MAHA BARATHI ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

SUBJECT CODE & NAME : CS3311 DATA STRUCTURES LABORATORY LAB
YEAR & SEMESTER : II / IV
REGULATION 2021

PREPARED BY	APPROVED BY
P.Kumareson ME .	Mr.N.Khadirkumar M.TECH(CSE HOD).

CS3311 DATA STRUCTURES LABORATORY

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)

EX.NO:1(a) ARRAY IMPLEMENTATION OF STACK ADT

AIM

To write a C program to implement stack operations using array.

ALGORITHM

STEP 1: Start the program

STEP2: Define an array stack of size max = 5

STEP 3: Initialize top = -1

STEP 4: Display a menu listing stack operation

STEP 5: Accept choice

STEP 6: If choice = 1 then

If top < max -1

Increment top

Store element at current position of top

Else

Print Stack overflow

Else If choice = 2 then

If top < 0 then

Print Stack underflow

Else

Display current top element

Decrement top

Else If choice = 3 then

Display stack elements starting from top

STEP 7: Stop the program

PROGRAM:

```
#include  <stdio.h>
#include  <conio.h>
#define max 5
static int stack[max];
int top = -1;
void push(int x)
{
stack[++top] = x;
}
int pop()
{
return (stack[top--]);
}
void view()
{
int i;
if (top < 0)
printf("\n Stack Empty \n");
else
{
printf("\n Top-->");
for(i=top; i>=0; i--)
{
printf("%4d", stack[i]);
}
printf("\n");
```

```
}
}
main()
{
int ch=0, val;
clrscr();
while(ch != 4)
{
printf("\n STACK OPERATION \n");
printf("1.PUSH ");
printf("2.POP ");
printf("3.VIEW ");
printf("4.QUIT \n");
printf("Enter Choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
if(top < max-1)
{
printf("\nEnter Stack element : ");
scanf("%d", &val);
push(val);
}
else
printf("\n Stack Overflow \n");
break;
```

```
case 2:
if(top < 0)
printf("\n Stack Underflow \n");
else
{
val = pop();
printf("\n Popped element is %d\n", val);
}
break;
case 3:
view();
break;
case 4:
exit(0);
default:
printf("\n Invalid Choice \n");
}
}
}
```

OUTPUT

```
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element : 12
```

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 1

Enter Stack element : 23

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 1

Enter Stack element : 34

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 1

Enter Stack element : 45

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 3

Top--> 45 34 23 12

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 2

Popped element is 45

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 3

Top--> 34 23 12

STACK OPERATION

1.PUSH 2.POP 3.VIEW 4.QUIT

Enter Choice : 4

RESULT

Thus push and pop operations of a stack was demonstrated using arrays.

EX.NO:1(b) ARRAY IMPLEMENTATION OF QUEUE ADT

AIM

To write a C program to implement queue operations using array.

ALGORITHM

STEP1: Start the program

STEP2: Define an array queue of size max = 5

STEP3: Initialize front = rear = –1

STEP4: Display a menu listing queue operation

STEP 5: Accept choice

STEP 6: If choice = 1 then

If rear < max -1

Increment rear

Store element at current position of rear

Else

Print Queue Full

Else If choice = 2 then

If front = –1 then

Print Queue empty

Else

Display current front element

Increment front

Else If choice = 3 then

Display queue elements starting from front to rear.

STEP 7: Stop the program

PROGRAM

```
#include  <stdio.h>
#include <conio.h>
#define max 5
static int queue[max];
int front = -1;
int rear = -1;
void insert(int x)
{
queue[++rear] = x;
if (front == -1)
front = 0;
}
int remove()
{
int val;
val = queue[front];
if (front==rear && rear==max-1)
front = rear = -1;
else
front ++;
return (val);
}
void view()
{
int i;
if (front == -1)
```

```
printf("\n Queue Empty \n");
else
{
printf("\n Front-->");
for(i=front; i<=rear; i++)
printf("%4d", queue[i]);
printf(" <--Rear\n");
}
}
main()
{
int ch= 0,val;
clrscr();
while(ch != 4)
{
printf("\n QUEUE OPERATION \n");
printf("1.INSERT ");
printf("2.DELETE ");
printf("3.VIEW ");
printf("4.QUIT\n");
printf("Enter Choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
if(rear < max-1)
{
```

```
printf("\n Enter element to be inserted : ");
scanf("%d", &val);
insert(val);
}
else
printf("\n Queue Full \n");
break;
case 2:
if(front == -1)
printf("\n Queue Empty \n");
else
{
val = remove();
printf("\n Element deleted : %d \n", val);
}
break;
case 3:
view();
break;
case 4:
exit(0);
default:
printf("\n Invalid Choice \n");
}
}
}
```

OUTPUT

QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 12

QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 23

QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 34

QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 45

QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Enter element to be inserted : 56

QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 1

Queue Full

QUEUE OPERATION

1.INSERT 2.DELETE 3.VIEW 4.QUIT

Enter Choice : 3

Front--> 12 23 34 45 56 <--Rear

RESULT

Thus insert and delete operations of a queue was demonstrated using arrays.

EX.NO:1(C) ARRAY IMPLEMENTATION OF CIRCULAR QUEUE

AIM

To write a C program for circular queue by using array implementation.

ALGORITHM

STEP1: Start the program

STEP2: Follow the below enqueue and dequeue algorithm

ENQUEUE(X) OPERATION:

You should follow the following steps to insert (Enqueue) a data element into a circular queue.

STEP1: Check if the queue is full (rear+1 % Max size=front)

STEP2: If the queue is full, there will be an overflow error

STEP3: Check if the queue is empty, and set both front and rear to 0

STEP4: If rear = Max size-1& front! =0 (rear pointer is at the end of the queue and front is not at 0th index), then set rear =0

STEP5: Otherwise set rear = (rear+1) % Max size

STEP6: Insert the element into the queue (queue [rear] = X)

STEP7: Exit

DEQUEUE(X) OPERATION:

You should follow the following to delete (Dequeue) a data element into a circular queue

STEP1: Check if the queue is empty (front= -1 & rear= -1)

STEP2: If the queue is empty, there will be an underflow error

STEP3: Set element =queue [front]

STEP4: If there is only one element in a queue set both front and rear to -1

(if front =rear, set front= rear = -1)

STEP5: And if front = Max size -1 set front = 0

STEP6: Otherwise, set front = front + 1

STEP7: Exit

STEP3: Stop the program.

