DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE
(AUTONOMOUS)
(Approved by AICTE & Affiliated to Anna University, Chennai)
Accredited with 'A' Grade by NAAC, Accredited by TCS
Accredited by NBA with BME, ECE & EEE
PERAMBALUR - 621 212. Tamil Nadu.
website : www.dsengg.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

U23CSP32 DATA STRUCTURES LABORATORY

REGULATIONS 2023

YEAR / SEM : II / III

PREPARED BY **VERIFIED BY**

U23CS32 DATA STRUCTURES LABORATORY L T P C 0 0 4 2

COURSE OBJECTIVES:

- To demonstrate array implementation of linear data structure algorithms.
- To implement the applications using Stack.
- To implement the applications using Linked list
- To implement Binary search tree and AVL tree algorithms.
- To implement the Heap algorithm.
- To implement Dijkstra's algorithm.
- To implement Prim's algorithm
- To implement Sorting, Searching and hashing algorithms.

LIST OF EXERCISES:

1. Array implementation of Stack, Queue and Circular Queue ADTs
2. Implementation of Singly Linked List
3. Linked list implementation of Stack and Linear Queue ADTs
4. Implementation of Polynomial Manipulation using Linked list
5. Implementation of Evaluating Postfix Expressions, Infix to Postfix conversion
6. Implementation of Binary Search Trees
7. Implementation of AVL Trees
8. Implementation of Heaps using Priority Queues
9. Implementation of Dijkstra's Algorithm
10. Implementation of Prim's Algorithm
11. Implementation of Linear Search and Binary Search
12. Implementation of Insertion Sort and Selection Sort
13. Implementation of Merge Sort
14. Implementation of Open Addressing (Linear Probing and Quadratic Probing)

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO1: Develop and array implement of Stack and Queue ADTs
CO2: Develop and array implement of List ADT
CO3: Develop and implement List, Stack and Queue ADTs.
CO4: Apply the concept of Binary Trees , Binary Search Trees, AVL Trees
CO5: Develop and implement Heaps using Priority Queues
CO6: Apply the concept of searching and sorting algorithms

LIST OF EXERCISES

DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE (AUTONOMOUS)

Vision of the Institute

An active and committed centre of advanced learning focused on research and training in the fields of Engineering, Technology and Management to serve the nation better.

Mission of the Institute

M1: To develop eminent scholar with a lifelong follow up of global standards by offering UG,PG and Doctoral Programmes.

M2: To pursue Professional and Career growth by collaborating mutually beneficial partnership with industries and higher institutes of research.

M3: To promote sustained research and training with emphasis on human values and leadership qualities.

M4: To contribute solutions for the need based issues of our society by proper ways and means as dutiful citizen.

Vision of the Department

To produce globally competent, socially responsible professionals in the field ofComputer Science and Engineering.

Mission of the Department

M1: Impart high quality experiential learning to get expertise in modern software tools

M2: Inculcate industry exposure and build inter disciplinary research skills.

M3: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M4: Acquire Innovative skills and promote lifelong learning with a sense of societal and ethical responsibilities

Program Educational Objectives (PEO's)

PEO1: Graduates of the program will develop proficiency in identifying, formulating, and resolving complex computing problems.

PEO2: Graduates of the program will achieve successful careers in the field of computer science and engineering, pursue advanced degrees, or demonstrate entrepreneurial success.

PEO3: Graduates of the program will cultivate effective communication skills, teamwork abilities, ethical values, and leadership qualities for professional engagement in industry and research organizations.

Program Outcomes (PO'S)

PO1: Engineering knowledge: Apply the basic knowledge of science, mathematics and engineering fundamentals in the field of Computer Science and Engineering to solve complex engineering problems.

PO2: Problem analysis: Ability to use basic principles of mathematics, natural sciences, and engineering sciences to Identify, formulate, review research literature and analyze Computer Science and engineering problems.

PO3: Design/development of solutions: Ability to design solutions for complex Computer Science and engineering problems and basic design system to meet the desired needs within realistic constraints such as manufacturability, durability, reliability, sustainability and economy with appropriate consideration for the public health, safety, cultural, societal, and environmental considerations

PO4: Conduct investigations of complex problems: Ability to execute the experimental activities using research-based knowledge and methods including analyze, interpret the data and results with valid conclusion.

PO5: Modern tool usage: Ability to use state of the art of techniques, skills and modern engineering tools necessary for engineering practice to satisfy the needs of the society with an understanding of the limitations.

PO6: The Engineer and Society: Ability to apply reasoning informed by the contextual knowledge to assess the impact of Computer Science and engineering solutions in legal, health, cultural, safety and societal context and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Ability to understand the professional responsibility and accountability to demonstrate the need for sustainable development globally in Computer Science domain with consideration of environmental effect.

PO8: Ethics: Ability to understand and apply ethical principles and commitment to address the professional ethical responsibilities of an engineer.

PO9: Individual and team work: Ability to function efficiently as an individual or as a group member or leader in a team in multidisciplinary environment.

PO10: Communication: Ability to communicate, comprehend and present effectivelywith engineering community and the society at large on complex engineering activities by receiving clear instructions for preparing effective reports, design documentation and presentations.

PO11: Project management and finance: Ability to acquire and demonstrate the knowledge of contemporary issues related to finance and managerial skills in one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Ability to recognize and adapt to the emerging field of application in engineering and technology by developing self-confidence for lifelonglearning process.

Program Specific Outcome (PSO's)

The graduates of Bachelor of Engineering in Computer Science and Engineering Programme will be able to

PSO1: Analyze, develop and provide solutions to industrial problems in computer domain using programming, data processing and analytical skills.

PSO2: Apply software application oriented skills to innovate solution to meet the ever changing demands of IT industry.

Ex.No:1a ARRAY IMPLEMENTATION OF STACK ADT

AIM

To write a C program for implementing Stack ADT using Array.

ALGORITHM

Step 1: Start the program.
Step 2: Declare the required variables
Step 3: Define Push() to insert the element into stack.
Step 4: Define Pop() to delete an element from the stack.
Step 5: Display the values from the stack.
Step 5: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<string.h>
#define MAX 4 //you can take any number to limit your stack size
int stack[MAX];
int top;
void push()
{
int token;
if(top==MAX-1)
{
printf("\nStack full");
return;
}
printf("\nEnter the element to be inserted:");
scanf("%d",&token);
top=top+1;
stack[top]=token;
}
int pop()
{
int t;
if(top==-1)
{
return -1;
}
t=stack[top];
top=top-1;
return t;
}
void show()
{
int i;
printf("\nThe Stack elements are:\nTOP-->");
for(i=top;i>=0;i--)
{
printf("\t|%d|\n",stack[i];
}
}
void main()
```

```
{
int choice,token;
top=-1;
clrscr();
printf("STACK USING ARRAY");
do
{
printf("\n1.PUSH\n2.POP\n3.show or display\n4.exit"); printf("\nEnter your
choice for the operation: "); scanf("%d",&choice);
switch(choice)
{
case 1:
push();
show();
break;
case 2:
token=pop();
if(token==-1)
printf("\nStack empty");
else
{
printf("\nThe element deleted is:%d",token);
show();
break;
}
case 3:
show();
break;
case 4:
exit(0);
default:printf("\nWrong choice");
break;
}
}while(choice<5);
getch();
}
```

OUTPUT:

```
STACK USING ARRAY
1.PUSH
2.POP
3.SHOW OR DISPLAY
4.EXIT
Enter your choice for the operation:1
Enter the element to be inserted: 4
```

The stack elements are:

TOP→|4|

STACK USING ARRAY

```
1.PUSH
2.POP
3.SHOW OR DISPLAY
4.EXIT
Enter your choice for the operation:1
Enter the element to be inserted: 2
```

The stack elements are:

TOP→ |2|

|4|

STACK USING ARRAY

1.PUSH
2.POP
3.SHOW OR DISPLAY
4.EXIT
Enter your choice for the operation:1
Enter the element to be inserted: 8
The stack elements are:
TOP→ |8|
|2|
|4|

1.PUSH
2.POP
3.SHOW OR DISPLAY
4.EXIT
Enter your choice for the operation:2
Enter the element to be deleted: 8
The stack elements are:
TOP→ |2|
|4|

1.PUSH
2.POP
3.SHOW OR DISPLAY
4.EXIT
Enter your choice for the operation:3
The stack elements are:
TOP→ |2|
|4|

1.PUSH
2.POP
3.SHOW OR DISPLAY
4.EXIT
Enter your choice for the operation:4
Press any key to continue…

RESULT

Thus C program to implement Stack ADT using Array was written, executed and output is verified successfully.

Ex.No:1b **ARRAY IMPLEMENTATION OF QUEUE ADT**

AIM

To write a C program for implementing Queue ADT using Array.

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Ask the user for the operation like insert, delete, display and exit.

Step 4: According to the option entered, access its respective function using switch statement.

Step 5: In the function insert(), firstly check if the queue is full. If it is, then print the output as "Queue Overflow". Otherwise take the number to be inserted as input and store it in the variable add_item. Copy the variable add_item to the array queue_array[] and increment the variable rear by 1.

Step 6: In the function delete(), firstly check if the queue is empty. If it is, then print the output as "Queue Underflow". Otherwise print the first element of the array queue_array[] and decrement the variable front by 1.

Step 7: In the function display(), using for loop print all the elements of the array starting from front to rear.

Step 8: Stop

PROGRAM

