VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Amogh Krishna J S (1BM23CS029)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
September 2024-January 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by Amogh Krishna J S **(1BM23CS029)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

Prof. Basavaraj Jakalli
Assistant Professor
Department of CSE
BMSCE, Bengaluru

D**r. Kavitha Sooda**
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

```
#include<conio.h>
#include<stdio.h>
#define MAX 5
int s[10],top=-1,i,item,ch;
void main(){
   while ()
   {
      printf("1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n5.ISEMPTY\n6.ISFULL\n7.TOP\n");
      printf("Enter your choice: ");
      scanf("%d",&ch);
      swich(ch){
         case 1:push();
         break;
         case 2:item=pop();
         if(item!=-1){
            printf("POPPED ELEMENT=%d",item);
         }
         break;
         case 3:display();
         break;
         case 5:ISEMPTY();
         break;
         case 6:ISFULL();
         break;
         case 7:TOP();
         break;
         case 4:exit(0);}
      getch()
   }
}
void push(){
   if(top=MAX-1){
      printf("STACK OVERFLOW");
      return;
   }
   printf("Enter element to be pushed:\n");
   scanf("%d",&item);
   top=top+1;
   s[top]=item;
}
int pop(){
   if(top==-1){
      printf("STACK underflow");
```

```
        return -1;
    }
    item=s[top];
    top=top-1;
    return item;
}
void display(){
    if(top==-1){
        printf("Stack is Empty");
        return;
    }
    printf("Stack Contents is\n");
    for(i=top;i>=0,i--){
        printf("Element is %d",s[i])
    }
}
void ISEMPTY(){
    if(top==-1){
        printf("Stack is Empty");
        return;}
}
void ISFULL(){
    if(top=MAX-1){
        printf("Stack is full");
        return;}
}
void TOP(){
    printf("Topmost element is: %d",s[top]);
    return;
}
```

Output:

```
1.PUSH
2.POP
3.DISPLAY
4.EXIT
5.ISEMPTY
6.ISFULL
7.TOP
Enter your choice: 1
Enter element to be pushed:
1
1.PUSH
2.POP
3.DISPLAY
4.EXIT
5.ISEMPTY
6.ISFULL
7.TOP
Enter your choice: 1
Enter element to be pushed:
2
1.PUSH
2.POP
3.DISPLAY
4.EXIT
5.ISEMPTY
6.ISFULL
7.TOP
Enter your choice: 1
Enter element to be pushed:
3
1.PUSH
2.POP
3.DISPLAY
4.EXIT
5.ISEMPTY
6.ISFULL
7.TOP
Enter your choice: 3
Stack Contents is
Element is 3
Element is 2
Element is 1
```

```
1.PUSH
2.POP
3.DISPLAY
4.EXIT
5.ISEMPTY
6.ISFULL
7.TOP
Enter your choice: 2
POPPED ELEMENT=3
1.PUSH
2.POP
3.DISPLAY
4.EXIT
5.ISEMPTY
6.ISFULL
7.TOP
Enter your choice: 4
```

Lab Program 2:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

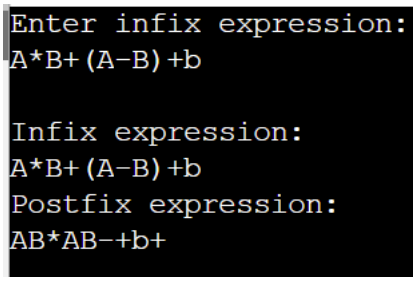
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int index=0,pos=0,top=-1,length;
char symbol,temp,infix[40],postfix[40],stack[20];
void infixtopostfix();
void push(char);
char pop();
int pred(char);
void main(){

   printf("Enter infix expression: \n");
   scanf("%s",infix);
   infixtopostfix();
   printf("\nInfix expression: \n%s",infix);
   printf("\nPostfix expression:\n%s",postfix);
   getch();
}

void infixtopostfix(){
   length=strlen(infix);
   push('#');
   while(index<length)
      {
      symbol=infix[index];
```

```
switch(symbol)
{
   case '(':push(symbol);
   break;
   case ')':temp=pop();
   while(temp!='('){
      postfix[pos]=temp;
      pos++;
      temp=pop();
   }
   break;
   case '+':
   case '-':
   case '*':
   case '/':
   case '^': while(pred(stack[top])>=pred(symbol))
   {
      temp=pop();
      postfix[pos++]=temp;
   }
   push(symbol);
   break;
   default: postfix[pos++]=symbol;
}
   index++;
}
while(top>0)
{
   temp=pop();
```

