

S

LAB MANUAL

DATA STRUCTURES

**(R22CSE2126)**

**II Year I Semester**

**DEPARTMENT OF**

**COMPUTER SCIENCE AND ENGINEERING**

**ACADEMIC YEAR 2022-2023**



|  |
| --- |
| **SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY** |
| **(An Autonomous Institution under UGC, New Delhi)** |
| **(Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi and Accredited by NBA, NAAC)** |
| **Sheriguda Village, Ibrahimpatnam Mandal, Ranga Reddy Dist. – 501 510** |

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**Contents**

* Institution Vision
* Institution Mission
* Department Vision
* Department Mission
* Program Educational Objectives (PEO’s)
* Program Specific Outcomes (PSO’s)
* Program Outcomes (PO’s)
* Academic Calendar
* Course Outcomes (CO’s)
* Mapping of Course Outcomes (CO’s) with PO’s:
* Syllabus
* Manual

|  |
| --- |
| **SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY** |
| **(An Autonomous Institution under UGC, New Delhi)** |
| **(Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi and Accredited by NBA, NAAC)** |
| **Sheriguda Village, Ibrahimpatnam Mandal, Ranga Reddy Dist. – 501 510** |



**INSTITUTION VISION**

To be a premier institution in engineering & technology and management with competence, values and social consciousness.

## INSTITUTION MISSION

**IM1:** Provide high quality academic programmes, training activities and research facilities.

**IM2:** Promote continuous industry-institute interaction for employability, entrepreneurship, leadership and research aptitude among stakeholders.

**IM3:** Contribute to the economic and technological development of the region, state and nation.



|  |
| --- |
| **SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY** |
| **(An Autonomous Institution under UGC, New Delhi)** |
| **(Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi and Accredited by NBA, NAAC)** |
| **Sheriguda Village, Ibrahimpatnam Mandal, Ranga Reddy Dist. – 501 510** |

**VISION OF THE DEPARTMENT**

To be a technologically adaptive centre for computing by grooming the students as top notch professionals.

**MISSION OF THE DEPARTMENT**

**DM1:** To offer quality education in computing.

**DM2:** To provide an environment that enables overall development of all the stakeholders.

**DM3:** To impart training on emerging technologies like Data Analytics, Artificial Intelligence and Internet Of Things.

**DM4:** To encourage participation of stakeholders in research and development

**Program Educational Objectives(PEO’s)**

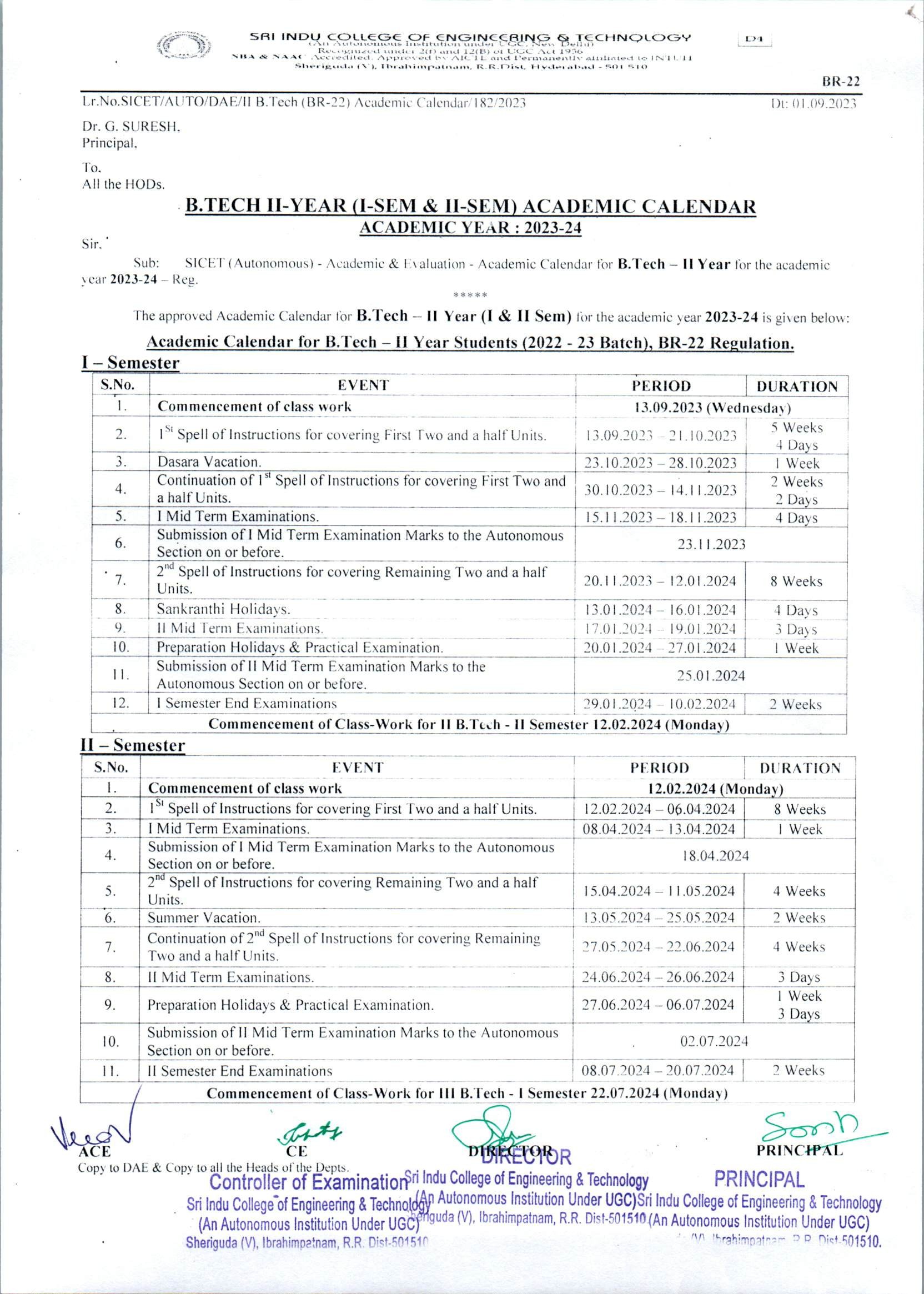
|  |  |
| --- | --- |
| **PEO1** | **Higher Studies:** Graduates with an ability to apply knowledge of Basic Sciences and programming skills in their career and higher education. |
| **PE02** | **Lifelong Learning:** Graduates with an ability to adopt new technologies for ever changing IT industry needs through Self-Study, Critical thinking and Problem-solving skills. |
| **PEO3** | **Professional Skills:** Graduates will be ready to work in projects related to complex problems involving multidisciplinary projects with effective analytical skills |
| **PEO4** | **Engineering citizenship:** Graduates with an ability to communicate well and exhibit social, technical and ethical responsibility in process or product. |

**Program Specific Outcomes(PSO’s)**

|  |  |
| --- | --- |
| **PSO1** | **Software Development:** To apply the knowledge of Software Engineering, Data Communication, Web Technology and Operating Systems for building  IOT and Cloud Computing applications. |
| **PSO2** | **Industrial Skills Ability:** Design, develop and test software systems for world- wide network of computers to provide solutions to real world  problems. |
| **PSO3** | **Project Implementation:** Analyze and recommend the appropriate IT infrastructure required for the implementation of a project. |

**Program Outcomes(PO’s)**

|  |  |
| --- | --- |
| **PO1** | **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. |
| **PO2** | **Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. |
| **PO3** | **Design / Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. |
| **PO4** | **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. |
| **PO5** | **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. |
| **PO6** | **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. |
| **PO7** | **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| **PO8** | **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| **PO9** | **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings. |
| **PO10** | **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. |
| **PO11** | **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. |
| **PO12** | **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change. |



**AcademicYear**:2023-24

# COURSEOUTCOMES(CO’s)

**Class:**II YEAR-ISEM**.**

**Course Name: Data Structures Lab (R22CSE2126)**

At the end of the course, the student will be able to

|  |  |
| --- | --- |
| **CourseOutcomes (COs)** | |
| **C21L1.1** | Design a program to implement the linear data structures using static and dynamic memory allocation. (Create)**)** |
| **C21L1.2** | Design a program to implement searching ,sorting techniques for the given problem.(Create) |
| **C21L1.3** | Demonstrate the fundamental algorithms of tree data structures by experimenting the programs.(Apply) |
| **C21L1.4** | Examine the traversing of a given graph by using the respect to graph traversal techniques(Apply) |
| **C21L1.5** | Design a program to implement the pattern matching algorithms for the given problem.(Create) |
| **C21L1.6** | Apply data structures in the real time applications |

# MappingofCourseOutcomes(CO’s)withPO’s:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CO** | **PO** | | | | | | | | | | | | |
| **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | | **PO12** |
| **C21L1.1** | 3 | 2 | 3 | 3 | 1 | 1 | - | - | - | - | | 3 | 1 |
| **C21L1.2** | 3 | 3 | 3 | 3 | - | - | - | - | - | - | | - | 1 |
| **C21L1.3** | 3 | 3 | 3 | 3 | - | - | - | - | 1 | - | | 1 | 1 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C21L1.4** | 3 | 2 | 3 | 2 | - | 1 | - | - | - | - | - | 1 |
| **C21L1.5** | 3 | 3 | 3 | 3 | - | - | - | - | 1 | - | - | 2 |
| **C21L1.6** | 3 | 3 | 3 | 3 | - | - | - | - | - | - | - | 1 |
|  | **3** | **2.6** | **3** | **2.6** | **-** | **1** | **-** | **-** | **1** | **-** | **3** | **1.5** |

**3:High 2.Medium 1. Low**

# MappingofCourseOutcomes(CO’s)withPSO’s:

|  |  |  |  |
| --- | --- | --- | --- |
| **COs** | **PSO1** | **PSO2** | **PSO3** |
| **C21L1.1** | 3 | 3 | - |
| **C21L1.2** | 3 | 3 | - |
| **C21L1.3** | 3 | 3 | 2 |
| **C21L1.4** | 3 | 3 | 3 |
| **C21L1.5** | 3 | 3 | 2 |
| **C21L1.6** | 3 | 3 | - |
|  | 3 | 3 | 3 |

