Sri Indu
College of Engineering & Technology
UGC Autonomous Institution
Recognized under 2(f) & 12(B) of UGC Act 1956,
NAAC, Approved by AICTE &
Permanently Affiliated to JNTUH
Estd.2001

NAAC
NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL

HANDOUT

II Year - I Semester

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

ACADEMIC YEAR 2025-2026

INSTITUTION VISION

To be a premier Institution in Engineering & Technology and Management with competency, values and social consciousness.

INSTITUTION MISSION

IM1 Provide high quality academic programs, training activities and research facilities.

IM2 Promote Continuous Industry-Institute interaction for employability, Entrepreneurship, leadership and research aptitude among stakeholders.

IM3 Contribute to the economical and technological development of the region, state and nation.

DEPARTMENT VISION

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

DEPARTMENT MISSION

The Department has following Missions:

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 stake holders in research and development.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: **Higher Studies:** Graduates with an ability to pursue higher studies and get employment in reputed institutions and organizations.

PEO2: **Domain knowledge:** Graduate with an ability to design and develop a product.

PEO3: **Professional Career:** Graduate with an ability to design and develop a product.

PEO4: **Life Long Learning:** Graduate with an ability to learn advanced skills to face professional competence through lifelong learning.

PROGRAMOUTCOMES(POs)

PO	Description
PO 1	**Engineering Knowledge:** To be able to apply knowledge of computing, mathematics, Science and Engineering appropriate to the discipline
PO 2	**Problem Analysis:** To be able identify, formulate & analyze a problem, and ascertain and define the computing requirements appropriate to its solution.
PO 3	**Design & Development Solutions:** To be able to design, implement, and evaluatea computer-based system, process, component, or program to meet desired needs.
PO 4	**Investigation of complex problems:** To be able to identify and analyze user needs And consider them in the selection, creation, evaluation and administration of computer-based systems for providing valid solutions to complex problems.
PO 5	**Modern Tool Usage:** To posses skills for creating and in using contemporary techniques, skills, and tools necessary for computing Practice.
PO 6	**Engineering&Society:**Toapplyconceptualknowledgerelevanttoprofessionalengineeringpracticesinsocietal,health,safety,legalandculturalissuesandtheirconsequences
PO 7	**Environment &Sustainability:** To be able to Analyze the local and global impact of computing on individuals, organizations, and society and work towards sustainable development.
PO 8	**Ethics:** To understand contemporary professional, ethical, legal, security and social issue sand responsibilities.
PO 9	**Individual & Team work:** To Be able to function effectively as an individual andon teams to accomplish a common goal.
PO 10	**Communication:** To communicate precisely and effectively both in oral andwritten form with a range of audiences.
PO 11	**Project management & finance:** To apply engineering and management principles For managing and leading economically feasible projects in multi-disciplinaryenvironments with an effective project plan.
PO 12	**Life Long Learning:** To recognize the need for and an ability to engage in independent& lifelong learning for continuing professional development.
Program Specific Outcomes	
PSO 1	Develop software projects using standard practices and suitable programming environment.
PSO 2	Identify , formulate and solve the real life problems faced in the society, industry and other areas by applying the skills of the programming languages, networks and databases learned.
PSO 3	To apply computer science knowledge in exploring and adopting latest technologies in different co-curricular activities.

COURSE OUTCOMES

C216.1	Implement data link layer farming methods and Analyze error detection and error correction codes.
C216.2	Implement and analyze routing and congestion issues in network design.
C216.3	Implement Encoding and Decoding techniques used in presentation layer.

COsMAPPINGWITHPOs&PSOs

Course Outcome	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO11	PO 12	PSO 1	PSO2	PSO3
C216.1	2	2	3	2	2	-	-	-	-	1	-	-	1	2	1
C216.2	2	1	2	1	2	-	-	-	-	-	2	-	1	1	1
C216.3	1	2	1	2	1	-	-	--	-	-	-	-	2	1	1
C216	**1.6**	**1.6**	**2.0**	**1.6**	**1.6**	**-**	**-**	**-**	**-**	**0.3**	**0.6**	**-**	**1.3**	**1.3**	**1.16**

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), thosewho come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with thesynopsis / program / experiment details.
3. Student should enter into the laboratory with:
   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure,Program, Expected Output, etc.,) filled in for the lab session.
   b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed inthe lab.
   c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allottedto you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book,and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain thediscipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should beutilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severepunishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody foundloitering outside the lab / class without permission during working hours will be treated seriously andpunished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, New Delhi)

B.Tech II year – I Sem

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 linked list.:
i) Creation ii) Insertion iii) Deletion iv) Traversal

2. Write a program that uses functions to perform the following operations on doubly linked list.:
i) Creation ii) Insertion iii) Deletion iv) Traversal

3. Write a program that uses functions to perform the following operations on circular linked list.:
i) Creation ii) Insertion iii) Deletion iv) Traversal

4. Write a program that implement stack (its operations) using
i) Arrays ii) Pointers

5. Write a program that implement Queue (its operations) using
i) Arrays ii) Pointers

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

7. Write a program to implement the tree traversal methods.(Recurssive and non Recurssive)

8. Write a program to implement
i)Bnary 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

TEXTBOOKS:

1. Fundamentals of Data Structures in C, 2nd Edition, E. Horowitz, S. Sahni and Susan Anderson Freed, Universities Press.
2. Data Structures using C – A. S. Tanenbaum, Y. Langsam, and M. J. Augenstein, PHI/PearsonEducation.

REFERENCE:

1. Data Structures: A Pseudocode Approach with C, 2nd Edition, R. F. Gilberg and B. A. Forouzan,Cengage Learning

LabPlan

Sno	Topics	No. Of weeks
1.	1. Write a program that uses functions to perform the following operations on singly linked list.: i) Creation ii) Insertion iii) Deletion iv) Traversal	2
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	2
5.	Write a program that implement Queue (its operations) using i) Arrays ii) Pointers	2
6.	. Write a program that implements the following sorting methods to sort a given list of integers inascending order i) Quick sort ii) Heap sort iii) Merge sort	2
7.	Write a program to implement the tree traversal methods.(Recurssive and non Recurssive)	2
8.	Write a program to implement i) Binary search ii)B Trees iii)B+ Trees iv)AVL Trees v)Red-Black Trees.	1
9.	Write a program to implement the graph traversal methods.	1
10	Implement a Pattern Matching Algorithms using Boyer-Moore,Knuth-Morris-Pratt	
	ADDITIONAL PROGRAMS	
1	Program to convert infix expression into postfix expression	1
2	Program to evaluate Postfix expression	1
3	Program to implement Heap sort	1
4	Program to implement merge sort	1
5	Program to implement pattern matching algorithm	1

Data Structures Lab Manual

(Subject Code : R22CSE2126)

DEPARTMENT OF INFORMATION TECHNOLOGY

Lab Manual

Data Structures Lab

1. Write a program that uses functions to perform the following operations on singly linked list.:
 i) Creation ii) Insertion iii) Deletion iv) Traversal

2. Write a program that uses functions to perform the following operations on doubly linked list.:
 i) Creation ii) Insertion iii) Deletion iv) Traversal

3. Write a program that uses functions to perform the following operations on circular linked list.:
 i) Creation ii) Insertion iii) Deletion iv) Traversal

4. Write a program that implement stack (its operations) using
 i) Arrays ii) Pointers

5. Write a program that implement Queue (its operations) using
 i) Arrays ii) Pointers

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

7. Write a program to implement the tree traversal methods

8. Write a program to implement
 i) Binary search 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 algorithm using Boyer-Moore,Knuth-Morris-Pratt

PROGRAMS

Week1.

Aim: Write a program that uses functions to perform the following operations on singly linked list.:
i) 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)
        {
```

```
        temp = prev = start;
        while(ctr < pos)
        {
        prev = temp;
        temp = temp -> next;
        ctr++;
        }
        prev -> next = newnode;
        newnode -> next = temp;
        }
else
        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()
{
        node *temp, *prev;
        if(start == NULL)
        {
        printf("\n Empty List..");
return ;
        }
        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);
```

```
                printf("\n List created..");
        }
        else
                printf("\n List is alreadycreated..");
        break;
  case 2:
  insert_at_beg();
        break;
  case 3:
        insert_at_end();
        break;
  case 4:
        insert_at_mid();
        break;