```
            postfix[pos++]=temp;
        }
    }
void push(char symbol){
    top=top+1;
    stack[top]=symbol;
}
char pop(){
    char symb;
    symb=stack[top];
    top=top-1;
    return(symb);
}
int pred(char symbol){
    int p;
    switch(symbol){
        case '^':p=3;
        break;
        case '*':
        case '/':p=2;
        break;
        case '+':
        case '-':p=1;
        break;
        case '(':p=0;
        break;
        case '#':p=-1;
        break;
    }
```

```
    return(p);
}
```

Output



Lab Program 3:

a) Queue Question

WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display. The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include <stdio.h>
int front=-1, rear=-1, i, queue[10], ch, item;
#define MAX 3
void insert();
int delete();
void display();
void main()
{printf("AMOGH KRISHNA J S USN 1BM23CS029 \n");
   while (1)
   {
      printf(" 1. INSERT \n 2. DELETE \n 3. DISPLAY \n 4. EXIT \n ");
      printf("enter your choice \n");
      scanf("%d",&ch);
      switch (ch)
      {
         case 1: insert();
                 break;
         case 2: item=delete();
                 if (item!=-1)
                 {
                    printf("deleted item is:%d\n", item);
                 } break;
         case 3: display();
                 break;
```

```
            case 4: exit(0);
        }
    }
}
void insert()
{
    if (rear==MAX-1)
    {
        printf("queue is full.\n");
        return;
    }
    printf("enter element to be inserted: \n");
    scanf("%d",&item);
    if (rear ==-1 && front ==-1)
    {
        rear=0;
        front =0;
    }
    else
    {
        rear=rear+1;
    }
    queue[rear]=item;
    return;
}

int delete()
{
    if (front ==-1 && rear==-1)
```

```
    {
        printf("queue is empty.\n");
        return -1;
    }
    item=queue[front];
    if (front==rear)
    {
        front=-1;
        rear=-1;
    }
    else
    {
        front=front+1;
    }
    return item;
}

void display()
{
    if (front==-1 && rear==-1)
    {
        printf("queue is empty.\n");
        return;
    }
    printf("queue: \n");
    for (i=front;i<=MAX-1;i++)
    {
        printf("%d  ", queue[i]);
    }
```

```
    printf("\n");
    return;
}
```

Output

```
 1. INSERT
 2. DELETE
 3. DISPLAY
 4. EXIT
 enter your choice
1
enter element to be inserted:
1
 1. INSERT
 2. DELETE
 3. DISPLAY
 4. EXIT
 enter your choice
1
enter element to be inserted:
2
 1. INSERT
 2. DELETE
 3. DISPLAY
 4. EXIT
 enter your choice
1
enter element to be inserted:
3
 1. INSERT
 2. DELETE
 3. DISPLAY
 4. EXIT
 enter your choice
2
deleted item is:1
 1. INSERT
 2. DELETE
 3. DISPLAY
 4. EXIT
 enter your choice
3
queue:
2  3
 1. INSERT
 2. DELETE
 3. DISPLAY
 4. EXIT
```

Leetcode:

The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.

You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return an array ans of length nums1.length such that ans[i] is the next greater element as described above.

Solution

```
int* nextGreaterElement(int* nums1, int nums1Size, int* nums2, int nums2Size, int* returnSize) {


    int* ans = (int*)calloc(nums2Size, sizeof(int));

    int* stack = (int*)calloc(nums2Size, sizeof(int));

    int top = -1;

    int max = nums2[0];


    for(int i = nums2Size-1; i >=0; i--){

        max = ( nums2[i] > max )? nums2[i] : max ;

        while( top != -1 && nums2[stack[top]] <= nums2[i]){

            top--;

        }

        ans[i] = (top == -1)? -1 : nums2[stack[top]];

        stack[++top] = i;

    }

    free(stack);

    int* map = (int*)calloc(max+1, sizeof(int));
```

```
    for(int i = 0; i < nums2Size; i++){
        map[nums2[i]] = ans[i];
    }
    free(ans);
    int* result = (int*)calloc(nums1Size, sizeof(int));
    for(int i = 0; i< nums1Size; i++){
        result[i] = map[nums1[i]];
    }

    *returnSize = nums1Size;
    return result;
}
```

b) Circular Queue Question

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 5
void insert();
int delete();
void display();
int cq[20],front=-1,rear=-1,item,ch,i;
void main(){
    while(1){
        printf("Circular Queue MENU\n");
        printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
```

```
        printf("Enter your choice:\n");
        scanf("%d",&ch);
        switch(ch){
            case 1:insert();
            break;
            case 2:item=delete();
            if(item!=-1){
                printf("The deleted element is:%d\n",item);
            }
            break;
            case 3:display();
            break;
            case 4:exit(0);
        }
    }
}
void insert(){
    if (front==(rear+1)%MAX){
            printf(" circular queue is full\n");
            return;
        }
    if (rear==-1&&front==-1){
        rear=0;
        front=0;
    }
    else{
        rear=(rear+1)%MAX;
    }
    printf("enter the element to be inserted:\n");
```

