CS3361 DATA STRUCTURES LABORATORY

L T P C
0 0 3 1.5

COURSE OBJECTIVES:

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

TOTAL:45 PERIODS

COURSE OUTCOMES:

At the end of this course, the students will be able to:
CO1: Implement Linear data structure algorithms.
CO2: Implement applications using Stacks and Linked lists
CO3: Implement Binary Search tree and AVL tree operations.
CO4: Implement graph algorithms.
CO5: Analyze the various searching and sorting algorithms.

INDEX

S.NO.	NAME OF THE PROGRAM	MARK	SIGN

Ex. No. 1a **Stack Array**
Date:

Aim

To implement stack operations using array.

Algorithm

```
1. Start
2. Define a array stack of size max = 5
3. Initialize top = -1
4. Display a menu listing stack operations
5. Accept choice
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
7. Stop
```

Program

```
/* Stack Operation using Arrays */
#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");
}
}
}
```

Viva Questions
1. List the applications of stacks
2. Define a stack
3. List out the basic operations that can be performed on a stack
4. Mention the advantages of representing stacks using linked lists than arrays
5.Define Data Structures

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

Ex. No. 1b **Queue Array**
Date:

Aim

To implement queue operations using array.

Algorithm

```
1. Start
2. Define a array queue of size max = 5
3. Initialize front = rear = –1
4. Display a menu listing queue operations
5. Accept choice
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.
7. Stop
```

Program

```
/* Queue Operation using Arrays */
#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");
}
}
}
```

viva question
1.List the applications of queues
2.Define a queue
3.Define a priority queue
4.State the difference between queues and linked lists
5.Define a Deque

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

Ex. No. 1c **Circular Queue Array**
Date:

Aim

To implement circular queue operations using array.

Algorithm

```
1. Start
2. Define a array queue of size max = 5
3. Initialize front = rear = –1
4. Display a menu listing queue operations
5. Accept choice
6. If choice = 1 then
        check for overflow
        Increment rear
        Store element at current position of rear
        Else
        Print Queue Full
        Else If choice = 2 then
        check for underlfow
        Print Queue empty
        Else
        Display current front element
        Increment front
        Else If choice = 3 then
        Display queue elements starting from front to rear.
7. Stop
```

Program:

```
#include<stdio.h>
# define MAX 5
int cqueue_arr[MAX];
int front = -1;
int rear = -1;
void insert(int item)
{
if((front == 0 && rear == MAX-1) || (front == rear+1))
{
printf("Queue Overflow n");
return;
}
if(front == -1)
{
front = 0;
rear = 0;
}
else
{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
cqueue_arr[rear] = item ;
}
void deletion()
{
if(front == -1)
{
printf("Queue Underflown");
return ;
}
printf("Element deleted from queue is : %dn",cqueue_arr[front]);
if(front == rear)
{
front = -1;
rear=-1;
}
else
{
if(front == MAX-1)
front = 0;
else
front = front+1;
}
```

```
}
void display()
{
int front_pos = front,rear_pos = rear;
if(front == -1)
{
printf("Queue is emptyn");
return;
}
printf("Queue elements :n");
if( front_pos <= rear_pos )
while(front_pos <= rear_pos)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
else
{
while(front_pos <= MAX-1)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
front_pos = 0;
while(front_pos <= rear_pos)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
}
printf("n");
}
int main()
{
int choice,item;
do
{
printf("1.Insert");
printf("2.Delete");
printf("3.Display");
printf("4.Quit");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1 :
printf("Input the element for insertion in queue : ");
scanf("%d", &item);
insert(item);
break;
case 2 :
deletion();
break;
case 3:
display();
break;
case 4:
break;
default:
printf("Wrong choice");
}
}while(choice!=4);
return 0;
}
```

viva question
1.What is the need for Priority queue?
2.What is a Circular Queue?
3.Mention the advantages of representing stacks using linked lists than arrays
4,State the difference between queues and linked lists
5.What are the types of queues?

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

Ex. No. 2 **Singly Linked List**
Date:

Aim

To define a singly linked list node and perform operations such as insertions and deletions dynamically.