  case 5:
        delete_at_beg();
        break;
  case 6:
        delete_at_last();
        break;
  case 7:
        delete_at_mid();
        break;
  case 8:
        traverse();
        break;
  case 9:
        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();
}
}
```

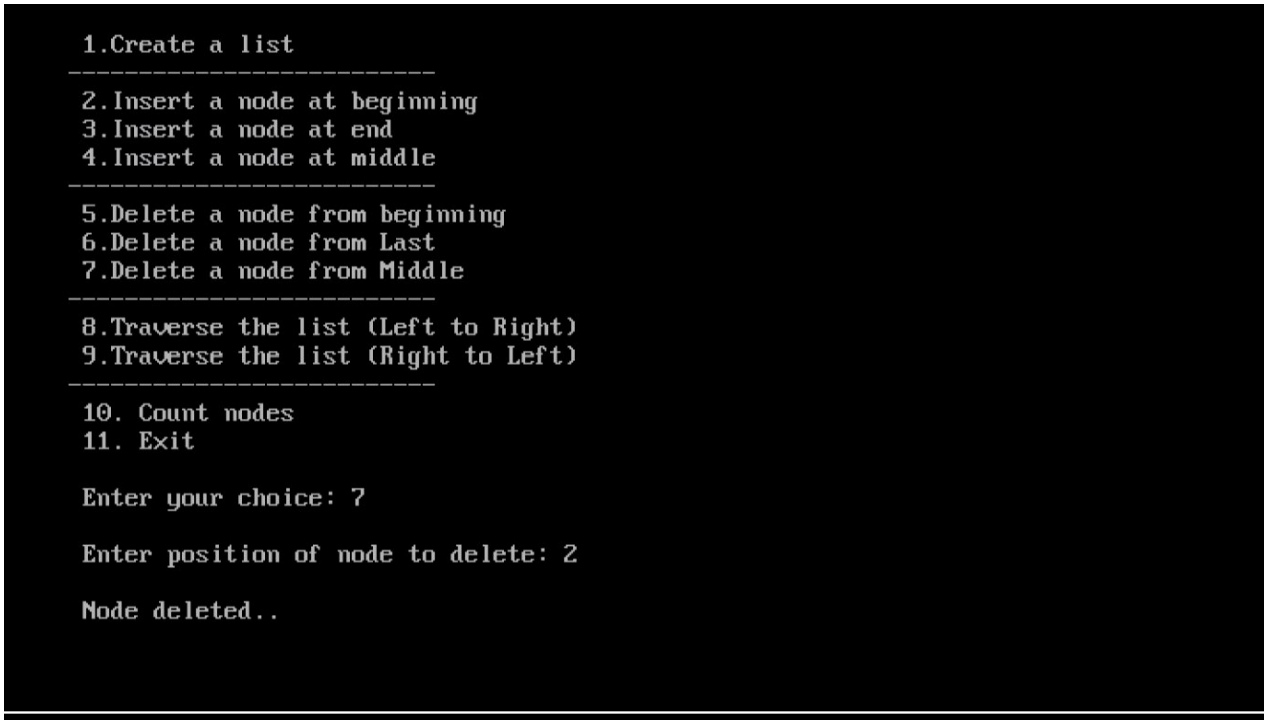
OUTPUT :

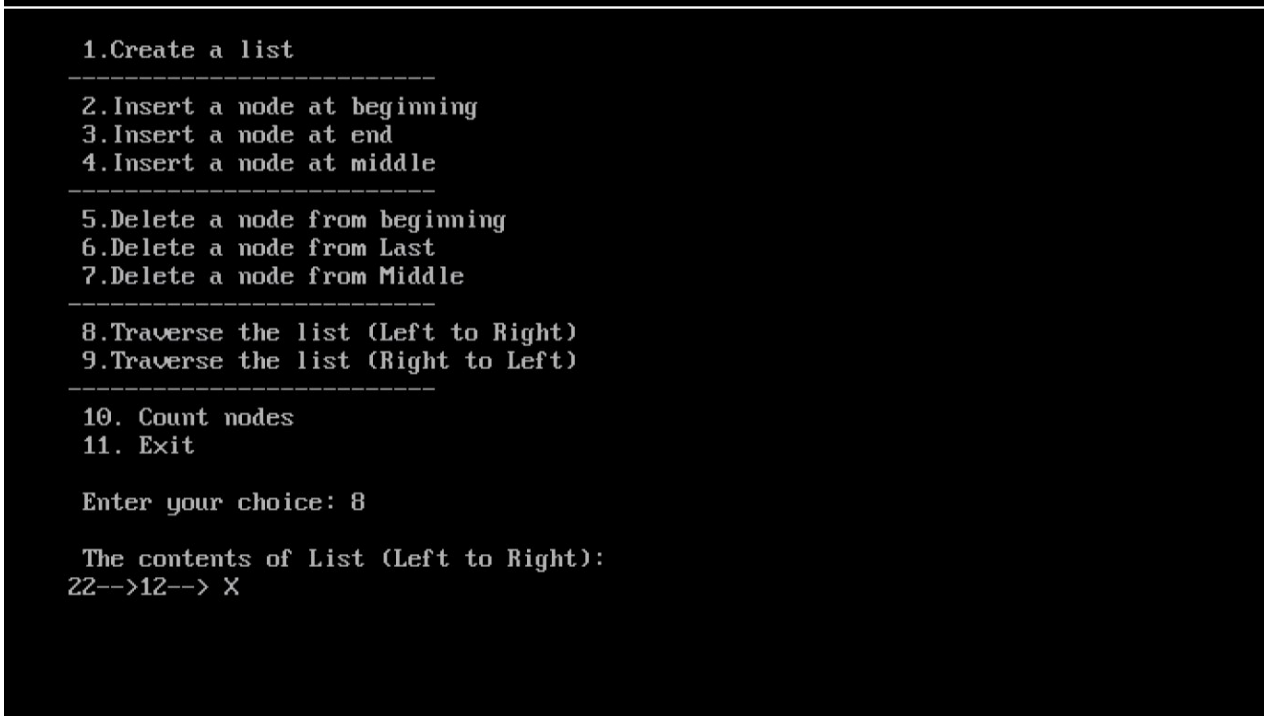
```
 1.Create a list
--------------------------
 2.Insert a node at beginning
 3.Insert a node at end
 4.Insert a node at middle
--------------------------
 5.Delete a node from beginning
 6.Delete a node from Last
 7.Delete a node from Middle
--------------------------
 8.Traverse the list (Left to Right)
 9.Traverse the list (Right to Left)
--------------------------
 10. Count nodes
 11. Exit

 Enter your choice: 1

 Number of nodes you want to create: 2

 Enter data: 11

 Enter data: 12

 List created..
```

```
 1.Create a list
--------------------------
 2.Insert a node at beginning
 3.Insert a node at end
 4.Insert a node at middle
--------------------------
 5.Delete a node from beginning
 6.Delete a node from Last
 7.Delete a node from Middle
--------------------------
 8.Traverse the list (Left to Right)
 9.Traverse the list (Right to Left)
--------------------------
 10. Count nodes
 11. Exit

 Enter your choice: 8

 The contents of List (Left to Right):
11-->12--> X
```

```
 1.Create a list
--------------------------
 2.Insert a node at beginning
 3.Insert a node at end
 4.Insert a node at middle
--------------------------
 5.Delete a node from beginning
 6.Delete a node from Last
 7.Delete a node from Middle
--------------------------
 8.Traverse the list (Left to Right)
 9.Traverse the list (Right to Left)
--------------------------
 10. Count nodes
 11. Exit

 Enter your choice: 2

 Enter data: 22
_
```

```
 1.Create a list
--------------------------
 2.Insert a node at beginning
 3.Insert a node at end
 4.Insert a node at middle
--------------------------
 5.Delete a node from beginning
 6.Delete a node from Last
 7.Delete a node from Middle
--------------------------
 8.Traverse the list (Left to Right)
 9.Traverse the list (Right to Left)
--------------------------
 10. Count nodes
 11. Exit

 Enter your choice: 8

 The contents of List (Left to Right):
22-->11-->12--> X
```





Week I Viva Questions

1. **Define self referential structure and give one example**
2. **Draw one example node of single linked list**
3. **What is NULL?**
4. **List out the operations on linked list.**
5. **Differentiate Array and linked list.**

Week2:

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

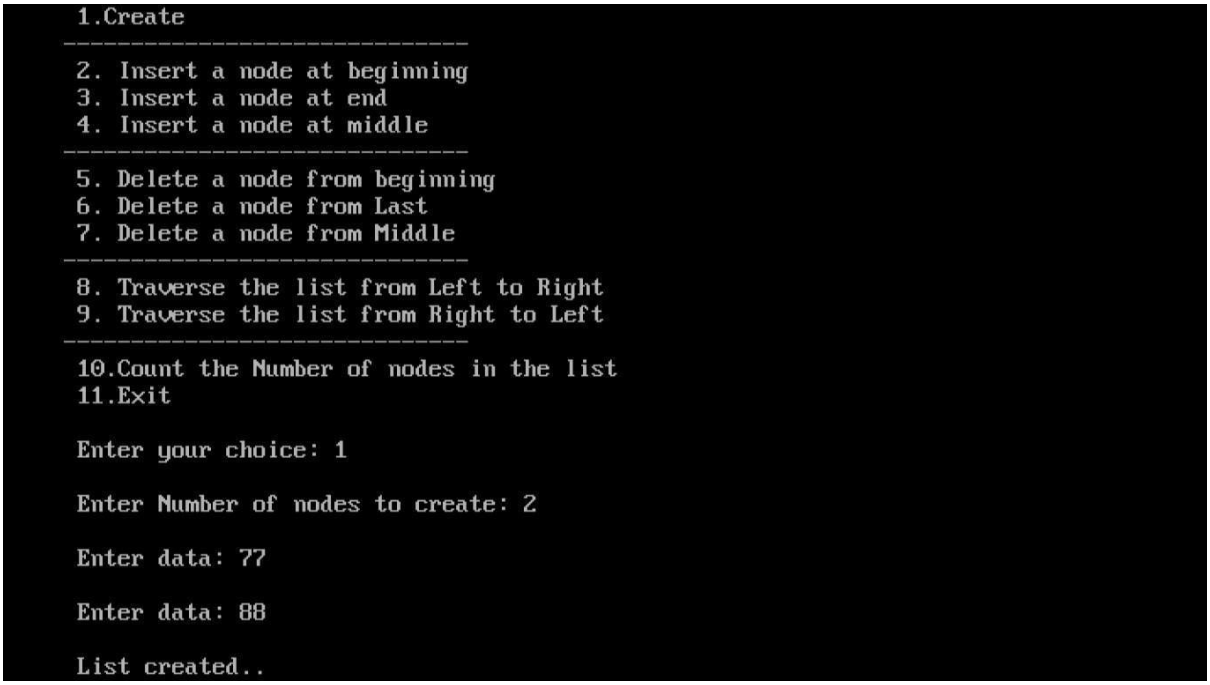
```
temp = temp -> right;
}
while(temp != NULL)
{
printf("\t%d", temp -> data);
temp = temp -> left;
}
}
void dll_insert_beg()
{
node *newnode;
newnode = getnode();
if(start == NULL)
start = newnode;
else
{
newnode -> right = start;
start -> left = newnode;
start = newnode;
}
}
void dll_insert_end()
{
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 Emptylist");
getch();
return ;
}
else
{
temp = start;
start = start -> right;
start -> left = NULL;
free(temp);
}
}
void dll_delete_last()
{
node *temp;
if(start == NULL)
{
printf("\n Emptylist");
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.***
2. ***Difference between single and double linked list.***
3. ***Define node structure of double linked list.***
4. ***What is traversal?***
5. ***List types of Linked lists.***