PROGRAM

```
#include <stdio.h>
#include <Windows.h>
#define MAX_SIZE 5
int a[MAX_SIZE];
int front = -1;
int rear = -1;
void enqueue(int x)
{
if (front == -1 && rear == -1)
{
front = rear = 0;
}
else if ((rear + 1) % MAX_SIZE == front)
{
printf("queue is full\n");
return;
}
else
rear = (rear + 1) % MAX_SIZE;
```

```
a[rear] = x;
}
void dequeue()
{
if (front == -1 && rear == -1)
{
printf("queue is empty \n");
return;
}
else if (front == rear)
{
front = rear = -1;
}
else
front = (front + 1) % MAX_SIZE;
}
int Peek()
{
if (a[front] == -1)
return -1;
return a[front];
}
void Print()
{
int count = ((rear + MAX_SIZE - front) % MAX_SIZE)+1;
int i;
for (i = 0; i < count; i++)
```

```
{
printf("%d ", a[(front+i)%MAX_SIZE]);
}
printf("\n");
}
int main(void)
{
enqueue(5);
printf("\nFirst insertion in circular Queue\n");
Print();
printf("\n Second insertion in circular Queue\n");
enqueue(7);
Print();
printf("\n Third insertion in circular Queue\n");
enqueue(-3);
Print();
printf("\n Fourth insertion in circular Queue\n");
enqueue(9);
Print();
printf("\n Circular Queue after first deletion\n");
dequeue();
Print();
printf("\n Circular Queue after 2nd deletion\n");
dequeue();
Print();
printf("\n Insertion in circular Queue\n");
enqueue(14);
```

```
system("pause");
return 0;
}
```

OUTPUT

```
First insertion in Circular Queue
5
Second insertion in Circular Queue
5 7
Third insertion in Circular Queue
5 7 -3
Fourth insertion in Circular Queue
5 7 -3 9
Circular Queue after first deletion
7 -3 9
Circular Queue after Second deletion
-3 9
Insertion in Circular Queue
Press any key to continue……
```

RESULT

Thus enqueue and dqueue operations of a circular queue was demonstrated successfully using arrays.

EX.NO:2 IMPLEMENTATION OF SINGLY LINKED LIST

AIM

To write a C program for a singly linked list node and perform operations such as insertions and deletions dynamically.

ALGORITHM

STEP1: Start the program

STEP2: Define single linked list node as self-referential structure

STEP3: Create Head node with label = -1 and next = NULL using

STEP4: Display menu on list operation

STEP5: Accept user choice

STEP6: If choice = 1 then

Locate node after which insertion is to be done

Create a new node and get data part

Insert new node at appropriate position by manipulating address

Else if choice = 2

Get node's data to be deleted.

Locate the node and delink the node

Rearrange the links

Else

Traverse the list from Head node to node which points to null

STEP7: Stop the program

PROGRAM

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

```
#include <alloc.h>
#include <string.h>
struct node
{
int label;
struct node *next;
};
main()
{
int ch, fell=0;
int k;
struct node *h, *temp, *head, *h1;
/* Head node construction */
head = (struct node*) malloc(sizeof(struct node));
head->label = -1;
head->next = NULL;
while(-1)
{
clrscr();
printf("\n\n SINGLY LINKED LIST OPERATIONS \n");
printf("1->Add ");
printf("2->Delete ");
printf("3->View ");
printf("4->Exit \n");
printf("Enter your choice : ");
scanf("%d",  &ch);
switch(ch)
```

```
{
/* Add a node at any intermediate location */
case 1:
printf("\n Enter label after which to add : ");
scanf("%d", &k);
h = head;
fell = 0;
if (h->label == k)
fell = 1;
while(h->next != NULL)
{
if (h->label == k)
{
fell=1;
break;
}
h = h->next;
}
if (h->label == k)
fell = 1;
if (fell != 1)
printf("Node not found\n");
else
{
temp=(struct node *)(malloc(sizeof(struct node)));
printf("Enter label for new node : ");
scanf("%d", &temp->label);
```

```
temp->next = h->next;
h->next = temp;
}
break;
/* Delete any intermediate node */
case 2:
printf("Enter label of node to be deleted\n");
scanf("%d", &k);
fell = 0;
h = h1 = head;
while (h->next != NULL)
{
h = h->next;
if (h->label == k)
{
fell = 1;
break;
}
}
if (fell == 0)
printf("Sorry Node not found\n");
else
{
while (h1->next != h)
h1 = h1->next;
h1->next = h->next;
free(h);
```

```
printf("Node deleted successfully \n");
}
break;
case 3:
printf("\n\n HEAD -> ");
h=head;
while (h->next != NULL)
{
h = h->next;
printf("%d -> ",h->label);
}
printf("NULL");
break;
case 4:
exit(0);
}
}
}
```

OUTPUT

```
SINGLY LINKED LIST OPERATIONS
1->Add 2->Delete 3->View 4->Exit
Enter your choice : 1
Enter label after which new node is to be added : -1
Enter label for new node : 23

SINGLY LINKED LIST OPERATIONS
```

1->Add 2->Delete 3->View 4->Exit

Enter your choice : 1

Enter label after which new node is to be added : 23

Enter label for new node : 67

SINGLY LINKED LIST OPERATIONS

1->Add 2->Delete 3->View 4->Exit

Enter your choice : 3

HEAD -> 23 -> 67 -> NULL

RESULT

Thus the program for single linked list was executed and the output was verified successfully.

EX.NO:3(a) LINKED LIST IMPLEMENTATION OF STACK ADT

AIM

To implement stack operations using linked list.

ALGORITHM

STEP 1: Start the program

STEP 2: Define a singly linked list node for stack

STEP 3: Create Head node

STEP 4: Display a menu listing stack operation

STEP 5: Accept choice

STEP 6: If choice = 1 then

Create a new node with data

Make new node point to first node

Make head node point to new node

Else If choice = 2 then

Make temp node point to first node

Make head node point to next of temp node

Release memory

Else If choice = 3 then

Display stack elements starting from head node till null

STEP 7: Stop the program

PROGRAM

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

```
#include <process.h>
#include <alloc.h>
struct node
{
int label;
struct node *next;
};
main()
{
int ch = 0;
int k;
struct node *h, *temp, *head;
/* Head node construction */
head = (struct node*) malloc(sizeof(struct node));
head->next = NULL;
while(1)
{
printf("\n Stack using Linked List \n");
printf("1->Push ");
printf("2->Pop ");
printf("3->View ");
printf("4->Exit \n");
printf("Enter your choice : ");
scanf("%d",  &ch);
switch(ch)
{
case 1:
```

```
/* Create a new node */
temp=(struct node *)(malloc(sizeof(struct node)));
printf("Enter label for new node : ");
scanf("%d", &temp->label);
h = head;
temp->next = h->next;
h->next = temp;
break;
case 2:
/* Delink the first node */
h = head->next;
head->next = h->next;
printf("Node %s deleted\n", h->label);
free(h);
break;
case 3:
printf("\n HEAD -> ");
h = head;
/* Loop till last node */
while(h->next != NULL)
{
h = h->next;
printf("%d -> ",h->label);
}
printf("NULL \n");
break;
case 4:
```

```
exit(0);
}
}
}
```

OUTPUT

```
Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 1
Enter label for new node : 23
New node added

Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 1
Enter label for new node : 34

Stack using Linked List
1->Push 2->Pop 3->View 4->Exit
Enter your choice : 3
HEAD -> 34 -> 23 -> NULL
```

RESULT

Thus push and pop operations of a stack was demonstrated using linked list.

EX.NO:3(b) LINKED LIST IMPLEMENTATION OF QUEUE ADT

AIM

To implement queue operations using linked list.