```
#include <stdio.h>
#define MAX 50
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
int choice;
while (1)
{
printf("1.Insert element to queue \n");
printf("2.Delete element from queue \n");
printf("3.Display all elements of queue \n");
printf("4.Quit \n");
printf("Enter your choice : ");
scanf("%d", &choice);
switch (choice)
{
case 1:
insert();
break;
case 2:
delete();
break;
case 3:
display();
break;
```

```
case 4:
exit(1);
default:
printf("Wrong choice \n");
} /*End of switch*/
} /*End of while*/
} /*End of main()*/
insert()
{
int add_item;
if (rear == MAX - 1)
printf("Queue Overflow \n");
else
{
if (front == - 1)
/*If queue is initially empty */
front = 0;
printf("Inset the element in queue : ");
scanf("%d", &add_item);
rear = rear + 1;
queue_array[rear] = add_item;
}
return 0;
}
/*End of insert()*/

delete()
{
if (front == - 1 || front > rear)
{
printf("Queue Underflow \n");
return ;
}
else
{
printf("Element deleted from queue is : %d\n", queue_array[front]);
front = front + 1;
}
return;
} /*End of delete() */
display()
{
int i;
if (front == - 1)
printf("Queue is empty \n");
else
{
printf("Queue is : \n");
for (i = front; i <= rear; i++)
printf("%d ", queue_array[i]);
printf("\n");
}
return 0;
}
/*End of display() */
```

OUTPUT

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 10

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 15

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 20

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 30

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2
Element deleted from queue is : 10

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 3
Queue is :
15 20 30

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 4
```

RESULT

Thus C program to implement Queue ADT using Array was written, executed and output is verified successfully.

Ex.No:1c **ARRAY IMPLEMENTATION OF CIRCULAR QUEUE ADT**

AIM

To write a C program for implementing circular Queue ADT using Array.

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Ask the user for the operation like insert, delete, display and exit.

Step 4: According to the option entered, access its respective function using switch statement.

Step 5: In the function insert(), firstly check if the queue is full. If it is, then print the output as "Queue Overflow". Otherwise take the number to be inserted as input and store it in the variable add_item. Copy the variable add_item to the array queue_array[] and increment the variable rear by 1.

Step 6: In the function delete(), firstly check if the queue is empty. If it is, then print the output as "Queue Underflow". Otherwise print the first element of the array queue_array[] and decrement the variable front by 1.

Step 7: In the function display(), using for loop print all the elements of the array starting from front to rear.

Step 8: Stop

PROGRAM

```
/*static circular queue*/
#include <stdio.h>
#define size 5

void insertq(int[], int);
void deleteq(int[]);
void display(int[]);

int front =  - 1;
int rear =  - 1;

int main()
{
   int n, ch;
   int queue[size];
   do
   {
      printf("\n\n Circular Queue:\n1. Insert \n2. Delete\n3. Display\n0. Exit");
```

```
        printf("\nEnter Choice 0-3? : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("\nEnter number: ");
                scanf("%d", &n);
                insertq(queue, n);
                break;
            case 2:
                deleteq(queue);
                break;
            case 3:
                display(queue);
                break;
        }
    }while (ch != 0);
}


void insertq(int queue[], int item)
{
    if ((front == 0 && rear == size - 1) || (front == rear + 1))
    {
        printf("queue is full");
        return;
    }
    else if (rear ==  - 1)
    {
        rear++;
        front++;
    }
    else if (rear == size - 1 && front > 0)
    {
        rear = 0;
    }
    else
    {
        rear++;
    }
    queue[rear] = item;
}

void display(int queue[])
{
    int i;
    printf("\n");
    if (front > rear)
    {
        for (i = front; i < size; i++)
        {
```

```
            printf("%d ", queue[i]);
        }
        for (i = 0; i <= rear; i++)
            printf("%d ", queue[i]);
    }
    else
    {
        for (i = front; i <= rear; i++)
            printf("%d ", queue[i]);
    }
}

void deleteq(int queue[])
{
    if (front ==  - 1)
    {
        printf("Queue is empty ");
    }
    else if (front == rear)
    {
        printf("\n %d deleted", queue[front]);
        front =  - 1;
        rear =  - 1;
    }
    else
    {
        printf("\n %d deleted", queue[front]);
        front++;
    }
}
```

RESULT

Thus C program to implement Circular Queue ADT using Array was written, executed and output is verified successfully.

Ex.No: 2 **IMPLEMENTATION OF SINGLY LINKED LIST**

AIM

To write a C program for implementing singly linked list ADT.

ALGORTIHM

Step 1: Start the program.

Step 2: Declare a structure for Linked List.

Step 3: Declare the operations involved in Linked List.

Step 4: In main (), using do-while loop call the different operations based on user choice.

Step 5: Perform the operations such as creating a list, Inserting an element into the list, Deleting an element from the list, Check for duplication, Find the previous and next element in the list, sort the list and display the required result. Step 6: Stop the program.

PROGRAM

//Implementation of Singly Linked List:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
}*head=NULL;

void insertend()
{
struct node *new,*temp;
new=(struct node*)malloc(sizeof(struct node));
printf("Enter the data ");
scanf("%d",&new->data);
new->next=NULL;
if(head==NULL)
{
head=new;
}
else
{
temp=head;
while(temp->next!=NULL)
{
temp=temp->next;
```

```
}
temp->next=new;
}
}

void insertbeg()
{
struct node *new;
new=(struct node*)malloc(sizeof(struct node));
printf("Enter the data ");
scanf("%d",&new->data);
new->next=head;
head=new;
}
void insertafter()
{
int pos,i;
struct node *new,*temp;
new=(struct node*)malloc(sizeof(struct node));
printf("Enter the data ");
scanf("%d",&new->data);
printf("Enter the position after which data to be added ");
scanf("%d",&pos);
temp=head;
for(i=0;i<pos-1;i++)
{
temp=temp->next;
if(temp==NULL)
{
printf("There are less than %d elements",pos);
return;
}
}
new->next=temp->next;
temp->next=new;
}
void delnode()
{
int x;
struct node *temp,*a;
printf("Enter data to be deleted ");
scanf("%d",&x);
temp=head;
while(temp->data!=x&&temp!=NULL)
{
a=temp;
temp=temp->next;
}
if(temp==NULL)
{
printf("Data does not exist\n");
return;
```

```
}

if(temp==head)
{
head=temp->next;
}
else
{
a->next=temp->next;
free(temp);
}
}
void display()
{
struct node *temp;
temp=head;
printf("List contains:\n");
while(temp!=NULL)
{
printf("%d\n",temp->data);
temp=temp->next;
}
}
void main()
{
int ch;
clrscr();
printf("Menu\n");
printf("1.Insert at the end\n");
printf("2.Insert at the beginning\n");
printf("3.Insert in between nodes\n");
printf("4.Delete a node\n");
printf("5.Display list\n");
printf("6.Exit\n");
do
{
printf("Enter your choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1:insertend();
break;
case 2:insertbeg();
break;
case 3:insertafter();
break;
case 4:if(head==NULL)
printf("List underflow\n");
else
delnode();
break;
case 5:if(head==NULL)
```

```
printf("List underflow\n");
else
display();
break;
case 6:break;
default:printf("Wrong Choice\n");
}
}
while(ch!=6);
getch();
}
```

OUTPUT

```
Menu
1.Insert at the end
2. Insert at the beginning
3. Insert in between nodes
4. Delete a node
5. Display list
6.Exit

Enter your choice: 1
Enter the data : 5
Enter your choice 5
List contains:  5 6

Enter your choice: 2
Enter the data : 2
Enter your choice 5
List contains:  2 5 6

Enter your choice: 3
Enter the data : 8
Enter the position after which the data to be added : 2
Enter your choice 5
List contains:  2 5 8  6

Enter your choice: 4
Enter the data to be deleted : 8
Enter your choice 5
List contains:  2 5 6