Week3:

Aim: Write a program that uses functions to perform the following operations on circular linked list.:
i)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..");
```

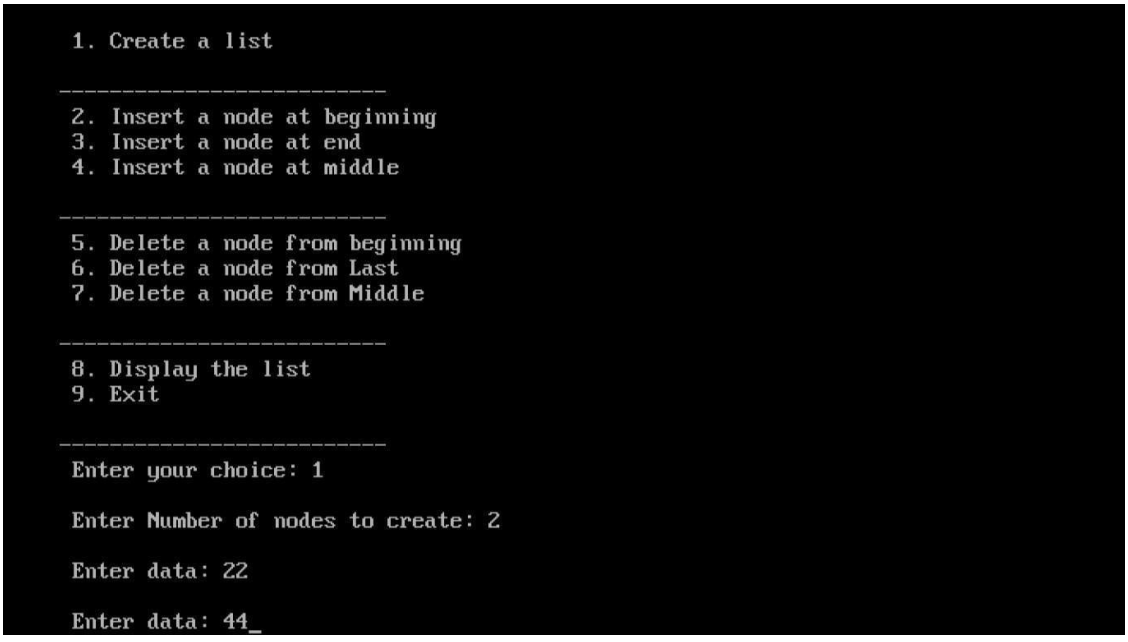
```
nodectr++;
}
void cll_insert_end()
{
node *newnode, *temp;
newnode = getnode();
if(start == NULL )
{
start = newnode;
newnode -> next = start;
}
else
{
temp = start;
while(temp -> next != start)
temp = temp -> next;
temp -> next = newnode;
newnode -> next = start;
}
printf("\n Node inserted at end..");
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..");
```

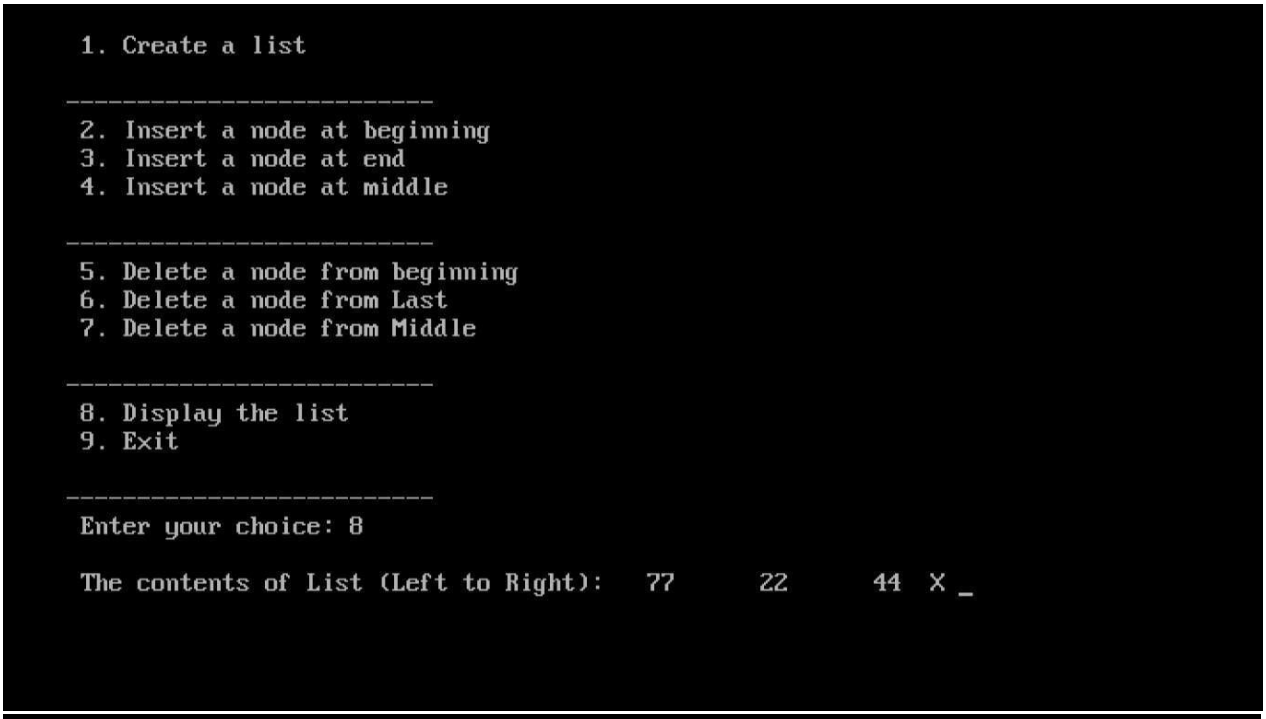
```
break;
case 2 :
cll_insert_beg();
break;
case 3 :
cll_insert_end();
break;
case 4 :
cll_insert_mid();
break;
case 5 :
cll_delete_beg();
break;
case 6 :
cll_delete_last();
break;
case 7 :
cll_delete_mid();
break;
case 8 :
display();
break;
case 9 :
exit(0);
}
getch();
}
 }
```

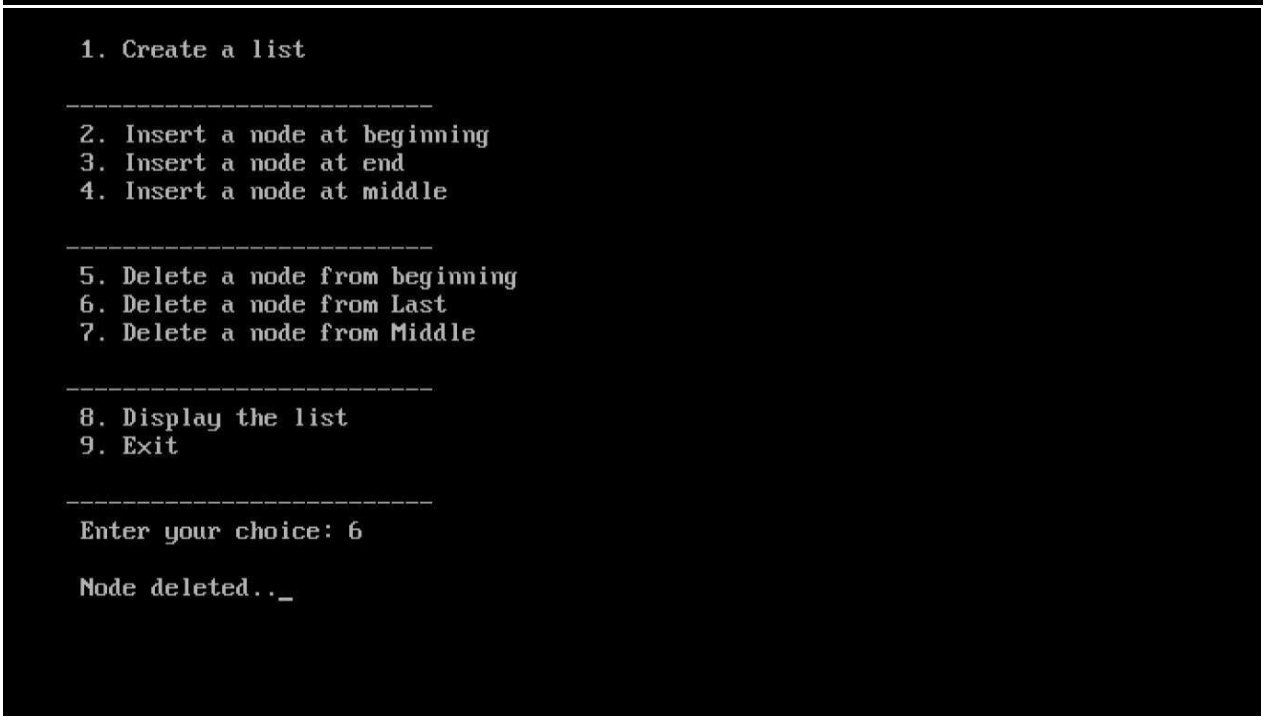
Output :



```
 1. Create a list

--------------------------
 2. Insert a node at beginning
 3. Insert a node at end
 4. Insert a node at middle

--------------------------
 5. Delete a node from beginning
 6. Delete a node from Last
 7. Delete a node from Middle

--------------------------
 8. Display the list
 9. Exit

--------------------------
 Enter your choice: 2

 Enter data: 77

 Node inserted at beginning..
```

```
 1. Create a list

--------------------------
 2. Insert a node at beginning
 3. Insert a node at end
 4. Insert a node at middle

--------------------------
 5. Delete a node from beginning
 6. Delete a node from Last
 7. Delete a node from Middle