Algorithm

```
1. Start
2. Define single linked list node as self referential structure
3. Create Head node with label = -1 and next = NULL using
4. Display menu on list operation
5. Accept user choice
6. 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
7. Stop
```

Program
```
/* Single Linked List */
#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, fou=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;
fou = 0;
if (h->label == k)
fou = 1;
while(h->next != NULL)
{
if (h->label == k)
{
fou=1;
break;
}
h = h->next;
}
if (h->label == k)
fou = 1;
if (fou != 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);
fou = 0;
h = h1 = head;
while (h->next != NULL)
{
h = h->next;
if (h->label == k)
{
fou = 1;
break;
}
}
if (fou == 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);
}
}
}
```

viva question
1. Define Data Structures
2. Define Linked Lists
3. State the different types of linked lists
4. List the basic operations carried out in a linked list
5. List out the advantages of using a linked list

Result
Thus operation on single linked list is performed.

Ex. No. 3a. **Stack Using Linked List**
Date:

Aim

To implement stack operations using linked list.

Algorithm

```
1. Start
2. Define a singly linked list node for stack
3. Create Head node
4. Display a menu listing stack operations
5. Accept choice
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
7. Stop
```

Program

```
/* Stack using Single Linked List */
#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);
}
}
}
```

viva question
1.Define Linked Lists
2.List out the basic operations that can be performed on a stack
3.Mention the advantages of representing stacks using linked lists than arrays
4.Mention the advantages of representing stacks using linked lists than arrays
5.State the difference between queues and linked lists

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

Ex. No.3.b **Queue Using Linked List**
Date:

Aim

To implement queue operations using linked list.

Algorithm

```
1. Start
2. Define a singly linked list node for stack
3. Create Head node
4. Display a menu listing stack operations
5. Accept choice
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
7. Stop
```

Program

```
/* Queue using Single Linked List */
#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 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:
/* Create a new node */
temp=(struct node *)(malloc(sizeof(struct node)));
printf("Enter label for new node : ");
scanf("%d", &temp->label);
/* Reorganize the links */
h = head;
while (h->next != NULL)
h = h->next;
h->next = temp;
temp->next = NULL;
break;
case 2:
/* Delink the first node */
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);
}
}
}
```

viva

1.Define a queue
2.Define a priority queue
3.Define a Deque
4.What is the need for Priority queue?
5.List the applications of queues

Result

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

Ex. No. 4 **Polynomial manipulation using linked list**
Date:

Aim

To implement Polynomial manipulation using linked list.

.

Algorithm

1. start the program.
2. Get the polynomial 1 and 2 from the user.
3. If addition, add the coefficient of the corresponding powers.
4. Print the resultant polynomial.
5. Stop

Program

```
#include<math.h>
#include<stdio.h>
#include<conio.h>
#define MAX 17
typedef struct node
  {
int coeff;
struct node *next;
  }node;
node *  init();
void read(node *h1);
void print(node *h1);
node * add(node *h1,node *h2);
node * multiply(node *h1,node *h2);
/*Polynomial is stored in a linked list, ith node  gives coefficient of x^i .
  a polynomial 3x^2 + 12x^4 will be represented as (0,0,3,0,12,0,0,….)
*/
void main()
{
node *h1=NULL,*h2=NULL,*h3=NULL;
int option;
do
{
printf("nn1 : create 1'st polynomial");
printf("n2 : create 2'nd polynomial");
printf("n3 : Add polynomials");
printf("n4 : Multiply polynomials");
printf("n5 : Quit");
printf("nEnter your choice :");
scanf("%d",&option);
switch(option)
{
case 1:h1=init();read(h1);break;
case 2:h2=init();read(h2);break;
case 3:h3=add(h1,h2);
      printf("n1'st polynomial -> ");
      print(h1);
      printf("n2'nd polynomial -> ");
      print(h2);
      printf("n Sum = ");
      print(h3);
      break;
case 4:h3=multiply(h1,h2);
      printf("n1'st polynomial -> ");
      print(h1);
      printf("n2'nd polynomial -> ");
      print(h2);
      printf("n Product = ");
      print(h3);
      break;
}
}while(option!=5);
}
void  read(node *h)
{
int n,i,j,power,coeff;
node *p;
p=init();
printf("n Enter number of terms :");
scanf("%d",&n);
/* read n terms */
for (i=0;i<n;i++)
{       printf("nenter a term(power  coeff.)");
scanf("%d%d",&power,&coeff);
for(p=h,j=0;j<power;j++)
   p=p->next;
p->coeff=coeff;
}
}
void print(node *p)
{
   int i;
for(i=0;p!=NULL;i++,p=p->next)
if(p->coeff!=0)
printf("%dX^%d   ",p->coeff,i);
}
node * add(node *h1, node *h2)
{
    node *h3,*p;
    h3=init();
    p=h3;
    while(h1!=NULL)
      {
h3->coeff=h1->coeff+h2->coeff;
h1=h1->next;
h2=h2->next;
h3=h3->next;
      }
 return(p);
}
node * multiply(node *h1, node *h2)
{
node *h3,*p,*q,*r;
```

```
int i,j,k,coeff,power;
h3=init();
for(p=h1,i=0;p!=NULL;p=p->next,i++)
for(q=h2,j=0;q!=NULL;q=q->next,j++)
   {
coeff=p->coeff * q->coeff;
power=i+j;
for(r=h3,k=0;k<power;k++)
r=r->next;
r->coeff=r->coeff+coeff;
    }
  return(h3);
}
node * init()
{
    int i;
    node *h=NULL,*p;
    for(i=0;i<MAX;i++)
{
p=(node*)malloc(sizeof(node));
p->next=h;
p->coeff=0;
h=p;
}
  return(h);
}
```