```
    scanf("%d",&item);
    cq[rear]=item;
    return;

}

int delete(){
    if(front==-1&&rear==-1)    {
        printf("circular queue is empty\n");
        return (-1);
    }
    item=cq[front];
    if(front==rear)    {
        front=-1;
        rear=-1;
    }
    else
        front=(front+1)%MAX;
    return item;
}

void display(){
    if(front==-1&&rear==-1)    {
        printf("circular queue is empty\n");
        return;
    }

    printf("CIRCULAR QUEUE\n");
    if (front<=rear){
```

```
        for (int i=front;i<=rear;i++){
            printf("%d\n",cq[i]);
        }
    }

  else{
        for(int i=front;i<=MAX-1;i++){
            printf("%d\n",cq[i]);
        }
        for (int i=0;i<=rear;i++){
            printf("%d\n",cq[i]);
        }
    }
  return;
}
```

Leetcode:

You have a RecentCounter class which counts the number of recent requests within a certain time frame.

Implement the RecentCounter class:

- **RecentCounter() Initializes the counter with zero recent requests.**
- **int ping(int t) Adds a new request at time t, where t represents some time in milliseconds, and returns the number of requests that has happened in the past 3000 milliseconds (including the new request). Specifically, return the number of requests that have happened in the inclusive range [t - 3000, t].**

It is guaranteed that every call to ping uses a strictly larger value of t than the previous call.

Solution

```
#define MAX_SIZE 10000


typedef struct {
```

```
    int* pingHistory;
    int head;
    int tail;
} RecentCounter;

RecentCounter* recentCounterCreate() {
    RecentCounter* obj = malloc(sizeof(RecentCounter));
    obj -> pingHistory = calloc(10000, sizeof(int));
    obj -> head = 0;
    obj -> tail = 0;
    return obj;
}

int recentCounterPing(RecentCounter* obj, int t) {
    obj -> pingHistory[obj->head++] = t;

    for(; obj->tail < obj -> head; obj->tail++)
    {
        if((t-3000) <= obj -> pingHistory[obj->tail])
        {
            break;
        }
    }
    return obj->head-obj->tail;
}

void recentCounterFree(RecentCounter* obj) {
    free(obj->pingHistory);
    free(obj);
```

}

Output

```
Circular Queue MENU
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:
1
enter the element to be inserted:
1
Circular Queue MENU
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:
1
enter the element to be inserted:
2
Circular Queue MENU
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:
1
enter the element to be inserted:
3
Circular Queue MENU
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:
2
The deleted element is:1
Circular Queue MENU
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:
3
CIRCULAR QUEUE
2
3
Circular Queue MENU
1.Insert
2.Delete
3.Display
4.Exit
```

Lab Program 4:

WAP to Implement Singly Linked List with following operations

a) Create a linkedlist.

b) Insertion of a node at first position, at any position and at end of list.

c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *link;
};

typedef struct Node node;
node *start = NULL;
node *new1, *curr, *ptr;

void create();
void display();
void InsertStart();
void Insertposition();
void InsertEnd();

void main() {
    int ch;
    while (1) {
        printf("\n1. Create \n2. Display \n3. Insert at Beginning \n4. Insert at Position \n5. Insert at End \n6. Exit");
```

```
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1: create();
                break;
            case 2: display();
                break;
            case 3: InsertStart();
                break;
            case 4: Insertposition();
                break;
            case 5: InsertEnd();
                break;
            case 6: exit(0);
        }
    }
}

void create() {
    char ch;
    do {
        new1 = (node*)malloc(sizeof(node));
       printf("\nEnter the Value: ");
       scanf("%d",&new1->data);
       if (start==NULL)
       {
          start=new1;
          curr=new1;
       }
```

```
        else {
            curr->link = new1;
            curr=new1;
        }

        printf("Do you want to add any more elements (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.");
        return;
    }

    ptr = start;
    printf("\nElements in the Linked List: \n");

    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
    printf("\n");
}

void InsertStart() {
    new1 = (node*)malloc(sizeof(node));
```

```
    printf("\nEnter the value: ");
    scanf("%d",&new1->data);
    if(start==NULL)
    {
        start=new1;
        new1->link=NULL;
        return;
    }
    else {
        new1->link=start;
        start=new1;
        return;
    }
}

void InsertEnd() {
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter the cvalue: ");
    scanf("%d",&new1->data);
    if(start==NULL)
    {
        start=new1;
        new1->link=NULL;
        return;
    }

    ptr=start;
    while(ptr->link !=NULL)
    {
```

```
        ptr=ptr->link;
    }
    ptr->link=new1;
    new1->link=NULL;
    return;
}

void Insertposition() {
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter the value: ");
    scanf("%d",&new1->data);
    if(start==NULL)
    {
        start=new1;
        new1->link=NULL;
        return;
    }

    int i=1, pos;
    ptr=start;
    printf("\nEnter the position you would like to insert from: ");
    scanf("%d",&pos);
    while (ptr!=NULL && i<pos-1)
    {
        ptr=ptr->link;
        i++;
    }
    if(ptr==NULL)
    {
```

```
        return;
    }

    new1->link=ptr->link;
    ptr->link=new1;
}
```