--------------------------
 8. Display the list
 9. Exit

--------------------------
 Enter your choice: 8

 The contents of List (Left to Right):   77      22      44  X
```





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

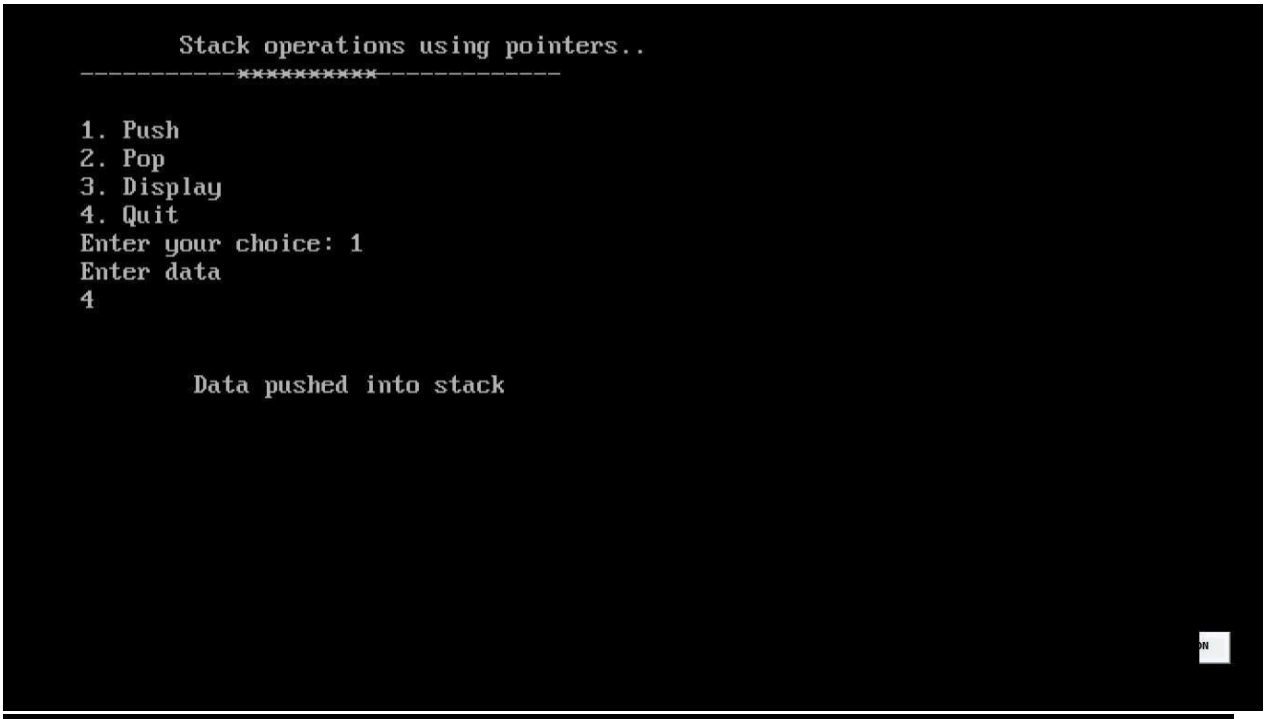
Source code to implement Stack using linked list :

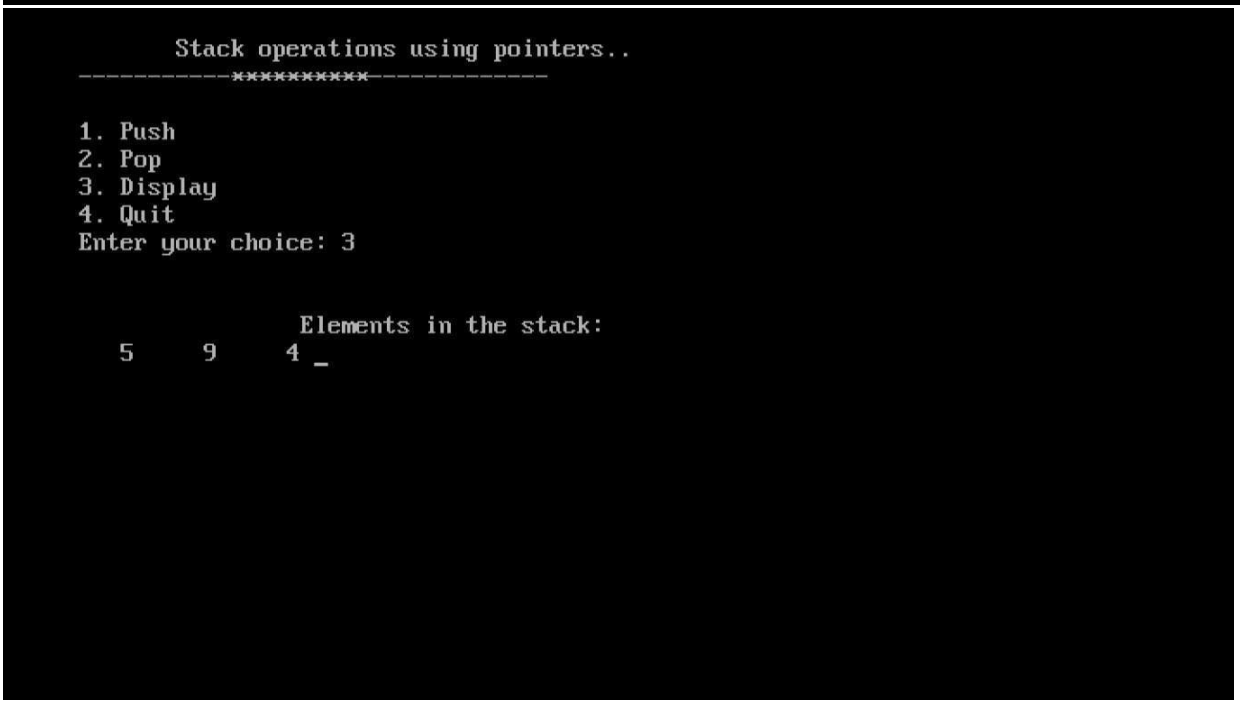
```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
struct stack
{
int data;
struct stack *next;
};
void push();
void pop();
void display();
typedef struct stack node;
node *start=NULL;
node *top = NULL;
node* getnode()
{
node *temp;
temp=(node *) malloc( sizeof(node)) ;
printf("\n Enter 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");
```

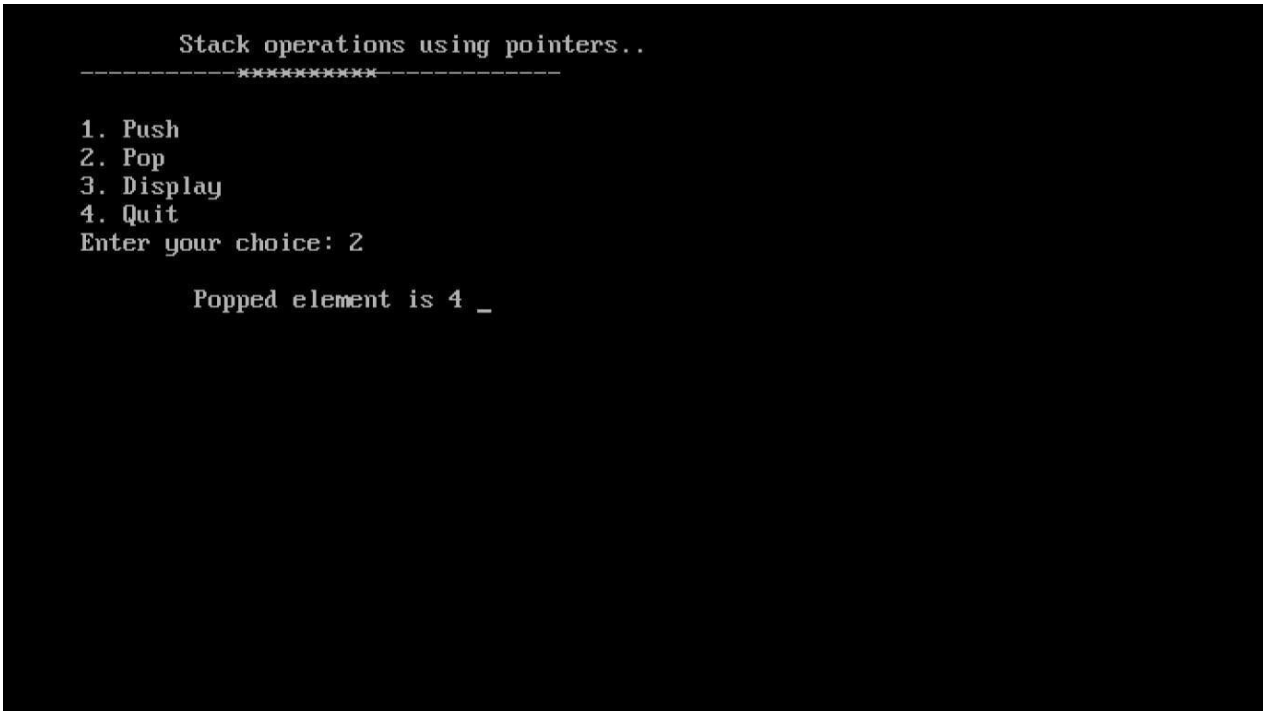
```
}
void pop()
{
node *temp;
if(top == NULL)
{
printf("\n\n\t Stack underflow");
return;
}
temp = start;
if( start -> next == NULL)
{
printf("\n\n\t Popped element is %d ", top -> data);
start = NULL;
free(top);
top = NULL;
}
else
{
while(temp -> next != top)
{
temp = temp -> next;
}
temp -> next = 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' );
}
```

OUTPUT :


Stack operations using pointers..
1. Push
2. Pop
3. Display
4. Quit
Enter your choice: 1
Enter data
4
Data pushed into stack


Stack operations using pointers..
1. Push
2. Pop
3. Display
4. Quit
Enter your choice: 3
Elements in the stack:
5 9 4



Viva Questions

1. ***Define Stack and list operations on stack.***
2. ***What is stack overflow?***
3. ***What is stack underflow?***
4. ***List any two stack applications.***
5. ***List types of expressions.***

Week5:

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

i) 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 :