viva
1.Define Linked Lists
2.State the different types of linked lists
3.List the basic operations carried out in a linked list
4.List out the advantages of using a linked list
5.List out the disadvantages of using a linked list

Result

Thus the C program to implement Polynomial manipulation using linked list was executed successfully.

Ex. No. 5.a **Postfix Expression Evaluation**
Date:

Aim

To evaluate the given postfix expression using stack operations.

Algorithm

```
1. Start
2. Define a array stack of size max = 20
3. Initialize top = -1
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,
5. Eventually only result will be in the stack at end of the expression.
6. Pop the result and print it.
7. Stop
```

Program

```
/* Evaluation of Postfix expression using stack */
#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();
}
```

viva

1.List the applications of stacks
2.State the different ways of representing expressions
3.State the rules to be followed during infix to postfix conversions
4.Mention the advantages of representing stacks using linked lists than arrays
5.Mention the advantages of representing stacks using linked lists than arrays

Result
Thus the given postfix expression was evaluated using stack.

Ex. No. 5.b. Infix To Postfix Conversion

Date:

Aim

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

Algorithm

```
1. Start
2. Define a array stack of size max = 20
3. Initialize top = -1
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
```

.

Program

```
/* Conversion of infix to postfix expression */
#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;
}
```

viva

1.List the applications of stacks
2.State the different ways of representing expressions
3.State the rules to be followed during infix to postfix conversions
4.Mention the advantages of representing stacks using linked lists than arrays
5.Mention the advantages of representing stacks using linked lists than arrays

Result

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

Ex. No. 6 **Binary Search Tree**

Date:

Aim

To insert and delete nodes in a binary search tree.

Algorithm

```
1. Create a structure with key and 2 pointer variable left and right.
2. Read the node to be inserted.
    If (root==NULL)
    root=node
    else if (root->key<node->key)
    root->right=NULL
    else
    Root->left=node
3. 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 a children
    Find min value in right subtree
    Copy min value to delnode place
    Delete the duplicate
4. Stop
```