Output

```
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter your choice: 1

Enter the Value: 1
Do you want to add any more elements (Y/N)? Y

Enter the Value: 1
Do you want to add any more elements (Y/N)? y

Enter the Value: 3
Do you want to add any more elements (Y/N)? n

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter your choice: 2

Elements in the Linked List:
1 1 3

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter your choice: 3

Enter the value: 4

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter your choice: 5

Enter the cvalue: 6
```

```
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter your choice: 4

Enter the value: 2

Enter the position you would like to insert from: 2

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter your choice: 2

Elements in the Linked List:
4 2 1 1 3 6

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter your choice: 6
```

Leetcode

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step:

If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue.

Otherwise, they will leave it and go to the queue's end.

This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays students and sandwiches where sandwiches[i] is the type of the ith sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the jth student in the initial queue (j = 0 is the front of the queue). Return the number of students that are unable to eat.

Solution

```
int countStudents(int* students, int studentsSize, int* sandwiches, int sandwichesSize) {
    int st = studentsSize;
    int count = 0;
    int i = 0;
    int flag = 0;
    while(1) {
        if(students[i] == sandwiches[count]) {
            count++;
            students[i] = 2;
            flag = 1;
            if(count == st) {
                break;
            }
        }
        i++;
        if(i == st && flag == 1) {
            i = 0;
            flag = 0;
            continue;
        }
        else if(i == st && flag == 0) {
            break;
        }
    }
    count = 0;
```

```
    for(int i = 0 ; i < st ; i++) {
        if(students[i] != 2) {
            count++;
        }
    }
    return count;
}
```

Lab Program 5:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *link;
};
typedef struct Node node;
node *start = NULL;
void create();
void display();
void DeletefromStart();
void DeleteatPosition();
void DeleteatEnd();
void main() {
   int ch;
   while (1) {
      printf("\n1. Create \n2. Display \n3. Delete from the Beginning \n4. Delete at Position \n5. Delete from the End \n6. Exit");
      printf("\nEnter your choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            create();
            break;
```

```
            case 2:
                display();
                break;
            case 3:
                DeletefromStart();
                break;
            case 4:
                DeleteatPosition();
                break;
            case 5:
                DeleteatEnd();
                break;
            case 6:
                exit(0);
            default:
                printf("Enter a VALID CHOICE (1-9).\n");
        }
    }
}
void create() {
    char ch;
    node *new1, *curr;

    do {
        new1 = (node*)malloc(sizeof(node));
       printf("\n enter value:\n");
       scanf("%d",&new1->data);
       if (start==NULL)
       {
```

```
            start=new1;
            curr=new1;
        }
        else {
            curr->link = new1;
            curr=new1;
        }
        printf("Do you want to add any more elements (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}
void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }
    node *temp = start;
    printf("\nElements in Linked List are: \n");

    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->link;
    }
    printf("\n");
}
void DeletefromStart() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
```

```
        return;
    }
    node *temp = start;
    start = start->link;
    free(temp);
    printf("\nFirst element has been deleted successfully.\n");
}
void DeleteatPosition() {
    int pos, i = 1;
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }
    printf("\nEnter the position of the element you want to delete: ");
    scanf("%d", &pos);

    node *temp = start;
    node *prev = NULL;
    if (pos == 1) {
        start = temp->link;
        free(temp);
        printf("\nElement at position %d has been deleted successfully.\n", pos);
        return;
    }
    while (temp != NULL && i < pos) {
        prev = temp;
        temp = temp->link;
        i++;
    }
```

```
    if (temp == NULL) {
        printf("\nPosition not found.\n");
        return;
    }
    prev->link = temp->link;
    free(temp);
    printf("\nElement at position %d has been deleted successfully.\n", pos);
}
void DeleteatEnd() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }
    node *temp = start;
    node *prev = NULL;
    if (start->link == NULL) {
        start = NULL;
        free(temp);
        printf("\nLast element has been deleted successfully.\n");
        return;
    }
    while (temp->link != NULL) {
        prev = temp;
        temp = temp->link;
    }
    prev->link = NULL;
    free(temp);
    printf("\nLast element has been deleted successfully.\n");
}
```

Output