```
       Queue operations using ARRAY..
----------**********-------------

1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 1

Enter data: 99

Data Inserted in the Queue _
```

```
       Queue operations using ARRAY..
----------**********-------------

1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 3

Elements in Queue are: 99      88      77      66      55      _
```

```
       Queue operations using ARRAY..
----------**********-------------

1. Insert
2. Delete
3. Display
4. Quit
Enter your choice:
2

Deleted element from Queue is 99
```

ii)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.***
2. ***List the condition for queue full and queue empty.***
3. ***List types of queues.***
4. ***Write the applications of queue.***
5. ***Limitation of linear queue.***

Week6:

Aim: 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

i)Quick sort

```
#include<stdio.h>
void quicksort(int number[25],int first,int
  last){ int i, j, pivot, temp;
  if(first<last){
    pivot=first;
    i=first;
    j=last;
    while(i<j){
      while(number[i]<=number[pivot]&&i<last)
      i++;
      while(number[j]>number[pivot])
      j--;
      if(i<j){  temp=number[i
        ];
        number[i]=number[j];
        number[j]=temp;
      }
    }
    temp=number[pivot];
    number[pivot]=number[j];
    number[j]=temp;
    quicksort(number,first,j-1);
    quicksort(number,j+1,last);
  }
}
int main(){
  int i, count, number[25];
  printf("How many elements are u going to enter?: ");
  scanf("%d",&count);
  printf("Enter %d elements: ", count);
  for(i=0;i<count;i++)
  scanf("%d",&number[i]);
  quicksort(number,0,count-1);
  printf("Order of Sorted elements: ");
  for(i=0;i<count;i++)
  printf(" %d",number[i]);
  return 0;
}
```

Output

How many elements are u going to enter?:10

Enter 10 elements:2 3 5 7 1 9 3 8 0 4

Order of sorted elements:0 1 2 3 4 5 7 8 9

ii) 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();
          }
```

Output

Enter the number of elements:6

Enter elements

30 8 99 11 24 39

Array before sorting:30 8 99 11 24 39

Array after sorting:8 11 24 30 39 99

iii) 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();
    }
```

```
list before  sorting
101419262731333542440
 list after sorting
00101419262731333350
```

Viva Questions

1. Define sorting.
2. What is external sorting?
3. What is internal sorting ?
4. Compare Merge and Heap sorts.
5. Give one numerical example for each sorting .

Week7:

Write a program to implement the tree traversal methods.

Source code:

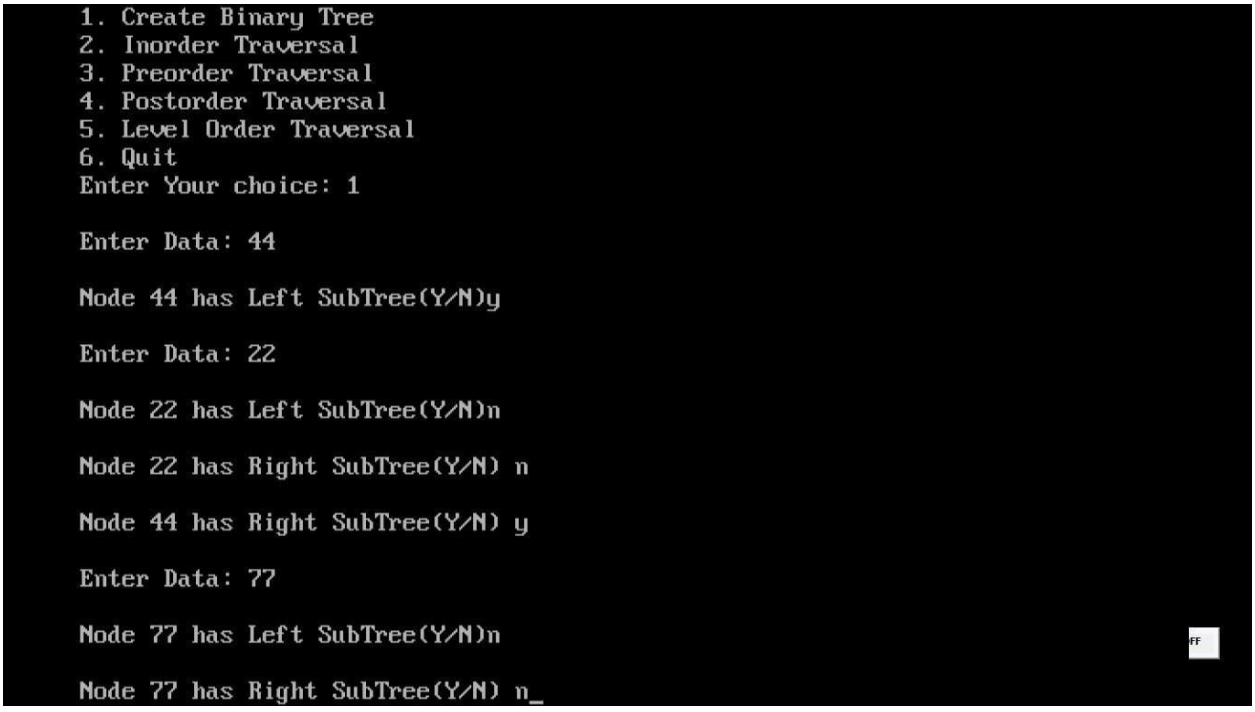
```
# include <stdio.h>
# include <stdlib.h>
   struct tree
    {
      struct tree* lchild;
      char data[10];
      struct tree* rchild;
    };

     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 ..");
                      }
                      break;
            case 2 :
                      printf("\n Inorder Traversal: ");
                      inorder(root);
                      break;
```

```
        case 3 :
                printf("\n Preorder Traversal: ");
                preorder(root);
                break;

        case 4 :
                printf("\n Postorder Traversal: ");
                postorder(root);
                break;
        case 5:
                printf("\n Level Order Traversal ..");
                make_Queue(root,0);
                level_order(Q,node_ctr);
                break;
        case 6 :
                exit(0);
      }
  getch();
   }while(1);
 }
```

OUTPUT :



```
1. Create Binary Tree
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Level Order Traversal
6. Quit
Enter Your choice: 2

Inorder Traversal:  22 44 77
```

```
1. Create Binary Tree
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Level Order Traversal
6. Quit
Enter Your choice:
4

Postorder Traversal:  22 77 44
```

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.

Week8:

8. Write a program to implement
i)Bnary search tree ii)B Trees iii)B+ Trees iv)AVL Trees v)Red-Black Trees

i)Binary Search Tree

```
#include  <stdio.h>
#include <stdlib.h>
struct node {
  int data;
  struct node *right_child;
  struct node *left_child;
};
struct node* new_node(int
  x){ struct node *temp;
   temp = malloc(sizeof(struct node));
   temp->data = x;
   temp->left_child = NULL;
   temp->right_child = NULL;
   return temp;
  }
   struct node* search(struct node * root, int
   x){ if (root == NULL || root->data == x)
     return root;
   else if (x > root->data)
         return search(root->right_child, x);
   else
     return search(root->left_child, x);
  }
  struct node* insert(struct node * root, int
   x){ if (root == NULL)
     return new_node(x);
   else if (x > root->data)
     root->right_child = insert(root->right_child, x);
   else
root -> left_child = insert(root->left_child, x);
   return root;
  }
  struct node* find_minimum(struct node * root)
   { if (root == NULL)
     return NULL;
   else if (root->left_child != NULL)
     return find_minimum(root->left_child);
   return root;
  }
   struct node* delete(struct node * root, int x)
     { if (root == NULL)
     return NULL;
   if (x > root->data)
     root->right_child = delete(root->right_child, x);
   else if (x < root->data)
     root->left_child = delete(root->left_child, x);
```

```
  else {
    if (root->left_child == NULL && root->right_child ==
      NULL){ free(root);
           return NULL;
    }
    else if (root->left_child == NULL || root->right_child ==
      NULL){ struct node *temp;
      if (root->left_child == NULL)
        temp = root->right_child;
      else
        temp = root->left_child;
      free(root);
      return temp;
    }
    else {
           struct node *temp = find_minimum(root->right_child);
      root->data = temp->data;
      root->right_child = delete(root->right_child, temp->data);
    }
  }
  return root;
}
  void inorder(struct node
  *root){ if (root != NULL)
  {
        inorder(root->left_child);
    printf(" %d ", root->data);
    inorder(root->right_child);
  }
}
  int main()
  {  struct node
  *root;
  root = new_node(20);
  insert(root, 5);
  insert(root, 1);
  insert(root, 15);
  insert(root, 9);
  insert(root, 7);
  insert(root, 12);
  insert(root, 30);
  insert(root, 25);
  insert(root, 40);
  insert(root, 45);
  insert(root, 42);
inorder(root);
printf("\n");
    root = delete(root, 1);
  root = delete(root, 40);
  root = delete(root, 45);
  root = delete(root, 9);
  inorder(root);
  printf("\n");
  return 0;
}
```

Output

```
1  5  7  9  12  15  20  25  30  40  42  45
 5  7  12  15  20  25  30  42
```

ii) B Trees

```
// insertion of a key in a B Tree in C Language

// including required header files
#include <stdio.h>
#include <stdlib.h>

// defining required macros
#define MAX 3
#define MIN 2

// defining the structure of the node
struct btree_node {
  int data_item[MAX + 1], counter;
  struct btree_node *the_link[MAX + 1];
};

struct btree_node *root_node;

// creating a Node
struct btree_node *create_node(int data_item, struct btree_node *child_node)
  { struct btree_node *new_node;
  new_node = (struct btree_node *)malloc(sizeof(struct btree_node));
  new_node -> data_item[1] = data_item;
  new_node -> counter = 1;
  new_node -> the_link[0] = root_node;
  new_node -> the_link[1] = child_node;
  return new_node;
}

// insert
void insert_value(int data_item, int position, struct btree_node *the_node,
        struct btree_node *child_node) {
  int j = the_node -> counter;
  while (j > position) {
    the_node -> data_item[j + 1] = the_node -> data_item[j];
    the_node -> the_link[j + 1] = the_node -> the_link[j];
    j--;
  }
  the_node -> data_item[j + 1] = data_item;
  the_node -> the_link[j + 1] = child_node;
  the_node -> counter++;
}

