VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

RANJAN DEVI (1BM22CS219)

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
Dec 2023- March 2024

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 RANJAN DEVI **(1BM22CS219)**, who is a 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 2023-24. 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. Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Write a program to simulate the working of stack using an array.	4
2	WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. Leetcode account creation.	7
3	3a) Working of a queue of integers using an array. 3b) Working of a circular queue of integers using an array.	10
4	WAP to Implement Singly Linked List(insertion). Leetcode program	15
5	WAP to Implement Singly Linked List(deletion).Leetcode program	21
6	6a) Sort the linked list, Reverse the linked list, Concatenation of two linked lists. 6b) Single Link List to simulate Stack and Queue operations.	27
7	WAP to Implement doubly link list. Leetcode program	36
8	WAP to construct a binary Search tree and to traverse the tree using all the methods. Leetcode program	41
9	9a) Write a program to traverse a graph using BFS method. 9b) Write a program to check whether given graph is connected or not using DFS method. Hackerrank program	45
10	Hashing table using linear probing	50

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 <stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
void push(int st[],int *top)
{
	int item;
	if(*top==STACK_SIZE-1)
		printf("Stack overflow\n");
	else
	{
		printf("\nEnter an item :");
		scanf("%d",&item);
		(*top)++;
		st[*top]=item;
	}
}
void pop(int st[],int *top)
{
	if(*top==-1)
		printf("Stack underflow\n");
	else
	{
		printf("\n%d item was deleted",st[(*top)--]);
	}
}
void display(int st[],int *top)
{
	int i;
	if(*top==-1)
		printf("Stack is empty\n");
	for(i=0;i<=*top;i++)
		printf("%d\t",st[i]);
}
void main()
{
	int st[10],top=-1, c,val_del;
	while(1)
	{
		printf("\n1. Push\n2. Pop\n3. Display\n");
		printf("\nEnter your choice :");
		scanf("%d",&c);
		switch(c)
		{
```

```
                    case 1: push(st,&top);
                            break;
                    case 2: pop(st,&top);
                            break;
                    case 3: display(st,&top);
                            break;
                    default: printf("\nInvalid choice!!!");
                            exit(0);
                }
        }
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\jyothika\DST> cd "d:\jyothika\DST\" ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :12

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :65

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :45

1. Push
2. Pop
3. Display

Enter your choice :1
Stack overflow
```

```
1. Push
2. Pop
3. Display

Enter your choice :2

45 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2

65 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :3
12
1. Push
2. Pop
3. Display

Enter your choice :2

12 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2
Stack underflow

1. Push
2. Pop
3. Display

Enter your choice :4

Invalid choice!!!
```

Lab program 2:

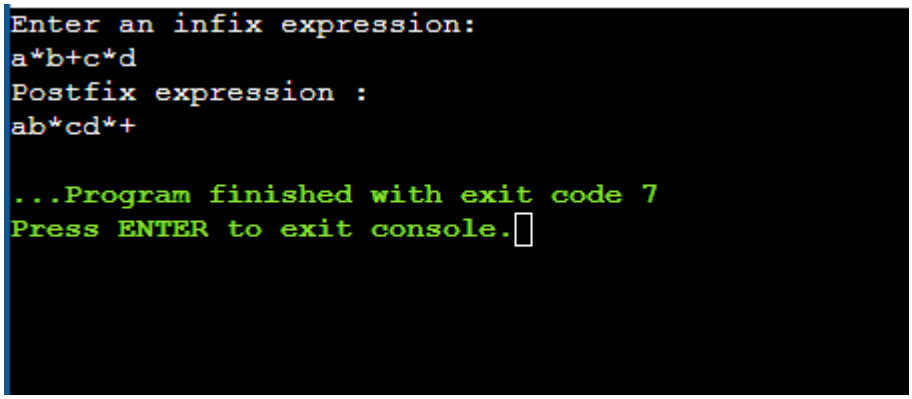
WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and /(divide)