```
1. Create
2. Display
3. Delete from the Beginning
4. Delete at Position
5. Delete from the End
6. Exit
Enter your choice: 1

 enter value:
1
Do you want to add any more elements (Y/N)? y

 enter value:
2
Do you want to add any more elements (Y/N)? y

 enter value:
3
Do you want to add any more elements (Y/N)? y

 enter value:
4
Do you want to add any more elements (Y/N)? y

 enter value:
5
Do you want to add any more elements (Y/N)? y

 enter value:
6
Do you want to add any more elements (Y/N)? 2

1. Create
2. Display
3. Delete from the Beginning
4. Delete at Position
5. Delete from the End
6. Exit
Enter your choice: 2

Elements in Linked List are:
1 2 3 4 5 6
```

```
1. Create
2. Display
3. Delete from the Beginning
4. Delete at Position
5. Delete from the End
6. Exit
Enter your choice: 3

First element has been deleted successfully.

1. Create
2. Display
3. Delete from the Beginning
4. Delete at Position
5. Delete from the End
6. Exit
Enter your choice: 4

Enter the position of the element you want to delete: 2

Element at position 2 has been deleted successfully.

1. Create
2. Display
3. Delete from the Beginning
4. Delete at Position
5. Delete from the End
6. Exit
Enter your choice: 5

Last element has been deleted successfully.

1. Create
2. Display
3. Delete from the Beginning
4. Delete at Position
5. Delete from the End
6. Exit
Enter your choice: 2

Elements in Linked List are:
2 4 5
```

```
1. Create
2. Display
3. Delete from the Beginning
4. Delete at Position
5. Delete from the End
6. Exit
Enter your choice: 6
```

Lab Program 6:

a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int value;
   struct node *next;
};

typedef struct node* NODE;
NODE getnode(){
   NODE new_node = (NODE)malloc(sizeof(struct node));
   if (new_node==NULL) {
      printf("Memory allocation failed.\n");
      exit(1);
   }
   return new_node;
}

NODE insert_end(int item, NODE START){
   NODE new_end = getnode();
   new_end->value = item;
   new_end->next = NULL;
   if (START==NULL){
      return new_end;
   }
   NODE current = START;
   while (current->next!=NULL) {
```

```
        current = current->next;
    }
    current->next = new_end;
    return START;
}

NODE reverse(NODE START){
    NODE current,temp;
    current=NULL;
    while (START!=NULL){
        temp=START;
        START=START->next;
        temp->next=current;
        current=temp;
    }
    return current;
}

NODE concatenate(NODE START_1, NODE START_2){
     NODE last1;
     if (START_1==NULL && START_2==NULL){
        return NULL;
     }
     else if(START_1==NULL){
        return START_2;
     }
     else if (START_2==NULL){
        return START_1;
     }
```

```
    else{
      last1=START_1;
      while(last1->next!=NULL){
         last1=last1->next;
      }
      last1->next=START_2;
    }
    return START_1;
}

NODE sort(NODE start) {
   NODE temp1, temp2;
   temp1=start;
   while (temp1!=NULL) {
      temp2=temp1->link;
      while (temp2!=NULL) {
         if (temp1->data > temp2->data) {
            int x = temp1->data;
            temp1->data = temp2->data;
            temp2->data = x;
         }
         temp2 = temp2->link;
      }
      temp1 = temp1->link;
   }
   return start;
}
```

```
void display(NODE START)
{ NODE temp;

   if(START==NULL)
   {
      printf("linked list is empty");
      return;
   }
    temp=START;
   while(temp!=NULL)
   {
      printf("%d\t",temp->value);
      temp=temp->next;
   }
}

int main(){
   NODE START_1= NULL;
   NODE START_2= NULL;
   int choice, item, pos;
   while(1){
      printf("\nMenu:\n");
      printf("1. Insert in linked list 1\n ");
      printf("2. Insert in linked list 2\n");
      printf("3. Sort in linked list 1\n");
      printf("4. Sort in linked list 2\n");
      printf("5. Reverse in linked list 1\n");
      printf("6. Reverse in linked list 2\n");
      printf("7. Concatenate the two lists\n");
```

```
printf("8. Display LL 1\n");
printf("9. Display LL 2\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch(choice){
   case 1:
      printf("Enter value to insert: ");
      scanf("%d", &item);
      START_1 = insert_end(item, START_1);
      break;
   case 2:
      printf("Enter value to insert: ");
      scanf("%d", &item);
      START_2 = insert_end(item, START_2);
      break;
   case 3:
      printf("Sorting LL1");
      START_1=sort(START_1);
      break;
   case 4:
      printf("Sorting LL2");
      START_2=sort(START_2);
      break;
   case 5:
      printf("LL1 being reversed");
      START_1 =reverse(START_1);
      break;
   case 6:
      printf("LL2 being reversed");
```

```
            START_2=reverse (START_2);
            break;
        case 7:
            START_1=concatenate (START_1,START_2);
            break;
        case 8:
            display(START_1);
            break;
        case 9:
            display(START_2);
            break;
        default:
            printf("Invalid choice. \n");
      }
   }
   return 0;
}
```