// splitting the node
void splitNode(int data_item, int *p_value, int position, struct btree_node *the_node,
```

```
        struct btree_node *child_node, struct btree_node **new_node)
  { int median_key, j;

  if (position > MIN)
  median_key = MIN + 1;
  else
    median_key = MIN;

  *new_node = (struct btree_node *)malloc(sizeof(struct btree_node));
  j = median_key + 1;
  while (j <= MAX) {
    (*new_node)->data_item[j - median_key] = the_node -> data_item[j];
    (*new_node)->the_link[j - median_key] = the_node -> the_link[j];
    j++;
  }
  the_node -> counter = median_key;
  (*new_node) -> counter = MAX - median_key;

  if (position <= MIN) {
    insert_value(data_item, position, the_node, child_node);
  } else {
    insert_value(data_item, position - median_key, *new_node, child_node);
  }
  *p_value = the_node -> data_item[the_node->counter];
  (*new_node) -> the_link[0] = the_node -> the_link[the_node->counter];
  the_node -> counter--;
}

// setting the value of node
int set_node_value(int data_item, int *p_value,
          struct btree_node *the_node, struct btree_node **child_node)
  { int position;
  if (!the_node) {
    *p_value = data_item;
    *child_node = NULL;
    return 1;
  }

  if (data_item < the_node -> data_item[1])
    { position = 0;
  } else {
    for (position = the_node -> counter;
       (data_item < the_node -> data_item[position] && position > 1); position--)
      ;
    if (data_item == the_node -> data_item[position])
      { printf("Duplicates are not allowed\n");
      return 0;
    }
  }
  if (set_node_value(data_item, p_value, the_node->the_link[position], child_node))
    { if (the_node->counter < MAX) {
      insert_value(*p_value, position, the_node, *child_node);
    } else {
      splitNode(*p_value, p_value, position, the_node, *child_node, child_node);
```

```
      return 1;
    }
  }
  return 0;
}

// insertion operation
void insertion_operation(int data_item)
  { int the_flag, i;
  struct btree_node *child_node;

  the_flag = set_node_value(data_item, &i, root_node, &child_node);
  if (the_flag)
    root_node = create_node(i, child_node);
}

// copying the successor
void copy_successor(struct btree_node *my_node, int position)
  { struct btree_node *dummy_node;
  dummy_node = my_node -> the_link[position];

  for (; dummy_node -> the_link[0] != NULL;)
    dummy_node = dummy_node -> the_link[0];
  my_node -> data_item[position] = dummy_node -> data_item[1];
}

// performing the rightshift
void right_shift(struct btree_node *my_node, int position)
  { struct btree_node *x = my_node -> the_link[position];
  int j = x -> counter;

  while (j > 0) {
    x -> data_item[j + 1] = x -> data_item[j];
    x -> the_link[j + 1] = x -> the_link[j];
  }
  x -> data_item[1] = my_node->data_item[position];
  x -> the_link[1] = x->the_link[0];
  x -> counter++;

  x = my_node -> the_link[position - 1];
  my_node -> data_item[position] = x -> data_item[x -> counter];
  my_node -> the_link[position] = x -> the_link[x -> counter];
  x -> counter--;
  return;
}

// performing the leftshift
void left_shift(struct btree_node *my_node, int position)
  { int j = 1;
  struct btree_node *x = my_node -> the_link[position - 1];

  x -> counter++;
  x -> data_item[x -> counter] = my_node -> data_item[position];
  x -> the_link[x -> counter] = my_node -> the_link[position] -> the_link[0];
```

```
  x = my_node -> the_link[position];
  my_node -> data_item[position] = x -> data_item[1];
  x -> the_link[0] = x -> the_link[1];
  x -> counter--;

  while (j <= x -> counter) {
    x -> data_item[j] = x -> data_item[j + 1];
    x -> the_link[j] = x -> the_link[j + 1];
    j++;
  }
  return;
}

// merging the nodes
void merge_nodes(struct btree_node *my_node, int position)
  { int j = 1;
  struct btree_node *x1 = my_node -> the_link[position], *x2 = my_node -> the_link[position - 1];

  x2 -> counter++;
  x2 -> data_item[x2 -> counter] = my_node -> data_item[position];
  x2 -> the_link[x2 -> counter] = my_node -> the_link[0];

  while (j <= x1 -> counter)
    { x2 -> counter++;
    x2 -> data_item[x2 -> counter] = x1 -> data_item[j];
    x2 -> the_link[x2 -> counter] = x1 -> the_link[j];
    j++;
  }

  j = position;
  while (j < my_node -> counter) {
    my_node -> data_item[j] = my_node -> data_item[j + 1];
    my_node -> the_link[j] = my_node -> the_link[j + 1];
    j++;
  }
  my_node -> counter--;
  free(x1);
}

// adjusting the node
void adjustNode(struct btree_node *my_node, int position)
  { if (!position) {
    if (my_node -> the_link[1] -> counter > MIN)
      { left_shift(my_node, 1);
    } else
      { merge_nodes(my_node,
      1);
    }
  } else {
    if (my_node -> counter != position) {
      if (my_node -> the_link[position - 1] -> counter > MIN)
        { right_shift(my_node, position);
      } else {
        if (my_node -> the_link[position + 1] -> counter > MIN) {
```

```
          left_shift(my_node, position + 1);
        } else {
          merge_nodes(my_node, position);
        }
      }
    } else {
      if (my_node -> the_link[position - 1] -> counter > MIN)
        right_shift(my_node, position);
      else
        merge_nodes(my_node, position);
    }
  }
}

// Traversing the tree
void tree_traversal(struct btree_node *my_node)
  { int i;
  if (my_node) {
    for (i = 0; i < my_node -> counter; i++)
      { tree_traversal(my_node -> the_link[i]);
      printf("%d ", my_node -> data_item[i + 1]);
    }
    tree_traversal(my_node -> the_link[i]);
  }
}

// main function
int main() {
  int data_item, ch;

  insertion_operation(4);
  insertion_operation(6);
  insertion_operation(2);
  insertion_operation(8);
  insertion_operation(10);
  insertion_operation(9);
  insertion_operation(1);
  insertion_operation(3);
  insertion_operation(12);
  insertion_operation(11);
  insertion_operation(13);

  printf("The B Tree is : ");
  tree_traversal(root_node);
}
```

Output:

The B Tree is : 1 2 3 4 6 8 9 10 11 12 13

iii) B+ Trees

```
#include<stdio.h>
#include<conio.h>
#include<iostream>
using namespace std;
struct BplusTreeNode
{
   int *data;
   BplusTreeNode **child_ptr;
   bool leaf;
   int n;
}*root = NULL, *np = NULL, *x = NULL;
BplusTreeNode * init()
{
   int i;
   np = new BplusTreeNode;
   np->data = new int[5];
   np->child_ptr = new BplusTreeNode *[6];
   np->leaf = true;
   np->n = 0;
   for (i = 0; i < 6; i++)
   {
      np->child_ptr[i] = NULL;
   }
   return np;
}
void traverse(BplusTreeNode *p)
{
   cout<<endl;
   int i;
   for (i = 0; i < p->n; i++)
   {
      if (p->leaf == false)
      {
         traverse(p->child_ptr[i]);
      }
      cout << " " << p->data[i];
   }
   if (p->leaf == false)
   {
      traverse(p->child_ptr[i]);
   }
   cout<<endl;
}
void sort(int *p, int n)
{
   int i, j, temp;
   for (i = 0; i < n; i++)
   {
      for (j = i; j <= n; j++)
      {
         if (p[i] > p[j])
         {
```

```
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}
int split_child(BplusTreeNode *x, int i)
{
    int j, mid;
    BplusTreeNode *np1, *np3, *y;
    np3 = init();
    np3->leaf = true;
    if (i == -1)
    {
        mid = x->data[2];
        x->data[2] = 0;
        x->n--;
        np1 = init();
        np1->leaf = false;
        x->leaf = true;
        for (j = 3; j < 5; j++)
        {
            np3->data[j - 3] = x->data[j];
            np3->child_ptr[j - 3] = x->child_ptr[j];
            np3->n++;
            x->data[j] = 0;
            x->n--;
        }
        for(j = 0; j < 6; j++)
        {
            x->child_ptr[j] = NULL;
        }
        np1->data[0] = mid;
        np1->child_ptr[np1->n] = x;
        np1->child_ptr[np1->n + 1] = np3;
        np1->n++;
        root = np1;
    }
    else
    {
        y = x->child_ptr[i];
        mid = y->data[2];
        y->data[2] = 0;
        y->n--;
        for (j = 3; j < 5; j++)
        {
            np3->data[j - 3] = y->data[j];
            np3->n++;
            y->data[j] = 0;
            y->n--;
        }
        x->child_ptr[i + 1] = y;
        x->child_ptr[i + 1] = np3;
```

```
    }
    return mid;
}
void insert(int a)
{
    int i, temp;
    x = root;
    if (x == NULL)
    {
        root = init();
        x = root;
    }
    else
    {
        if (x->leaf == true && x->n == 5)
        {
            temp = split_child(x, -1);
            x = root;
            for (i = 0; i < (x->n); i++)
            {
                if ((a > x->data[i]) && (a < x->data[i + 1]))
                {
                    i++;
                    break;
                }
                else if (a < x->data[0])
                {
                    break;
                }
                else
                {
                    continue;
                }
            }
            x = x->child_ptr[i];
        }
        else
        {
            while (x->leaf == false)
            {
            for (i = 0; i < (x->n); i++)
            {
                if ((a > x->data[i]) && (a < x->data[i + 1]))
                {
                    i++;
                    break;
                }
                else if (a < x->data[0])
                {
                    break;
                }
                else
                {
                    continue;
```