```
#include <stdio.h>
#include <string.h>
int i,j=0,n,top=-1;
char val,temp,c;
char stack[100];
char infix[100],postfix[100];
void push(char );
char pop();
int precedence( char);
void main()
{
	printf("Enter an infix expression:\n");
	scanf("%s",infix);
	n=strlen(infix);
	for(i=0;i<=n;i++)
	{
		if(infix[i]=='(')
			push(infix[i]);
		else if(infix[i]>='a' && infix[i]<='z' || infix[i]>='A' && infix[i]<='Z' || infix[i]>='0'
		&& infix[i]<='9')
		{
			postfix[j]=infix[i];
			j++;
		}
		else if(infix[i]==')')
		{
			while(stack[top]!='(')
			{	postfix[j]=pop();
				j++;
			}
			temp=pop();
		}
		else
		{
			while(precedence(infix[i])<=precedence(stack[top]) && top>=0)
			{
				postfix[j]=pop();
				j++;
			}
```

```
                        push(infix[i]);
                }
        }
        while(stack[top]!='\0')
        {
                postfix[j]=pop();
                j++;
        }
        postfix[j]='\0';
        printf("Postfix expression :\n");
        printf("%s",postfix);
}
void push(char val)
{
        if(top==n-1)
                printf("Stack is full");
        else
        {
                top+=1;
                stack[top]=val;
        }
}
char pop()
{
        if(top==-1)
                printf("Stack is empty ");
        else
        {
                val=stack[top];
                top-=1;
                return val;
        }
}
int precedence(char c)
{
        if(c=='^')
                return 5;
        else if(c=='/')
                return 4;
        else if(c=='*')
                return 3;
        else if(c=='+')
                return 2;
        else if(c=='-')
```

```
            return 1;
        else
            return -1;
}
```

OUTPUT:



Lab program 3:

3a) 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 q[50],rear=-1,front=-1,max=50;
void enqueue();
void dequeue();
void display();
void main()
{
	int ch;
	printf("Press-1.insert, 2.delete, 3.Display and 4.Exit\n");
	while(ch!=4)
	{
		printf("Enter choice:");
		scanf("%d",&ch);
		switch(ch){
			case 1:
				enqueue();
				break;
			case 2:
				dequeue();
				break;
			case 3:
				display();
				break;
		}
	}
	printf("Exited");
}

void enqueue()
{
	int item;
	if(rear==max-1)
		printf("Queue overflow\n");
	else
	{	if(front==-1)
			front=0;
		printf("Insert an element:");
		scanf("%d",&item);
```

```
                rear+=1;
                q[rear]=item;
        }
}
void dequeue()
{
        if(front==-1||front>rear)
                printf("Queue underflow\n");
        else
        {
                printf("Deleted element is:%d\n",q[front]);
                front+=1;
        }
}
void display()
{
        int i;
        if(front==-1)
                printf("Queue is empty");
        else
        {
                printf("Queue is:\n");
                for(i=front;i<=rear;i++)
                        printf("%d\t",q[i]);
                printf("\n");
        }
}
```

OUTPUT:

```
Press-1.insert, 2.delete, 3.Display and 4.Exit
Enter choice:2
Queue underflow
Enter choice:1
Insert an element:6
Enter choice:1
Insert an element:8
Enter choice:1
Insert an element:5
Enter choice:2
Deleted element is:6
Enter choice:3
Queue is:
8       5
Enter choice:4
Exited
```

3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete and Display The program should print appropriate messages for queue empty and queue overflow conditions.

```
#include <stdio.h>
int q[50],front=-1,rear=-1,size;
void display();
void enqueue();
void dequeue();
void main()
{
	int ch;
	printf("Enter no. of elements:");
	scanf("%d",&size);
	while(ch!=4)
	{
		printf("1.Insert  2.Delete  3.Display  4.Exit\n");
		printf("Enter your choice:");
		scanf("%d",&ch);
		switch (ch)
		{
			case 1:
				enqueue();
				break;
			case 2:
				dequeue();
				break;
			case 3:
				display();
				break;
		}
	}
	printf("Exited");
}
void enqueue()
{
	int item;
	if((front == rear+1)||(front==0 && rear==size-1))
		printf("Queue is full\n");
	else
	{
		if(front == -1)
			front=0;
```

```
                printf("Enter element:");
                scanf("%d",&item);
                rear=(rear + 1)%size;
                q[rear] = item;
        }
}
void dequeue()
{
        int ele;
        if(front==-1)
                printf("Queue is empty\n");
        else
        {
                ele = q[front];
                if(front==rear)
                {
                        front = -1;
                        rear = -1;
                }
                else
                        front = (front+1) % size;
                printf("Deleted element = %d\n",ele);
        }
}
void display()
{
        int i;
        if(front == -1)
                printf("Queue is empty");
        else
        {
                printf("Front = %d\t",front);
                printf("Rear = %d\n",rear);
                printf("Queue:");
                for(i=front;i!=rear;i=(i+1)%size)
                        printf("%d",q[i]);
                printf("%d\n",q[i]);
        }
}
```