**BR22 – B.TECH. – COMPUTER SCIENCE & ENGINEERING**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY** | | | | |
| **(An Autonomous Institution under UGC, New Delhi)** | | | | |
| **B.Tech. - II Year – I Semester** | **L** | **T** | **P** | **C** |
|  | **0** | **0** | **3** | **1.5** |
| **(R22CSE2126) DATA STRUCTURES LAB** | | | | |
| **Course Objectives:**   * It covers various concepts of C programming language * It introduces searching and sorting algorithms * It provides an understanding of data structures such as stacks and queues.   **Course Outcomes:**   * Ability to develop C programs for computing and real-life applications using basic elements like control statements, arrays, functions, pointers and strings, and data structures like stacks, queues and linked lists. * Ability to Implement searching and sorting algorithms   **List of Experiments:**   * 1. Write a program that uses functions to perform the following operations on singly linkedlist.:      1. Creation ii) Insertion iii) Deletion iv) Traversal   2. Write a program that uses functions to perform the following operations on doubly linkedlist.:      1. Creation ii) Insertion iii) Deletion iv) Traversal   3. Write a program that uses functions to perform the following operations on circular linkedlist.:      1. Creation ii) Insertion iii) Deletion iv) Traversal   4. Write a program that implement stack (its operations) using      1. Arrays ii) Pointers   5. Write a program that implement Queue (its operations) using      1. Arrays ii) Pointers   6. Write a program that implements the following sorting methods to sort a given list of integersin ascending order      1. Quick sort ii) Heap sort iii) Merge sort   7. Write a program to implement the tree traversal methods( Recursive and Non Recursive).   8. Write a program to implement      1. Binary Search tree ii) B Trees iii) B+ Trees iv) AVL trees v) Red - Black trees   9. Write a program to implement the graph traversal methods.   10. Implement a Pattern matching algorithms using Boyer- Moore, Knuth-Morris-Pratt   **TEXT BOOKS:**   1. Fundamentals of Data Structures in C, 2nd Edition, E. Horowitz, S. Sahni and Susan AndersonFreed, Universities Press. 2. Data Structures using C – A. S. Tanenbaum, Y. Langsam, and M. J. Augenstein, PHI/PearsonEducation.   **REFERENCE BOOK:**   * 1. Data Structures: A Pseudocode Approach with C, 2nd Edition, R. F. Gilberg and B. A. Forouzan, Cengage Learning | | | | |

**SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY (AUTONOMOUS)** Page 78

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **SRI INDU COLLEGE OF ENGG & TECH** | | | |  |
| **LAB PLAN** | | | |
| **Regulation: R22** | | | |
| **Department of Computer Science and Engineering** | | | |
|  | **(R22CSE2126)** | | | |
| **Sub. Code & Title** | **Data Structures** | | | |
| **Academic Year: 2023-24** | | **Year/Sem./Section** | **II/I/A&B&C&D** | |
| **Faculty Name & Designation** | |  | | |

**Lab Plan**

**2023-24 II Year –I Semester CSE**

|  |  |  |
| --- | --- | --- |
| **S No** | **Topics** | **No. of weeks** |
| 1. | Write a program that uses functions to perform the following operations on singly linked list.:  i) Creation ii) Insertion iii) Deletion iv) Traversal | 1 |
| 2. | Write a program that uses functions to perform the following operations on doubly linked list.:  i) Creation ii) Insertion iii) Deletion iv) Traversal | 1 |
| 3. | Write a program that uses functions to perform the following operations on circular linked list.:  i) Creation ii) Insertion iii) Deletion iv) Traversal | 1 |
| 4. | Write a program that implement stack (its operations) using  i) Arrays ii) Pointers | 1 |
| 5. | Write a program that implement Queue (its operations) using  i) Arrays ii) Pointers | 1 |
| 6. | Write a program that implements the following sorting methods to sort a given list of integers in ascending order   * + 1. Quick sort ii) Heap sort iii) Merge sort | 1 |
| 7. | Write a program to implement the tree traversal methods( Recursive and Non Recursive). | 1 |

|  |  |  |
| --- | --- | --- |
| 8. | Write a program to implement   1. Binary Search tree ii) B Trees iii) B+ Trees iv)AVL trees   v) Red - Black trees | 2 |
| 9. | Write a program to implement the graph traversal methods. | 1 |
| 10. | Implement a Pattern matching algorithms using Boyer- Moore, Knuth-Morris-Pratt | 1 |

## PROGRAMS

**Week1.**

**Aim:** Write a program that uses functions to perform the following operations on singly linked list.:

* 1. Creation ii) Insertion iii) Deletion iv) Traversal

## SourceCode:

# include <stdio.h> # include <conio.h> # include <stdlib.h> struct slinklist

{

int data;

struct slinklist \*next;

};

typedef struct slinklist node; node \*start = NULL;

int menu()

{

int ch; clrscr();

printf("\n 1.Create a list "); printf("\n ");

printf("\n 2.Insert a node at beginning "); printf("\n 3.Insert a node at end"); printf("\n 4.Insert a node at middle"); printf("\n ");

printf("\n 5.Delete a node from beginning"); printf("\n 6.Delete a node from Last"); printf("\n 7.Delete a node from Middle"); printf("\n ");

printf("\n 8.Traverse the list (Left to Right)"); printf("\n 9.Traverse the list (Right to Left)"); printf("\n ");

printf("\n 10. Count nodes "); printf("\n 11. Exit ");

printf("\n\n Enter your choice: "); scanf("%d",&ch);

return ch;

}

node\* getnode()

{

node \* newnode;

newnode = (node \*) malloc(sizeof(node)); printf("\n Enter data: ");

scanf("%d", &newnode -> data); newnode -> next = NULL; return newnode;

}

int countnode(node \*ptr)

{

int count=0; while(ptr != NULL)

{

count++;

ptr = ptr -> next;

}

return (count); } void createlist(int n)

{

int i; node \*newnode; node \*temp;

for(i = 0; i < n; i++)

{

newnode = getnode(); if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL) temp = temp -> next;

temp -> next = newnode;

}

}

}

void traverse()

{

node \*temp; temp = start;

printf("\n The contents of List (Left to Right): \n"); if(start == NULL)

{

printf("\n Empty List"); return;

}

else

{

while(temp != NULL)

{

printf("%d-->", temp -> data); temp = temp -> next;

}

}

printf(" X ");

}

void rev\_traverse(node \*start)

{

if(start == NULL)

{

return;

}

else

{

rev\_traverse(start -> next); printf("%d -->", start -> data);

}

}

void insert\_at\_beg()

{

node \*newnode; newnode = getnode(); if(start == NULL)

{

start = newnode;

}

else

{

newnode -> next = start; start = newnode;

}

}

void insert\_at\_end()

{

node \*newnode, \*temp; newnode = getnode(); if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL) temp = temp -> next;

temp -> next = newnode;

}

}

void insert\_at\_mid()

{

node \*newnode, \*temp, \*prev; int pos, nodectr, ctr = 1; newnode = getnode(); printf("\n Enter the position: "); scanf("%d", &pos);

nodectr = countnode(start);

if(pos > 1 && pos < nodectr)

{

else

temp = prev = start; while(ctr < pos)

{

prev = temp;

temp = temp -> next; ctr++;

}

prev -> next = newnode; newnode -> next = temp;

}

printf("position %d is not a middle position", pos);

}

void delete\_at\_beg()

{

node \*temp; if(start == NULL)

{

printf("\n No nodes are exist.."); return ;

}

else

{

temp = start;

start = temp -> next; free(temp);

printf("\n Node deleted ");

}

}

void delete\_at\_last()

{

return ;

}

node \*temp, \*prev; if(start == NULL)

{

printf("\n Empty List..");

}

else

{

temp = start;

prev = start;

while(temp -> next != NULL)

{

prev = temp;

temp = temp -> next;

}

prev -> next = NULL; free(temp);

printf("\n Node deleted ");

}

void delete\_at\_mid()

{

int ctr = 1, pos, nodectr; node \*temp, \*prev; if(start == NULL)

{

printf("\n Empty List..");

return ;

}

else

{

printf("\n Enter position of node to delete: "); scanf("%d", &pos);

nodectr = countnode(start); if(pos > nodectr)

{

printf("\nThisnode doesnot exist");

}

if(pos > 1 && pos < nodectr)

{

temp = prev = start; while(ctr < pos)

{

prev = temp;

temp = temp -> next; ctr ++;

}

prev -> next = temp -> next; free(temp);

printf("\n Node deleted..");

}

else

{

printf("\n Invalid position.."); getch();

}

}

}

void main(void)

{

int ch, n; clrscr();

while(1)

{

ch = menu(); switch(ch)

{

case 1:

if(start == NULL)

{

printf("\n Number of nodes you want to create: "); scanf("%d", &n);

createlist(n);

case 2:

}

else break;

printf("\n List created..");

printf("\n List is already created..");

insert\_at\_beg();

break;

case 3:

case 4:

insert\_at\_end(); break;

insert\_at\_mid(); break;

case 5:

case 6:

case 7:

case 8:

case 9:

delete\_at\_beg(); break;

delete\_at\_last(); break;

delete\_at\_mid(); break;

traverse(); break;

printf("\n The contents of List (Right to Left): \n"); rev\_traverse(start);

printf(" X "); break;

case 10:

printf("\n No of nodes : %d ", countnode(start)); break;

case 11 :

exit(0);

}

getch();

}

## }

**OUTPUT :**







## Week I Viva Questions

1. **Define self referential structure and give one example**

## Draw one example node of single linked list

1. **What is NULL?**

## List out the operations on linked list.

1. **Differentiate Array and linked list.**

## Week2:

**Aim:** Write a program that uses functions to perform the following operations on doubly linked list.:

* 1. Creation ii) Insertion iii) Deletion iv) Traversal

## SourceCode:

#include <stdio.h> #include <stdlib.h> #include <conio.h> struct dlinklist

{

struct dlinklist \*left; int data;

struct dlinklist \*right;

};

typedef struct dlinklist node; node \*start = NULL;

node\* getnode()

{

node \* newnode;

newnode = (node \*) malloc(sizeof(node)); printf("\n Enter data: ");

scanf("%d", &newnode -> data); newnode -> left = NULL; newnode -> right = NULL; return newnode;

}

int countnode(node \*start)

{

if(start == NULL) return 0;

else

return 1 + countnode(start -> right);

}

int menu()