ALGORITHM

STEP 1: Start the program

STEP 2: Define a singly linked list node for stack

STEP 3: Create Head node

STEP 4: Display a menu listing stack operation

STEP 5: Accept choice

STEP 6: If choice = 1 then

Create a new node with data

Make new node point to first node

Make head node point to new node

Else If choice = 2 then

Make temp node point to first node

Make head node point to next of temp node

Release memory

Else If choice = 3 then

Display stack elements starting from head node till null

STEP 7: Stop the program

PROGRAM

```
#include <conio.h>
#include <process.h>
#include <alloc.h>
```

```
struct node
{
int label;
struct node *next;
};
main()
{
int ch=0;
int k;
struct node *h, *temp, *head;
head = (struct node*) malloc(sizeof(struct node));
head->next = NULL;
while(1)
{
printf("\n Queue using Linked List \n");
printf("1->Insert ");
printf("2->Delete ");
printf("3->View ");
printf("4->Exit \n");
printf("Enter your choice : ");
scanf("%d",  &ch);
switch(ch)
{
case 1:
temp=(struct node *)(malloc(sizeof(struct node)));
printf("Enter label for new node : ");
scanf("%d", &temp->label);
```

```
h = head;
while (h->next != NULL)
h = h->next;
h->next = temp;
temp->next = NULL;
break;
case 2:
h = head->next;
head->next = h->next;
printf("Node deleted \n");
free(h);
break;
case 3:
printf("\n\nHEAD -> ");
h=head;
while (h->next!=NULL)
{
h = h->next;
printf("%d -> ",h->label);
}
printf("NULL \n");
break;
case 4:
exit(0);
}
}
}
```

OUTPUT

Queue using Linked List

1->Insert 2->Delete 3->View 4->Exit

Enter your choice : 1

Enter label for new node : 12

Queue using Linked List

1->Insert 2->Delete 3->View 4->Exit

Enter your choice : 1

Enter label for new node : 23

Queue using Linked List

1->Insert 2->Delete 3->View 4->Exit

Enter your choice : 3

HEAD -> 12 -> 23 -> NULL

RESULT

Thus insert and delete operations of a queue was demonstrated using linked list.

EX.NO:4(a) POLYNOMIAL ADDITION

AIM

To add any two given polynomial using linked lists.

ALGORITHM

STEP 1: Start the program

STEP 2: Create a structure for polynomial with exp and coeff terms.

STEP 3: Read the coefficient and exponent of given two polynomials p and q.

STEP 4: While p and q are not null, repeat step 4.

If powers of the two terms are equal then

Insert the sum of the terms into the sum Polynomial

Advance p and q

Else if the power of the first polynomial> power of second then

Insert the term from first polynomial into sum polynomial

Advance p

Else

Insert the term from second polynomial into sum polynomial

Advance q

STEP 5: Copy the remaining terms from the non-empty polynomial into the Sum polynomial

STEP 6: Stop the program.

PROGRAM

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

```
struct link
{
int coeff;
int pow;
struct link *next;
};
struct link *poly1=NULL,*poly2=NULL,*poly=NULL;
void create(struct link *node)
{
char ch;
do
{
printf("\nEnter coefficient: ");
scanf("%d", &node->coeff);
printf("Enter exponent: ");
scanf("%d", &node->pow);
node->next = (struct link*)malloc(sizeof(struct link));
node = node->next;
node->next = NULL;
printf("\n continue(y/n): ");
fflush(stdin);
ch=getch();
} while(ch=='y' || ch=='Y');
}
void show(struct link *node)
{
while(node->next!=NULL)
```

```
{
printf("%dx^%d", node->coeff, node->pow);
node=node->next;
if(node->next!=NULL)
printf(" + ");
}
}
void polyadd(struct link *poly1, struct link *poly2, struct link *poly)
{
while(poly1->next && poly2->next)
{
if(poly1->pow > poly2->pow)
{
poly->pow = poly1->pow;
poly->coeff = poly1->coeff;
poly1 = poly1->next;
}
else if(poly1->pow < poly2->pow)
{
poly->pow = poly2->pow;
poly->coeff = poly2->coeff;
poly2 = poly2->next;
}
else
{
poly->pow = poly1->pow;
poly->coeff = poly1->coeff + poly2->coeff;
```

```
poly1 = poly1->next;
poly2 = poly2->next;
}
poly->next=(struct link *)malloc(sizeof(struct link));
poly=poly->next;
poly->next=NULL;
}
while(poly1->next || poly2->next)
{
if(poly1->next)
{
poly->pow = poly1->pow;
poly->coeff = poly1->coeff;
poly1 = poly1->next;
}
if(poly2->next)
{
poly->pow = poly2->pow;
poly->coeff = poly2->coeff;
poly2 = poly2->next;
}
poly->next = (struct link *)malloc(sizeof(struct link));
poly = poly->next;
poly->next = NULL;
}
}
main()
```

```
{
poly1 = (struct link *)malloc(sizeof(struct link));
poly2 = (struct link *)malloc(sizeof(struct link));
poly = (struct link *)malloc(sizeof(struct link));
printf("Enter 1st Polynomial:");
create(poly1);
printf("\nEnter 2nd Polynomial:");
create(poly2);
printf("\nPoly1: ");
show(poly1);
printf("\nPoly2: ");
show(poly2);
polyadd(poly1, poly2, poly);
printf("\nAdded Polynomial: ");
show(poly);
}
```

OUTPUT

```
Enter 1st Polynomial:
Enter coefficient: 5
Enter exponent: 2
continue(y/n): y
Enter coefficient: 4
Enter exponent: 1
continue(y/n): y
Enter coefficient: 2
Enter exponent: 0
```

continue(y/n): n

Enter 2nd Polynomial:

Enter coefficient: 5

Enter exponent: 1

continue(y/n): y

Enter coefficient: 5

Enter exponent: 0

continue(y/n): n

Poly1: 5x^2 + 4x^1 + 2x^0

Poly2: 5x^1 + 5x^0

Added Polynomial: 5x^2 + 9x^1 + 7x^0

RESULT

Thus the polynomial operations using linked list was executed successfully and the output was verified successfully.

EX.NO:4 (b) POLYNOMIAL SUBTRACTION

AIM

To sub any two given polynomial using linked lists.

ALGORITHM

STEP 1: Start the program

STEP 2: Create a structure for polynomial with exp and coeff terms.

STEP 3: Read the coefficient and exponent of given two polynomials p and q.

STEP 4: While p and q are not null, repeat step 4.

If powers of the two terms are equal then

Insert the sub of the terms into the sub Polynomial

Advance p and q

Else if the power of the first polynomial> power of second then

Insert the term from first polynomial into sub polynomial

Advance p

Else

Insert the term from second polynomial into sub polynomial

Advance q

STEP 5: Copy the remaining terms from the non-empty polynomial into the sub polynomial

STEP 6: Stop the program.