Output

```
Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 1
Enter value to insert: 1

Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 1
Enter value to insert: 2

Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 1
Enter value to insert: 3
```

```
Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 2
Enter value to insert: 7

Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 8
1       2       3
Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 9
1       3       7
```

```
Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 7

Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 6
LL2 being reversed
Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 2
Enter value to insert: 1
```

```
Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 4
Sorting LL2
Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 8
1       2       3       3       7
Menu:
1. Insert in linked list 1
 2. Insert in linked list 2
3. Sort in linked list 1
4. Sort in linked list 2
5. Reverse in linked list 1
6. Reverse in linked list 2
7. Concatenate the two lists
8. Display LL 1
9. Display LL 2
Enter your choice: 9
1       1       3       7
```

b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

```
#include <stdio.h>
#include <stdlib.h>
void push();
int pop();
void insert();
int delete();
void display();
struct Node {
   int data;
   struct Node *link;
};
typedef struct Node node;
node *start=NULL;
node *curr, *temp, *new1;
int main(){
   while(1){
      printf("1. STACK \n2. QUEUE\n3. EXIT\n");
      printf("Make your choice:\n");
      int ch;
      scanf("%d", &ch);

      switch(ch){
         case 1:{clearList();
            while (1){
               printf("Linked list is being implemented as stack!\n1.PUSH
\n2.POP\n3.DISPLAY \n4.EXIT\n");
               int ch1;
               scanf("%d", &ch1);
               switch (ch1){
```

```
                case 1: push(); break;
                case 2:{
                    int poppedData=pop();
                    if (poppedData != -1){
                        printf("Popped element is: %d\n", poppedData);
                    }
                    break;
                }
                case 3: display(); break;
                case 4: break;
                default: printf("Invalid choice\n");
            }
            if (ch1==4){break;}
        }
    } break;
    case 2:{clearList();
        while (1){
            printf("Linked list is being implemented as a queue!\n1.INSERT
\n2.DELETE\n3.DISPLAY\n4.EXIT\n");
            int ch1;
            scanf("%d", &ch1);
            switch (ch1){
                case 1: insert(); break;
                case 2:{
                    int deletedData = delete();
                    if (deletedData!=-1) {
                        printf("Deleted element is: %d\n", deletedData);
                    }
                    break;
                }
```

```
                    case 3: display(); break;
                    case 4: break;
                    default: printf("Invalid choice\n");
                }
                if (ch1==4){break;}
            }
        } break;
        case 3: return 0;
        default: printf("Invalid choice\n");
    }
  }
}
void push(){
  new1 = (node*)malloc(sizeof(node));
  printf("Enter element:\n");
  scanf("%d", &new1->data);
  if (start==NULL) {
    start=new1;
    new1->link=NULL;
  } else {
    new1->link= tart;
    start = new1;
  }
}
int pop(){
  if (start==NULL) {
    printf("Stack is empty!\n");
    return -1;
  }
```

```
    temp=start;
    start=start->link;
    int data=temp->data;
    free(temp);
    return data;
}
void insert(){
    new1=(node*)malloc(sizeof(node));
    printf("Enter element:\n");
    scanf("%d", &new1->data);

    if (start==NULL){
        start=new1;
        new1->link=NULL;
    } else{
        temp=start;
        while (temp->link!=NULL) {
            temp=temp->link;
        }
        temp->link=new1;
        new1->link=NULL;
    }
}
int delete(){
    if (start==NULL){
        printf("Queue is empty.\n");
        return -1;
    }
    temp=start;
```

```
    start=start->link;
    int data=temp->data;
    free(temp);
    return data;
}
void display(){
    if (start==NULL){
        printf("\nLinked list is empty.\n");
        return;
    }
    temp=start;
    printf("Elements are:\n");
    while (temp!=NULL) {
        printf("%d ", temp->data);
        temp=temp->link;
    }
    printf("\n");
}
void clearList(){
    temp=start;
    while (temp!=NULL){
        start=temp->link;
        free(temp);
        temp=start;
    }
    printf("All nodes have been deleted. The list is now empty.\n");
}
```