{

int ch; clrscr();

printf("\n 1.Create");

printf("\n "); printf("\n 2. Insert a node at beginning "); printf("\n 3. Insert a node at end"); printf("\n 4. Insert a node at middle"); printf("\n ");

printf("\n 5. Delete a node from beginning"); printf("\n 6. Delete a node from Last"); printf("\n 7. Delete a node from Middle"); printf("\n ");

printf("\n 8. Traverse the list from Left to Right "); printf("\n 9. Traverse the list from Right to Left "); printf("\n ");

printf("\n 10.Count the Number of nodes in the list"); printf("\n 11.Exit");

printf("\n\n Enter your choice: "); scanf("%d", &ch);

return ch;

}

void createlist(int n)

{

int i;

node \*newnode; node \*temp;

for(i = 0; i < n; i++)

{

newnode = getnode(); if(start == NULL) start = newnode;

else

{

temp = start; while(temp -> right) temp = temp -> right;

temp -> right = newnode; newnode -> left = temp;

}

}

}

void traverse\_left\_to\_right()

{

node \*temp; temp = start;

printf("\n The contents of List: "); if(start == NULL )

printf("\n Empty List"); else

{

while(temp != NULL)

{

printf("\t %d ", temp -> data); temp = temp -> right;

}

}

}

void traverse\_right\_to\_left()

{

node \*temp; temp = start;

printf("\n The contents of List: "); if(start == NULL)

printf("\n Empty List"); else

{

while(temp -> right != NULL)

temp = temp -> right;

}

while(temp != NULL)

{

printf("\t%d", temp -> data); temp = temp -> left;

}

}

void dll\_insert\_beg()

{

node \*newnode; newnode = getnode(); if(start == NULL) start = newnode;

else

{

newnode -> right = start; start -> left = newnode; start = newnode;

}

}

void dll\_insert\_end()

{

node \*newnode, \*temp; newnode = getnode(); if(start == NULL)

start = newnode; else

{

temp = start;

while(temp -> right != NULL) temp = temp -> right;

temp -> right = newnode; newnode -> left = temp;

}

}

void dll\_insert\_mid()

{

node \*newnode,\*temp; int pos, nodectr, ctr = 1; newnode = getnode();

printf("\n Enter the position: "); scanf("%d", &pos);

nodectr = countnode(start); if(pos - nodectr >= 2)

{

printf("\n Position is out of range.."); return;

}

if(pos > 1 && pos < nodectr)

{

temp = start; while(ctr < pos - 1)

{

temp = temp -> right; ctr++;

}

newnode -> left = temp; newnode -> right = temp -> right; temp -> right -> left = newnode; temp -> right = newnode;

}

else

printf("position %d of list is not a middle position ", pos);

}

void dll\_delete\_beg()

{

node \*temp; if(start == NULL)

{

printf("\n Empty list"); getch();

return ;

}

else

{

temp = start;

start = start -> right; start -> left = NULL; free(temp);

}

}

void dll\_delete\_last()

{

node \*temp; if(start == NULL)

{

printf("\n Empty list"); getch();

return ;

}

else

{

temp = start;

while(temp -> right != NULL) temp = temp -> right;

temp -> left -> right = NULL; free(temp);

temp = NULL;

}

}

void dll\_delete\_mid()

{

int i = 0, pos, nodectr; node \*temp;

if(start == NULL)

{

printf("\n Empty List");

getch(); return;

}

else

{

printf("\n Enter the position of the node to delete: "); scanf("%d", &pos);

nodectr = countnode(start); if(pos > nodectr)

{

printf("\nthis node does not exist"); getch();

return;

}

if(pos > 1 && pos < nodectr)

{

temp = start; i= 1;

while(i < pos)

{

temp = temp -> right; i++;

}

temp -> right -> left = temp -> left; temp -> left -> right = temp -> right; free(temp);

printf("\n node deleted..");

}

else

{

printf("\n It is not a middle position.."); getch();

}

}

}

void main(void)

{

int ch, n; clrscr(); while(1)

{

ch = menu(); switch( ch)

{

case 1 :

printf("\n Enter Number of nodes to create: "); scanf("%d", &n);

createlist(n);

printf("\n List created.."); break;

case 2 : dll\_insert\_beg(); break;

case 3 :

dll\_insert\_end(); break;

case 4 :

dll\_insert\_mid(); break;

case 5 :

dll\_delete\_beg(); break;

case 6 :

dll\_delete\_last(); break;

case 7 :

dll\_delete\_mid(); break;

case 8 : traverse\_left\_to\_right(); break;

case 9 : traverse\_right\_to\_left(); break;

case 10 :

printf("\n Number of nodes: %d", countnode(start)); break;

case 11: exit(0);

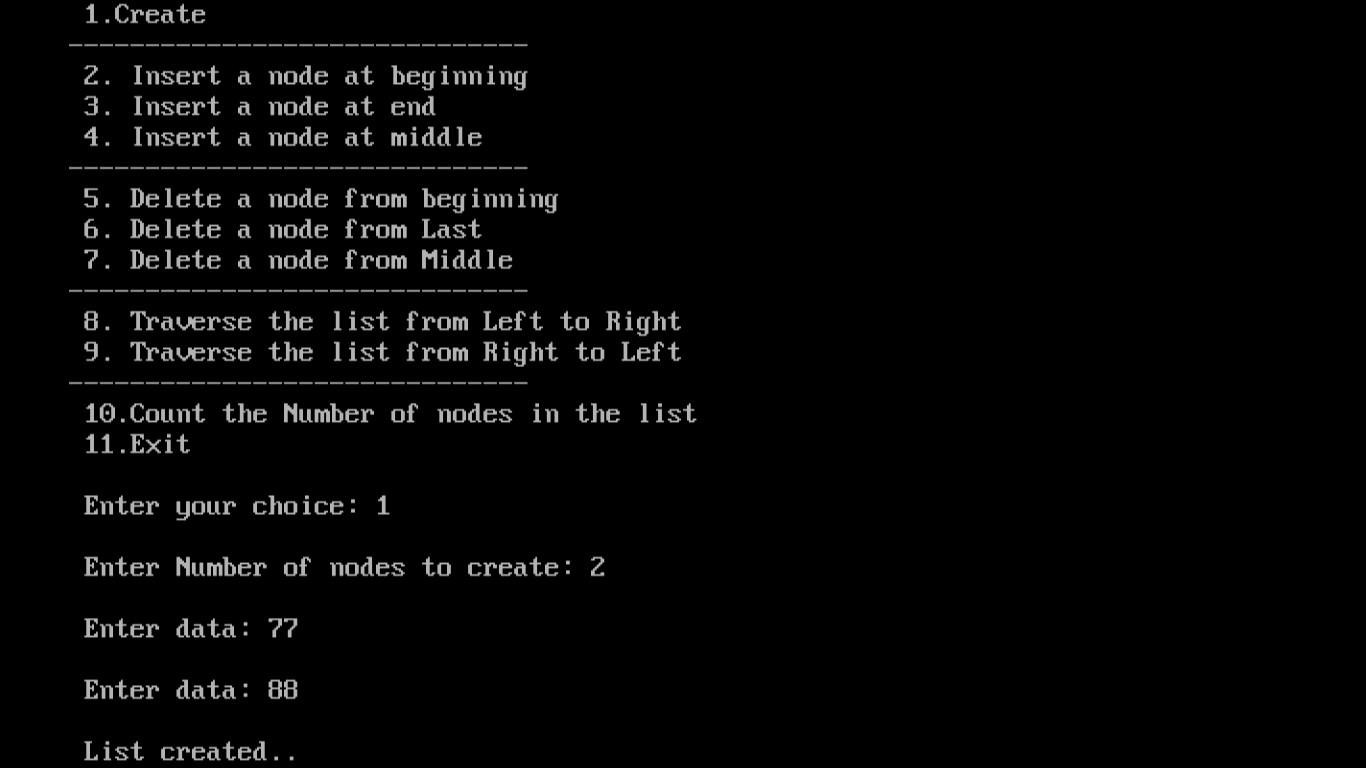
}

getch();

}

}

OUTPUT :





### Week II Viva Questions

1. ***Advantage of linked lists.***

### Difference between single and double linked list.

1. ***Define node structure of double linked list.***

### What is traversal?

1. ***List types of Linked lists.***

### Week3:

**Aim:** Write a program that uses functions to perform the following operations on circular linked list.:

* 1. Creation ii) Insertion iii) Deletion iv) Traversal

Sourcecode:

# include <stdio.h> # include <conio.h> # include <stdlib.h> struct cslinklist

{

int data;

struct cslinklist \*next;

};

typedef struct cslinklist node; node \*start = NULL;

int nodectr; node\* getnode()

{

node \* newnode;

newnode = (node \*) malloc(sizeof(node)); printf("\n Enter data: ");

scanf("%d", &newnode -> data); newnode -> next = NULL; return newnode;

}

int menu()

{

int ch; clrscr();

printf("\n 1. Create a list "); printf("\n\n ");

printf("\n 2. Insert a node at beginning "); printf("\n 3. Insert a node at end"); printf("\n 4. Insert a node at middle"); printf("\n\n ");

printf("\n 5. Delete a node from beginning"); printf("\n 6. Delete a node from Last"); printf("\n 7. Delete a node from Middle"); printf("\n\n "); printf("\n 8. Display the list");

printf("\n 9. Exit");

printf("\n\n "); printf("\n Enter your choice: "); scanf("%d", &ch);

return ch;

}

void createlist(int n)

{

int i;

node \*newnode; node \*temp;

nodectr = n;

for(i = 0; i < n ; i++)

{

newnode = getnode(); if(start == NULL)

{

start = newnode;

}

else

{

temp = start;

while(temp -> next != NULL) temp = temp -> next;

temp -> next = newnode;

}

}

newnode ->next = start; /\* last node is pointing to starting node \*/

}

void display()

{

node \*temp; temp = start;

printf("\n The contents of List (Left to Right): "); if(start == NULL )

printf("\n Empty List"); else

{

do

{

printf("\t %d ", temp -> data); temp = temp -> next;

} while(temp != start); printf(" X ");

}

}

void cll\_insert\_beg()

{

node \*newnode, \*last; newnode = getnode(); if(start == NULL)

{

start = newnode; newnode -> next = start;

}

else

{

last = start;

while(last -> next != start) last= last -> next; newnode -> next = start; start = newnode;

last -> next = start;

}

printf("\n Node inserted at beginning..");

nodectr++;

}

void cll\_insert\_end()

{

node \*newnode, \*temp; newnode = getnode(); if(start == NULL )

{

start = newnode; newnode -> next = start;