PROGRAM

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

```
struct link
{
int coeff;
int pow;
struct link *next;
};
struct link *poly1=NULL,*poly2=NULL,*poly=NULL;
void create(struct link *node)
{
char ch;
do
{
printf("\nEnter coefficient: ");
scanf("%d", &node->coeff);
printf("Enter exponent: ");
scanf("%d", &node->pow);
node->next = (struct link*)malloc(sizeof(struct link));
node = node->next;
node->next = NULL;
printf("\n continue(y/n): ");
fflush(stdin);
ch=getch();
} while(ch=='y' || ch=='Y');
}
void show(struct link *node)
{
while(node->next!=NULL)
```

```
{
printf("%dx^%d", node->coeff, node->pow);
node=node->next;
if(node->next!=NULL)
printf(" + ");
}
}
void polysub(struct link *poly1, struct link *poly2, struct link *poly)
{
while(poly1->next && poly2->next)
{
if(poly1->pow > poly2->pow)
{
poly->pow = poly1->pow;
poly->coeff = poly1->coeff;
poly1 = poly1->next;
}
else if(poly1->pow < poly2->pow)
{
poly->pow = poly2->pow;
poly->coeff = poly2->coeff;
poly2 = poly2->next;
}
else
{
poly->pow = poly1->pow;
poly->coeff = poly1->coeff - poly2->coeff;
```

```
poly1 = poly1->next;
poly2 = poly2->next;
}
poly->next=(struct link *)malloc(sizeof(struct link));
poly=poly->next;
poly->next=NULL;
}
while(poly1->next || poly2->next)
{
if(poly1->next)
{
poly->pow = poly1->pow;
poly->coeff = poly1->coeff;
poly1 = poly1->next;
}
if(poly2->next)
{
poly->pow = poly2->pow;
poly->coeff = poly2->coeff;
poly2 = poly2->next;
}
poly->next = (struct link *)malloc(sizeof(struct link));
poly = poly->next;
poly->next = NULL;
}
}
main()
```

```
{
poly1 = (struct link *)malloc(sizeof(struct link));
poly2 = (struct link *)malloc(sizeof(struct link));
poly = (struct link *)malloc(sizeof(struct link));
printf("Enter 1st Polynomial:");
create(poly1);
printf("\nEnter 2nd Polynomial:");
create(poly2);
printf("\nPoly1: ");
show(poly1);
printf("\nPoly2: ");
show(poly2);
polysub(poly1, poly2, poly);
printf("\nSubtracted Polynomial: ");
show(poly);
}
```

OUTPUT

```
Enter 1st Polynomial:
Enter coefficient: 5
Enter exponent: 2
continue(y/n): y
Enter coefficient: 4
Enter exponent: 1
continue(y/n): y
Enter coefficient: 2
Enter exponent: 0
```

continue(y/n): n

Enter 2nd Polynomial:

Enter coefficient: 5

Enter exponent: 1

continue(y/n): y

Enter coefficient: 5

Enter exponent: 0

continue(y/n): n

Poly1: 5x^2 + 4x^1 + 2x^0

Poly2: 5x^1 + 5x^0

Subtracted Polynomial: 5x^2 - 1x^1 - 3x^0

RESULT

Thus the polynomial operations using linked list was executed successfully and the output was verified successfully.

EX.NO:5(a) Infix to Postfix conversion

AIM

To convert infix expression to its postfix form using stack operations.

ALGORITHM

STEP 1: Start the program

STEP 2: Define a array stack of size max = 20

STEP 3: Initialize top = -1

STEP 4: Read the infix expression character-by-character

If character is an operand print it

If character is an operator

Compare the operator's priority with the stack[top] operator.

If the stack [top] has higher/equal priority than the input operator,

Pop it from the stack and print it.

Else

Push the input operator onto the stack

If character is a left parenthesis, then push it onto the stack.

If character is a right parenthesis, pop all operators from stack and print it until a left parenthesis is encountered. Do not print the parenthesis.

If character = $ then Pop out all operators, Print them and Stop

STEP 5: Stop the program.

PROGRAM

```
#include  <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 20
```

```
int top = -1;
char stack[MAX];
char pop();
void push(char item);
int prcd(char symbol)
{
switch(symbol)
{
case '+':
case '-':
return 2;
break;
case '*':
case '/':
return 4;
break;
case '^':
case '$':
return 6;
break;
case '(':
case ')':
case '#':
return 1;
break;
}
}
```

```
int isoperator(char symbol)
{
switch(symbol)
{
case '+':
case '-':
case '*':
case '/':
case '^':
case '$':
case '(':
case ')':
return 1;
break;
default:
return 0;
}
}
void convertip(char infix[],char postfix[])
{
int i,symbol,j = 0;
stack[++top] = '#';
for(i=0;i<strlen(infix);i++)
{
symbol = infix[i];
if(isoperator(symbol) == 0)
{
```

```
postfix[j] = symbol;
j++;
}
else
{
if(symbol == '('
push(symbol);
else if(symbol == ')')
{
while(stack[top] != '('
{
postfix[j] = pop();
j++;
}
pop(); //pop out (.
}
else
{
if(prcd(symbol) > prcd(stack[top]))
push(symbol);
else
{
while(prcd(symbol) <= prcd(stack[top]))
{
postfix[j] = pop();
j++;
}
```

```
push(symbol);
}
}
}
}
while(stack[top] != '#')
{
postfix[j] = pop();
j++;
}
postfix[j] = '\0';
}
main()
{
char infix[20],postfix[20];
clrscr();
printf("Enter the valid infix string: ");
gets(infix);
convertip(infix, postfix);
printf("The corresponding postfix string is: ");
puts(postfix);
getch();
}
void push(char item)
{
top++;
stack[top] = item;
```

```
}
char pop()
{
char a;
a = stack[top];
top--;
return a;
}
```

OUTPUT

```
Enter the valid infix string: (a+b*c)/(d$e)
The corresponding postfix string is: abc*+de$/
Enter the valid infix string: a*b+c*d/e
The corresponding postfix string is: ab*cd*e/+
Enter the valid infix string: a+b*c+(d*e+f)*g
The corresponding postfix string is: abc*+de*f+g*+
```

RESULT

Thus the given infix expression was converted into postfix form using stack.

EX.NO:5(B) EVALUATING POSTFIX EXPRESSIONS

AIM

To evaluate the given postfix expression using stack operations.

ALGORITHM

STEP 1: Start the program

STEP 2: Define a array stack of size max = 20

STEP 3: Initialize top = -1

STEP 4: Read the postfix expression character-by-character

If character is an operand push it onto the stack

If character is an operator

Pop topmost two elements from stack.

Apply operator on the elements and push the result onto the stack,

STEP 5: Eventually only result will be in the stack at end of the expression.

STEP 6: Pop the result and print it.

STEP 7: Stop the program

PROGRAM

```
#include  <stdio.h>
#include  <conio.h>
struct stack
{
int top;
float a[50];
}s;
main()
```

```
{
char pf[50];
float d1,d2,d3;
int i;
clrscr();
s.top = -1;
printf("\n\n Enter the postfix expression: ");
gets(pf);
for(i=0; pf[i]!='\0'; i++)
{
switch(pf[i])
{
case '0':
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
s.a[++s.top] = pf[i]-'0';
break;
case '+':
d1 = s.a[s.top--];
d2 = s.a[s.top--];
```

```
s.a[++s.top] = d1 + d2;
break;
case '-':
d2 = s.a[s.top--];
d1 = s.a[s.top--];
s.a[++s.top] = d1 - d2;
break;
case '*':
d2 = s.a[s.top--];
d1 = s.a[s.top--];
s.a[++s.top] = d1*d2;
break;
case '/':
d2 = s.a[s.top--];
d1 = s.a[s.top--];
s.a[++s.top] = d1 / d2;
break;
}
}
printf("\n Expression value is %5.2f", s.a[s.top]);
getch();
}
```

OUTPUT

```
Enter the postfix expression: 6523+8*+3+*
Expression value is 288.00
```

RESULT

Thus the given postfix expression was evaluated using stack.

EX.NO:6 IMPLEMENTATION OF BINARY SEARCH TREES

AIM

To insert and delete nodes in a binary search tree.