```
            }
        }
            if ((x->child_ptr[i])->n == 5)
            {
                temp = split_child(x, i);
                x->data[x->n] = temp;
                x->n++;
                continue;
            }
            else
            {
                x = x->child_ptr[i];
            }
        }
    }
    x->data[x->n] = a;
    sort(x->data, x->n);
    x->n++;
}
int main()
{
    int i, n, t;
    cout<<"enter the no of elements to be inserted\n";
    cin>>n;
    for(i = 0; i < n; i++)
    {
        cout<<"enter the element\n";
        cin>>t;
        insert(t);
    }
    cout<<"traversal of constructed tree\n";
    traverse(root);
    return 0;
}
```

Output

```
enter the no of elements to be inserted
5
enter the element
10
enter the element
30
enter the element
20
enter the element
40
enter the element
60
traversal of constructed tree

 10 20 30 40 60
```

iv) AVL Trees

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

struct Node
{
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};

int getHeight(struct Node
    *n){ if(n==NULL)
        return 0;
    return n->height;
}

struct Node *createNode(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;
}

int max (int a, int
    b){ return (a>b)?a:b;
}

int getBalanceFactor(struct Node *
    n){ if(n==NULL){
        return 0;
    }
```

```
    return getHeight(n->left) - getHeight(n->right);
}

struct Node* rightRotate(struct Node*
    y){ struct Node* x = y->left;
    struct Node* T2 = x->right;

    x->right = y;
    y->left = T2;

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

    return x;
}

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

    return y;
}

struct Node *insert(struct Node* node, int
    key){ if (node == NULL)
        return  createNode(key);

    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    node->height = 1 + max(getHeight(node->left), getHeight(node->right));
    int bf = getBalanceFactor(node);

    // Left Left Case
        if(bf>1 && key <
            node->left->key){ return
            rightRotate(node);
        }
    // Right Right Case
        if(bf<-1 && key >
            node->right->key){ return
            leftRotate(node);
        }
    // Left Right Case
    if(bf>1 && key >
            node->left->key){ node->left =
            leftRotate(node->left); return
```

```
    rightRotate(node);
}
```

```
    // Right Left Case
    if(bf<-1 && key < node->right->key){
            node->right = rightRotate(node->right);
            return leftRotate(node);
        }
    return node;
}

void preOrder(struct Node *root)
{
    if(root != NULL)
    {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

int main(){
    struct Node * root = NULL;


    root = insert(root, 1);
    root = insert(root, 2);
    root = insert(root, 4);
    root = insert(root, 5);
    root = insert(root, 6);
    root = insert(root, 3);
    preOrder(root);
    return 0;
}
```

Output

4 2 1 3 5 6

v) Red –Black Trees

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

enum nodeColor
  { RED,
  BLACK
};

struct rbNode
  { int data,
  color;
  struct rbNode *link[2];
};

struct rbNode *root = NULL;

struct rbNode *createNode(int data)
{
  struct rbNode *newnode;
  newnode = (struct rbNode *)malloc(sizeof(struct rbNode));
  newnode->data = data;
  newnode->color = RED;
  newnode->link[0] = newnode->link[1] = NULL;
  return newnode;
}

void insertion(int data) {
  struct rbNode *stack[98], *ptr, *newnode, *xPtr, *yPtr;
  int dir[98], ht = 0, index;
  ptr = root;
  if (!root) {
    root = createNode(data);
    return;
  }

  stack[ht] = root;
  dir[ht++] = 0;
  while (ptr != NULL) {
    if (ptr->data == data)
      { printf("Duplicates Not
      Allowed!!\n"); return;
    }
    index = (data - ptr->data) > 0 ? 1 : 0;
    stack[ht] = ptr;
    ptr = ptr->link[index];
    dir[ht++] = index;
  }
  stack[ht - 1]->link[index] = newnode = createNode(data);
  while ((ht >= 3) && (stack[ht - 1]->color == RED)) {
    if (dir[ht - 2] == 0) {
      yPtr = stack[ht - 2]->link[1];
      if (yPtr != NULL && yPtr->color == RED)
        { stack[ht - 2]->color = RED;
        stack[ht - 1]->color = yPtr->color = BLACK;
        ht = ht - 2;
      } else {
        if (dir[ht - 1] == 0)
          { yPtr = stack[ht - 1];
        } else {
          xPtr = stack[ht - 1];
          yPtr = xPtr->link[1];
          xPtr->link[1] = yPtr->link[0];
          yPtr->link[0] = xPtr;
          stack[ht - 2]->link[0] = yPtr;
```

```
        }
        xPtr = stack[ht - 2];
        xPtr->color = RED;
        yPtr->color = BLACK;
        xPtr->link[0] = yPtr->link[1];
        yPtr->link[1] = xPtr;
        if (xPtr == root)
          { root = yPtr;
        } else {
          stack[ht - 3]->link[dir[ht - 3]] = yPtr;
        }
        break;
      }
    } else {
      yPtr = stack[ht - 2]->link[0];
      if ((yPtr != NULL) && (yPtr->color == RED))
        { stack[ht - 2]->color = RED;
        stack[ht - 1]->color = yPtr->color = BLACK;
        ht = ht - 2;
      } else {
        if (dir[ht - 1] == 1)
          { yPtr = stack[ht - 1];
        } else {
          xPtr = stack[ht - 1];
          yPtr = xPtr->link[0];
          xPtr->link[0] = yPtr->link[1];
          yPtr->link[1] = xPtr;
          stack[ht - 2]->link[1] = yPtr;
        }
        xPtr = stack[ht - 2];
        yPtr->color = BLACK;
        xPtr->color = RED;
        xPtr->link[1] = yPtr->link[0];
        yPtr->link[0] = xPtr;
        if (xPtr == root)
          { root = yPtr;
        } else {
          stack[ht - 3]->link[dir[ht - 3]] = yPtr;
        }
        break;
      }
    }
  }
  root->color = BLACK;
}

void deletion(int data) {
  struct rbNode *stack[98], *ptr, *xPtr, *yPtr;
  struct rbNode *pPtr, *qPtr, *rPtr;
  int dir[98], ht = 0, diff, i;
  enum nodeColor color;

  if (!root) {
    printf("Tree not available\n");
    return;
  }

  ptr = root;
  while (ptr != NULL) {
    if ((data - ptr->data) == 0)
      break;
    diff = (data - ptr->data) > 0 ? 1 : 0;
    stack[ht] = ptr;
    dir[ht++] = diff;
    ptr = ptr->link[diff];
  }
```

```
if (ptr->link[1] == NULL) {
  if ((ptr == root) && (ptr->link[0] == NULL))
    { free(ptr);
    root = NULL;
  } else if (ptr == root)
    { root = ptr->link[0];
    free(ptr);
  } else {
    stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];
  }
} else {
  xPtr = ptr->link[1];
  if (xPtr->link[0] == NULL)
    { xPtr->link[0] =
    ptr->link[0]; color =
    xPtr->color;
    xPtr->color = ptr->color;
    ptr->color = color;

    if (ptr == root)
      { root = xPtr;
    } else {
      stack[ht - 1]->link[dir[ht - 1]] = xPtr;
    }

    dir[ht] = 1;
    stack[ht++] = xPtr;
  } else {
    i = ht++;
    while (1) {
      dir[ht] = 0;
      stack[ht++] = xPtr;
      yPtr = xPtr->link[0];
      if (!yPtr->link[0])
        break;
      xPtr = yPtr;
    }

    dir[i] = 1;
    stack[i] = yPtr;
    if (i > 0)
      stack[i - 1]->link[dir[i - 1]] = yPtr;

    yPtr->link[0] = ptr->link[0];

    xPtr->link[0] = yPtr->link[1];
    yPtr->link[1] = ptr->link[1];

    if (ptr == root)
      { root = yPtr;
    }

    color = yPtr->color;
    yPtr->color = ptr->color;
    ptr->color = color;
  }
}

if (ht < 1)
  return;

if (ptr->color == BLACK)
  { while (1) {
    pPtr = stack[ht - 1]->link[dir[ht - 1]];
    if (pPtr && pPtr->color == RED) {
      pPtr->color = BLACK;
```

```
  break;
}

if (ht < 2)
  break;

if (dir[ht - 2] == 0) {
  rPtr = stack[ht - 1]->link[1];

  if (!rPtr)
    break;

  if (rPtr->color == RED)
    { stack[ht - 1]->color =
    RED; rPtr->color = BLACK;
    stack[ht - 1]->link[1] = rPtr->link[0];
    rPtr->link[0] = stack[ht - 1];

    if (stack[ht - 1] == root)
      { root = rPtr;
    } else {
      stack[ht - 2]->link[dir[ht - 2]] = rPtr;
    }
    dir[ht] = 0;
    stack[ht] = stack[ht - 1];
    stack[ht - 1] = rPtr;
    ht++;

    rPtr = stack[ht - 1]->link[1];
  }

  if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&
    (!rPtr->link[1] || rPtr->link[1]->color == BLACK))
    { rPtr->color = RED;
  } else {
    if (!rPtr->link[1] || rPtr->link[1]->color == BLACK)
      { qPtr = rPtr->link[0];
      rPtr->color = RED;
      qPtr->color = BLACK;
      rPtr->link[0] = qPtr->link[1];
      qPtr->link[1] = rPtr;
      rPtr = stack[ht - 1]->link[1] = qPtr;
    }
    rPtr->color = stack[ht - 1]->color;
    stack[ht - 1]->color = BLACK;
    rPtr->link[1]->color = BLACK;
    stack[ht - 1]->link[1] = rPtr->link[0];
    rPtr->link[0] = stack[ht - 1];
    if (stack[ht - 1] == root)
      { root = rPtr;
    } else {
      stack[ht - 2]->link[dir[ht - 2]] = rPtr;
    }
    break;
  }
} else {
  rPtr = stack[ht - 1]->link[0];
  if (!rPtr)
    break;

  if (rPtr->color == RED)
    { stack[ht - 1]->color =
    RED; rPtr->color = BLACK;
    stack[ht - 1]->link[0] = rPtr->link[1];
    rPtr->link[1] = stack[ht - 1];
```