Enter your choice: 6
```

RESULT

Thus C program to implement List ADT using Linked List was written, executed and output is verified successfully.

Ex.No: 3a **LINKED LIST IMPLEMENTATION OF LIST ADT**

AIM

To write a C program for implementing List ADT using Linked List.

ALGORTIHM

Step 1: Start the program.

Step 2: Declare a structure for Linked List.

Step 3: Declare the operations involved in Linked List.

Step 4: In main (), using do-while loop call the different operations based on user choice.

Step 5: Perform the operations such as creating a list, Inserting an element into the list, Deleting an element from the list, Check for duplication, Find the previous and next element in the list, sort the list and display the required result. Step 6: Stop the program.

PROGRAM

//Implementation of Singly Linked List:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
}*head=NULL;

void insertend()
{
struct node *new,*temp;
new=(struct node*)malloc(sizeof(struct node));
printf("Enter the data ");
scanf("%d",&new->data);
new->next=NULL;
if(head==NULL)
{
head=new;
}
else
{
temp=head;
while(temp->next!=NULL)
{
temp=temp->next;
}
```

```
temp->next=new;
}
}

void insertbeg()
{
struct node *new;
new=(struct node*)malloc(sizeof(struct node));
printf("Enter the data ");
scanf("%d",&new->data);
new->next=head;
head=new;
}
void insertafter()
{
int pos,i;
struct node *new,*temp;
new=(struct node*)malloc(sizeof(struct node));
printf("Enter the data ");
scanf("%d",&new->data);
printf("Enter the position after which data to be added ");
scanf("%d",&pos);
temp=head;
for(i=0;i<pos-1;i++)
{
temp=temp->next;
if(temp==NULL)
{
printf("There are less than %d elements",pos);
return;
}
}
new->next=temp->next;
temp->next=new;
}
void delnode()
{
int x;
struct node *temp,*a;
printf("Enter data to be deleted ");
scanf("%d",&x);
temp=head;
while(temp->data!=x&&temp!=NULL)
{
a=temp;
temp=temp->next;
}
if(temp==NULL)
{
printf("Data does not exist\n");
return;
}
```

```
if(temp==head)
{
head=temp->next;
}
else
{
a->next=temp->next;
free(temp);
}
}
void display()
{
struct node *temp;
temp=head;
printf("List contains:\n");
while(temp!=NULL)
{
printf("%d\n",temp->data);
temp=temp->next;
}
}
void main()
{
int ch;
clrscr();
printf("Menu\n");
printf("1.Insert at the end\n");
printf("2.Insert at the beginning\n");
printf("3.Insert in between nodes\n");
printf("4.Delete a node\n");
printf("5.Display list\n");
printf("6.Exit\n");
do
{
printf("Enter your choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1:insertend();
break;
case 2:insertbeg();
break;
case 3:insertafter();
break;
case 4:if(head==NULL)
printf("List underflow\n");
else
delnode();
break;
case 5:if(head==NULL)
printf("List underflow\n");
```

```
else
display();
break;
case 6:break;
default:printf("Wrong Choice\n");
}
}
while(ch!=6);
getch();
}
```

OUTPUT

```
Menu
1.Insert at the end
2. Insert at the beginning
3. Insert in between nodes
4. Delete a node
5. Display list
6.Exit

Enter your choice: 1
Enter the data : 5
Enter your choice 5
List contains:  5 6

Enter your choice: 2
Enter the data : 2
Enter your choice 5
List contains:  2 5 6

Enter your choice: 3
Enter the data : 8
Enter the position after which the data to be added : 2
Enter your choice 5
List contains:  2 5 8  6

Enter your choice: 4
Enter the data to be deleted : 8
Enter your choice 5
List contains:  2 5 6

Enter your choice: 6
```

RESULT

Thus C program to implement List ADT using Linked List was written, executed and output is verified successfully.

Ex.No: 3b LINKED LIST IMPLEMENTATION OF STACK ADT

AIM

To write a C program for implementing Stack ADT using Linked List.

ALGORITHM

Step 1: Start the program.
Step 2: Define a structure for Linked list.
Step 3: Define Push() to insert value into the stack.
Step 4: Define Pop() to delete an element from the stack.
Step 5: Display the stack values.
Step 6: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct Node
{
int Data;
struct Node *next;
}*top;
void popStack()
{
struct Node *var=top;
if(var==top)
{
top = top->next;
free(var);
}
else
printf("\nStack Empty");
}
void push(int value)
{
struct Node *temp;
temp=(struct Node *)malloc(sizeof(struct Node));
temp->Data=value;
if (top == NULL)
{
top=temp;
top->next=NULL;
}
else
{
temp->next=top;
top=temp;
}
}
```

```
void display()
{
struct Node *var=top;
if(var!=NULL)
{
printf("\nElements are as:\nTOP->");
while(var!=NULL)
{
printf("\t|%d|\n",var->Data);
var=var->next;
}
}
else
printf("\nStack is Empty");
}
void main()
{
int i=0,value;
top=NULL;
clrscr();
while(1)
{
printf(" \n1. Push to stack");
printf(" \n2. Pop from Stack");
printf(" \n3. Display data of Stack");
printf(" \n4. Exit\n");
printf(" \nChoose Option: ");
scanf("%d",&i);
switch(i)
{
case 1:
printf("\nEnter a value to push into Stack: ");
scanf("%d",&value);
push(value);
display();
break;
case 2:
popStack();
display();
break;
case 3:
display();
break;
case 4:
exit(0);
default:
printf("\nwrong choice for operation");
getch();
}
}
}
```

OUTPUT

```
1. Push to stack
2. Pop from stack
3. Display data of stack

Choose Option: 1

Enter a value to push into Stack: 4
Elements are as Top→ |4|

Choose Option: 1

Enter a value to push into Stack: 8
Elements are as Top→ |8|
                     |4|
Choose Option: 1

Enter a value to push into Stack: 8
Elements are as Top→ |1|
                     |8|
                     |4|
1. Push to stack
2. Pop from stack
3. Display data of stack

Choose Option: 2
Element are as TOP→|8 |
                   |4|

1. Push to stack
2. Pop from stack
3. Display data of stack

Choose Option: 3

Element are as TOP→|8 |
                   |4|

1. Push to stack
2. Pop from stack
3. Display data of stack

Choose Option: 4
Press any key to continue. . .
```

RESULT

Thus C program to implement Stack ADT using Linked List was written, executed and output is verified successfully.

Ex.No: 3c LINKED LIST IMPLEMENTATION OF QUEUE ADT

AIM

To write a C program for implementing Queue ADT using Linked List.

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Ask the user for the operation like insert, delete, display and exit.

Step 4: According to the option entered, access its respective function using switch statement.

Step 5: In the function insert(), firstly check if the queue is full. If it is, then print the output as "Queue Overflow". Otherwise take the number to be inserted as input and store it in the variable add_item. Copy the variable add_item to the array queue_array[] and increment the variable rear by 1.

Step 6: In the function delete(), firstly check if the queue is empty. If it is, then print the output as "Queue Underflow". Otherwise print the first element of the array queue_array[] and decrement the variable front by 1.

Step 7: In the function display(), using for loop print all the elements of the array starting from front to rear.

Step 8: Stop

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int info;
struct node *ptr;
}*front,*rear,*temp,*front1;

int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();
int count = 0;
```

```
void main()
{
int no, ch, e;
printf("\n 1 - Enque");
printf("\n 2 - Deque");
printf("\n 3 - Front element");
printf("\n 4 - Empty");
printf("\n 5 - Exit");
printf("\n 6 - Display");
printf("\n 7 - Queue size");
create();
while (1)
{
printf("\n Enter choice : ");
scanf("%d", &ch);
switch (ch)
{
case 1:
printf("Enter data : ");
scanf("%d", &no);
enq(no);
break;
case 2:
deq();
break;
case 3:
e = frontelement();
if (e != 0)
printf("Front element : %d", e);
else
printf("\n No front element in Queue as queue is empty");
break;
case 4:
empty();
break;
case 5:
exit(0);
case 6:
display();
break;
case 7:
queuesize();
break;
default:
printf("Wrong choice, Please enter correct choice  ");
break;
}
}
}

void create()
```

```
{
front = rear = NULL;
}

void queuesize()
{
printf("\n Queue size : %d", count);
}

void enq(int data)
{
if (rear == NULL)
{
rear = (struct node *)malloc(1*sizeof(struct node));
rear->ptr = NULL;
rear->info = data;
front = rear;
}
else
{
temp=(struct node *)malloc(1*sizeof(struct node));
rear->ptr = temp;
temp->info = data;
temp->ptr = NULL;
rear = temp;
}
count++;
}

void display()
{
front1 = front;
if ((front1 == NULL) && (rear ==NULL))
{
printf("Queue is empty");
return;
}
while (front1 != rear)
{
printf("%d ", front1->info);
front1 = front1->ptr;
}
if (front1 == rear)
printf("%d", front1->info);
}

void deq()
{
front1 = front;
if (front1 == NULL)
{
```

```
printf("\n Error: Trying to display elements from empty queue");
return;
}
else
if (front1->ptr != NULL)
{
front1 = front1->ptr;
printf("\n Dequed value : %d",front->info);
free(front);
front = front1;
}
else
{
printf("\n Dequed value : %d",front->info);
free(front);
front = NULL;
rear = NULL;
}
count--;
}
int frontelement()
{
if ((front != NULL) && (rear != NULL))
return(front->info);
else
return 0;
}
void empty()
{
if ((front == NULL) && (rear ==NULL))
printf("\n Queue empty");
else
printf("Queue not empty");
}
```

OUTPUT

```
1 - Enque
2 - Deque
3 - Front element
4 - Empty
5 - Exit
6 - Display
7 - Queue size
Enter choice : 1
Enter data : 14

Enter choice : 1
Enter data : 85

Enter choice : 1
Enter data : 38

Enter choice : 3
Front element : 14
Enter choice : 6
14 85 38
Enter choice : 7

Queue size : 3
Enter choice : 2

Dequed value : 14
Enter choice : 6
85 38
Enter choice : 7

Queue size : 2
Enter choice : 4
Queue not empty
Enter choice : 5
```

RESULT

Thus C program to implement Queue ADT using Linked List was written, executed and output is verified successfully.

Ex.No: 4 IMPLEMENTATION OF POLYNOMIAL MANIPULATION USING LINKED LIST

AIM

To write a C program for implementing Polynomial Manipulation using linked list.