}

else

{

temp = start;

while(temp -> next != start) temp = temp -> next;

temp -> next = newnode; newnode -> next = start;

}

printf("\n Node inserted at end.."); nodectr++;

}

void cll\_insert\_mid()

{

node \*newnode, \*temp, \*prev; int i, pos ;

newnode = getnode(); printf("\n Enter the position: "); scanf("%d", &pos);

if(pos > 1 && pos < nodectr)

{

temp = start; prev = temp; i= 1;

while(i < pos)

{

prev = temp;

temp = temp -> next; i++;

}

prev -> next = newnode; newnode -> next = temp;

nodectr++;

printf("\n Node inserted at middle..");

}

else

{

printf("position %d of list is not a middle position ", pos);

}

}

void cll\_delete\_beg()

{

node \*temp, \*last; if(start == NULL)

{

printf("\n No nodes exist.."); getch();

return ;

}

else

{

last = temp = start; while(last -> next != start) last= last -> next;

start = start -> next; last -> next = start; free(temp);

nodectr--;

printf("\n Node deleted.."); if(nodectr == 0)

start = NULL;

}

}

void cll\_delete\_last()

{

node \*temp,\*prev; if(start == NULL)

{

printf("\n No nodes exist.."); getch();

return ;

}

else

{

temp = start; prev = start;

while(temp -> next != start)

{

prev = temp;

temp = temp -> next;

}

prev -> next = start; free(temp);

nodectr--; if(nodectr == 0) start = NULL;

printf("\n Node deleted..");

}

}

void cll\_delete\_mid()

{

int i = 0, pos;

node \*temp, \*prev; if(start == NULL)

{

printf("\n No nodes exist.."); getch();

return ;

}

else

{

printf("\n Which node to delete: "); scanf("%d", &pos);

if(pos > nodectr)

{

printf("\nThis node does not exist"); getch();

return;

}

if(pos > 1 && pos < nodectr)

{

temp=start; prev = start; i= 0;

while(i < pos - 1)

{

prev = temp;

temp = temp -> next ; i++;

}

prev -> next = temp -> next; free(temp);

nodectr--;

printf("\n Node Deleted..");

}

else

{

printf("\n It is not a middle position.."); getch();

}

}

}

void main(void)

{

int result; int ch, n; clrscr(); while(1)

{

ch = menu(); switch(ch)

{

case 1 :

if(start == NULL)

{

printf("\n Enter Number of nodes to create: "); scanf("%d", &n);

createlist(n); printf("\nList created..");

}

else

printf("\n List is already Exist..");

break; case 2 :

cll\_insert\_beg(); break;

case 3 : cll\_insert\_end(); break;

case 4 : cll\_insert\_mid(); break;

case 5 : cll\_delete\_beg(); break;

case 6 : cll\_delete\_last(); break;

case 7 : cll\_delete\_mid(); break;

case 8 : display(); break; case 9 : exit(0);

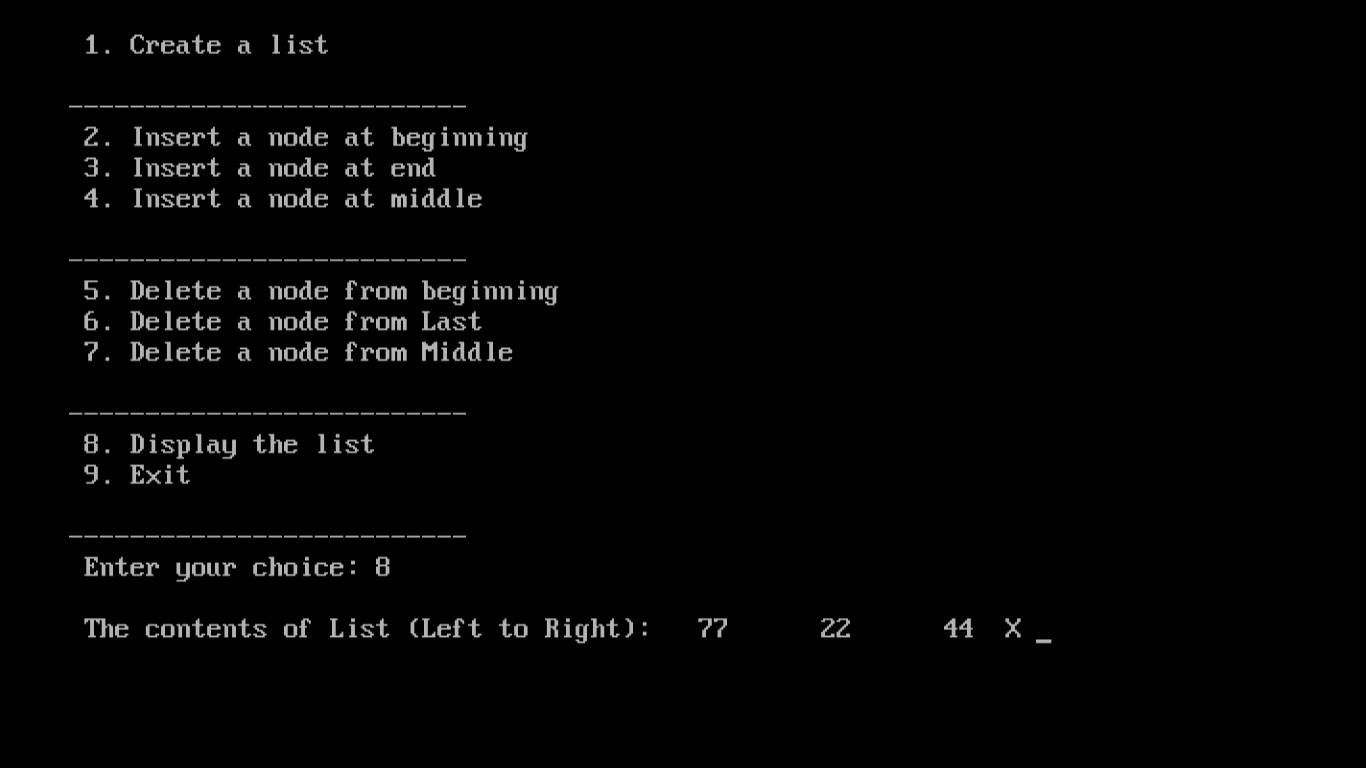
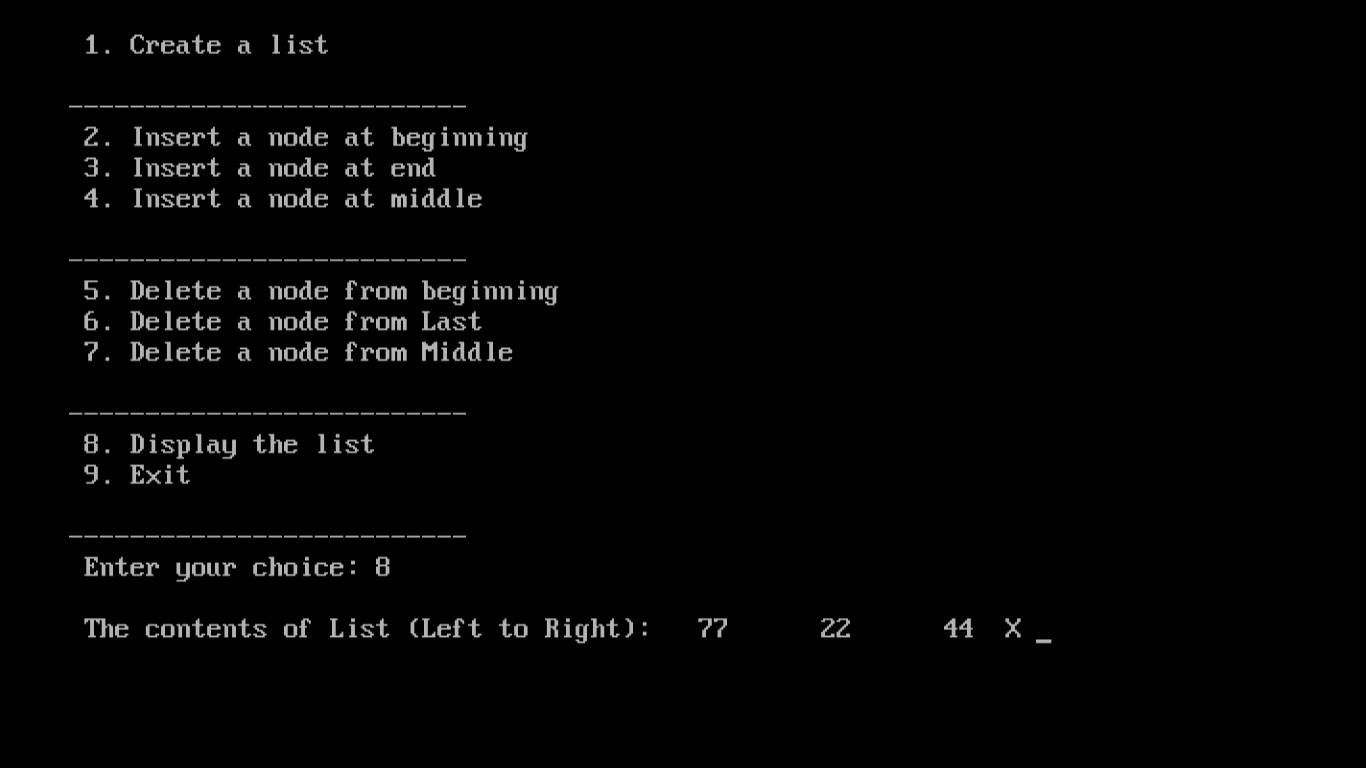
}

getch();

}

}

## Output :



Viva Questions

1. Draw an example to insert node in Circular linked list.
2. Write the Advantage of Circular linked list.
3. Differentiate single , double ,circular lists.
4. List applications of linked list.
5. What is while(1)?