Program

```
/* Binary Search Tree */
#include <stdio.h>
#include <stdlib.h>
struct node
{
int key;
struct node *left;
struct node *right;
};
struct node *newNode(int item)
{
struct node *temp = (struct node *)malloc(sizeof(struct
node));
temp->key = item;
temp->left = temp->right = NULL;
return temp;
}
void inorder(struct node *root)
{
if (root != NULL)
{
inorder(root->left);
printf("%d ", root->key);
inorder(root->right);
}
}
struct node* insert(struct node* node, int key)
{
if (node == NULL)
return newNode(key);
if (key < node->key)
node->left = insert(node->left, key);
else
node->right = insert(node->right, key);
return node;
}
struct node * minValueNode(struct node* node)
{
struct node* current = node;
while (current->left != NULL)
current = current->left;
return current;
}
struct node* deleteNode(struct node* root, int key)
{
```

```
struct node *temp;
if (root == NULL)
return root;
if (key < root->key)
root->left = deleteNode(root->left, key);
else if (key > root->key)
root->right = deleteNode(root->right, key);
else
{
if (root->left == NULL)
{
temp = root->right;
free(root);
return temp;
}
else if (root->right == NULL)
{
temp = root->left;
free(root);
return temp;
}
temp = minValueNode(root->right);
root->key = temp->key;
root->right = deleteNode(root->right, temp->key);
}
return root;
}
main()
{
struct node *root = NULL;
root = insert(root, 50);
root = insert(root, 30);
root = insert(root, 20);
root = insert(root, 40);
root = insert(root, 70);
root = insert(root, 60);
root = insert(root, 80);
printf("Inorder traversal of the given tree \n");
inorder(root);
printf("\nDelete 20\n");
root = deleteNode(root, 20);
printf("Inorder traversal of the modified tree \n");
inorder(root);
printf("\nDelete 30\n");
root = deleteNode(root, 30);
printf("Inorder traversal of the modified tree \n");
inorder(root);
printf("\nDelete 50\n");
root = deleteNode(root, 50);
```

```
printf("Inorder traversal of the modified tree \n");
inorder(root);
}
```

viva

1.Define a tree
2.Define root
3.Define degree of the node
4.Define leaves
5.Define internal nodes

Result
Thus nodes were inserted and deleted from a binary search tree.

Ex. No. 7 AVL Tree
Date:

Aim

To implement operations on AVL tree using a C program.

Algorithm

```
1. Create a structure with key and 2 pointer variable left and right.
2. Read the node to be inserted.
    If (root==NULL)
    root=node
    else if (root->key<node->key)
    root->right=NULL
    else
    Root→left=node
    if balancing factor is 1 or -1 or 0
    perform rotation
3. 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 a children
    Find min value in right subtree
    Copy min value to delnode place
    Delete the duplicate
    if balancing factor is 1 or -1 or 0
    perform rotation
4. Stop
```

Program

```
// AVL tree implementation in C

#include <stdio.h>
#include <stdlib.h>
// Create Node
struct Node {
  int key;
  struct Node *left;
  struct Node *right;
  int height;
};
int max(int a, int b);
// Calculate height
int height(struct Node *N) {
  if (N == NULL)
    return 0;
  return N->height;
}
int max(int a, int b) {
  return (a > b) ? a : b;
}
// Create a node
struct Node *newNode(int key) {
  struct Node *node = (struct Node *)
    malloc(sizeof(struct Node));
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}
// Right rotate
struct Node *rightRotate(struct Node *y) {
  struct Node *x = y->left;
  struct Node *T2 = x->right;
  x->right = y;
  y->left = T2;
  y->height = max(height(y->left), height(y->right)) + 1;
  x->height = max(height(x->left), height(x->right)) + 1;
  return x;
}
// Left rotate
struct Node *leftRotate(struct Node *x) {
  struct Node *y = x->right;
  struct Node *T2 = y->left;
  y->left = x;
  x->right = T2;
  x->height = max(height(x->left), height(x->right)) + 1;
  y->height = max(height(y->left), height(y->right)) + 1;
  return y;
}
// Get the balance factor
int getBalance(struct Node *N) {
  if (N == NULL)
    return 0;
  return height(N->left) - height(N->right);
}
// Insert node
struct Node *insertNode(struct Node *node, int key) {
  // Find the correct position to insertNode the node and insertNode it
  if (node == NULL)
    return (newNode(key));
  if (key < node->key)
    node->left = insertNode(node->left, key);
  else if (key > node->key)
    node->right = insertNode(node->right, key);
  else
    return node;
  // Update the balance factor of each node and
  // Balance the tree
  node->height = 1 + max(height(node->left),
             height(node->right));
  int balance = getBalance(node);
  if (balance > 1 && key < node->left->key)
    return rightRotate(node);
  if (balance < -1 && key > node->right->key)
    return leftRotate(node);
  if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
  }
  if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
  }
  return node;
}
struct Node *minValueNode(struct Node *node) {
  struct Node *current = node;
  while (current->left != NULL)
    current = current->left;
  return current;
```

```
}
// Delete a nodes
struct Node *deleteNode(struct Node *root, int key) {
  // Find the node and delete it
  if (root == NULL)
    return root;
  if (key < root->key)
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);
  else {
    if ((root->left == NULL) || (root->right == NULL)) {
      struct Node *temp = root->left ? root->left : root->right;
      if (temp == NULL) {
        temp = root;
        root = NULL;
      } else
        *root = *temp;
      free(temp);
    } else {
      struct Node *temp = minValueNode(root->right);
      root->key = temp->key;
      root->right = deleteNode(root->right, temp->key);
    }
  }
  if (root == NULL)
    return root;
  // Update the balance factor of each node and
  // balance the tree
  root->height = 1 + max(height(root->left),
              height(root->right));
  int balance = getBalance(root);
  if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);
  if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
  }
  if (balance < -1 && getBalance(root->right) <= 0)
    return leftRotate(root);
  if (balance < -1 && getBalance(root->right) > 0) {
    root->right = rightRotate(root->right);
    return leftRotate(root);
  }
  return root;
}
// Print the tree
void printPreOrder(struct Node *root) {
  if (root != NULL) {
    printf("%d ", root->key);
    printPreOrder(root->left);
    printPreOrder(root->right);
  }
}
int main() {
  struct Node *root = NULL;
  root = insertNode(root, 2);
  root = insertNode(root, 1);
  root = insertNode(root, 7);
  root = insertNode(root, 4);
  root = insertNode(root, 5);
  root = insertNode(root, 3);
  root = insertNode(root, 8);
  printPreOrder(root);
  root = deleteNode(root, 3);
  printf("\nAfter deletion: ");
  printPreOrder(root);
  return 0;
}
```