ALGORITHM

1. Input the multiplicand and multiplier and multiplier
2. Set both the polynomial in descending order of the coefficient
3. Multiply each node of multiplicand with each node of the multiplier (multiplication of the coefficient part and addition of the exponent part) and add them into a newly formed linked list in descending order
4. Coefficient having the same exponent value is added up with each other in the list and no two nodes have the same exponent value.
5. Then the product is to be displayed in a proper way in the form of ax^n+bx^n-1+…
6. Certain points to be noted before displaying a polynomial: Any coefficient with value 0 must not be displayed, 1x^n+2x^n-1 must not be displayed … node having coefficient value 1 must be displayed as x^n, node with exponent value 0 must be displayed as x not x^0, 1x^n-2x^n-1 format should be maintained not standard like errors 1x^n+-2x^n-1 should come up.

PROGRAM:

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

struct node {
  int coefficient, exponent;
  struct node *next;
};

struct node *hPtr1, *hPtr2, *hPtr3;
/*
 * creates new node and fill the given data
 */
struct node * buildNode(int coefficient, int exponent) {
  struct node *ptr = (struct node *) malloc(sizeof (struct node));
  ptr->coefficient = coefficient;
  ptr->exponent = exponent;
  ptr->next = NULL;
  return ptr;
}

/* insert data in decending order - based on exponent value */
void polynomial_insert(struct node ** myNode, int coefficient, int exponent) {
```

```
struct node *lPtr, *pPtr, *qPtr = *myNode;
lPtr = buildNode(coefficient, exponent);

/* inserting new node at appropriate position */
if (*myNode == NULL || (*myNode)->exponent < exponent) {
 *myNode = lPtr;
 (*myNode)->next = qPtr;
 return;
}

/* placing new node between two nodes or end of node */
while (qPtr) {
 pPtr = qPtr;
 qPtr = qPtr->next;
 if (!qPtr) {
  pPtr->next = lPtr;
  break;

 }
 else if ((exponent < pPtr->exponent) &&       (exponent > qPtr->exponent){
 lPtr->next = qPtr;
 pPtr->next = lPtr;
 break;
  }
  }
 return;
}

/* inserting new node with resultant data into the output list (n1) */
void polynomial_add(struct node **n1, int coefficient, int exponent) {
 struct node *x = NULL, *temp = *n1;
 if (*n1 == NULL || (*n1)->exponent < exponent) {
  /* adding at the front */
  *n1 = x = buildNode(coefficient, exponent);
  (*n1)->next = temp;
 } else {
  while (temp) {
 if (temp->exponent == exponent) {
  /* updating the co-efficient value  alone */
  temp->coefficient = temp->coefficient + coefficient;
  return;
 }
 if (temp->exponent > exponent && (!temp->next || temp->next->exponent < exponent)) {
  /* inserting in the middle or end */
  x = buildNode(coefficient, exponent);
  x->next = temp->next;
  temp->next = x;
  return;
 }
 temp = temp->next;
  }
```

```
    x->next = NULL;
    temp->next = x;
   }
}

void polynomial_multiply(struct node **n1, struct node *n2, struct node *n3) {
  struct node * temp;
  int coefficient, exponent;

  temp = n3;

  /* if both input list are absent, then output list is NULL */
  if (!n2 && !n3)
    return;

  /* input list 1(n2) is absent, then output list is input list2 (n3) */
  if (!n2) {
    *n1 = n3;
  } else if (!n3) {

    /*
     * list n3 is absent, then o/p list is n2
     */
    *n1 = n2;
  } else {
    while (n2) {
  while (n3) {
    /* multiply coefficient & add exponents */
    coefficient = n2->coefficient * n3->coefficient;
    exponent = n2->exponent + n3->exponent;
    n3 = n3->next;
    /* insert the above manipulated data to o/p list */
    polynomial_add(n1, coefficient, exponent);
  }
  n3 = temp;
  n2 = n2->next;
    }
  }
  return;
}

/* delete the given input list */
struct node * polynomial_deleteList(struct node *ptr) {
  struct node *temp;
  while (ptr){
    temp = ptr->next;
    free(ptr);
    ptr = temp;
  }
  return NULL;
}
```

```
void polynomial_view(struct node *ptr) {
  int i = 0;
  int flag=0;
  while (ptr) {
    if(ptr->exponent != 0 || ptr->exponent != 1 ){
  if(ptr->coefficient > 0 && flag==0 ){
    printf("%dx^%d", ptr->coefficient,ptr->exponent);
    flag++;
  }
  else if (ptr->coefficient > 0 && flag==1 )
    printf("+%dx^%d", ptr->coefficient,ptr->exponent);
  else if(ptr->coefficient < 0)
    printf("%dx^%d", ptr->coefficient,ptr->exponent);
    }
    else if (ptr->exponent == 0){
  if(ptr->coefficient > 0 && flag==0 ){
    printf("%d", ptr->coefficient);
    flag++;
  }
  else if (ptr->coefficient > 0 && flag==1 )
    printf("+%d", ptr->coefficient);
  else if(ptr->coefficient < 0)
    printf("%d", ptr->coefficient);
    }
    else if( ptr->exponent == 1 ){
  if(ptr->coefficient > 0 && flag==0 ){
    printf("%dx", ptr->coefficient);
    flag++;
  }
  else if (ptr->coefficient > 0 && flag==1 )
    printf("+%dx", ptr->coefficient);
  else if(ptr->coefficient < 0)
    printf("%dx", ptr->coefficient);
    }
    ptr = ptr->next;
    i++;
  }
  printf("\n");
  return;
}

int main (int argc, char *argv[]) {
  int coefficient, exponent, i, n;
  int count;
  printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=-=-=-=--=\n");
  printf("            Multiplication of Two Polynomials\n");
  printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=-=-=-=--=\n");
  printf("Enter the number of coefficients in the multiplicand:");
  scanf("%d",&count);
  for(i=0;i<count;i++){
```

```
  printf("Enter the coefficient part:");
  scanf("%d", &coefficient);
  printf("Enter the exponent part:");
  scanf("%d",&exponent);
      polynomial_insert(&hPtr1, coefficient, exponent);
}
printf("Enter the number of coefficients in the multiplier:");
scanf("%d",&count);
for(i=0;i<count;i++){
  printf("Enter the coefficient part:");
  scanf("%d", &coefficient);
  printf("Enter the exponent part:");
  scanf("%d",&exponent);
      polynomial_insert(&hPtr2, coefficient, exponent);
}
    printf("Polynomial Expression 1: ");
    polynomial_view(hPtr1);
    printf("Polynomial Expression 2: ");
    polynomial_view(hPtr2);

    polynomial_multiply(&hPtr3, hPtr1, hPtr2);

    printf("Output:\n");
    polynomial_view(hPtr3);

printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
    hPtr1 = polynomial_deleteList(hPtr1);
    hPtr2 = polynomial_deleteList(hPtr2);
    hPtr3 = polynomial_deleteList(hPtr3);

    return 0;
```

OUTPUT:

```
Multiplication of Two Polynomials
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=–=
Enter the number of coefficients in the multiplicand:2
Enter the coefficient part:3
Enter the exponent part:2
Enter the coefficient part:2
Enter the exponent part:3
Enter the number of coefficients in the multiplier:2
Enter the coefficient part:4
Enter the exponent part:2
Enter the coefficient part:1
Enter the exponent part:3
Polynomial Expression 1: 2x^3+3x^2
Polynomial Expression 2: 1x^3+4x^2
```

Output:
2x^6+11x^5+12x^4

RESULT:

Thus C program to implement Polynomial Manipulation using linked list was executed and verified successfully.

Ex.No: 5 a CONVERTION OF INFIX TO POSTFIX EXPRESSION USING STACK

AIM

To write a C program to implement the conversion of infix to postfix expression using stack

ALGORITHM

Step 1: Start the program.

Step 2: Declare the variables

Step 3: Read one character from the input

Step 4: If the character is operator , push it onto the stack. If the stack operator has higher or equal priority, then pop that operator from the stack and place it onto the output.

Step 5: If the character is left parenthesis, push it onto the stack.

Step 6: If the character is a right parenthesis, pop all the operator from the stack till it encounter left parenthesis, discard both parenthesis in the output.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
void main()
{
char exp[30],post[25];
int top=-1,i;
clrscr();
printf("enter the notation:");
scanf("%s",exp);
printf("postfix notation:");
for(i=0;i<strlen(exp);i++)
{
if(isalpha(exp[i]))
{
printf("%c",exp[i]);
while(post[top]=='*'|| post[top]=='/'|| post[top]=='^')
printf("%c",post[top--]);
}
else if(exp[i]==')')
{
while(post[top]!='(')
printf("%c",post[top--]);
```

```
top--;
}
else
{
post[++top]=exp[i];
}  }
for(i=0;i<=top;i++)
if(post[i]!='('&&(post[i]!='*'|| post[i]!='-'|| post[i]!='^'))
printf("%c",post[i]);
getch();
}
```

OUTPUT

```
Enter the Notation : (a+b)*c
Postfix Notation is : ab+c*
```

RESULT:

Thus C program to implement conversion of infix to postfix was executed and verified successfully

Ex.No: 5b EVALUATION OF POSTFIX EXPRESSION USING STACK

AIM

To write a C program to evaluate postfix expression using stack

ALGORITHM

Step 1: Start the program.

Step 2: Declare the functions to be performed.