### Week4:

**Aim:**Write a program that implement stack (its operations) using i)Arrays ii) Pointers

### Source code to implement Stack using linked list :

# include <stdio.h> # include <conio.h> # include <stdlib.h> struct stack

{

int data;

struct stack \*next;

};

void push(); void pop(); void display();

typedef struct stack node; node \*start=NULL;

node \*top = NULL; node\* getnode()

{

node \*temp;

temp=(node \*) malloc( sizeof(node)) ; printf("\n Enter data ");

scanf("%d", &temp -> data); temp -> next = NULL; return temp;

}

void push(node \*newnode)

{

node \*temp;

if( newnode == NULL )

{

printf("\n Stack Overflow.."); return;

if(start == NULL)

{

start = newnode; top = newnode;

}

else

{

temp = start;

while( temp -> next != NULL) temp = temp -> next;

temp -> next = newnode; top = newnode;

}

printf("\n\n\t Data pushed into stack");

}

void pop()

{

node \*temp; if(top == NULL)

{

printf("\n\n\t Stack underflow"); return;

}

temp = start;

if( start -> next == NULL)

{

printf("\n\n\t Popped element is %d ", top -> data); start = NULL;

free(top); top = NULL;

}

else

{

while(temp -> next != top)

{

temp = temp -> next;

}

temp -> next = NULL;

printf("\n\n\t Popped element is %d ", top -> data); free(top);

top = temp;

}

}

void display()

{

node \*temp; if(top == NULL)

{

printf("\n\n\t\t Stack is empty ");

}

else

{

temp = start;

printf("\n\n\n\t\t Elements in the stack: \n"); printf("%5d ", temp -> data);

while(temp != top)

{

temp = temp -> next; printf("%5d ", temp -> data);

}

}

}

char menu()

{

char ch; clrscr();

printf("\n \tStack operations using pointers.. "); printf("\n -----------\*\*\*\*\*\*\*\*\*\* \n");

printf("\n 1. Push ");

printf("\n 2. Pop "); printf("\n 3. Display"); printf("\n 4. Quit ");

printf("\n Enter your choice: "); ch = getche();

return ch;

}

void main()

{

char ch;

node \*newnode; do

{

ch = menu(); switch(ch)

{

case '1' :

newnode = getnode(); push(newnode); break;

case '2' :

pop(); break;

case '3' :

display(); break;

case '4':

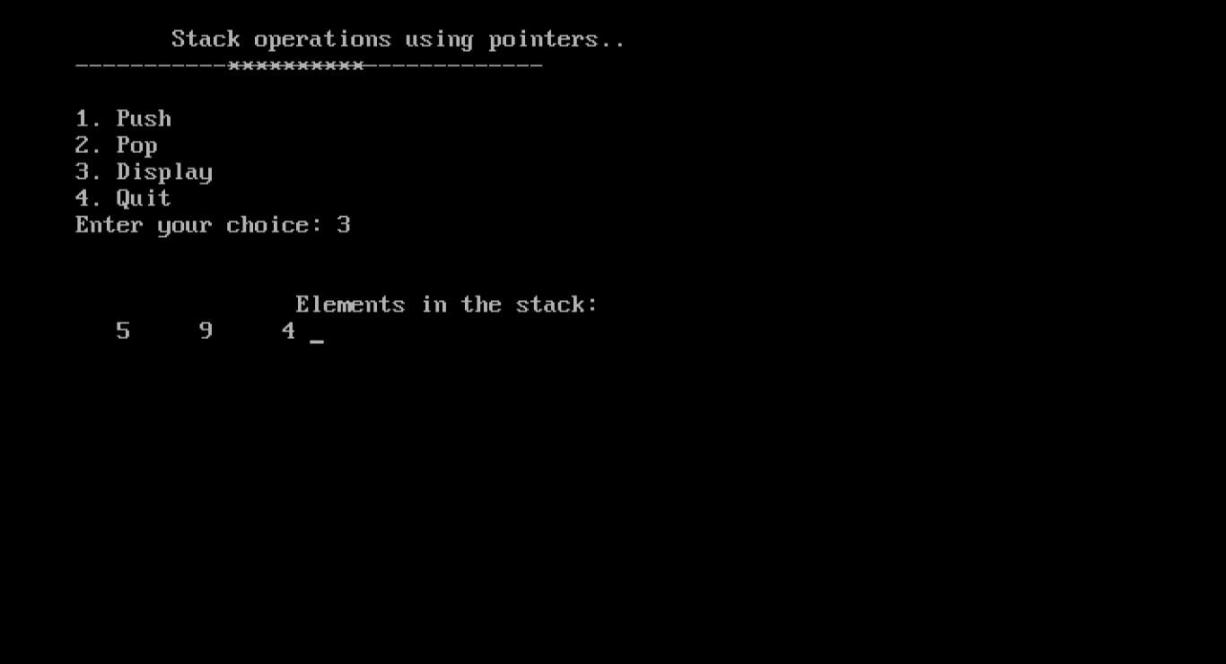
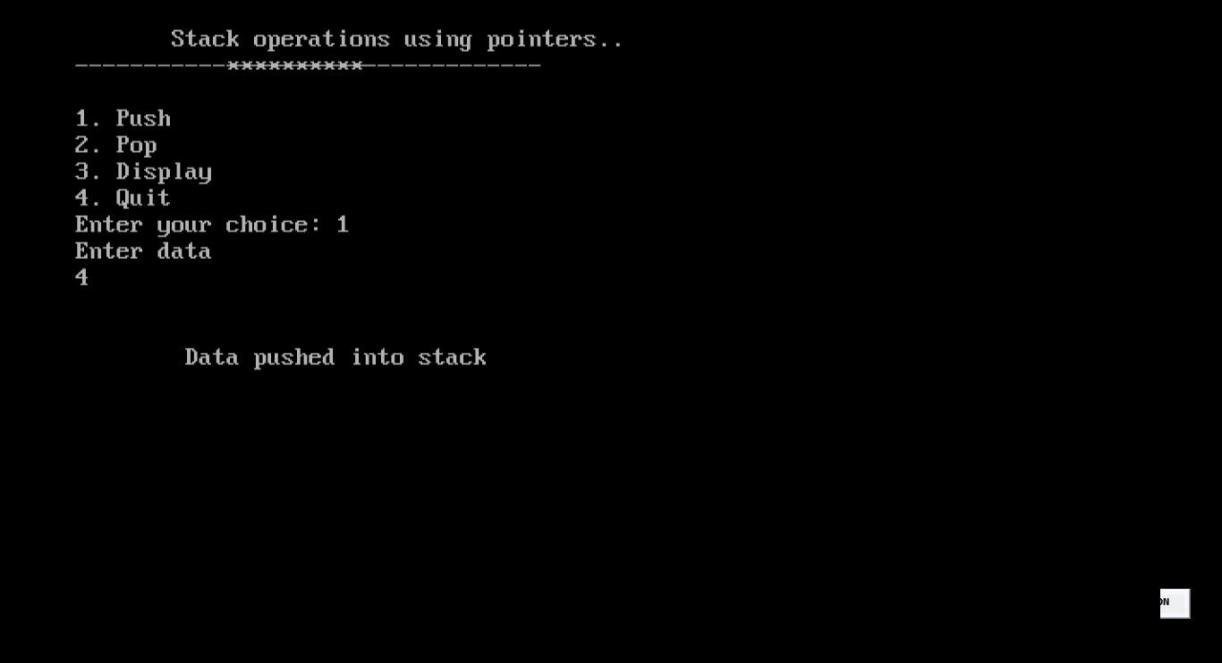
return;

}

getch();

} while( ch != '4' );

}





### Viva Questions

* 1. ***Define Stack and list operations on stack.***

### What is stack overflow?

* 1. ***What is stack underflow?***

### List any two stack applications.

* 1. ***List types of expressions.***

### Week5:

**Aim:**Write a program that implement Queue (its operations) using

* + 1. Arrays ii) Pointers

Sourcecode (Using array ) :

# include <conio.h> # define MAX 6

int Q[MAX]; int front, rear; void insertQ()

{

int data;

if(rear == MAX)

{

printf("\n Linear Queue is full"); return;

}

else

{

printf("\n Enter data: "); scanf("%d", &data); Q[rear] = data;

rear++;

printf("\n Data Inserted in the Queue ");

}

}

void deleteQ()

{

if(rear == front)

{

printf("\n\n Queue is Empty.."); return;

}

else

{

printf("\n Deleted element from Queue is %d", Q[front]); front++;

}

}

void displayQ()

{

int i;

if(front == rear)

{

printf("\n\n\t Queue is Empty"); return;

}

else

{

printf("\n Elements in Queue are: "); for(i = front; i < rear; i++)

{

printf("%d\t", Q[i]);

}

}

}

int menu()

{

int ch; clrscr();

printf("\n \tQueue operations using ARRAY.."); printf("\n -----------\*\*\*\*\*\*\*\*\*\* \n");

printf("\n 1. Insert "); printf("\n 2. Delete "); printf("\n 3. Display"); printf("\n 4. Quit ");

printf("\n Enter your choice: "); scanf("%d", &ch);

return ch;

}

void main()

{

int ch; do

{

ch = menu(); switch(ch)

{

case 1: insertQ(); break; case 2: deleteQ(); break; case 3:

displayQ(); break;

case 4: return;