```
          if (stack[ht - 1] == root)
            { root = rPtr;
          } else {
            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
          }
          dir[ht] = 1;
          stack[ht] = stack[ht - 1];
          stack[ht - 1] = rPtr;
          ht++;

          rPtr = stack[ht - 1]->link[0];
        }
        if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&
          (!rPtr->link[1] || rPtr->link[1]->color == BLACK))
          { rPtr->color = RED;
        } else {
          if (!rPtr->link[0] || rPtr->link[0]->color == BLACK)
            { qPtr = rPtr->link[1];
            rPtr->color = RED;
            qPtr->color = BLACK;
            rPtr->link[1] = qPtr->link[0];
            qPtr->link[0] = rPtr;
            rPtr = stack[ht - 1]->link[0] = qPtr;
          }
          rPtr->color = stack[ht - 1]->color;
          stack[ht - 1]->color = BLACK;
          rPtr->link[0]->color = BLACK;
          stack[ht - 1]->link[0] = rPtr->link[1];
          rPtr->link[1] = stack[ht - 1];
          if (stack[ht - 1] == root)
            { root = rPtr;
          } else {
            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
          }
          break;
        }
      }
      ht--;
    }
  }
}

void inorderTraversal(struct rbNode *node)
  { if (node) {
    inorderTraversal(node->link[0]);
    printf("%d  ", node->data);
    inorderTraversal(node->link[1]);
  }
  return;
}

int main()
  { int ch,
  data; while
  (1) {
    printf("1. Insertion\t2. Deletion\n");
    printf("3. Traverse\t4. Exit");
    printf("\nEnter your choice:");
    scanf("%d", &ch);
    switch (ch)
      { case 1:
        printf("Enter the element to insert:");
        scanf("%d", &data);
        insertion(data);
        break;
      case 2:
        printf("Enter the element to delete:");
```

```
        scanf("%d", &data);
        deletion(data);
        break;
      case 3:
        inorderTraversal(root);
        printf("\n");
        break;
      case 4:
        exit(0);
      default:
        printf("Not available\n");
        break;
    }
    printf("\n");
  }
  return 0;
}
```

OUTPUT

1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:1
Enter the element to insert:4
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice: 3
4
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:2
Enter the element to delete:4
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:3
1. Insertion2. Deletion
3. Traverse 4. Exit
Enter your choice

EXPERIMENT 9

AIM: Write a program to implement the Graph traversal methods.

i) Depth First Search ii) Breadth First Search

i) Write a C program to implement Depth First Search

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v) {
int i;
reach[v]=1;
for (i=1;i<=n;i++)
if(a[v][i] && !reach[i]) {
printf("\n %d->%d",v,i);
dfs(i);
}
}
void main()
{ int
i,j,count=0;
clrscr();
printf("\n Enter number of vertices:");
scanf("%d",&n);
Scanned with OKEN Scanner
for (i=1;i<=n;i++)
{ reach[i]=0;
for (j=1;j<=n;j++)
a[i][j]=0;
}
printf("\n Enter the adjacency matrix:\n");
for (i=1;i<=n;i++)
for (j=1;j<=n;j++)
scanf("%d",&a[i][j]);
dfs(1);
printf("\n");
for (i=1;i<=n;i++)
{ if(reach[i])
count++;
}
if(count==n)
printf("\n Graph is connected"); else
printf("\n Graph is not connected");
getch();
}
```

OUTPUT:

Enter number of vertices:3
Enter the adjacency matrix:
1 1 1
0 1 1
010
1->2
2->3
Graph is connected

AIM: ii) Write a C program to implement Breadth First Search

SOURCE CODE:

```
#include<stdio.h>
 #include<conio.h>
int a[20][20],q[20],visited[20],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;
bfs(q[f++]);
}
}
void main()
{ int v;
clrscr();
printf("\n Enter the number of vertices:");
scanf("%d",&n);
for (i=1;i<=n;i++) {
}
q[i]=0; visited[i]=0;
printf("\n Enter graph data in matrix form:\n');
for (i=1;i<=n;i++)
for (j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter the starting vertex:");
scanf("%d",&v);
bfs(v);
printf("\n The node which are reachable are:\n");
for (i=1;i<=n;i++)
if(visited[i])
printf("%d\t",i); else
printf("\n Bfs is not possible");
getch();
```

}

OUTPUT:

```
Enter the number of vertices:4
Enter graph data in matrix form:
1 1 1 1
1010
0 0 1 1
0001
```

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?***

EXPERIMENT 10

AIM: Implement a pattern matching algorithm using boyer-moore,knuth-morris-pratt

Boyer-moore

```
# include <limits.h>
# include <string.h>
# include <stdio.h>
# define NO_OF_CHARS 256
 void badCharHeuristic( char *str, int size,  int badchar[NO_OF_CHARS])
{
    int i;
 for (i = 0; i < NO_OF_CHARS; i++)
         badchar[i] = -1;
     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;
```

```
    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;
}
```

Output

pattern occurs at shift = 4

Knuth-morris-pratt

```
#include<iostream>
#include<string.h>
using namespace std;
void prefixSuffixArray(char* pat, int M, int* pps)
   { int length = 0;
   pps[0] = 0;
   int i = 1;
   while (i < M) {
      if (pat[i] == pat[length])
         { length++;
         pps[i] = length;
         i++;
      } else {
         if (length != 0)
         length = pps[length - 1];
         else {
            pps[i] = 0;
            i++;
         }
      }
   }
}
void KMPAlgorithm(char* text, char* pattern)
   { int M = strlen(pattern);
   int N = strlen(text);
   int pps[M];
   prefixSuffixArray(pattern, M, pps);
   int i = 0;
   int j = 0;
   while (i < N) {
      if (pattern[j] == text[i])
         { j++;
         i++;
      }
      if (j == M) {
         printf("Found pattern at index %d
", i - j);
         j = pps[j - 1];
      }
      else if (i < N && pattern[j] != text[i])
         { if (j != 0)
         j = pps[j - 1];
         else
         i = i + 1;
      }
   }
```

```
}
int main() {
   char text[] = "xyztrwqxyzfg";
   char pattern[] = "xyz";
   printf("The pattern is found in the text at the following index :
");
   KMPAlgorithm(text, pattern);
   return 0;
}
```

Output

The pattern is found in the text at the following index −

Found pattern at index 0
Found pattern at index 7

SRIINDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, New Delhi)

LAB INTERNAL – MID – I LAB QUESTION

B.Tech II – I Sem – Lab MID I Examinations 2023– 2024

DEPARTMENT OF INFORMATION TECHNOLOGY

(R22CSE2126) DATA STRUCTURES LAB

BATCH : 2023 - 2024

SET - 1

Write a program that uses functions to perform the following operations on singly linked list.:
i) Creation ii) Insertion at beg iii) Deletion at end iv) Traversal

SET - 2
Write a program that uses functions to perform the following operations on doubly linked list.:
i) Creation ii) Insertion at mid iii) Deletion at beg iv) Traversal

SET - 3
Write a program that uses functions to perform the following operations on circular linked list.:
i) Creation ii) Insertion at end iii) Deletion at mid iv) Traversal

SET - 4
Write a program that implement stack (its operations) using
i)Arrays ii) Pointers

SET – 5
Write a program that implement Queue (its operations) using
i) Arrays ii) Pointers

SET – 6
Write a program that implementsthe following sorting methodsto sort a given list of integersin ascending order
i)Quick sort ii) Heap sort iii) Merge sort

SET - 7
Write a program to implement the tree traversal methods(Recursive and Non Recursive).

SET - 8
Write a program to implement
i) Binary Search tree ii) B Trees iii) B+ Trees iv) AVLtrees v) Red - Black trees

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, New Delhi)

B.Tech II – I Sem – Lab Mid II Examinations 2023 – 2024

DEPARTMENT OF INFORMATION TECHNOLOGY

(R22CSE2126) DATA STRUCTURES LAB

BATCH : 2022 - 2026

SET – 1

Write a program that implements the Selection sorting method to sort a given list of integers in ascending order

SET – 2

Write a program that use Linear searching operations for a Key value in a given list of integers

SET - 3

Write a program to implement graph traversal method using BFS

SET – 4

Write a program that use Binary searching operations for a Key value in a given list of integers

SET – 5

Write a program to implement graph traversal method using DFS

SET – 6

Write a program that implements the following sorting methods to sort a given list of integers in ascendingorder
i) Quick sort ii) Heap sort iii) merge sort

SET – 7

Write a program to implement
i)Bnary search tree ii)B Trees iii)B+ Trees

MODEL PAPER FOR LAB EXTERNAL

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, New Delhi)

B.Tech II – I Semester – Lab End Examinations 2023– 2024

DEPARTMENT OF INFORMATION TECHNOLOGY

(R22CSE2126) DATA STRUCTURES LAB

BATCH : 2022 - 2026 **DATE : 24/01/2024**

SET - 1

1. Write a program that uses functions to perform the following operations on singly linked list.:
i) Creation ii) Insertion at beg iii) Deletion at end iv) Traversal

SET - 2

1. Write a program that uses functions to perform the following operations on doubly linked list.:
i) Creation ii) Insertion at mid iii) Deletion at beg iv) Traversal

SET - 3

1. Write a program that uses functions to perform the following operations on circular linked list.:
i) Creation ii) Insertion at end iii) Deletion at mid iv) Traversal

SET - 4

1. Write a program that implement stack (its operations) using Arrays
2. Write a program that implements the Bubble sorting method to sort a given list of integers in ascending order

SET – 5

1. Write a program that implement stack (its operations) using Pointers

SET – 6

1. Write a program that implement Queue (its operations) using Arrays
2. Write a program that implements the Selection sorting method to sort a given list of integers in ascending order

SET – 7

1. Write a program that implement Queue (its operations) using Pointers

SET - 8

1. Write a program that use Linear searching operations for a Key value in a given list of integers
2. Write a program to implement graph traversal method using BFS

SET – 9

Write a program to implement

i)Bnary search tree ii)B Trees iii)B+ Trees

SET – 10

Write a program that implements the following sorting methods to sort a given list of integers in ascendingorder

ii)Quick sort ii) Heap sort iii) merge sort