ALGORITHM

STEP1: Start the program

STEP2: Create a structure with key and 2 pointer variable left and right.

STEP3: Read the node to be inserted.

If (root==NULL)

root=node

else if (root->key<node->key)

root->right=NULL

else

Root->left=node

STEP4: For Deletion

if it is a leaf node

Remove immediately

Remove pointer between del node & child

if it is having one child

Remove link between del node&child

Link delnode is child with delnodes parent

If it is a node with two children

Find min value in right subtree

Copy min value to delnode place

Delete the duplicate

STEP5: Stop the program

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
// structure of a node
struct node
{
   int data;
   struct node *left;
   struct node *right;
};
struct node *root = NULL;
struct node *create_node(int);
void insert(int);
struct node *delete (struct node *, int);
int search(int);
void inorder(struct node *);
void postorder();
void preorder();
struct node *smallest_node(struct node *);
struct node *largest_node(struct node *);
int get_data();
int main()
{
int userChoice;
int userActive = 'Y';
int data;
struct node* result = NULL;
```

```
while (userActive == 'Y' || userActive == 'y')
{
printf("\n\n------- Binary Search Tree------ \n");
printf("\n1. Insert");
printf("\n2. Delete");
printf("\n3. Search");
printf("\n4. Get Larger Node Data");
printf("\n5. Get smaller Node data");
printf("\n\n-- Traversals --");
printf("\n\n6. Inorder ");
printf("\n7. Post Order ");
printf("\n8. Pre Oder ");
printf("\n9. Exit");
printf("\n\nEnter Your Choice: ");
scanf("%d", &userChoice);
printf("\n");
switch(userChoice)
{
case 1:
data = get_data();
insert(data);
break;
case 2:
data = get_data();
root = delete(root, data);
break;
case 3:
```

```
data = get_data();
if (search(data) == 1)
{
printf("\nData was found!\n");
}
else
{
printf("\nData does not found!\n");
}
break;
case 4:
result = largest_node(root);
if (result != NULL)
{
printf("\nLargest Data: %d\n", result->data);
}
break;
case 5:
result = smallest_node(root);
if (result != NULL)
{
printf("\nSmallest Data: %d\n", result->data);
}
break;
case 6:
inorder(root);
break;
```

```
case 7:
postorder(root);
break;
case 8:
preorder(root);
break;
case 9:
printf("\n\nProgram was terminated\n");
break;
default:
printf("\n\tInvalid Choice\n");
break;
}
printf("\n_____\nDo you want to continue? ");
fflush(stdin);
scanf(" %c", &userActive);
}
return 0;
}
// creates a new node
struct node *create_node(int data)
{
struct node *new_node = (struct node *)malloc(sizeof(struct node));
if (new_node == NULL)
{
printf("\nMemory for new node can't be allocated");
return NULL;
```

```
}
new_node->data = data;
new_node->left = NULL;
new_node->right = NULL;
return new_node;
}
// inserts the data in the BST
void insert(int data)
{
struct node *new_node = create_node(data);
if (new_node != NULL)
{
 // if the root is empty then make a new node as the root node
if (root == NULL)
{
root = new_node;
printf("\n* node having data %d was inserted\n", data);
return;
}
struct node *temp = root;
struct node *prev = NULL;
 // traverse through the BST to get the correct position for insertion
while (temp != NULL)
{
prev = temp;
if (data > temp->data)
{
```

```
temp = temp->right;
}
else
{
temp = temp->left;
}
}
 // found the last node where the new node should insert
if (data > prev->data)
{
prev->right = new_node;
}
else
{
prev->left = new_node;
}
printf("\n* node having data %d was inserted\n", data);
}
}
// deletes the given key node from the BST
struct node *delete (struct node *root, int key)
{
if (root == NULL)
{
return root;
}
if (key < root->data)
```

```
{
root->left = delete (root->left, key);
}
else if (key > root->data)
{
root->right = delete (root->right, key);
}
else
{
if (root->left == NULL)
{
struct node *temp = root->right;
free(root);
return temp;
}
else if (root->right == NULL)
{
struct node *temp = root->left;
free(root);
return temp;
}
struct node *temp = smallest_node(root->right);
root->data = temp->data;
root->right = delete (root->right, temp->data);
}
return root;
}
```

```
// search the given key node in BST
int search(int key)
{
struct node *temp = root;
while (temp != NULL)
{
 if (key == temp->data)
{
return 1;
}
else if (key > temp->data)
{
temp = temp->right;
}
 else
{
temp = temp->left;
}
}
return 0;
}
// finds the node with the smallest value in BST
struct node *smallest_node(struct node *root)
{
struct node *curr = root;
while (curr != NULL && curr->left != NULL)
{
```

```
curr = curr->left;
}
return curr;
}
// finds the node with the largest value in BST
struct node *largest_node(struct node *root)
{
struct node *curr = root;
while (curr != NULL && curr->right != NULL)
{
curr = curr->right;
}
return curr;
}
// inorder traversal of the BST
void inorder(struct node *root)
{
if (root == NULL)
{
 return;
}
inorder(root->left);
printf("%d ", root->data);
inorder(root->right);
}
// preorder traversal of the BST
void preorder(struct node *root)
```

```
{
if (root == NULL)
{
return;
}
printf("%d ", root->data);
preorder(root->left);
preorder(root->right);
}
// postorder travsersal of the BST
void postorder(struct node *root)
{
if (root == NULL)
{
return;
 }
postorder(root->left);
postorder(root->right);
printf("%d ", root->data);
}
// getting data from the user
int get_data()
{
int data;
printf("\nEnter Data: ");
scanf("%d", &data);
return data;
```

}

OUTPUT

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. In Order

7. Post Order

8. Pre Order

9. Exit

Enter Your Choice: 1

Enter Data: 20

* node having data 20 was inserted

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 1

Enter Data: 15

* node having data 15 was inserted

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 1

Enter Data: 25

* node having data 25 was inserted

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 1

Enter Data: 12

* node having data 12 was inserted

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 1

Enter Data: 18

* node having data 18 was inserted

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 1

Enter Data: 65

* node having data 65 was inserted

_____

Do you want to continue? n

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 6

12 15 18 20 25 65

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 2

Enter Data: 25

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 6

12 15 18 20 65

_____

Do you want to continue? n

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 3

Enter Data: 65

Data was found!

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 3

Enter Data: 64

Data does not found!

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. In Order

7. Post Order

8. Pre Order

9. Exit

Enter Your Choice: 4

Largest Data: 65

_____

Do you want to continue? y

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. In order

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 5

Smallest Data: 12

_____

Do you want to continue?

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 6

12 15 18 20 25 65

_____

Do you want to continue? n

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 8

20 15 12 18 25 65

_____

Do you want to continue? n

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 7

12 18 15 65 25 20

_____

Do you want to continue? n

RESULT

Thus the implementation of binary search tree was executed and the output was verified successfully.

EX.NO:7 IMPLEMENTATION OF AVL TREES

AIM

To perform insertion operation on an AVL tree and to maintain balance factor.

ALGORITHM

STEP1: Start the program

STEP2: Perform standard BST insert for w

STEP3: 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.

STEP4: 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.

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)

STEP5: Stop the program

PROGRAM

```
#include<stdio.h>
#include<stdlib.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 the implementation of AVL tree was executed and the output was verified successfully.