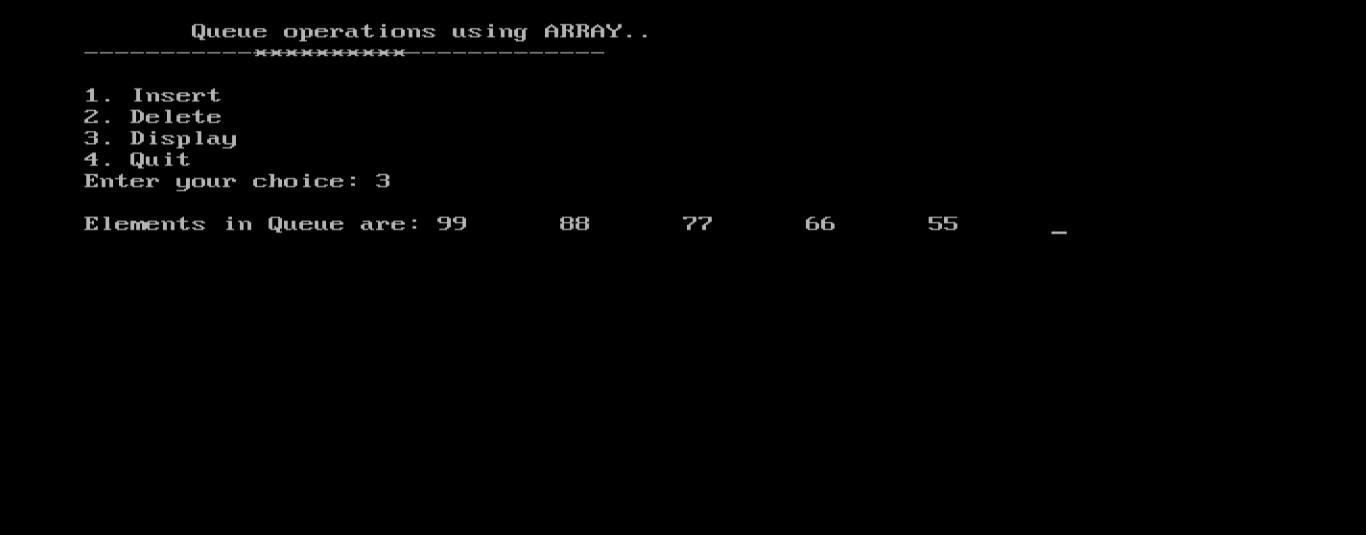
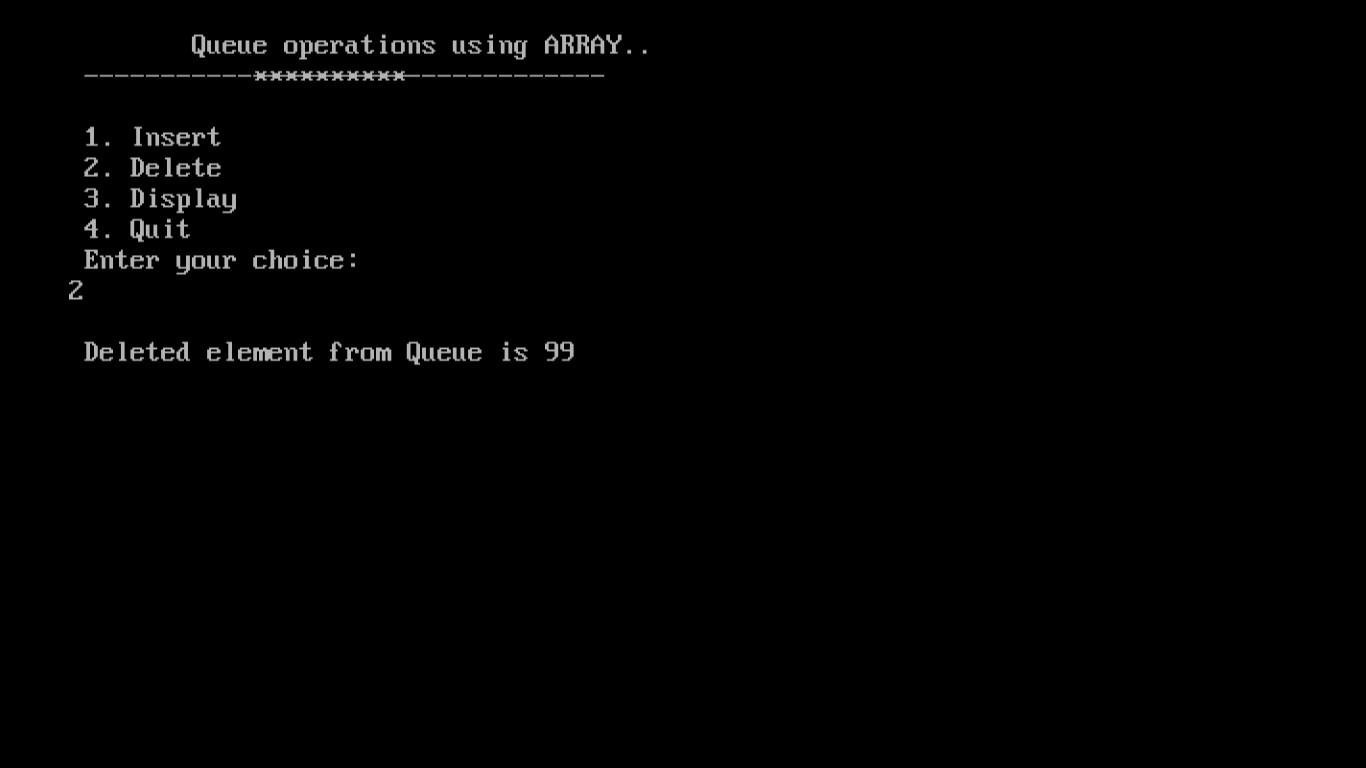
}

getch();

} while(1);

### }

***Output :***



### Source code to implement Queue operations using pointer

# include <stdlib.h> # include <conio.h> struct queue

{

int data;

struct queue \*next;

};

typedef struct queue node; node \*front = NULL; node \*rear = NULL; node\* getnode()

{

node \*temp;

temp = (node \*) malloc(sizeof(node)) ; printf("\n Enter data ");

scanf("%d", &temp -> data); temp -> next = NULL; return temp;

}

void insertQ()

{

node \*newnode; newnode= getnode(); if(newnode== NULL)

{

printf("\n Queue Full"); return;

}

if(front == NULL)

{

front = newnode; rear=newnode;

}

else

{

rear -> next = newnode; rear=newnode;

}

printf("\n\n\t Data Inserted into the Queue..");

}

void deleteQ()

{

node \*temp; if(front == NULL)

{

printf("\n\n\t Empty Queue.."); return;

}

temp = front;

front = front -> next;

printf("\n\n\t Deleted element from queue is %d ", temp -> data); free(temp);

}

void displayQ()

{

node \*temp; if(front == NULL)

{

printf("\n\n\t\t Empty Queue ");

}

else

{

temp = front;

printf("\n\n\n\t\t Elements in the Queue are: "); while(temp != NULL)

{

printf("%5d ", temp -> data); temp = temp -> next;

}

}

}

char menu()

{

char ch; clrscr();

printf("\n \t..Queue operations using pointers.. "); printf("\n\t -----------\*\*\*\*\*\*\*\*\*\* \n");

printf("\n 1. Insert "); printf("\n 2. Delete "); printf("\n 3. Display"); printf("\n 4. Quit ");

printf("\n Enter your choice: "); ch = getche();

return ch;

}

void main()

{

char ch; do

{

ch = menu(); switch(ch)

{

case '1' : insertQ(); break; case '2' : deleteQ(); break; case '3' :

displayQ(); break;

case '4':

return;

}

getch();

} while(ch != '4' }

### Viva Questions

1. ***Define Queue.***

### List the condition for queue full and queue empty.

1. ***List types of queues.***

### Write the applications of queue.

1. ***Limitation of linear queue.***

**6.Write a program that implements the following sorting methods to sort a given list of Integers in ascending order i) Quick sort ii) Heap sort iii) Merge sort**

1. **Quick sort**

#include <stdio.h>

/\* function that consider last element as pivot, place the pivot at its exact position, and place smaller elements to left of pivot and greater elements to right of pivot. \*/

int partition (int a[], int start, int end)

{

int pivot = a[end]; // pivot element

int i = (start - 1);

for (int j = start; j <= end - 1; j++)

{

// If current element is smaller than the pivot

if (a[j] < pivot)

{

i++; // increment index of smaller element

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}

int t = a[i+1];

a[i+1] = a[end];

a[end] = t;

return (i + 1);

}

/\* function to implement quick sort \*/

void quick(int a[], int start, int end) /\* a[] = array to be sorted, start = Starting index, end = Ending index \*/

{

if (start < end)

{

int p = partition(a, start, end); //p is the partitioning index

quick(a, start, p - 1);

quick(a, p + 1, end);

}

}

/\* function to print an array \*/

void printArr(int a[], int n)

{

int i;

for (i = 0; i < n; i++)

printf("%d ", a[i]);

}

int main()

{

int a[] = { 24, 9, 29, 14, 19, 27 };

int n = sizeof(a) / sizeof(a[0]);

printf("Before sorting array elements are - \n");

printArr(a, n);

quick(a, 0, n - 1);

printf("\n After sorting array elements are - \n");

printArr(a, n);

return 0;

}

1. **Heap sort :**

#include<stdio.h >

#include<conio.h>

void adjust(int i, int n, int a[])

{

int j, item; j=2 \* i; item = a[i];

while(j <= n)

{

if((j < n) && (a[j] < a[j+1])) j++;

if(item >= a[j]) break;

else

{

a[j/2] = a[j]; j=2\*j;

}

}

a[j/2] = item;

}

void heapify(int n, int a[])

{

int i;

for(i = n/2; i > 0; i--)

adjust(i, n, a);

}

void heapsort(int n,int a[])

{

int temp, i; heapify(n, a);

for(i = n; i > 0; i--)

{

temp = a[i]; a[i] = a[1]; a[1]=temp;

adjust(1, i - 1, a);

}

}

void main()

{

int i, n, a[20];

clrscr();

printf("\n How many element you want: "); scanf("%d",&n);

printf("Enter %d elements: ", n); for (i=1; i<=n; i++)

scanf("%d", &a[i]); heapsort(n,a);

printf("\n The sorted elements are: \n"); for (i=1; i<=n; i++)

printf("%5d", a[i]); getch();

***}***

**iii ) Program to implement merge sort**

#include<stdio.h> #include<conio.h>

void main( )

{

int a[20],b[20],c[20],i,j,k,temp,n;

printf(“enter the size “); scanf(“%d”,&n);

printf(“\n enter elements into array A”); for(i=0;i<n;i++)

scanf(“%d”,&a[i]);

printf(“\n enter elements into array B”); for(i=0;i<n;i++)

scanf(“%d”,&b[i]); for(i=0;i<=n-2;i++)

{

for(j=i+1;j<=n-1;j++)

{

If(a[i]>a[j])

{

temp=a[i]; a[i]=a[j]; a[j]=temp;

}

}

}

for(i=0;i<=n-2;i++)

{

for(j=i+1;j<=n-1;j++)

{

If(b[i]>b[j])

{

temp=b[i]; b[i]=b[j]; b[j]=temp;

}

}

}

for(1=j=k=0;i<(2\*n);)

{

If(a[j]<=b[k])

C[i++]=a[j++];

else

c[i++]=b[k++];

if(j==n ||

k==n) break;

}

for( ;j<n; )

c[i++]=a[j++];

for( ;k<n; )

c[i++]=b[k++];

printf(“ sorted elements (using merge sort) are \n”); for(i=0;i<(2\*n);i++)

printf(“%d” , c[i]);getch();

}

Week 7 :

Write a program to implement the tree traversal methods( Recursive and Non Recursive).