OUTPUT:

```
Enter no. of elements:4
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:1
Enter element:1
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:1
Enter element:2
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:1
Enter element:3
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:1
Enter element:4
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:1
Queue is full
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:2
Deleted element = 1
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:1
Enter element:5
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:3
Front = 1       Rear = 0
Queue:2345
1.Insert  2.Delete  3.Display  4.Exit
Enter your choice:4
Exited
```

Lab program 4:

WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.

```
#include <stdlib.h>
#include <stdio.h>
void push();
void append();
void display();
void insert_at_pos();
struct node
{
        int data;
        struct node *next;
};
struct node *head=NULL;
void main()
{
        printf("1.Insert from beginning\n2.Insert at end\n3.Insert at  particular position \n 4.
        Display \n5.Exit\n");
        int ch;
        while(ch!=6)
        {
                printf("Enter choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                push();
                                break;
                        case 2:
                                append();
                                break;
                        case 3:
                                insert_at_pos();
                                break;
                        case 4:
                                display();
                                break;
                        case 5:exit(0);
                        default: printf("Invalid choise");
                                break;
```

```
                }
        }
}
void push()
{
        int data;
        struct node *new_node;
        new_node=(struct node*)malloc(sizeof(struct node));
        printf("Enter the data to be inserted\n");
        scanf("%d",&data);
        new_node->data=data;
        new_node->next=head;
        head = new_node;
}
void append()
{
        int data;
        struct node *last=head;
        struct node *new_node;
        new_node=(struct node*)malloc(sizeof(struct node));
        printf("Enter the data\n");
        scanf("%d",&data);
        new_node -> data =data;
        new_node->next=NULL;
        if(head==NULL)
        {
                head=new_node;
                return;
        }
        while(last->next!=NULL)
        {
                last=last->next;
        }
        last-> next =new_node;
}
void insert_at_pos()
{
        int data;
        int pos;
        struct node *temp=head;
        struct node *new_node;
        new_node = (struct node*) malloc(sizeof(struct node));
        printf("Enter the data to be inserted\n");
        scanf("%d",&data);
```

```
        new_node->data=data;
        printf("enter the position\n");
        scanf("%d",&pos);
        new_node -> next = NULL;
        if(pos==0)
        {
                new_node->next=head;
                head = new_node;
        }
        else
        {
                for(int i=1 ; i<pos-1 ; i++)
                {
                        temp=temp->next;
                }
                struct node *temp1=temp->next;
                new_node->next= temp1;
                temp->next = new_node;
        }
}
void display()
{
        struct node *p= head;
        printf("List:\n");
        while(p != NULL)
        {
                printf("%d ->",p->data);
                p=p->next;
        }
        printf("NULL\n");
}
```

OUTPUT:

```
1.Insert from beginning
2.Insert at end
3.Insert at particular position
4.Display
5.Exit
Enter choice:1
Enter the data to be inserted
1
Enter choice:2
Enter the data
9
Enter choice:1
Enter the data to be inserted
2
Enter choice:2
Enter the data
8
Enter choice:3
Enter the data to be inserted
5
enter the position
3
Enter choice:4
List:
2 ->1 ->5 ->9 ->8 ->NULL
Enter choice:5
```