Ex.No:8 IMPLEMENTATION OF HEAPS USING PRIORITY QUEUE

AIM

To build a heaps using priority queue from an array of input elements.

ALGORITHM

STEP1: Start the program

STEP2: In a heap, for every node x with parent p, the key in p is smaller than or equal to the key in x.

STEP3: For insertion operation

a. Add the element to the bottom level of the heap.

b. Compare the added element with its parent; if they are in the correct order, stop.

c. If not, swap the element with its parent and return to the previous step.

STEP4: For deleteMin operation

a. Replace the root of the heap with the last element on the last level.

b. Compare the new root with its children; if they are in the correct order, stop.

c. If not, Swap with its smaller child in a min-heap

STEP5: Stop the program

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 20
void main()
```

```
{
        int choice,top=1,a[20],item,i,min;
        char str='c';
        void insert(int[],int,int);
        void DeleteMin(int[],int);
        int FindMin(int [],int);
        void display(int [],int);
        printf("Priority Queue Implementation(Binary Heaps)");
        printf("\n_____\n");
        while(str=='c')
        {
                printf("\n 1. Insert\n");
                printf(" 2. DeleteMin\n");
                printf(" 3. FindMin\n");
                printf(" 4. Display\n");
                printf(" 5. Exit\n");
                printf(" Enter your Choice :");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:
                                if(top<=MAX)
                                {
                                        printf("\n Enter the element :");
                                        scanf("%d",&item);
                                        insert(a,top,item);
                                        top++;
```

```
                display(a,top);
        }
        else
                printf("Priority Queue is Full");
        break;
case 2:
        if(top>0)
        {
                DeleteMin(a,top);
                top--;
                display(a,top);
        }
        else
                printf(" Priority Queue is Empty");
        break;
case 3:
        if(top>0)
        {
                min=FindMin(a,top);
                printf(" Minimum Element is:%d",min);
        }
        break;
case 4:
        printf(" Elements in the priority queue are:");
        display(a,top);
        break;
case 5:
```

```
                    exit(0);
                }
                printf(" \n Press c to continue:\n");
                str=getch();
        }
}
void insert(int a[20],int top,int item)
{
        int i,temp;
        a[top]=item;
        i=top;
        while((i/2)>0)
        {
                if(a[i]<a[i/2])
                {
                        temp=a[i];
                        a[i]=a[i/2];
                        a[i/2]=temp;
                }
                i=i/2;
        }
}
void DeleteMin(int a[20],int top)
{
        int i,j,temp;
        a[1]=a[top-1];
        top--;
```

```
        i=1;
        while((i*2)+1<top)
        {
                if(a[2*i]>a[(2*i)+1])
                        j=(2*i)+1;
                else
                        j=2*i;
                if(a[i]>a[j])
                {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                        i=j;
                }
        }
        if(2*i<top)
                if(a[i]>a[2*i])
                {
                        temp=a[i];
                        a[i]=a[2*i];
                        a[2*i]=temp;
                }
}
int FindMin(int a[20],int top)
{
        if(top<1)
        {
```

```
                printf( "Priority Queue is Empty" );
                return 0;
        }
        else
                return a[1];
}
void display(int a[20],int top)
{
        int i;
        for(i=1;i<top;i++)
                printf(" %d   ",a[i]);
}
```

OUTPUT

```
Priority Queue Implementation (Binary Heaps)
-------------------------------------------------------------
1. Insert
2. Deletemin
3. Findmin
4. Display
5. Exit
Enter Your Choice: 1
Enter the Elements: 10
10
Press c to continue:
```

1. Insert

2. Deletemin

3. Findmin

4. Display

5. Exit

Enter Your Choice: 1

Enter the Elements: 15

10 15

Press c to continue:

1. Insert

2. Deletemin

3. Findmin

4. Display

5. Exit

Enter Your Choice: 1

Enter the Elements: 5

5 15 10

Press c to continue:

1. Insert

2. Deletemin

3. Findmin

4. Display

5. Exit

Enter Your Choice: 3

Minimum Element is: 5

Press c to continue:

1. Insert

2. Deletemin

3. Findmin

4. Display

5. Exit

Enter Your Choice: 2

Enter the Elements:

10 15

Press c to continue:

1. Insert

2. Deletemin

3. Findmin

4. Display

5. Exit

Enter Your Choice: 4

Elements in the priority queue are: 10 15

Press c to continue:

1. Insert

2. Deletemin

3. Findmin

4. Display

5. Exit

Enter Your Choice: 5

RESULT

Thus the implementation of heaps using priority queue was executed and the output was verified successfully.

Ex.No:9 IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

AIM

To find the shortest path for the given graph from a specified source to all other vertices using Dijkstra's algorithm.

ALGORITHM

STEP1: Start the program

STEP2: Obtain no. of vertices and adjacency matrix for the given graph

STEP3: Create cost matrix from adjacency matrix. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity

STEP4: Initialize visited[] to zero

STEP5: Read source vertex and mark it as visited

STEP6: Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex
distance[i]=cost[0][i];

STEP7: Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.

STEP8: Recalculate the shortest distance of remaining vertices from the source.

STEP9: Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v if(visited[v]==0) distance[v]=min(distance[v] distance[w]+cost[w][v])

STEP10: Stop the program

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijikstra(int G[MAX][MAX], int n, int startnode);
void main()
{
        int G[MAX][MAX], i, j, n, u;
        clrscr();
        printf("\nEnter the 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);
        dijikstra(G,n,u);
        getch();
}
void dijikstra(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 %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 0 60 0

Enter the starting node: 0

Distance to node1 = 10

Path = 1<-0

Distance to node2 = 50

Path = 2<-3<-0

Distance to node3 = 30

Path = 3<-0

Distance to node4 = 60

Path = 4<-2<-3<-0

RESULT

Thus the implementation of Dijkstra's Algorithm was executed and the output was verified successfully.

Ex.No:10 IMPLEMENTATION OF PRIM'S ALGORITHM

AIM

To find the minimum spanning tree for the given graph from a specified source to all other vertices using prim's algorithm.

ALGORITHM

STEP 1: Start the program

STEP 2: Read a weighted Graph G(V, E)

STEP 3: Let n be the number of vertices in the graph G

STEP 4: Choose the starting vertex say V1 and add it to visited array

STEP 5: Find the lowest cost edge coming out of Vertices in visited array

STEP 6: if a cycle is formed if this lowest cost edge is drawn then

Take the next lowest cost edge

STEP 7: else

Draw the edge

Add the vertex to visited array

STEP 8: Go to step 3 until n-1 edges are drawn

STEP 9: STOP the program

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {0},min,mincost=0,cost[10][10];
void main()
{
printf("\n Enter the number of nodes:");
```

```
scanf("%d",&n);
printf("\n Enter the adjacency matrix:\n");
for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
        {
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                cost[i][j]=999;
        }
 visited[1]=1;
 printf("\n");
 while(ne<n)
{
        for(i=1,min=999;i<=n;i++)
                for (j=1;j<=n;j++)
                        if(cost[i][j]<min)
                                if(visited[i]!=0)
                                {
                                        min=cost[i][j];
                                        a=u=i;
                                        b=v=j;
                                }
if(visited[u]==0 || visited[v]==0)
                {
                        printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                        mincost+=min;
                        visited[b]=1;
```

```
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
        getch();
}
```

OUTPUT

Enter the number of nodes: 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

Edge 1:(1 3) Cost : 1

Edge 1:(1 2) Cost : 3

Edge 1:(2 5) Cost : 3

Edge 1:(1 6) Cost : 4

Edge 1:(6 4) Cost : 2

Minimum Cost=1

RESULT

Thus the implementation of prim's algorithm was executed and the output was verified successfully.