viva

1. Define parent node
2. Define depth and height of a node
3. Define depth and height of a tree
4. What do you mean by level of the tree?
5. Define a binary tree

Result

Thus AVL Tree was implemented successfully.

Ex. No. 8 **Heaps using Priority Queues**
Date:

Aim

To implement Heaps using Priority Queues.

Algorithm

1. Start the program.
2. Get the choice from the user
3. If insert, get the element to be inserted and get the priority of the element.
4. If delete, the element with the highest priority is deleted first
5. Stop

Program

```
#include<stdio.h>

#include<malloc.h>

void insert();

void del();
void display();
struct node
{
int priority;
int info;
struct node *next;
}*start=NULL,*q,*temp,*new;
typedef struct node N;
int main()
{
int ch;
do
{
printf("\n[1] INSERTION\t[2] DELETION\t[3] DISPLAY [4] EXIT\t:");
scanf("%d",&ch);
switch(ch)
{
case 1:insert();
break;
case 2:del();
break;
case 3:display();
break;
case 4:
break;
}
}
while(ch<4);
}
void insert()
{
int item,itprio;
new=(N*)malloc(sizeof(N));
printf("ENTER THE ELT.TO BE INSERTED :\t");
scanf("%d",&item);
printf("ENTER ITS PRIORITY :\t");
scanf("%d",&itprio);
```

```
new->info=item;
new->priority=itprio;
new->next=NULL;
if(start==NULL )
{
//new->next=start;
start=new;
}
else if(start!=NULL&&itprio<=start->priority)
{ new->next=start;
start=new;
}
else
{
q=start;
while(q->next != NULL && q->next->priority<=itprio)
{q=q->next;}
new->next=q->next;
q->next=new;
}
}
void del()
{
if(start==NULL)
{
printf("\nQUEUE UNDERFLOW\n");
}
else
{
new=start;
printf("\nDELETED ITEM IS %d\n",new->info);
start=start->next;
//free(start);
}
}
void display()
{
temp=start;
if(start==NULL)
printf("QUEUE IS EMPTY\n");
else
{
printf("QUEUE IS:\n");
if(temp!=NULL)
for(temp=start;temp!=NULL;temp=temp->next)
{
printf("\n%d priority =%d\n",temp->info,temp->priority);
//temp=temp->next;
}
}
}
```

viva
1.What is meant by binary tree traversal?
2.What are the different binary tree traversal techniques?
3.What are the tasks performed during inorder traversal?
4.What are the tasks performed during postorder traversal?
5.State the merits of linear representation of binary trees.

Result

Thus the C program to implement Heaps using Priority Queues was executed successfully.

Ex. No. 8 **Dijkstra's Algorithm**
Date:

Aim

To implement Dijkstra's Algorithm in C

.