## Source code:

# include <stdio.h> # include <stdlib.h>

struct tree

{

struct tree\* lchild; char data[10]; struct tree\* rchild;

};

typedef struct tree node; node \*Q[50];

int node\_ctr;

node\* getnode()

{

node \*temp ;

temp = (node\*) malloc(sizeof(node)); printf("\n Enter Data: "); fflush(stdin);

scanf("%s",temp->data); temp->lchild = NULL; temp->rchild = NULL;

return temp;

}

void create\_binarytree(node \*root)

{

char option; node\_ctr = 1;

if( root != NULL )

{

printf("\n Node %s has Left SubTree(Y/N)",root->data); fflush(stdin);

scanf("%c",&option);

if( option=='Y' || option == 'y')

{

root->lchild = getnode(); node\_ctr++; create\_binarytree(root->lchild);

}

else

{

root->lchild = NULL; create\_binarytree(root->lchild);

}

printf("\n Node %s has Right SubTree(Y/N) ",root->data); fflush(stdin);

scanf("%c",&option);

if( option=='Y' || option == 'y')

{

root->rchild = getnode(); node\_ctr++; create\_binarytree(root->rchild);

}

else

{

root->rchild = NULL; create\_binarytree(root->rchild);

}

}

}

void make\_Queue(node \*root,int parent)

{

if(root != NULL)

{

node\_ctr++; Q[parent] = root;

make\_Queue(root->lchild,parent\*2+1); make\_Queue(root->rchild,parent\*2+2);

}

}

void inorder(node \*root)

{

if(root != NULL)

{

inorder(root->lchild); printf("%3s",root->data); inorder(root->rchild);

}

}

void preorder(node \*root)

{

if( root != NULL )

{

printf("%3s",root->data); preorder(root->lchild); preorder(root->rchild);

}

}

void postorder(node \*root)

{

if( root != NULL )

{

postorder(root->lchild); postorder(root->rchild);

printf("%3s", root->data);

}

}

void level\_order(node \*Q[],int ctr)

{

int i;

for( i = 0; i < ctr ; i++)

{

if( Q[i] != NULL )

printf("%5s",Q[i]->data);

}

}

int menu()

{

int ch; clrscr();

printf("\n 1. Create Binary Tree "); printf("\n 2. Inorder Traversal "); printf("\n 3. Preorder Traversal "); printf("\n 4. Postorder Traversal "); printf("\n 5. Level Order Traversal"); printf("\n 6. Quit ");

printf("\n Enter Your choice: "); scanf("%d", &ch);

return ch;

}

void main()

{

int i,ch;

node \*root = NULL; do

{

ch = menu(); switch( ch)

{

case 1 :

if( root == NULL )

{

root = getnode(); create\_binarytree(root);

}

else

{

printf("\n Tree is already Created ..");

}

case 2 :

break;

printf("\n Inorder Traversal: "); inorder(root);

break;

case 3 :

printf("\n Preorder Traversal: "); preorder(root);

break;

case 4 :

case 5:

case 6 :

}

printf("\n Postorder Traversal: "); postorder(root);

break;

printf("\n Level Order Traversal .."); make\_Queue(root,0); level\_order(Q,node\_ctr);

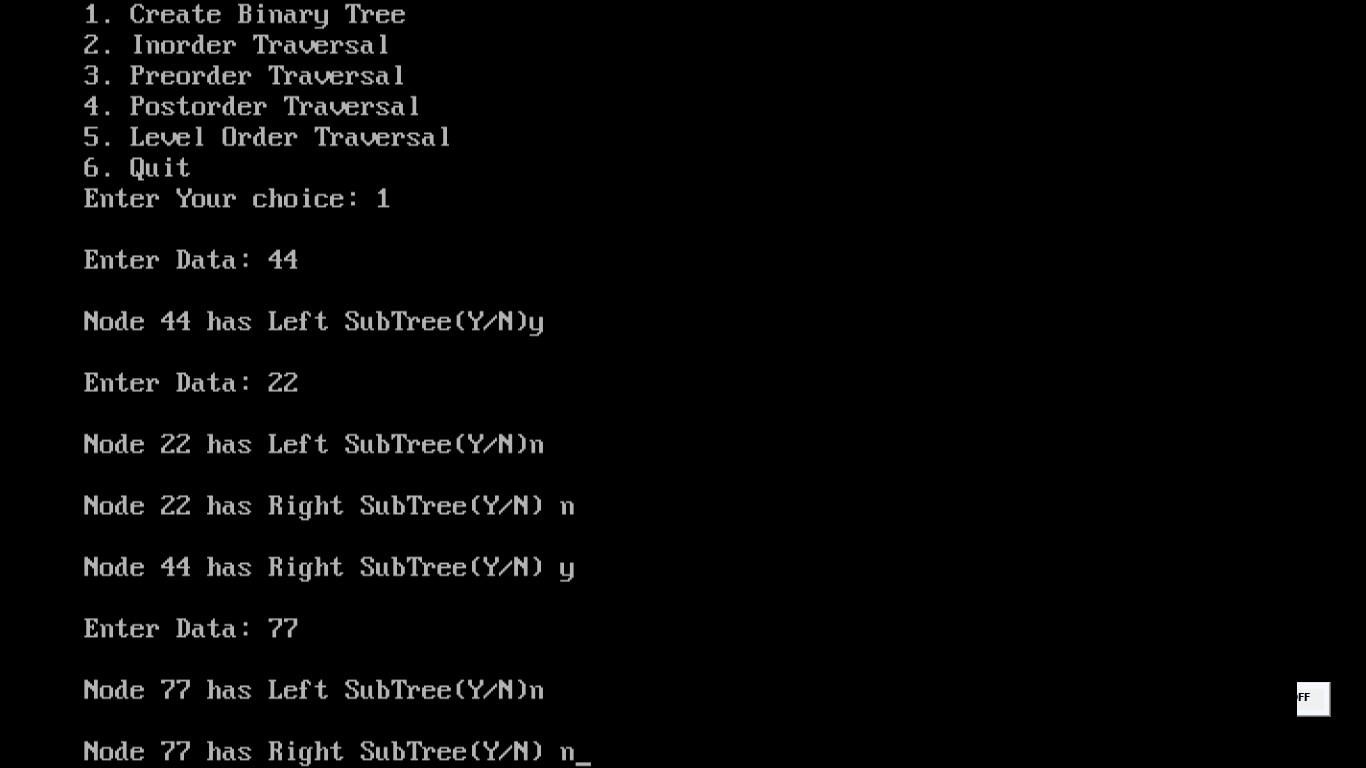
break; exit(0);

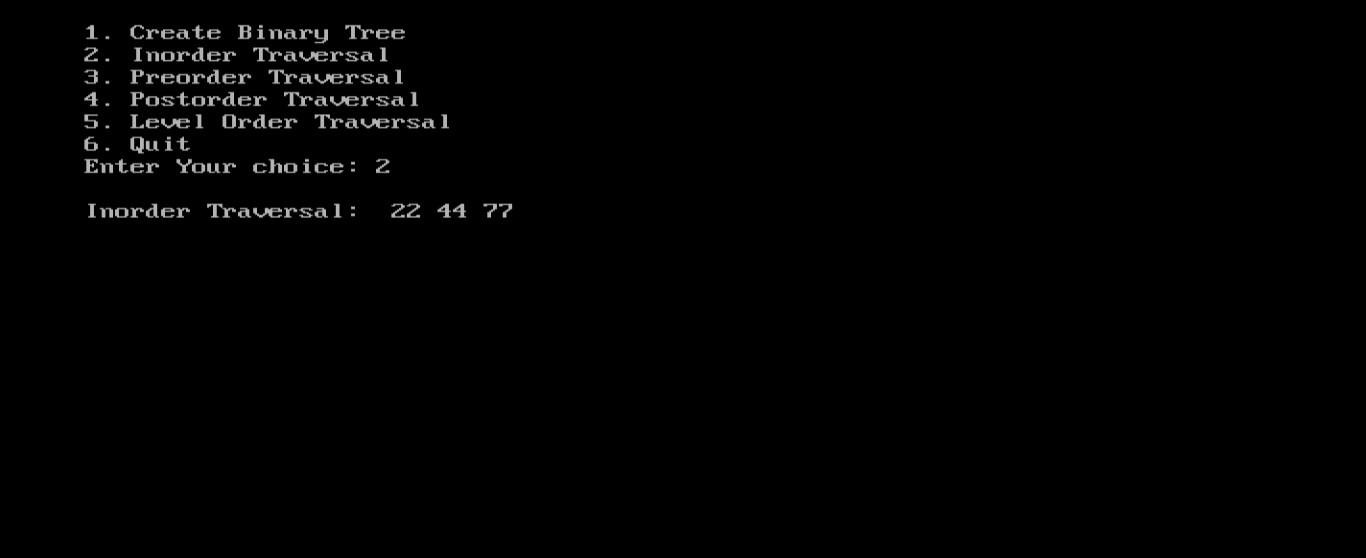
getch();

}while(1);

}

OUTPUT :







Viva Questions

* 1. Define binary tree
  2. List the properties of BST.
  3. What is complete binary tree?
  4. Give one example for Full binary tree.
  5. Give the node structure of a tree.