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

AIM

To perform linear search of an element on the given array.

ALGORITHM

STEP1: Start the program

STEP2: Read number of array elements n

STEP3: Read array elements Ai, i = 0,1,2,…n–1

STEP4: Read search value

STEP5: Assign 0 to found

STEP6: Check each array element against search

If Ai = search then

found = 1

Print "Element found"

Print position i

Stop

STEP7: If found = 0 then

print "Element not found"

STEP8: Stop the program

PROGRAM

```
#include  <stdio.h>
#include <conio.h>
main()
{
int a[50],i, n, val, found;
```

```
clrscr();
printf("Enter number of elements : ");
scanf("%d", &n);
printf("Enter Array Elements : \n");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("Enter element to locate : ");
scanf("%d", &val);
found = 0;
for(i=0; i<n; i++)
{
if (a[i] == val)
{
printf("Element found at position %d", i);
found = 1;
break;
}
}
if (found == 0)
printf("\n Element not found");
getch();
}
```

OUTPUT

Enter number of elements : 7

Enter Array Elements :

23 6 12 5 0 32 10

Enter element to locate : 5

Element found at position 3

RESULT

Thus an array was linearly searched for an element's existence.

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

AIM

To locate an element in a sorted array using Binary search method

ALGORITHM

STEP1: Start the program

STEP2: Read number of array elements, say n

STEP3: Create an array arr consisting n sorted elements

STEP4: Get element, say key to be located

STEP5: Assign 0 to lower and n to upper

STEP6: While (lower < upper)

Determine middle element mid = (upper+lower)/2

If key = arr[mid] then

Print mid

Stop

Else if key > arr[mid] then

lower = mid + 1

else

upper = mid – 1

STEP7: Print "Element not found"

STEP8: Stop the program

PROGRAM

```
#include  <stdio.h>
#include <conio.h>
main()
```

```
{
int a[50],i, n, upper, lower, mid, val, found;
clrscr();
printf("Enter array size : ");
scanf("%d", &n);
for(i=0; i<n; i++)
a[i] = 2 * i;
printf("\n Elements in Sorted Order \n");
for(i=0; i<n; i++)
printf("%4d", a[i]);
printf("\n Enter element to locate : ");
scanf("%d", &val);
upper = n;
lower = 0;
found = -1;
while (lower <= upper)
{
mid = (upper + lower)/2;
if (a[mid] == val)
{
printf("Located at position %d", mid);
found = 1;
break;
}
else if(a[mid] > val)
upper = mid - 1;
else
```

```
lower = mid + 1;
}
if (found == -1)
printf("Element not found");
getch();
}
```

OUTPUT

```
Enter array size : 9
Elements in Sorted Order
0 2 4 6 8 10 12 14 16
Enter element to locate : 12
Located at position 6
Enter array size : 10
Elements in Sorted Order
0 2 4 6 8 10 12 14 16 18
Enter element to locate : 13
Element not found
```

RESULT

Thus an element is located quickly using binary search method.

Ex.No:12(a) IMPLEMENTATION OF INSERTION SORT

AIM

To sort an array of N numbers using Insertion sort.

ALGORITHM

STEP1: Start the program

STEP2: Read number of array elements n

STEP3: Read array elements Ai

STEP4: Sort the elements using insertion sort

In pass p, move the element in position p left until its correct place is found among the first p + 1 elements.

Element at position p is saved in temp, and all larger elements (prior to position p) are moved one spot to the right. Then temp is placed in the correct spot.

STEP5: Stop the program

PROGRAM

```
main()
{
int i, j, k, n, temp, a[20], p=0;
printf("Enter total elements: ");
scanf("%d",&n);
printf("Enter array elements: ");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
for(i=1; i<n; i++)
{
temp = a[i];
j = i - 1;
```

```
while((temp<a[j]) && (j>=0))
{
a[j+1] = a[j];
j = j - 1;
}
a[j+1] = temp;
p++;
printf("\n After Pass %d: ", p);
for(k=0; k<n; k++)
printf(" %d", a[k]);
}
printf("\n Sorted List : ");
for(i=0; i<n;  i++)
printf(" %d", a[i]);
}
```

OUTPUT

```
Enter total elements: 6
Enter array elements: 34 8 64 51 32 21
After Pass 1: 8 34 64 51 32 21
After Pass 2: 8 34 64 51 32 21
After Pass 3: 8 34 51 64 32 21
After Pass 4: 8 32 34 51 64 21
After Pass 5: 8 21 32 34 51 64
Sorted List: 8 21 32 34 51 64
```

RESULT

Thus array elements was sorted using insertion sort.

Ex.No:12(b) IMPLEMENTATION OF SELECTION SORT

AIM

To sort an array of N numbers using selection sort.

ALGORITHM

STEP1: Start the program

STEP2: Set min to the first location

STEP3: Search the minimum element in the array

STEP4: swap the first location with the minimum value in the array

STEP5: assign the second element as min.

STEP6: Repeat the process until we get a sorted array.

STEP7: Stop the program

PROGRAM

```
#include <stdio.h>
void selectionSort(int arr[], int size);
void swap(int *a, int *b);
void selectionSort(int arr[], int size)
{
   int i, j;
   for (i = 0 ; i < size;i++)
   {
      for (j = i ; j < size; j++)
      {
         if (arr[i] > arr[j])
            swap(&arr[i], &arr[j]);
```

```
        }
    }
}
void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
int main()
{
    int array[10], i, size;
    printf("How many numbers you want to sort: ");
    scanf("%d", &size);
    printf("\nEnter %d numbers\t", size);
    printf("\n");
    for (i = 0; i < size; i++)
    scanf("%d", &array[i]);
    selectionSort(array, size);
    printf("\nSorted array is ");
    for (i = 0; i < size;i++)
    printf(" %d ", array[i]);
    return 0;
}
```

OUTPUT

How many numbers you want to sort: 6

Enter 6 numbers

4 6 1 2 5 3

Sorted array is 1 2 3 4 5 6

RESULT

Thus array elements was sorted using selection sort.

Ex.No:13 IMPLEMENTATION OF MERGE SORT

AIM

To sort an array of N numbers using Merge sort (Divide and Conquer method).

ALGORITHM

STEP1: Start the program

STEP2: Read number of array elements n

STEP3: Read array elements Ai

STEP4: Divide the array into sub-arrays with a set of elements

STEP5: Recursively sort the sub-arrays

STEP6: Merge the sorted sub-arrays onto a single sorted array.