Algorithm

1. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. 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.

```
2. Array visited[ ] is initialized to zero.
       for(i=0;i<n;i++)
       visited[i]=0;
3. If the vertex 0 is the source vertex then visited[0] is marked as 1.
4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the
   source vertex 0.
        for(i=1;i<n;i++)
        distance[i]=cost[0][i];
   Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;
5. for(i=1;i<n;i++)
       – Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark
         visited[w] as 1.
       – Recalculate the shortest distance of remaining vertices from the source.
       – 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])
```

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;
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
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];
//initialize pred[],distance[] and visited[]
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;
//nextnode gives the node at minimum distance
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
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++;
}
//print the path and distance of each node
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);
}
}
```

viva

1. Define Graph.
2. Define adjacent nodes.
3. What is a directed graph?
4. What is an undirected graph?
5. What is a loop?

Result

Thus the C program to implement Dijkstra's Algorithm was executed successfully.

Ex. No. 9 **Prim's Algorithm**

Date:

Aim

To implement Prims's Algorithm in C

.

Algorithm

1.Begin

2.Create edge list of given graph, with their weights.

3.Draw all nodes to create skeleton for spanning tree.

4.Select an edge with lowest weight and add it to skeleton and delete edge from edge list.

5.Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.

6.Repeat step 5 until n-1 edges are added.

7.Return.

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;
}
//initialise visited[],distance[] and from[]
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; //cost of spanning tree
no_of_edges=n-1; //no. of edges to be added
while(no_of_edges>0)
{
//find the vertex at minimum distance from the tree
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];
//insert the edge in spanning tree
spanning[u][v]=distance[v];
spanning[v][u]=distance[v];
no_of_edges--;
visited[v]=1;
//updated the distance[] array
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);
}
```

viva

1.Define path in a graph?
2.What is a simple path?
3.Define indegree of a graph?
4.Define outdegree of a graph?
5What is a weighted graph?

Result

Thus the C program to implement Prims’s Algorithm was executed successfully.

Ex. No. 11a Linear Search

Date:

Aim

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

Algorithm

```
1. Start
2. Read number of array elements $n$
3. Read array elements $Ai$, $i = 0,1,2,\ldots n-1$
4. Read search value
5. Assign 0 to found
6. Check each array element against search
        If $Ai$ = search then
        found = 1
        Print "Element found"
        Print position $i$
        Stop
7. If found = 0 then
        print "Element not found"
8. Stop
```

Program

```
/* Linear search on a sorted array */
#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();
}
```

viva

1. Define searching
2. What is meant by linear search?
3. What is binary search?
4. What do you mean by linear probing?
5. What do you mean by primary clustering?

Result

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

Ex. No. 11b **Binary Search**
Date:

Aim

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

Algorithm

```
1. Start
2. Read number of array elements, say n
3. Create an array arr consisting n sorted elements
4. Get element, say key to be located
5. Assign 0 to lower and n to upper
6. 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
7. Print "Element not found"
8. Stop
```

Program

```
/* Binary Search on a sorted array */
#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();
}
```

viva

1.Define searching

2.What is meant by linear search?

3.What is binary search?

4.What do you mean by linear probing?

5.What do you mean by primary clustering?

Result
Thus an element is located quickly using binary search method.

Ex. No. 12a **Insertion Sort**
Date:

Aim

To sort an array of N numbers using Insertion sort.

Algorithm

1. Start
2. Read number of array elements *n*
3. Read array elements *Ai*
4. 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.
5. Stop

Program

```
/* Insertion Sort */
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]);
}
```

viva
1.Define sorting
2.Mention the types of sorting
3.What do you mean by internal and external sorting?
4.Define bubble sort
5.How the insertion sort is done with the array?

Result
Thus array elements was sorted using insertion sort.

Ex. No. 12.b **Selection Sort**
Date:

Aim

To implement selection sort in C
.