Step 3: Create an empty stack and start scanning the postfix from left to right

Step 4: If the element is an operand, push it into the stack

Step 5: If the element is an operator O, pop twice and get A and B repectively. Calculate BOA and push back to the stack

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
#define MAX 20
typedef struct stack
{
int data[MAX];
int top;
}stack;
void init(stack *);
int empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *,int);
int evaluate(char x,int op1,int op2);
int main()
{
stack s;
char x;
int op1,op2,val;
init(&s);
printf("Enter the expression(eg: 59+3*)\nSingle digit operand and operators only:");
while((x=getchar())!='\n')
{
if(isdigit(x))
push(&s,x-48);        //x-48 for removing the effect of ASCII
else
```

```
{
op2=pop(&s);
op1=pop(&s);
val=evaluate(x,op1,op2);
push(&s,val);
}
}
val=pop(&s);
printf("\nValue of expression=%d",val);
getch();
return 0;
}
int evaluate(char x,int op1,int op2)
{
if(x=='+')
return(op1+op2);
if(x=='-')
return(op1-op2);
if(x=='*')
return(op1*op2);
if(x=='/')
return(op1/op2);
if(x=='%')
return(op1%op2);
}
void init(stack *s)
{
s->top=-1;
}
int empty(stack *s)
{
if(s->top==-1)
return(1);
return(0);
}
int full(stack *s)
{
if(s->top==MAX-1)
return(1);
return(0);
}
void push(stack *s,int x)
{
s->top=s->top+1;
s->data[s->top]=x;
}
int pop(stack *s)
{
int x;
x=s->data[s->top];
s->top=s->top-1;
```

```
return(x);
}
```

OUTPUT

```
Enter the expression(eg: 59+3*)
Single digit operand and operators only:74+5-
Value of expression=6
```

RESULT:

Thus C program to implement Evaluation of Postfix expression was executed and output is verified successfully.

Ex.No: 6 IMPLEMENTATION OF BINARY SEARCH TREES

AIM

To write a C program to implement Binary Search Trees

ALGORITHM

Step 1: Start the program.

Step 2: Declare the left node, right node and root as pointer

Step 3: For Inorder Traversal,

- First Visit the Root, then left subtree and then right subtree

Step 4: For Pre-order Traversal,

- First left subtree, then visit the Root and then right subtree

Step 5: For Post-order Traversal,

- First left subtree, then right subtree and then visit root

Step 6: Display the result based on the choice.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
typedef struct tree *node;
node insert(int,node T);
void inorder(node T);
void preorder(node T);
void postorder(node T);
struct tree
{
int data;
struct tree *right,*left;
}*root;
void main()
{
node T=NULL;
int data,ch,i=0,n;
clrscr();
printf("\n Enter the no of elements in the tree:");
scanf("%d",&n);
printf("\n The Elements are:\n");
while(i<n)
{
scanf("%d",&data);
T=insert(data,T);
```

```
i++;
}
printf("1.INORDER\t 2.PREORDER\t 3.POSTORDER\t4.EXIT");
do
{
printf("\nEnter ur choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("Inorder traversal of the given Tree \n");
inorder(T);
break;
case 2:
printf("preorder traversal of the given Tree \n");
preorder(T);
break;
case 3:
printf("postorder traversal of the given Tree \n");
postorder(T);
break;
default:
printf("Exit");
exit(0);
}
}
while(ch<4);
getch();
}
node insert(int X,node T)
{
struct tree*newnode;
newnode=malloc(sizeof(struct tree));
if(newnode==NULL)
printf("Out of space");
else
{
if(T==NULL)
{
newnode->data=X;
newnode->left=NULL;
newnode->right=NULL;
T=newnode;
}
else
{
if(X<T->data)
T->left=insert(X,T->left);
else
T->right=insert(X,T->right);
}}
```

```
return T;
}
void inorder(node T)
{
if(T!=NULL)
{
inorder(T->left);
printf("%d \t",T->data);
inorder(T->right);
}
}
void preorder(node T)
{
if(T!=NULL)
{
printf("%d \t",T->data);
preorder(T->left);
preorder(T->right);
}
}
void postorder(node T)
{
if(T!=NULL)
{
postorder(T->left);
postorder(T->right);
printf("%d \t",T->data);
}
}
```

OUTPUT

```
Enter the no of elements in the tree :
The Elements are:
3  10  6  4  2  9
1.INORDER     2.PREORDER  3.POSTORDER    4.EXIT
Enter ur choice : 1
Inorder traversal of the given Tree
2  3  4  6  9  10
Enter ur choice : 2
Preorder traversal of the given Tree
3  2  10  6  4  9
Enter ur choice : 3
Postorder traversal of the given Tree
2  4  9  6  10  3
```

RESULT:

Thus C program to implement Binary Search Tree was executed and output is verified successfully.

Ex.No: 7 IMPLEMENTATION OF AVL TREES

AIM

To write a C program to implement AVL Trees and its operations

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables

Step 3:Insertion:

- Perform standard BST insert for w.

1.Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.

2.Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:

a) y is left child of z and x is left child of y (Left Left Case)
b) y is left child of z and x is right child of y (Left Right Case)
c) y is right child of z and x is right child of y (Right Right Case)
d) y is right child of z and x is left child of y (Right Left Case)

Step 6: Based on the choice perform the operations such as creation of list, inserting an element, displaying the list, deleting an element & searching an element. Step 6: Display the result based on the choice.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
typedef struct node
{
int data;
struct node *left,*right;
int ht;
}node;
node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);
int main()
```

```
{
node *root=NULL;
int x,n,i,op;
do
{
printf("\n1)Create:");
printf("\n2)Insert:");
printf("\n3)Delete:");
printf("\n4)Print:");
printf("\n5)Quit:");
printf("\n\nEnter Your Choice:");
scanf("%d",&op);
switch(op)
{
case 1: printf("\nEnter no. of elements:");
scanf("%d",&n);
printf("\nEnter tree data:");
root=NULL;
for(i=0;i<n;i++)
{
scanf("%d",&x);
root=insert(root,x);
}
break;
case 2: printf("\nEnter a data:");
scanf("%d",&x);
root=insert(root,x);
break;
case 3: printf("\nEnter a data:");
scanf("%d",&x);
root=Delete(root,x);
break;
case 4: printf("\nPreorder sequence:\n");
preorder(root);
printf("\n\nInorder sequence:\n");
inorder(root);
printf("\n");
break;
}
}while(op!=5);
return 0;
}
node * insert(node *T,int x)
{
if(T==NULL)
{
T=(node*)malloc(sizeof(node));
T->data=x;
T->left=NULL;
T->right=NULL;
}
```

```
else
if(x > T->data)        // insert in right subtree
{
T->right=insert(T->right,x);
if(BF(T)==-2)
if(x>T->right->data)
T=RR(T);
else
T=RL(T);
}
else
if(x<T->data)
{
T->left=insert(T->left,x);
if(BF(T)==2)
if(x < T->left->data)
T=LL(T);
else
T=LR(T);
}
T->ht=height(T);
return(T);
}
node * Delete(node *T,int x)
{
node *p;
if(T==NULL)
{
return NULL;
}
else
if(x > T->data)        // insert in right subtree
{
T->right=Delete(T->right,x);
if(BF(T)==2)
if(BF(T->left)>=0)
T=LL(T);
else
T=LR(T);
}
else
if(x<T->data)
{
T->left=Delete(T->left,x);
if(BF(T)==-2)    //Rebalance during windup
if(BF(T->right)<=0)
T=RR(T);
else
T=RL(T);
}
else
```

```
{
//data to be deleted is found
if(T->right!=NULL)
{    //delete its inorder succesor
p=T->right;
while(p->left!= NULL)
p=p->left;
T->data=p->data;
T->right=Delete(T->right,p->data);
if(BF(T)==2)//Rebalance during windup
if(BF(T->left)>=0)
T=LL(T);
else
T=LR(T);\
}
else
return(T->left);
}
T->ht=height(T);
return(T);
}
int height(node *T)
{
int lh,rh;
if(T==NULL)
return(0);
if(T->left==NULL)
lh=0;
else
lh=1+T->left->ht;
if(T->right==NULL)
rh=0;
else
rh=1+T->right->ht;
if(lh>rh)
return(lh);
return(rh);
}

node * rotateright(node *x)
{
node *y;
y=x->left;
x->left=y->right;
y->right=x;
x->ht=height(x);
y->ht=height(y);
return(y);
}
node * rotateleft(node *x)
{
```

```
node *y;
y=x->right;
x->right=y->left;
y->left=x;
x->ht=height(x);
y->ht=height(y);
return(y);
}
node * RR(node *T)
{
T=rotateleft(T);
return(T);
}
node * LL(node *T)
{
T=rotateright(T);
return(T);
}
node * LR(node *T)
{
T->left=rotateleft(T->left);
T=rotateright(T);
return(T);
}
node * RL(node *T)
{
T->right=rotateright(T->right);
T=rotateleft(T);
return(T);
}

int BF(node *T)
{
int lh,rh;
if(T==NULL)
return(0);
if(T->left==NULL)
lh=0;
else
lh=1+T->left->ht;
if(T->right==NULL)
rh=0;
else
rh=1+T->right->ht;
return(lh-rh);
}
void preorder(node *T)
{
if(T!=NULL)
{
printf("%d(Bf=%d)",T->data,BF(T));
```