Output

```
1. STACK
2. QUEUE
3. EXIT
Make your choice:
1
All nodes have been deleted. The list is now empty.
Linked list is being implemented as stack!
1.PUSH
2.POP
3.DISPLAY
4.EXIT
1
Enter element:
1
Linked list is being implemented as stack!
1.PUSH
2.POP
3.DISPLAY
4.EXIT
1
Enter element:
2
Linked list is being implemented as stack!
1.PUSH
2.POP
3.DISPLAY
4.EXIT
1
Enter element:
3
Linked list is being implemented as stack!
1.PUSH
2.POP
3.DISPLAY
4.EXIT
3
Elements are:
3 2 1
Linked list is being implemented as stack!
1.PUSH
2.POP
3.DISPLAY
4.EXIT
4
1. STACK
2. QUEUE
3. EXIT
Make your choice:
2
```

```
All nodes have been deleted. The list is now empty.
Linked list is being implemented as a queue!
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
1
Enter element:
1
Linked list is being implemented as a queue!
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
1
Enter element:
2
Linked list is being implemented as a queue!
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
1
Enter element:
3
Linked list is being implemented as a queue!
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
3
Elements are:
1 2 3
Linked list is being implemented as a queue!
1.INSERT
2.DELETE
3.DISPLAY
4.EXIT
```

Lab Program 7:

WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.

b) Insert a new node to the left of the node.

c) Delete the node based on a specific value

d) Display the contents of the list

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

struct Node{
    int data;
    struct Node *left;
    struct Node *right;
};

typedef struct Node node;
node *start = NULL;
node *new1, *curr, *ptr;

void create();
void display();
void insertleft();
void deletespecific();

void main(){
    int ch;
    while(1){
        printf("\n1. Create \n2. Display \n3. Insert Left \n4. Delete Specific Element \n5. Exit");
```

```
        printf("\nEnter choice: ");
        scanf("%d", &ch);

        switch (ch){
            case 1: create();
                break;
            case 2: display();
                break;
            case 3: insertleft();
                break;
            case 4: deletespecific();
                break;
            case 5: exit(0);
        }
    }
}

void create(){
    char ch;

    do{
        new1 = (node*)malloc(sizeof(node));
        printf("\nEnter Value: ");
        scanf("%d", &new1->data);
        new1->left = NULL;
        new1->right = NULL;

        if (start == NULL) {
            start = new1;
```

```
            curr = new1;
        } else {
            curr->right = new1;
            new1->left = curr;
            curr = new1;
        }

        printf("Do you want to add Element (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.");
        return;
    }

    ptr = start;
    printf("\nElements in Linked List: \n");

    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->right;
    }
    printf("\n");
}

void insertleft() {
```

```
int val;
printf("\nEnter Value to Insert Left: ");
scanf("%d", &val);

new1 = (node*)malloc(sizeof(node));
new1->data = val;
new1->left = NULL;
new1->right = NULL;

printf("\nEnter the Value to Insert Left of: ");
scanf("%d", &val);

ptr = start;
while (ptr != NULL && ptr->data != val) {
   ptr = ptr->right;
}

if (ptr != NULL) {
   new1->right = ptr;
   new1->left = ptr->left;
   if (ptr->left != NULL) {
      ptr->left->right = new1;
   }
   ptr->left = new1;
   if (ptr == start) {
      start = new1;
   }
} else {
   printf("\nValue not found in the list.\n");
```

```
    }
}

void deletespecific() {
    int value;
    printf("\nEnter Value to Delete: ");
    scanf("%d", &value);

    ptr = start;
    while (ptr != NULL && ptr->data != value) {
        ptr = ptr->right;
    }

    if (ptr == NULL) {
        printf("\nValue not found in the list.\n");
        return;
    }

    if (ptr->left != NULL) {
        ptr->left->right = ptr->right;
    }
    if (ptr->right != NULL) {
        ptr->right->left = ptr->left;
    }
    if (ptr == start) {
        start = ptr->right;
    }

    free(ptr);
```

```
    printf("\nElement with value %d deleted.\n", value);
}
```

Output

```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter choice: 1

Enter Value: 1
Do you want to add Element (Y/N)? y

Enter Value: 2
Do you want to add Element (Y/N)? y

Enter Value: 3
Do you want to add Element (Y/N)? n

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter choice: 3

Enter Value to Insert Left: 4

Enter the Value to Insert Left of: 2

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter choice: 4

Enter Value to Delete: 3

Element with value 3 deleted.

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter choice: 2

Elements in Linked List:
1 4 2
```

```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter choice: 5
```

Lab Program 8:

Write a program:

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., inorder, preorder and post order.

c) To display the elements in the tree.

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

struct Node{
   int data;
   struct Node *left, *right;
};

typedef struct Node node;

node* createnode(int data) {
   node* new1 = (node*)malloc(sizeof(node));
   new1->data = data;
   new1->left = new1->right = NULL;
   return new1;
}

node* insertnode(node* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }
   if (data < root->data) {
      root->left = insertNode(root->left, data);
```

```
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}

void inordertraversal(node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

void preordertraversal(node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

void postordertraversal(node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
```

```
int main() {
    node* root = NULL;
    int choice, value;

    printf("Binary Search Tree Menu:\n");
    while (1) {
        printf("\n1. Insert\n2. Inorder Traversal\n3. Preorder Traversal\n4. Postorder
Traversal\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                root = insertnode(root, value);
                break;
            case 2:
                printf("Inorder Traversal: ");
                inordertraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Preorder Traversal: ");
                preordertraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder Traversal: ");
```

```
            postordertraversal(root);
            printf("\n");
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }

   return 0;
}
```

