୍କ ବୁଝିବା କ୍ଷମତା ବୃଦ୍ଧିପାଇଁ ସରଳ ଓ ସ୍ୱଷ୍ଟ ଭାଷାର ବ୍ୟବହାର।
- ରଚିପୂର୍ଣ୍ଣ ପ୍ରୋଗ୍ରାମିଂ ଶୈଳୀର ପ୍ରଦର୍ଶନ ତଥା ପ୍ରୋଗ୍ରାମିଂ ସମସ୍ୟାର ସମାଧାନପାଇଁ 90+ ପ୍ରୋଗ୍ରାମିଂ।
- ପ୍ରୋଗ୍ରାମ ବିକଶିତ ପ୍ରକ୍ରିୟା ଉଦାହରଣ ପ୍ରଦର୍ଶନପାଇଁ 165 + ଟି ପ୍ରସ୍ତୁତ ହୋଇଥିବା ପ୍ରୋଗ୍ରାମ।
- ଶିକ୍ଷାଲାଭ କରିଥିବା ମୌଳିକ ଅବଧାରଣାର ସ୍ୱମୂଲ୍ୟାଙ୍କନପାଇଁ 135+କ୍ଷୁଦ୍ର ଉତ୍ତରମୂଳକ ପ୍ରଶ୍ନ।
- ମୌଳିକ ଅବଧାରଣାର ସ୍ୱମୂଲ୍ୟାଙ୍କନପାଇଁ 165+ ବହୁ ବିକଳ୍ପିୟ ପ୍ରଶ୍ନ।

ISBN 978-93-91638-07-8

9 789391 638078

Published by: Institute of Odia Studies and Research,
3R9, BJB Nagar, Bhubaneswar, Odisha
ଓଡିଆ ଅଧ୍ୟୟନ ଓ ଗବେଷଣା ସଂସ୍ଥା, ୩ଆର୯, ବିଜେବି ନଗର ଭୁବନେଶ୍ୱର