```
preorder(T->left);
preorder(T->right);
}
}
void inorder(node *T)
{
if(T!=NULL)
{
inorder(T->left);
printf("%d(Bf=%d)",T->data,BF(T));
inorder(T->right);
}
}
```

OUTPUT

```
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:1
Enter no. of elements:4
Enter tree data:7 12 4 9
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:4
Preorder sequence:
7(Bf=-1)4(Bf=0)12(Bf=1)9(Bf=0)
Inorder sequence:
4(Bf=0)7(Bf=-1)9(Bf=0)12(Bf=1)

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:3
Enter a data:7

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:4
```

Preorder sequence:
9(Bf=0)4(Bf=0)12(Bf=0)

Inorder sequence:
4(Bf=0)9(Bf=0)12(Bf=0)

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:5

RESULT:

Thus C program to implement AVL Trees and its operations are executed and its output is verified successfully.

Ex.No: 8 IMPLEMENTATION OF HEAPS USING PRIORITY QUEUES

AIM

To write a C program to implement Heaps using Priority Queues

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Binary Heap must be complete binary tree

Step 4: A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap

Step 5: Based on the choice perform the operations such as creation of list, inserting an element, displaying the list, deleting an element & searching an element.

Step 6: Display the result based on the choice.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<math.h>
#define MAX 100/*Declaring the maximum size of the queue*/
void swap(int*,int*);
main()
{
int choice,num,n,a[MAX],data,s;
void display(int[],int);
void insert(int[],int,int,int);
int del_hi_priori(int[],int,int);
int lb=0;
n=0;/*Lower bound of the array is initialized to 0*/
while(1)
{
printf(".....MAIN MENU.....\n");
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Display\n");
printf("4.Quit \n");
printf("nEnter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter data to be inserted : ");
```

```
scanf("%d",&data);
insert(a,n,data,lb);
n++;
break;
case 2:
s=del_hi_priori(a,n+1,lb);
if(s!=0)
printf("\nThe deleted value is : %d \n",s);
if(n>0)
n--;
break;
case 3:
printf("\n");
display(a,n);
break;
case 4:
return;
default:
printf("Invalid choice.n");
}
printf("\n");
}
}
void insert(int a[],int heapsize,int data,int lb)
{
int i,p;
int parent(int);
if(heapsize==MAX)
{
printf("Queue Is Full!!\n");
return;
}
i=lb+heapsize;
a[i]=data;
while(i>lb&&a[p=parent(i)]<a[i])
{
swap(&a[p],&a[i]);
i=p;
}
}
int del_hi_priori(int a[],int heapsize,int lb)
{
int data,i,l,r,max_child,t;
int left(int);
int right(int);
if(heapsize==1)
{
printf("Queue Is Empty!!\n");
return 0;
}
t=a[lb];
```

```
swap(&a[lb],&a[heapsize-1]);
i=lb;
heapsize--;
while(1)
{
if((l=left(i))>=heapsize)
break;
if((r=right(i))>=heapsize)
max_child=l;
else
max_child=(a[l]>a[r])?l:r;
if(a[i]>=a[max_child])
break;
swap(&a[i],&a[max_child]);
i=max_child;
}
return t;
}
int parent(int i)
{
float p;
p=((float)i/2.0)-1.0;
return ceil(p);
}
int left(int i)
{
return 2*i+1;
}
int right(int i)
{
return 2*i+2;
}
void display(int a[],int n)
{
int i;
if(n==0)
{
printf("Queue Is Empty!!\n");
return;
}
for(i=0;i<n;i++)
printf("%d ",a[i]);
printf("\n");
}
void swap(int*p,int*q)
{
int temp;
temp=*p;
*p=*q;
*q=temp;
}
```

OUTPUT :

```
.....MAIN MENU.....

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 52

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 63

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 45

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 2

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 1
Enter data to be inserted : 99

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 3
99 63 45 2 52

.....MAIN MENU.....
```

1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 99

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 3
63 52 45 2

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 63

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 2
The deleted value is : 52

.....MAIN MENU.....
1.Insert.
2.Delete.
3.Display.
4.Quit.
Enter your choice : 3
45 2

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

RESULT:

Thus C program to implement Binary heap using priority queue was executed and output is verified successfully.

Ex.No: 9 IMPLEMENTATION OF DIJKSTRA ALGORITHM

AIM

To write a C program to implement Dijkstra Algorithm

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables

Step 3: Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty

Step 4: Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

Step 5: While *sptSet* doesn't include all vertices

Step 6: Pick a vertex u which is not there in *sptSet* and has minimum distance value

Step 7: include u to sptset

Step 8: Update distance val u.

Step 9: Stop the program

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
```

```
int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
}
```

OUTPUT

```
Enter no of vertices: 5
Enter the adjacency matrix:
0  10  0  30  100
10  0  50  0  0
0  50  0   20 10
30  0  20  0  60
100  0  10  60  0

Enter starting node:0
Distance of node 1=10
Path= 1<-0

Distance of node 2=50
Path= 2<-3<-1<-0

Distance of node 3=30
Path= 3<-0

Distance of node 4=60
Path=4<- 2<-3<-1<-0
```

RESULT:

Thus C program to implement Dijkstra's algorithm was executed and output is verified successfully.

Ex.No: 10 IMPLEMENTATION OF PRIMS ALGORITHM

AIM

To write a C program to implement Prim's Algorithm

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Initialize the set U as {1}

Step 4: Get the adjacency matrix and starting vertex

Step 5: At each setp, it finds a shortest edge(u,v) such that the cost of (u,v) is the smallest among all the edges, where 'u' is in MST and 'v' is not in MST

Step 6: Calculate the total cost of spanning Tree

Step 7: Display the Minimum spanning Tree

Step 8: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20
int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();
int main()
{
int i,j,total_cost;
printf("Enter no. of vertices:");
scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);

total_cost=prims();
printf("\nspanning tree matrix:\n");

for(i=0;i<n;i++)
{
printf("\n");
```

```
for(j=0;j<n;j++)
printf("%d\t",spanning[i][j]);
}
printf("\n\nTotal cost of spanning tree=%d",total_cost);
return 0;
}
int prims()
{
int cost[MAX][MAX];
int u,v,min_distance,distance[MAX],from[MAX];
int visited[MAX],no_of_edges,i,min_cost,j;

//create cost[][] matrix,spanning[][]
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(G[i][j]==0)
cost[i][j]=infinity;
else
cost[i][j]=G[i][j];
spanning[i][j]=0;
}

distance[0]=0;
visited[0]=1;
for(i=1;i<n;i++)
{
distance[i]=cost[0][i];
from[i]=0;
visited[i]=0;
}

min_cost=0;
no_of_edges=n-1;
while(no_of_edges>0)
{
min_distance=infinity;
for(i=1;i<n;i++)
if(visited[i]==0&&distance[i]<min_distance)
{
v=i;
min_distance=distance[i];
}
u=from[v];
spanning[u][v]=distance[v];
spanning[v][u]=distance[v];
no_of_edges--;
visited[v]=1;
for(i=1;i<n;i++)
if(visited[i]==0&&cost[i][v]<distance[i])
{
```

```
distance[i]=cost[i][v];
from[i]=v;
}
min_cost=min_cost+cost[u][v];
}
return(min_cost);
}
```

OUTPUT

```
Enter no. of vertices:6
Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

spanning tree matrix:
0 3 1 0 0 0
3 0 0 0 3 0
1 0 0 0 0 4
0 0 0 0 0 2
0 3 0 0 0 0
0 0 4 2 0 0

Total cost of spanning tree=13
```

RESULT:

Thus C program to implement Prim's algorithm was executed and output is verified successfully.

Ex.No: 11a IMPLEMENTATION OF SEARCHING AND SORTING (LINEAR SEARCH)

AIM

To write a C program to implement List ADT using Array.

ALGORITHM

Step 1: Start the program.

Step 2: Declare the functions to be performed.

Step 3: Declare the required variables.

Step 4: A Linear search is made over all items one by one.

Step 5: Every item is checked and if a match is found then that particular item is returned

Step 6: Otherwise the search continues till the end of the data collection

Step 7: Stop the program.

PROGRAM

```
#include <stdio.h>
int main()
{
int array[100], search, c, n;
printf("Enter the number of elements in array\n");
scanf("%d", &n);
printf("Enter %d integer(s)\n", n);
for (c = 0; c < n; c++)
scanf("%d", &array[c]);
printf("Enter a number to search\n");
scanf("%d", &search);
for (c = 0; c < n; c++)
{
if (array[c] == search)    /* If required element is found */
{
printf("%d is present at location %d.\n", search, c+1);
break;
}
}
if (c == n)
printf("%d isn't present in the array.\n", search);
return 0;
}
```

OUTPUT

```
Enter the number of elements in a array: 5
Enter 5 integers: 3  7  0  2  1
Enter the number to search : 0
The number 0 is present at the location 3
```

RESULT:

Thus C program to implement Linear Search was executed and output is verified successfully.

Ex.No: 11b IMPLEMENTATION OF SEARCHING AND SORTING (BINARY SEARCH)

AIM

To write a C program to implement List ADT using Array.

ALGORITHM

Step 1: Start the program.

Step 2: Start the mid element

- If the target value is equal to middle element of the array, then return index of the mid element
- If not, then compare the mid element with target value.
    - If target value is greater than mid index then pick elements to the right of mid index and start with step1.
    - If target value is less than mid index then pick elements to the left of mid index and start with step1

Step 3: When the match is found, return index of the element matched..