Output

```
Binary Search Tree Menu:

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 10

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 8

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 9

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 4

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 11

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 2
Inorder Traversal: 4 8 9 10 11
```

```
1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 3
Preorder Traversal: 10 8 4 9 11

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 4
Postorder Traversal: 4 9 8 11 10

1. Insert
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 5
```

Lab Program 9:

a) Write a program to traverse a graph using BFS method.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int graph[MAX][MAX];
int visited[MAX];
int queue[MAX];
int front = -1, rear = -1;
void enqueue(int item) {
    if (rear == MAX - 1) {
        printf("Queue overflow\n");
        return;
    }
    if (front == -1) front = 0;
    queue[++rear] = item;
}
int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue underflow\n");
        return -1;
    }
    return queue[front++];
}
```

```
void bfs(int start, int n) {
    int i;
    enqueue(start);
    visited[start] = 1;
    printf("BFS Traversal: ");
    while (front <= rear) {
        int current = dequeue();
        printf("%d ", current);

        for (i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}
int main() {
    int n, start;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
```

```
    }
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    printf("Enter the starting vertex: ");
    scanf("%d", &start);
    bfs(start, n);
    return 0;
}
```

Output

b) Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>
#define MAX 100
int graph[MAX][MAX];
int visited[MAX];
int n;
void dfs(int v){
   printf("visited vertex: %d\n",v);
   visited[v]=1;
   for (int i=0; i<n; i++) {
      if (graph[v][i] == 1 && !visited[i]) {
         dfs(i);
      }
   }
}
int isConnected() {
   for (int i=0; i<n; i++) {
      visited[i] = 0;
   }
   dfs(0);
   for (int i=0; i<n; i++) {
      if (!visited[i]) {
         return 0;
      }
   }
```

```
    return 1;
}
int main(){
    int i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j=0; j<n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    if (isConnected()){
        printf("Graph is connected.\n");
    } else {
        printf("Graph is not connected.\n");
    }
    return 0;
}
```

Output

```
Enter the number of vertices: 5
Enter adjacency matrix:
0
0
1
1
1
0
0
0
1
1
1
0
0
1
0
1
1
1
0
0
1
1
0
0
0
visited vertex: 0
visited vertex: 2
visited vertex: 3
visited vertex: 1
visited vertex: 4
Graph is connected.
```

Lab Program 10:

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the

keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash

function H: K -> L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_EMPLOYEES 100
#define m 100
typedef struct {
    int key;
    int address;
} EmployeeRecord;
int hashTable[m];
int hashFunction(int key) {
    return key % m;
}
int insert(int key) {
    int index = hashFunction(key);
    while (hashTable[index] != -1) {
        index = (index + 1) % m;
    }
    hashTable[index] = key;
    return index;
```

```
}
void displayHashTable() {
    printf("\nHash Table:\n");
    printf("Index  Key\n");
    for (int i = 0; i < m; i++) {
        if (hashTable[i] != -1) {
            printf("%d      %d\n", i, hashTable[i]);
        }
    }
}
int main() {
    for (int i = 0; i < m; i++) {
        hashTable[i] = -1;
    }
    int employeeKeys[MAX_EMPLOYEES];
    int numEmployees;

    printf("Enter number of employees: ");
    scanf("%d", &numEmployees);

    printf("Enter the employee keys (4-digit integers):\n");
    for (int i = 0; i < numEmployees; i++) {
        scanf("%d", &employeeKeys[i]);
    }
    for (int i = 0; i < numEmployees; i++) {
        int address = insert(employeeKeys[i]);
        printf("Employee key %d inserted at address %d\n", employeeKeys[i], address);
```

```
    }
    displayHashTable();
    return 0;
}
```

Output

```
Enter number of employees: 9
Enter the employee keys (4-digit integers):
1234
1111
1444
1342
1567
1777
1980
1665
1343
Employee key 1234 inserted at address 34
Employee key 1111 inserted at address 11
Employee key 1444 inserted at address 44
Employee key 1342 inserted at address 42
Employee key 1567 inserted at address 67
Employee key 1777 inserted at address 77
Employee key 1980 inserted at address 80
Employee key 1665 inserted at address 65
Employee key 1343 inserted at address 43

Hash Table:
Index  Key
11      1111
34      1234
42      1342
43      1343
44      1444
65      1665
67      1567
77      1777
80      1980
```