Week 8 : Write a program to implement

i) Binary Search tree ii) B Trees iii) B+ Trees iv) AVL trees v) Red - Black trees

# i)Program to implement Binary Search Tree

#include <stdio.h> #include <stdlib.h>

typedef struct node { int data;

struct node \*left; struct node \*right; struct node \*parent;

}node;

typedef struct binary\_search\_tree { node \*root;

}binary\_search\_tree;

node\* new\_node(int data) { node \*n = malloc(sizeof(node)); n->data = data;

n->left = NULL; n->right = NULL;

n->parent = NULL;

return n;

}

binary\_search\_tree\* new\_binary\_search\_tree() { binary\_search\_tree \*t = malloc(sizeof(binary\_search\_tree)); t->root = NULL;

return t;

}

node\* minimum(binary\_search\_tree \*t, node \*x) { while(x->left != NULL)

x = x->left; return x;

}

void insert(binary\_search\_tree \*t, node \*n) { node \*y = NULL;

node \*temp = t->root; while(temp != NULL) { y = temp;

if(n->data < temp->data) temp = temp->left;

else

temp = temp->right;

}

n->parent = y;

if(y == NULL) //newly added node is root t->root = n;

else if(n->data < y->data) y->left = n;

else

y->right = n;

}

void transplant(binary\_search\_tree \*t, node \*u, node \*v) { if(u->parent == NULL) //u is root

t->root = v;

else if(u == u->parent->left) //u is left child u->parent->left = v;

else //u is right child

u->parent->right = v;

if(v != NULL)

v->parent = u->parent;

}

void delete(binary\_search\_tree \*t, node \*z) { if(z->left == NULL) {

transplant(t, z, z->right); free(z);

}

else if(z->right == NULL) { transplant(t, z, z->left); free(z);

}

else {

node \*y = minimum(t, z->right); //minimum element in right subtree if(y->parent != z) {

transplant(t, y, y->right); y->right = z->right;

y->right->parent = y;

}

transplant(t, z, y); y->left = z->left;

y->left->parent = y; free(z);

}

}

void inorder(binary\_search\_tree \*t, node \*n)

{

if(n != NULL)

{

inorder(t, n->left);

printf("%d\n", n->data);

inorder(t, n->right);

}

}

int main()

{

binary\_search\_tree \*t = new\_binary\_search\_tree();

node \*a, \*b, \*c, \*d, \*e, \*f, \*g, \*h, \*i, \*j, \*k, \*l, \*m;

a = new\_node(10);

b = new\_node(20);

c = new\_node(30);

d = new\_node(100);

e = new\_node(90);

f = new\_node(40);

g = new\_node(50);

h = new\_node(60);

i = new\_node(70);

j = new\_node(80);

k = new\_node(150);

l = new\_node(110);

m = new\_node(120);

insert(t, a);

insert(t, b);

insert(t, c);

insert(t, d);

insert(t, e);

insert(t, f);

insert(t, g);

insert(t, h);

insert(t, i);

insert(t, j);

insert(t, k);

insert(t, l);

insert(t, m);

delete(t, a);

delete(t, m);

inorder(t, t->root); return 0;

}

# iv)AVL tree implementation

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int key;

struct Node \*left;

struct Node \*right;

int height;

};

int max(int a, int b);

// Calculate height

int height(struct Node \*N)

{

if (N == NULL)

return 0;

return N->height;

}

int max(int a, int b)

{ return (a > b) ? a : b;

}

// Create a node

struct Node \* newNode(int key)

{

struct Node \*node = (struct Node \*) malloc(sizeof(struct Node));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1;

return (node);

}

// Right rotate

struct Node \*rightRotate(struct Node \*y)

{

struct Node \*x = y->left;

struct Node \*T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

// Left rotate

struct Node \*leftRotate(struct Node \*x)

{

struct Node \*y = x->right;

struct Node \*T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

// Get the balance factor

int getBalance(struct Node \*N)

{ if (N == NULL)

return 0;

return height(N->left) - height(N->right);

}

// Insert node

struct Node \*insertNode(struct Node \*node, int key)

{

// Find the correct position to insertNode the node and insertNode

if (node == NULL)

return (newNode(key));

if (key < node->key)

node->left = insertNode(node->left, key);

else if (key > node->key)

node->right = insertNode(node->right, key);

else

return node;

// Update the balance factor of each node and

// Balance the tree

node->height = 1 + max(height(node->left),height(node->right));

int balance = getBalance(node);

if (balance > 1 && key < node->left->key)

return rightRotate(node);

if (balance < -1 && key > node->right->key)

return leftRotate(node);

if (balance > 1 && key > node->left->key)

{

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && key < node->right->key)

{

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

struct Node \*minValueNode(struct Node \*node)

{

struct Node \*current = node;

while (current->left != NULL)

current = current->left;

return current;

}

// Delete a nodes

struct Node \*deleteNode(struct Node \*root, int key)

{

// Find the node and delete it

if (root == NULL)

return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else

{

if ((root->left == NULL) || (root->right == NULL))

{

struct Node \*temp = root->left ? root->left : root->right;

if (temp == NULL)

{

temp = root;

root = NULL;

}

else

\*root = temp;

free(temp);

}

else

{

struct Node \*temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

}

if (root == NULL) return root;

// Update the balance factor of each node and

// balance the tree

root->height = 1 + max(height(root->left), height(root->right));

int balance = getBalance(root);

if (balance > 1 && getBalance(root->left) >= 0)

return rightRotate(root);

if (balance > 1 && getBalance(root->left) < 0)

{ root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && getBalance(root->right) <= 0)

return leftRotate(root);

if (balance < -1 && getBalance(root->right) > 0)

{ root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

// Print the tree

void printPreOrder(struct Node \*root)

{

if (root != NULL)

{

printf("%d ", root->key); printPreOrder(root->left); printPreOrder(root->right);

}

}

int main()

{

struct Node \*root = NULL;

root = insertNode(root, 2);

root = insertNode(root, 1);

root = insertNode(root, 7);

root = insertNode(root, 4);

root = insertNode(root, 5);

root = insertNode(root, 3);

root = insertNode(root, 8);

printPreOrder(root);

root = deleteNode(root, 3);

printf("\nAfter deletion: "); printPreOrder(root);

return 0;

}

Week 9 : **Write a program to implement the graph traversal methods.**

**// 9a) Breadth first search**

#include<stdio.h>

#include<conio.h>

int a[20][20],q[20],visited[20];

int n,i,j,f=0,r=-1;

void bfs(int v)

{

for(i=1;i<=n;i++)

if(a[v][i]&&!visited[i])

q[++r]=i;

if(f<=r)

{

visited[q[f]]=1;

printf("%d---",q[f]);

bfs(q[f++]);

}

}

void main()

{

int v;

clrscr();

printf("\n enter number of vertices");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

q[i]=0;

visited[i]=0;

}

printf("\n enter graph data in adjancet matrix form");

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

scanf("%d",&a[i][j]);

printf("\n enter source verter");

scanf("%d",&v);

visited[v]=1;

printf("\n bfs traversal of vertices:\n");

printf("\n %d---",v);

bfs(v);

getch();

}

//week 9 b) Depth First Search

#include<stdio.h>

#include<conio.h>

void dfs(int);

int a[10][10],visited[10],n,s;

void main()

{

int i,j;

printf("enter number of vertices");

scanf("\d",&n);

for(i=0;i<n;i++)

for(j=0;j<n;j++)

scanf("%d",&a[i][j]);

for(i=0;i<n;i++)

visited[i]=0;

printf("\n enter source vertex");

scanf("%d",&s);

dfs(s);

}

void dfs(int i)

{

int j;

printf("\n %d",i);

visited[i]=1;

for(j=0;j<n;j++)

if((!visited[j])&&(a[i][j]==1))

dfs(j);

}

*Week 9 : Viva questions*

1. Define graph.
2. List graph representations.
3. What is directed graph?
4. What is back edge, forward edge and cross edge?
5. What is weighted graph?

Week 10 :

**Implement a Pattern matching algorithms using Boyer- Moore, Knuth-Morris-Pratt**

// 10 a) Boyer- Moore

# include <limits.h>

# include <string.h>

# include <stdio.h>

# define NO\_OF\_CHARS 256

// A utility function to get maximum of two integers

int max(int a, int b)

{

return (a > b) ? a : b;

}

// The preprocessing function for Boyer Moore's bad character heuristic

void badCharHeuristic(char \*str, int size, int badchar[NO\_OF\_CHARS])

{

int i;

// Initialize all occurrences as -1

for (i = 0; i < NO\_OF\_CHARS; i++)

badchar[i] = -1;

// Fill the actual value of last occurrence of a character

for (i = 0; i < size; i++)

badchar[(int) str[i]] = i;

}

void search(char \*txt, char \*pat)

{

int m = strlen(pat);

int n = strlen(txt);

int badchar[NO\_OF\_CHARS];

badCharHeuristic(pat, m, badchar);

int s = 0; // s is shift of the pattern with respect to text

while (s <= (n - m))

{

int j = m - 1;

while (j >= 0 && pat[j] == txt[s + j])

j--;

if (j < 0)

{

printf("\n pattern occurs at shift = %d", s);

s += (s + m < n) ? m - badchar[txt[s + m]] : 1;

}

else

s += max(1, j - badchar[txt[s + j]]);

}

}

int main()

{

char txt[] = "ABAAABCD";

char pat[] = "ABC";

search(txt, pat);

return 0;

}

10 b) KNUTH MORRIS PRATT ALGORITHM

#include<stdio.h>

#include<string.h>

char txt[100],pat[100];

int M ,N ,lps[100],j=0,i=0;

void computeLPSArray()

{

int len = 0, i;

lps[0] = 0;

i = 1;

while(i < M)

{

if(pat[i] == pat[len])

{

len++;

lps[i] = len;

i++;

}

else

{

if( len != 0 )

len = lps[len-1];

else

{

lps[i] = 0;

i++;

}

}

}

}

void KMPSearch()

{

int j=0,i=0;

M = strlen(pat);

N = strlen(txt);

computeLPSArray();

while(i < N)

{

if(pat[j] == txt[i])

{

j++;

i++;

}

if (j == M)

{

printf("Found pattern at index %d \n", i-j);

j = lps[j-1];

}

else if(pat[j] != txt[i])

{

if(j != 0)

j = lps[j-1];

else

i = i+1;

}

}

}

int main()

{

printf("\n ENTER THE TEXT : ");

gets(txt);

printf("\n ENTER THE PATTERN : ");

gets(pat);

KMPSearch();

return 0;

}

**output:-**

ENTER THE TEXT : Welcome To CampusCoke

ENTER THE PATTERN : C

Found pattern at index 11

Found pattern at index 17

***Additional programs***

**Program to evaluate a postfix expression**

#include <conio.h> # include <math.h> # define MAX 20

int isoperator(char ch)

{

if(ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^') return 1;

else return 0;

}

void main(void)

{

char postfix[MAX]; int val;

char ch;

int i = 0, top = 0;

float val\_stack[MAX], val1, val2, res; clrscr();

printf("\n Enter a postfix expression: ); scanf("%s", postfix);

while((ch = postfix[i]) != '\0')

{

if(isoperator(ch) == 1)

{

val2= val\_stack[--top]; val1= val\_stack[--top]; switch(ch)

{

case '+':

res = val1 + val2; break;

case '-':

res = val1 - val2; break;

case '\*':

res = val1 \* val2; break;

case '/':

res = val1 / val2; break;

case '^':

res = pow(val1, val2); break;

}

val\_stack[top] = res;

}

else

val\_stack[top] = ch-48;

/\*convert character digit to integer digit \*/

top++;

i++;

}

printf("\n Values of %s is : %f ",postfix, val\_stack[0] ); getch(); ***}***