Step 4: If no match is found, then return -1

Step 5: Stop the program.

PROGRAM

```
//BINARY SEARCH
#include<stdio.h>
#include<conio.h>
void main()
{
int flag=0,a[20],n,pos,i,item,low,high,mid;
void sort(int[],int);
clrscr();
printf("Enter the no.of numbers\n");
scanf("%d",&n);
printf("Enter the numbers\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
sort(a,n);
printf("Enter the data to be searched");
scanf("%d",&item);
low=1;
high=n;
while(low<=high)
{
mid=(low+high)/2;
if(item==a[mid])
{
flag=1;
break;
```

```
}
else if(item>a[mid])
low=mid+1;
else
high=mid-1;
}
if(flag==0)
printf("The element %d is not present in the array \n",item);
else
printf("The element %d is present in the array \n",item);
getch();
}
void sort(int a[],int n)
{
int i,j,temp;
for(i=1;i<=n;i++)
for(j=i+1;j<=n;j++)
if(a[i]>a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
```

OUTPUT

```
Enter the number of elements in a array: 5
Enter the numbers: 3  7  0  2  1
Enter the data to be searched : 0
The number 0 is present in the array
```

RESULT:

Thus C program to implement Binary Search was executed and output is verified successfully.

Ex.No: 11c IMPLEMENTATION OF SEARCHING AND SORTING (BUBBLE SORT)

AIM

To write a C program to implement sorting and searching technique using bubble sort

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables

Step 3: Pairs of adjacent elements are compared

Step 4: The elements are swapped in case the one on the left is greater then the element on the right.

Step 5: It can bubble the largest element to the last position

Step 6: Display the numbers in sorted order

Step 7: Stop the program

PROGRAM

```
#include <stdio.h>
int main()
{
int array[100], n, c, d, swap;
clrscr();
printf("Enter number of elements\n");
scanf("%d", &n);
printf("Enter %d integers\n", n);
for (c = 0; c < n; c++)
scanf("%d", &array[c]);
for (c = 0 ; c < n - 1; c++)
{
for (d = 0 ; d < n - c - 1; d++)
{
if (array[d] > array[d+1]) /* For decreasing order use < */
{
swap       = array[d];
array[d]   = array[d+1];
array[d+1] = swap;
}
}
}
printf("Sorted list in ascending order:\n");
for (c = 0; c < n; c++)
```

```
printf("%d\n", array[c]);
getch();
return 0;

}
```

OUTPUT

```
Enter number of elements : 5
Enter 5 integers: 3 2 0 6  9
Sorted list in ascending order:  0  2   3  6  9
```

RESULT:

Thus C program to implement Bubble sort was executed and output is verified successfully.

Ex.No: 11d IMPLEMENTATION OF SEARCHING AND SORTING (INSERTION SORT)

AIM

To write a C program to implement searching and sorting technique using insertion sort

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Take the elements from the list one by one and inserting them in their position into new sorted list

Step 4: It usually consists of N-1 passes

Step 5: During ith pass, it will insert the ith element A[i] into right place.

Step 6: Display the numbers in sorted order

Step 7: Stop the program.

PROGRAM

```
#include <stdio.h>
int main()
{
int n, array[1000], c, d, t;
clrscr();
printf("Enter number of elements\n");
scanf("%d", &n);
printf("Enter %d integers\n", n);
for (c = 0; c < n; c++) {
scanf("%d", &array[c]);
}

for (c = 1 ; c <= n - 1; c++) {
d = c;
while ( d > 0 && array[d-1] > array[d]) {
t          = array[d];
array[d]   = array[d-1];
array[d-1] = t;
d--;
}
}

printf("Sorted list in ascending order:\n");
for (c = 0; c <= n - 1; c++) {
```

```
printf("%d\n", array[c]);
}
getch();
return 0;
}
```

OUTPUT:

```
Enter number of elements : 5
Enter 5 integers: 3 2 0 6  9
Sorted list in ascending order:  0  2   3  6  9
```

RESULT:

Thus C program to implement Insertion Sort was executed and output is verified successfully.

Ex.No: 11e IMPLEMENTATION OF SEARCHING AND SORTING (SHELL SORT)

AIM

To write a C program to implement searching and sorting technique using shell sort.

ALGORITHM

Step 1: Start the program.

Step 2: Declare the required variables.

Step 3: Divide the whole array into K-segments

Step 4: After first pass, whole array is partially sorted.

Step 5: In the next pass, value of K is reduced and then sort the reduced array elements

Step 6: This process continues until K=1

Step 7: Stop the program.

PROGRAM

```
#include <stdio.h>
void shellsort(int arr[], int num)
{
int i, j, k, tmp;
for (i = num / 2; i > 0; i = i / 2)
{
for (j = i; j < num; j++)
{
for(k = j - i; k >= 0; k = k - i)
{
if (arr[k+i] >= arr[k])
break;
else
{
tmp = arr[k];
arr[k] = arr[k+i];
arr[k+i] = tmp;
}
}
}
}
}
int main()
{
int arr[30];
```

```
int k,  num;
clrscr();
printf("Enter total no. of elements : ");
scanf("%d", &num);
printf("\nEnter %d numbers: ", num);

for (k =  0 ; k < num; k++)
{
scanf("%d", &arr[k]);
}
shellsort(arr, num);
printf("\n Sorted array is: ");
for (k = 0; k < num; k++)
printf("%d ", arr[k]);
getch();
return 0;
}
```

OUTPUT

```
Enter total no.of elements:5
Enter 4 numbers:
4  2  0  1  8
Sorted array is :  0  1  2  4  8
```

RESULT:

Thus C program to implement Shell Sort was executed and output is verified successfully.

Ex.No: 11f IMPLEMENTATION OF SEARCHING AND SORTING (MERGE SORT)

AIM

To write a C program to implement searching and sorting technique using merge sort

ALGORITHM

Step 1: Start the program.

Step 2: Declare the functions to be performed.

Step 3: Take two input array as A and B and one output array as C

Step 4: First element of A array and B array are compared, then the smallest element is stored in the output array C

Step 5: Corresponding pointer is incremented

Step 6: Display the numbers in sorted order

Step 7: Stop the program.

PROGRAM

```
#include <stdio.h>
#define max 10

int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
int b[10];
void merging(int low, int mid, int high) {
int l1, l2, i;
for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
if(a[l1] <= a[l2])
b[i] = a[l1++];
else
b[i] = a[l2++];
}
while(l1 <= mid)
b[i++] = a[l1++];
while(l2 <= high)
b[i++] = a[l2++];
for(i = low; i <= high; i++)
a[i] = b[i];
}
void sort(int low, int high) {
int mid;
if(low < high) {
mid = (low + high) / 2;
```

```
sort(low, mid);
sort(mid+1, high);
merging(low, mid, high);
} else {
return;
}
}
int main()
{
int i;
clrscr();
printf("List before sorting\n");
for(i = 0; i <= max; i++)
printf("%d ", a[i]);
sort(0, max);
printf("\nList after sorting\n");
for(i = 0; i <= max; i++)
printf("%d ", a[i]);
getch();
return 0;
}
```

OUTPUT

```
List before sorting
10  14  19  26  27  31  33  35  42  44  0

List after sorting
0  10  14  19  26  27  31  33  35  42  44
```

RESULT:

Thus C program to implement Shell Sort was executed and output is verified successfully.

Ex.No: 11g IMPLEMENTATION OF SEARCHING AND SORTING (QUICK SORT)

AIM

To write a C program to implement List ADT using Array.

ALGORITHM

Step 1: Start the program.

Step 2: Declare the functions to be performed.

Step 3: Pick last element as pivot

Step 4: The key process in quickSort is partition().

Step 5: Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

Step 6: All this should be done in linear time and display the numbers in sorted order

Step 7: Stop the program.

PROGRAM

```
#include <stdio.h>
void sort_numbers_ascending(int number[], int count)
{
int temp, i, j, k;
for (j = 0; j < count; ++j)
{
for (k = j + 1; k < count; ++k)
{
if (number[j] > number[k])
{
temp = number[j];
number[j] = number[k];
number[k] = temp;
}
}
}
printf("Numbers in ascending order:\n");
for (i = 0; i < count; ++i)
printf("%d\n", number[i]);
}
void main()
{
```

```
int i, count, number[20];
clrscr();
printf("How many numbers you are gonna enter:");
scanf("%d", &count);
printf("\nEnter the numbers one by one:");
for (i = 0; i < count; ++i)
scanf("%d", &number[i]);
sort_numbers_ascending(number, count);
getch();
}
```

OUTPUT

```
How many numbers you are gonna enter: 8
Enter the numbers one by one
2  10  5  2  89  3  1  9

Numbers in ascending order:
1  2  2  3  5   9  10  89
```

RESULT:

Thus C program to implement Quick Sort was executed and output is verified successfully.

Ex.No: 12a **IMPLEMENTATION OF HASHING (LINEAR PROBING)**

AIM

To write a C program to implement Hashing using Linear Probing

ALGORITHM

Step 1: Start the program.

Step 2: Declare the functions to be performed.

Step 3: In collision situation, the **linear probing** table will look onto to subsequent hash elements until the first free space is found.