STEP7: Stop the program

PROGRAM

```
#include<stdio.h>
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
void main()
{
        int a[30],n,i;
        printf("Enter no of elements:");
        scanf("%d",&n);
        printf("Enter array elements:");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        mergesort(a,0,n-1);
```

```
        printf("\nSorted array is :");
        for(i=0;i<n;i++)
                printf("%d ",a[i]);
}
void mergesort(int a[],int i,int j)
{
        int mid;
        if(i<j)
        {
                mid=(i+j)/2;
                mergesort(a,i,mid);
                mergesort(a,mid+1,j);
                merge(a,i,mid,mid+1,j);
        }
}
void merge(int a[],int i1,int j1,int i2,int j2)
{
        int temp[50];
        int i,j,k;
        i=i1;
        j=i2;
        k=0;
        while(i<=j1 && j<=j2)
        {
                if(a[i]<a[j])
                        temp[k++]=a[i++];
                else
```

```
                temp[k++]=a[j++];
        }
        while(i<=j1)
                temp[k++]=a[i++];
        while(j<=j2)
                temp[k++]=a[j++];
        for(i=i1,j=0;i<=j2;i++,j++)
                a[i]=temp[j];
}
```

OUTPUT

Enter no of elements: 5

Enter array elements: 10 7 3 6 2

Sorted array is: 2 3 6 7 10

RESULT

Thus array elements was sorted using merge sort's divide and conquer method.

Ex.No:14(a) IMPLEMENTATION OF HASHING USING LINEAR PROBING

AIM

To perform the implementation of hashing using linear probing method.

ALGORITHM

Algorithm to insert a value in linear probing

Hash table is an array of size = TABLE_SIZE

Step 1: Read the value to be inserted, key

Step 2: let i = 0

Step 3: hkey = key% TABLE_SIZE

Step 4: compute the index at which the key has to be inserted in hash table

index = (hkey + i) % TABLE_SIZE

Step 5: if there is no element at that index then insert the value at index and STOP

Step 6: If there is already an element at that index i = i+1

step 7: if i < TABLE_SIZE then go to step 4

Algorithm to search a value in linear probing

Hash table is an array of size = TABLE_SIZE

Step 1: Read the value to be searched, key

Step 2: let i = 0

Step 3: h(key) = key % TABLE_SIZE

Step 4: compute the index at which the key can be found

index = (h(key)+ i) % TABLE_SIZE

Step 5: if the element at that index is same as the search value then print

element found and STOP

Step 6: else i = i+1

step 7: if i < TABLE_SIZE then go to step 4

PROGRAM

```
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};
void insert()
{
 int key,index,i,flag=0,hkey;
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
    {
     index=(hkey+i)%TABLE_SIZE;
     if(h[index] == NULL)
     {
        h[index]=key;
         break;
     }
    }
    if(i == TABLE_SIZE)
     printf("\nelement cannot be inserted\n");
}
```

```
void search()
{
 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
    index=(hkey+i)%TABLE_SIZE;
    if(h[index]==key)
    {
      printf("value is found at index %d",index);
      break;
    }
  }
  if(i == TABLE_SIZE)
    printf("\n value is not found\n");
}
void display()
{
  int i;
  printf("\nelements in the hash table are \n");
  for(i=0;i< TABLE_SIZE; i++)
  printf("\nat index %d \t value =  %d",i,h[i]);
}
main()
{
```

```
    int opt,i;
    while(1)
    {
        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                insert();
                break;
            case 2:
                display();
                break;
            case 3:
                search();
                break;
            case 4:exit(0);
        }
    }
}
```

OUTPUT

```
Press 1. Insert     2. Display     3. Search     4.Exit
1
enter a value to insert into hash table      12

Press 1. Insert     2. Display     3. Search     4.Exit
```

```
1
enter a value to insert into hash table   13

Press 1. Insert    2. Display    3. Search    4.Exit
1
enter a value to insert into hash table   22

Press 1. Insert    2. Display    3. Search    4.Exit
2

elements in the hash table are
at index 0      value =  0
at index 1      value =  0
at index 2      value =  12
at index 3      value =  13
at index 4      value =  22
at index 5      value =  0
at index 6      value =  0
at index 7      value =  0
at index 8      value =  0
at index 9      value =  0

Press 1. Insert    2. Display    3. Search    4.Exit
3
enter search element    12
value is found at index 2
```

Press 1. Insert 2. Display 3. Search 4.Exit

3

enter search element 23

value is not found

Press 1. Insert 2. Display 3. Search 4.Exit

4

RESULT

Thus the program for implementation of linear searching was executed and verified successfully.

Ex.No:14(b) IMPLEMENATATION OF HASHING USING QUADRATIC PROBING

AIM

To perform the implementation of hashing using Quadratic probing method.

ALGORITHM

Algorithm to insert a value in quadratic probing

Hash table is an array of size = TABLE_SIZE

Step 1: Read the value to be inserted, key

Step 2: let i = 0

Step 3: h(key)=key % TABLE_SIZE

Step 4: compute the index at which the value has to be inserted in hash table

index = (h(key)+ i * i) % TABLE_SIZE

Step 5: if there is no element at that index then insert the value at index and STOP

Step 6: If there is already an element at that index i = i+1

step7: if i < TABLE_SIZE then go to step 4

Algorithm to search a value in quadratic probing

Hash table is an array of size = TABLE_SIZE

Step 1: Read the value to be searched, key

Step 2: let i = 0

Step 3: h(key)=key % TABLE_SIZE

Step 4: compute the index at which the value can be found

index = (h(key) + i * i) % TABLE_SIZE

Step 5: if the element at that index is same as the search value then print

element found and STOP

Step 6: else i = i+1

step7: if i < TABLE_SIZE then go to step 4

PROGRAM

```
#include <stdio.h>
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};
void insert()
{
 int key,index,i,flag=0,hkey;
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
{
    index=(hkey+i*i)%TABLE_SIZE;
    if(h[index] == NULL)
    {
       h[index]=key;
        break;
    }
}
if(i == TABLE_SIZE)
    printf("\nelement cannot be inserted\n");
}
```

```
void search()
{
 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key % TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
    index=(hkey+i*i)%TABLE_SIZE;
    if(h[index]==key)
    {
      printf("value is found at index %d",index);
      break;
    }
  }
  if(i == TABLE_SIZE)
    printf("\n value is not found\n");
}
void display()
{
  int i;
  printf("\nelements in the hash table are \n");
  for(i=0;i< TABLE_SIZE; i++)
            printf("\nat index %d \t value =  %d",i,h[i]);
}
main()
{
```

```
    int opt,i;
    while(1)
    {
        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                insert();
                break;
            case 2:
                display();
                break;
            case 3:
                search();
                break;
            case 4:exit(0);
        }
    }
}
```

OUTPUT

```
Press 1. Insert     2. Display     3. Search     4.Exit
1
enter a value to insert into hash table     12

Press 1. Insert     2. Display     3. Search     4.Exit
```

1

enter a value to insert into hash table 22

Press 1. Insert 2. Display 3. Search 4.Exit

1

enter a value to insert into hash table 32

Press 1. Insert 2. Display 3. Search 4.Exit

2

elements in the hash table are

at index 0 value = 0

at index 1 value = 0

at index 2 value = 12

at index 3 value = 22

at index 4 value = 0

at index 5 value = 0

at index 6 value = 32

at index 7 value = 0

at index 8 value = 0

at index 9 value = 0

Press 1. Insert 2. Display 3. Search 4.Exit

3

enter search element 22

value is found at index 3

Press 1. Insert 2. Display 3. Search 4.Exit

3

enter search element 123

value is not found

Press 1. Insert 2. Display 3. Search 4.Exit

4

RESULT

Thus the program for implementation of quadratic probing was executed and verified successfully.