Algorithm

Let ARR is an array having N elements

```
1. Read ARR
2. Repeat step 3 to 6 for I=0 to N-1
3. Set MIN=ARR[I] and Set LOC=I
4. Repeat step 5 for J=I+1 to N
5. If MIN>ARR[J], then
       (a) Set MIN=ARR[J]
       (b) Set LOC=J
       [End of if]
       [End of step 4 loop]
6. Interchange ARR[I] and ARR[LOC] using temporary variable
       [End of step 2 outer loop]
7. Exit
```

Program

```
#include<stdio.h>
 int main()
{
   int i,j,n,loc,temp,min,a[30];
   printf("Enter the number of elements:");
   scanf("%d",&n);
   printf("\nEnter the elements\n");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   for(i=0;i<n-1;i++)
   {
      min=a[i];
      loc=i;
      for(j=i+1;j<n;j++)
      {
         if(min>a[j])
         {
            min=a[j];
            loc=j;
         }
      }
      temp=a[i];
      a[i]=a[loc];
      a[loc]=temp;
   }
   printf("\nSorted list is as follows\n");
   for(i=0;i<n;i++)
```

```
    {
        printf("%d ",a[i]);
    }
    return 0;
}
```

viva

1. What are the steps for selection sort?
2. What is meant by shell sort?
3. What are the steps in quick sort?
4. Define radix sort
5. What are the advantages of insertion sort

Result

Thus the C program to implement selection sort was executed successfully.

Ex. No. 13 **Merge Sort**
Date:

Aim

To sort an array of N numbers using Merge sort.

Algorithm

1. Start
2. Read number of array elements *n*
3. Read array elements *Ai*
4. Divide the array into sub-arrays with a set of elements
5. Recursively sort the sub-arrays
6. Merge the sorted sub-arrays onto a single sorted array.
7. Stop

Program

```
/* Merge sort */
#include <stdio.h>
#include <conio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int );
int size;
main()
{
int i, arr[30];
printf("Enter total no. of elements : ");
scanf("%d", &size);
printf("Enter array elements : ");
for(i=0; i<size; i++)
scanf("%d", &arr[i]);
part(arr, 0, size-1);
printf("\n Merge sorted list : ");
for(i=0; i<size; i++)
printf("%d ",arr[i]);
getch();
}
void part(int arr[], int min, int max)
{
int mid;
if(min < max)
{
mid = (min + max) / 2;
part(arr, min, mid);
part(arr, mid+1, max);
merge(arr, min, mid, max);
}
if (max-min == (size/2)-1)
{
printf("\n Half sorted list : ");
for(i=min; i<=max; i++)
printf("%d ", arr[i]);
}
}
void merge(int arr[],int min,int mid,int max)
{
int tmp[30];
int i, j, k, m;
j = min;
m = mid + 1;
for(i=min; j<=mid && m<=max; i++)
{
if(arr[j] <= arr[m])
{
tmp[i] = arr[j];
j++;
}
else
{
tmp[i] = arr[m];
m++;
}
}
if(j > mid)
{
for(k=m; k<=max; k++)
{
tmp[i] = arr[k];
i++;
}
}
else
{
for(k=j; k<=mid; k++)
{
tmp[i] = arr[k];
i++;
}
}
for(k=min; k<=max; k++)
arr[k] = tmp[k];
}
```

viva

1. What are the steps for selection sort?
2. What is meant by shell sort?
3. What are the steps in quick sort?
4. Define radix sort
5. What are the advantages of insertion sort

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

Ex. No. 14 **Open Addressing Hashing Technique**
Date:

Aim

To implement hash table using a C program.

Algorithm

1. Create a structure, data (hash table item) with key and value as data.
2. Now create an array of structure, data of some certain size (10, in this case). But, the size of array must be immediately updated to a prime number just greater than initial array capacity (i.e 10, in this case).
3. A menu is displayed on the screen.
4. User must choose one option from four choices given in the menu
5. Perform all the operations
6. Stop

Program

```
/* Open hashing */
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
main()
{
int a[MAX], num, key, i;
char ans;
int create(int);
void linearprobing(int[], int, int);
void display(int[]);
printf("\nCollision handling by linear probing\n\n");
for(i=0; i<MAX; i++)
a[i] = -1;
do
{
printf("\n Enter number:");
scanf("%d", &num);
key = create(num);
linearprobing(a, key, num);
printf("\nwish to continue?(y/n):");
ans = getch();
} while( ans == 'y');
display(a);
}
int create(int num)
{
int key;
key = num % 10;
return key;
}
void linearprobing(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("hash 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("\n Hash table is:");
for(i=0; i<MAX; i++)
printf("\n %d\t\t%d",i,a[i]);
}
```

viva

1.Define hashing function

2.What is open addressing?

3.What are the collision resolution methods?

4.Define separate chaining

5.Define Hashing.

Result

Thus hashing has been performed successfully