Step 4: This traversal is known as **probing** the table; and as it goes by one element at a time, it is **linear probing**.

Step 6: Display the result

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10
void main()
{
int a[MAX],num,key,i;
char ans;
int create(int);
void linear_prob(int[],int,int);
void display(int[]);
clrscr();
printf("\nCollision Handling by Linear Probing");
for(i=0;i<MAX;i++)
a[i]=-1;
do
{
printf("\nEnter the number:");
scanf("%d",&num);
key=create(num);
linear_prob(a,key,num);
printf("\nDo you want to continue?");
ans=getch();
}while(ans=='y');
```

```
display(a);
}
int create(int num)
{
int key;
key=num%10;
return key;
}
void linear_prob(int a[MAX],int key,int num)
{
int flag,i,count=0;
void display(int a[]);
flag=0;
if(a[key]==-1)
a[key]=num;
else
{
i=0;
while(i<MAX)
{
if(a[i]!=-1)
count++;
i++;
}
if(count==MAX)
{
printf("\nHash Table is full");
display(a);
getch();
exit(1);
}
for(i=key+1;i<MAX;i++)
if(a[i]==-1)
{
a[i]=num;
flag=1;
break;
}
for(i=0;i<key&&flag==0;i++)
if(a[i]==-1)
{
a[i]=num;
flag=1;
break;
}
}

}

void display(int a[MAX])
{
```

```
int i;
printf("\nThe Hash Table is ...\n");
for(i=0;i<MAX;i++)
printf("\n %d  %d",i,a[i]);
getch();
}
```

OUTPUT

Collision handling by Linear Probing

Do u want to continue?
Enter the number:
2

Do u want to continue?
Enter the number:
6

Do u want to continue?
The Hash Table is ….

0	-1
1	-1
2	2
3	-1
4	-1
5	5
6	8
7	-1

RESULT:

Thus C program to implement Hashing using Linear Probing was executed and output is verified successfully.

Ex.No: 12b **IMPLEMENTATION OF HASHING (QUADRATIC PROBING)**

AIM

To write a C program to implement Hashing using Quadratic Probing

ALGORITHM

1.Create an array of structure (i.e a hash table).

2. Take a key and a value to be stored in hash table as input.

3. Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.

4. Using the generated index, access the data located in that array index.

5. In case of absence of data, create one and insert the data item (key and value) into it and increment the size of hash table.

6. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item.

(currentPosition + (h * h)) % arraySize => Quadratic Probing
where h = 1, 2, 3, 4 and so on.

7. To display all the elements of hash table, element at each index is accessed (via for loop).

8. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements until we find key or an empty space where not a single data has been entered (means data does not exist in the hash table).

9. Exit

PROGRAM

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

/* to store a data (consisting of key and value) in hash table array */
struct item
{
        int key;
        int value;
};

/* each hash table item has a flag (status) and data (consisting of key and value) */
struct hashtable_item
{

        int flag;
```

```
        /*
         * flag = 0 : data does not exist
         * flag = 1 : data exists at given array location
         * flag = 2 : data was present at least once
        */
        struct item *data;

};

struct hashtable_item *array;
int size = 0;
int max = 10;

/* this function returns corresponding index of the given key */
int hashcode(int key)
{
        return (key % max);
}

/* this  function initializes the hash table array */
void init_array()
{
        int i;
        for (i = 0; i < max; i++)
    {
                array[i].flag = 0;
                array[i].data = NULL;
        }
}

/* this function inserts an element in the hash table */
void insert(int key, int value)
{
        int index = hashcode(key);

        int i = index;
        int h = 1;

        struct item *new_item = (struct item*) malloc(sizeof(struct item));
        new_item->key = key;
        new_item->value = value;

        /* probing through the array until an empty space is found */
        while (array[i].flag == 1)
    {

                if (array[i].data->key == key)
          {

        /* case when already present key matches the given key */
        printf("\n This key is already present in hash table, hence updating it's value \n");
                        array[i].data->value = value;
                        return;
```

```
                        }
                        i = (i + (h * h)) % max;
                        h++;
                        if (i == index)
                  {
                                printf("\n Hash table is full, cannot add more elements \n");
                                return;
                        }

                }

                array[i].flag = 1;
                array[i].data = new_item;
                printf("\n Key (%d) has been inserted\n", key);
                size++;

        }

        /* to remove an element form the hash table array */
        void remove_element(int key)
        {
                int index = hashcode(key);
                int i = index;
                int h = 1;

                /* probing through the hash table until we reach at location where there had not been
an element even once */
                while (array[i].flag != 0)
              {
                        if (array[i].flag == 1  &&  array[i].data->key == key)
                  {

                                /* case where data exists at the location and its key matches to the
given key */
                                array[i].flag = 2;
                                array[i].data = NULL;
                                size--;
                                printf("\n Key (%d) has been removed \n", key);
                                return;

                        }
                        i = (i + (h * h)) % max;
                        h++;
                        if (i == index)
                  {
                                break;
                        }
                }
                printf("\n Key does not exist \n");

        }

        /* to display the contents of hash table */
        void display()
```

```
{

        int i;
        for(i = 0; i < max; i++)
    {
            if (array[i].flag != 1)
      {
                printf("\n Array[%d] has no elements \n", i);
            }
            else
      {
                printf("\n Array[%d] has elements \n  %d (key) and %d (value) \n",
i, array[i].data->key, array[i].data->value);
            }
        }

}

int size_of_hashtable()
{
        return size;
}

void main()
{
        int choice, key, value, n, c;
        clrscr();

        array = (struct hashtable_item*) malloc(max * sizeof(struct hashtable_item*));
        init_array();

        do {
            printf("Implementation of Hash Table in C with Quadratic Probing.\n\n");
            printf("MENU-: \n1.Inserting item in the Hash table"
               "\n2.Removing item from the Hash table"
               "\n3.Check the size of Hash table"
               "\n4.Display Hash table"
                  "\n\n Please enter your choice-:");

            scanf("%d", &choice);

            switch(choice)
      {

            case 1:

                printf("Inserting element in Hash table \n");
                printf("Enter key and value-:\t");
                scanf("%d %d", &key, &value);
                insert(key, value);

                break;

            case 2:
```

```
                printf("Deleting in Hash table \n Enter the key to delete-:");
                scanf("%d", &key);
                remove_element(key);

                break;

            case 3:

                n = size_of_hashtable();
                printf("Size of Hash table is-:%d\n", n);

                break;

            case 4:

                display();

                break;

            default:

                 printf("Wrong Input\n");

            }

            printf("\n Do you want to continue-:(press 1 for yes)\t");
            scanf("%d", &c);

        }while(c == 1);

        getch();

    }
```

OUTPUT:

```
Implementation of Hash Table in C with Quadratic Probing
MENU-:
1. Inserting item in the Hash table
2. Removing item from the Hash table
3. Check the size of Hash table
4. Display Hash table

Please enter your choice-: 3
Size of hash table is-: 0

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Quadratic Probing
MENU-:
1. Inserting item in the Hash table
```

2. Removing item from the Hash table
3. Check the size of Hash table
4. Display Hash table

Please enter your choice-: 1
Inserting element in Hash table
Enter key and value-: 12 10

Key (12) has been inserted

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Quadratic Probing
MENU-:
1. Inserting item in the Hash table
2. Removing item from the Hash table
3. Check the size of Hash table
4. Display Hash table

Please enter your choice-: 1
Inserting element in hash table
Enter key and value-: 122 4

Key (122) has been inserted

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Quadratic Probing
MENU-:
1. Inserting item in the Hash table
2. Removing item from the Hash table
3. Check the size of Hash table
4. Display Hash table

Please enter your choice-: 1
Inserting element in hash table
Enter key and value-: 82 5

Key (82) has been inserted

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Quadratic Probing
MENU-:
1. Inserting item in the Hash table
2. Removing item from the Hash table
3. Check the size of Hash table
4. Display Hash table

Please enter your choice-: 3
Size of hash table is-: 3

Do you want to continue-:(press 1 for yes) 1

Implementation of Hash Table in C with Quadratic Probing
MENU-:
1. Inserting item in the Hash table
2. Removing item from the Hash table
3. Check the size of Hash table
4. Display Hash table

Please enter your choice-: 4
Array[0] has no elements

Array[1] has no elements

Array[2] has elements-:
12 (key) and 10 (value)

Array[3] has elements-:
122(key) and 4(value)

Array[4] has no elements

Array[5] has no elements

Array[6] has no elements

Array[7] has no elements
82(key) and 5(value)
Array[8] has no elements

Array[9] has no elements

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Quadratic Probing
MENU-:
1. Inserting item in the Hash table
2. Removing item from the Hash table
3. Check the size of Hash table
4. Display Hash table

Please enter your choice-: 2
Deleting in hash table
Enter the key to delete-: 122

Key (122) has been removed

Do you want to continue-:(press 1 for yes) 1
Implementation of Hash Table in C with Quadratic Probing
MENU-:
1. Inserting item in the Hash table
2. Removing item from the Hash table
3. Check the size of Hash table

4. Display Hash table

Please enter your choice-: 2
Deleting in hash table
Enter the key to delete-: 56

This key does not exist

Do you want to continue-:(press 1 for yes) 2

RESULT:

Thus C program to implement Hashing using Quadratic Probing was executed and output is verified successfully.