LEETCODE 1: MINSTACK

```
typedef struct {
    int size;
    int top;
    int *s;
    int *minstack;
} MinStack;

MinStack* minStackCreate() {
    MinStack *st=(MinStack*) malloc(sizeof(MinStack));
    if(st==NULL)
    {
        exit(0);
    }
    st->size=1000;
    st->top=-1;
    st->s=(int*) malloc (st->size*sizeof(int));
    st->minstack = (int*) malloc (st->size * sizeof(int));
    if(st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
```

```
        exit(0);
    }
    return st;
}

void minStackPush(MinStack* obj, int val) {
    if(obj->top==obj->size-1)
        printf("stack is overflow");
    else{
        obj->top++;
        obj->s[obj->top]=val;
        if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
            obj->minstack[obj->top] = val;
        } else {
            obj->minstack[obj->top] = obj->minstack[obj->top - 1];
        }
    }
}

void minStackPop(MinStack* obj) {
    int value;
    if(obj->top==-1)
        printf("underflow");
    else
    {
        value=obj->s[obj->top];
        obj->top--;
    }
}

int minStackTop(MinStack* obj) {
    int value=-1;
    if(obj->top==-1)
    {
        exit(0);
    }
    else
    {
        value=obj->s[obj->top];
        return value;
    }
}

int minStackGetMin(MinStack* obj) {
    if(obj->top==-1)
    {
        exit(0);
    }
    else
    {
```

```
        return obj->minstack[obj->top];
    }
}

void minStackFree(MinStack* obj) {
    free(obj->s);
    free(obj->minstack);
    free(obj);
}
/**
 * Your MinStack struct will be instantiated and called as such:
 * MinStack* obj = minStackCreate();
 * minStackPush(obj, val);
 * minStackPop(obj);
 * int param_3 = minStackTop(obj);

 * int param_4 = minStackGetMin(obj);
 * minStackFree(obj);
*/
```

OUTPUT:

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>
void pop();
void end_delete();
void delete_at_pos();
void display();
void append();
struct node
{
        int data;
        struct node *next;
};
struct node *head = NULL;
void main()
{
        printf("Insert the elements in list\n");
        append();
        printf("1.Delete from beginning\n2.Delete at end\n3.Delete at particular
position\n4.Display\n5.Exit\n");
        int ch;
        while(ch!=5)
        {
                printf("Enter choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                pop();
                                break;
                        case 2:
                                end_delete();
                                break;
                        case 3:
                                delete_at_pos();
                                break;
                        case 4:
                                display();
                                break;
```

```
                default: printf("Invalid choice");
                        break;
                }
        }
}
void append()
{
        int data,n;
        printf("Enter no. of nodes:");
         scanf("%d",&n);
        for(int i=0;i<n;i++)
        {
                struct node *last=head;
                struct node *new_node;
                new_node=(struct node*)malloc(sizeof(struct node));
                printf("Enter the data:");
                scanf("%d",&data);
                new_node -> data=data;
                new_node -> next =NULL;
                if(head==NULL)
                        head = new_node;
                else
                {
                        while(last->next!=NULL)
                        {
                                last=last->next;
                        last-> next =new_node;
                        }
                }
        }
}
void pop()
{
        struct node *ptr;
        if(head == NULL)
                printf("List is empty\n");
        else
        {
                ptr=head;
                head = ptr->next;
                free(ptr);
                printf("Node deleted from beginning\n");
        }
}
```

```
void end_delete()
{
	struct node *ptr;
	struct node *ptr1;
	if(head == NULL)
		printf("List is empty\n");
	else if(head -> next ==NULL)
	{
		free(head);
		head = NULL;
	}
	else
	{
		ptr = head;
		ptr1 = head;
		while(ptr -> next !=NULL)
		{
			ptr1 = ptr;
			ptr = ptr->next;
		}
		ptr1 -> next =NULL;
		free(ptr);
		printf("Node deleted from end\n");
	}
}
void delete_at_pos()
{
	struct node *ptr;
	struct node *ptr1;
	int pos;
	printf("Enter the position of deletion:\n");
	scanf("%d",&pos);
	ptr = head;
	for(int i=0;i<pos-1;i++)
	{
		ptr1 = ptr;
		ptr = ptr->next;
		if(ptr == NULL)
		{
			printf("There are less elements in the list\n");
			return;
		}
	}
	ptr1->next = ptr->next;
```

```
        free(ptr);
        printf("Node deleted from position %d\n",pos);
}
void display()
{
        struct node *p= head;
        printf("List:\n");
        while(p != NULL)
        {
                printf("%d ->",p->data);
                p=p->next;
        }
        printf("NULL\n");
}
```

OUTPUT:

```
Insert the elements in list
Enter no. of nodes:5
Enter the data:1
Enter the data:2
Enter the data:3
Enter the data:4
Enter the data:5
1.Delete from beginning
2.Delete at end
3.Delete at particular position
4.Display
5.Exit
Enter choice:1
Node deleted from beginning
Enter choice:2
Node deleted from end
Enter choice:4
List:
2 ->3 ->4 ->NULL
Enter choice:3
Enter the position of deleteion:
3
Node deleted from position 3
Enter choice:4
List:
2 ->3 ->NULL
Enter choice:5
Invalid choice
```

LEETCODE 2: Reverse Linked List

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
    if (head == NULL || left == right) {
        return head;
    }
    struct ListNode* dummy = (struct ListNode*)malloc(sizeof(struct ListNode));
    dummy->next = head;
    struct ListNode* preLeft = dummy;
    for (int i = 1; i < left; i++) {
        preLeft = preLeft->next;
    }
    struct ListNode* current = preLeft->next;
    struct ListNode* prev = NULL;
    struct ListNode* next = NULL;
    for (int i = 0; i <= right - left; i++) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    preLeft->next->next = current;
    preLeft->next = prev;
    if (left == 1) {
        head = dummy->next;
    }
    free(dummy);
    return head;
}
```

OUTPUT:

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

```
[1,2,3,4,5]
```

left =

```
2
```

right =

```
4
```

Output

```
[1,4,3,2,5]
```

Expected

```
[1,4,3,2,5]
```

Lab program 6:

6a) 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 data;
	struct node *next;
};
struct node *head1=NULL;
struct node *head2=NULL;
void insert1()
{
	int n,i,data;
	printf("Enter the no of nodes:");
	scanf("%d",&n);
	printf("Enter the data:\n");
	for(i=0;i<n;i++)
	{
		struct node *last1=head1;
		struct node *new_node1;
		new_node1=(struct node*)malloc(sizeof(struct node));
		scanf("%d",&data);
		new_node1->data=data;
		new_node1->next=NULL;
		if(head1==NULL)
		{
			head1=new_node1;
		}
		else
		{
			while(last1->next!=NULL)
			{
				last1=last1->next;
			}
			last1->next=new_node1;
		}
	}
}
void insert2()
{
```

```
	int n,i,data;
	printf("Enter the number of nodes:");
	scanf("%d",&n);
	printf("Enter the data:\n");
	for(i=0;i<n;i++)
	{
		struct node *last2=head2;
		struct node *new_node2;
		new_node2=(struct node*)malloc(sizeof(struct node));
		scanf("%d",&data);
		new_node2->data=data;
		new_node2->next=NULL;
		if(head2==NULL)
		{
			head2=new_node2;
		}
		else
		{
			while(last2->next!=NULL)
			{
				last2=last2->next;
			}
			last2->next=new_node2;
		}
	}
}
void sort()
{
	struct node *curr = head1;
	struct node *ptr = NULL;
	int temp;
	while(curr != NULL)
	{
		ptr = curr->next;
		while(ptr !=NULL)
		{
			if(curr->data > ptr->data)
			{
				temp = curr->data;
				curr->data=ptr->data;
				ptr->data=temp;
			}
			ptr = ptr->next;
		}
```

```
                curr = curr->next;
        }
        display();
}
void reverse()
{
        struct node* prev=NULL;
        struct node* ptr=NULL;
        while(head1!=NULL)
        {
                ptr=head1->next;
                head1->next=prev;
                prev=head1;
                head1=ptr;
        }
        head1=prev;
        display();
}
void concate()
{
        struct node*temp=head1;
        if(head1==NULL)
        {
                struct node *node=head2;
                while(node!=NULL)
                {
                        printf("%d->",node->data);
                        node=node->next;
                }
                printf("\n");
        }
        else
        {
                while(temp->next!=NULL)
                {
                        temp=temp->next;
                }
                temp->next=head2;
        }
        display();
}
void display()
{
        struct node *node=head1;
```

```
        if(head1==NULL)
        {
                printf("List is empty\n");
        }
        else
        {
                while(node!=NULL)
                {
                        printf("%d -> ",node->data);
                        node=node->next;
                }
                printf("NULL\n");
        }
}
void main()
{
        printf("Insert elemnts in first list:\n");
        insert1();
        printf("Sorted list:\n");
        sort();
        printf("Reversed list:\n");
        reverse();
        printf("Insert the elements in second list:\n");
        insert2();
        printf("Concatenated list:\n");
        concate();
}
```

OUTPUT:

```
Insert elemnts in first list:
Enter the no of nodes:5
Enter the data:
4 6 1 2 7
Sorted list:
1 -> 2 -> 4 -> 6 -> 7 -> NULL
Reversed list:
7 -> 6 -> 4 -> 2 -> 1 -> NULL
Insert the elements in second list:
Enter the number of nodes:3
Enter the data:
3 2 1
Concatenated list:
7 -> 6 -> 4 -> 2 -> 1 -> 3 -> 2 -> 1 -> NULL
```

6b) WAP to Implement Single Link List to simulate Stack and Queue operations.

STACK implementation

```
#include <stdio.h>
struct node
{
	int data;
	struct node *next;
};
struct node *head=NULL;
void push()
{
	int i,data;
	printf("Enter the data :");
	struct node *last=head;
	struct node *new_node;
	new_node=(struct node*)malloc(sizeof(struct node));
	scanf("%d",&data);
	new_node->data=data;
	new_node->next=NULL;
	if(head==NULL)
	{
		head=new_node;
	}
	else
	{
		while(last->next!=NULL)
		{
			last=last->next;
		}
		last->next=new_node;
	}
	printf("Element pushed successfully\n");
}
void pop()
{
	struct node*ptr;
	struct node*ptr1;
	if(head==NULL)
		printf("List is empty\n");
	else if(head->next==NULL)
		free(head);
	else
	{
```

```
                ptr=head;
                while(ptr->next!=NULL)
                {
                        ptr1=ptr;
                        ptr=ptr->next;
                }
                free(ptr);
                ptr1->next=NULL;
                printf("Element popped successfully\n");
        }
}
void display()
{
        struct node *node=head;
        if(head==NULL)
                printf("List is empty\n");
        else
        {
                printf("Stack :\n");
                while(node!=NULL)
                {
                        printf("%d -> ",node->data);
                        node=node->next;
                }
                printf("NULL\n");
        }
}
void main()
{
        int ch;
        printf("Enter 1:pop \n 2:push \n 3:display \n 4:exit\n");
        while(ch!=4)
        {
                printf("Enter the choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: pop();
                                break;
                        case 2:push();
                                break;
                        case 3:display();
                                break;
                }
```

```
        }
}
```

OUTPUT:

```
Enter 1:pop
 2:push
 3:display
 4:exit
Enter the choice:2
Enter the data :1
Element pushed successfully
Enter the choice:2
Enter the data :2
Element pushed successfully
Enter the choice:2
Enter the data :3
Element pushed successfully
Enter the choice:3
Stack :
1 -> 2 -> 3 -> NULL
Enter the choice:2
Enter the data :4
Element pushed successfully
Enter the choice:1
Element popped successfully
Enter the choice:3
Stack :
1 -> 2 -> 3 -> NULL
Enter the choice:4
```

Queue Implementation

```
#include <stdio.h>
struct node
{
        int data;
        struct node *next;
};
struct node *head=NULL;
void enqueue()
{
        int i,data;
        printf("Enter the data :");
        struct node *last=head;
        struct node *new_node;
        new_node=(struct node*)malloc(sizeof(struct node));
        scanf("%d",&data);
        new_node->data=data;
        new_node->next=NULL;
```

```
        if(head==NULL)
        {
                head=new_node;
        }
        else
        {
                while(last->next!=NULL)
                {
                        last=last->next;
                }
                last->next=new_node;
        }
}
void dequeue()
{
        struct node *ptr;
        if(head == NULL)
                printf("List is empty\n");
        else
        {
                ptr=head;
                head = ptr->next;
                free(ptr);
                printf("Node deleted from beginning\n");
        }
}
void display()
{
        struct node *node=head;
        if(head==NULL)
                printf("List is empty\n");
        else
        {
                printf("Queue:\n");
                while(node!=NULL)
                {
                        printf("%d -> ",node->data);
                        node=node->next;
                }
                printf("NULL\n");
        }
}
void main()
{
```

```
        int ch;
        printf("Enter 1:Enqueue \n 2:Dequeue \n 3:Display \n 4:exit\n");
        while(ch!=4)
        {
                printf("Enter the choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:enqueue();
                                break;
                        case 2:dequeue();
                                break;
                        case 3:display();
                                break;
                }
        }
}
```

OUTPUT:

```
Enter 1:Enqueue
 2:Dequeue
 3:Display
 4:exit
Enter the choice:1
Enter the data :1
Enter the choice:1
Enter the data :2
Enter the choice:1
Enter the data :3
Enter the choice:1
Enter the data :4
Enter the choice:3
Queue:
1 -> 2 -> 3 -> 4 -> NULL
Enter the choice:2
Node deleted from beginning
Enter the choice:3
Queue:
2 -> 3 -> 4 -> NULL
Enter the choice:4
```

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 *prev;
        struct node *next;
};
struct node *head = NULL;
void insert();
void delete();
void display();
void main()
{
        int ch;
        printf("Press - 1.Insert\n 2.Delete\n 3.Display\n");
        while(ch!=4)
        {
                printf("Enter choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case1:insert();
                                break;
                        case 2:delete();
                                break;
                        case 3:display();
                                break;
                }
        }
}
void insert()
{
        int pos,i=1;
        struct node *new = (struct node*)malloc(sizeof(struct node));
        struct node *ptr=head;
        printf("Enter data:");
        scanf("%d",&new->data);
```

```
        if(head == NULL)
        {
                new -> prev = NULL;
                new -> next = NULL;
                head = new;
                printf("Node inserted\n");
                return;
        }
        printf("Enter the position:");
        scanf("%d",&pos);
        if(pos==1)
        {
                new -> prev =NULL;
                new->next = head;
                head -> prev =new;
                head=new;
                printf("Node inserted\n");
        }
        else{
                for(i=0;i<pos-1;i++)
                {
                        ptr = ptr->next;
                }
                new->prev = ptr->prev;
                new->next = ptr;
                ptr->prev->next=new;
                ptr->prev = new;
                printf("Node inserted\n");
        }
}
void delete()
{
        int val;
        printf("Enter the value:");
        scanf("%d",&val);
        struct node *ptr=head;
        if(head->data == val)
        {
                head = ptr->next;
                free(ptr);
                printf("Node deleted\n");
                return;
        }
        while(ptr->data != val)
```

```
        {
                ptr = ptr->next;
                if(ptr->next == NULL)
                {
                        ptr->prev->next = NULL;
                        free(ptr);
                        printf("Node deleted\n");
                        return;
                }
        }
        ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
        free(ptr);
        printf("Node deleted\n");
}
void display()
{
        struct node *p=head;
        while(p != NULL)
        {
                printf("%d -> ",p->data);
                p=p->next;
        }
        printf("NULL\n");
}
```

OUTPUT:

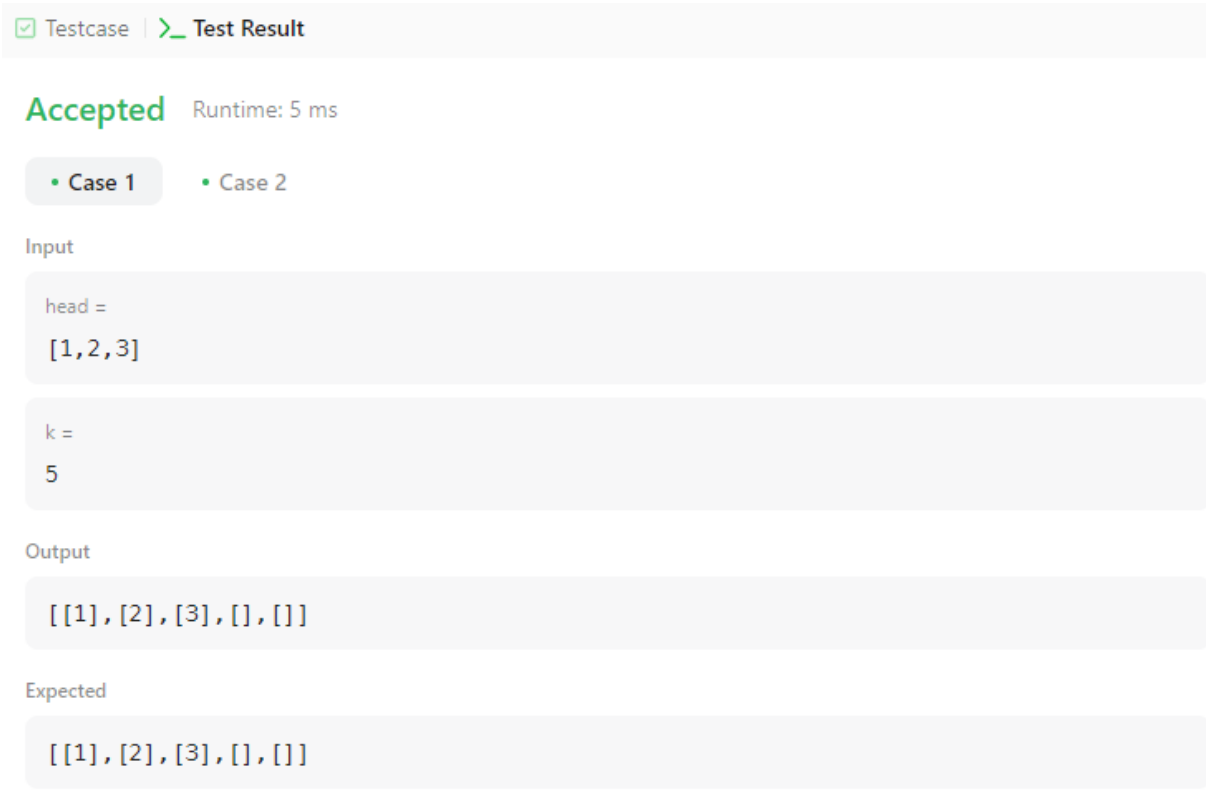
```
Press - 1.Insert
 2.Delete
 3.Display
Enter choice:1
Enter data:4
Node inserted
Enter choice:1
Enter data:5
Enter the position:1
Node inserted
Enter choice:3
5 -> 4 -> NULL
Enter choice:1
Enter data:6
Enter the position:2
Node inserted
Enter choice:3
5 -> 6 -> 4 -> NULL
Enter choice:2
Enter the value:5
Node deleted
Enter choice:3
6 -> 4 -> NULL
Enter choice:4
```

LEETCODE 3: Split list into parts

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
int findLength(struct ListNode *head) {
    int length = 0;
    while (head != NULL) {
        length++;
        head = head->next;
    }
    return length;
}
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
    int length = findLength(head);
    int partSize = length / k;
    int extraNodes = length % k;
    struct ListNode **result = (struct ListNode **)malloc(k * sizeof(struct ListNode *));
    for (int i = 0; i < k; i++) {
        result[i] = head;
        int currentPartSize = partSize + (extraNodes-- > 0 ? 1 : 0);
        for (int j = 0; j < currentPartSize - 1; j++) {
            if (head != NULL) {
                head = head->next;
            }
        }
```

```
        if (head != NULL) {
            struct ListNode *temp = head;
            head = head->next;
            temp->next = NULL;
        }
    }
    *returnSize = k;
    return result;
}
```

OUTPUT:



Lab program 8:
Write a program
a) To construct a binary Search tree.
b) To traverse the tree using all the methods i.e., in-order, 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;
        struct node*right;
};
struct node*root=NULL;
void create()
{
        struct node*new_node = (struct node*)malloc(sizeof(struct node));
        int val;
        printf("Enter data:");
        scanf("%d",&val);
        new_node->data=val;
        new_node->left=NULL;
        new_node->right=NULL;
        if (root==NULL)
        {
                root=new_node;
        }
        else
        {
                struct node*ptr=root;
                struct node*ptr1=NULL;
                while(ptr!=NULL)
                {
                        ptr1=ptr;
                        if(val<ptr->data)
                        {
                                ptr=ptr->left;
                        }
                        else
                        {
                                ptr=ptr->right;
                        }
                }
```

```
                if(val<ptr1->data)
                        ptr1->left=new_node;
                else
                        ptr1->right=new_node;
        }
}
void postorder(struct node* ptr)
{
        if(ptr!=NULL)
        {
                postorder(ptr->left);
                postorder(ptr->right);
                printf("%d ",ptr->data);
        }
}
void preorder(struct node* ptr)
{
        if(ptr!=NULL)
        {
                printf("%d ",ptr->data);
                preorder(ptr->left);
                preorder(ptr->right);
        }
}
void inorder(struct node* ptr)
{
        if(ptr!=NULL)
        {
                inorder(ptr->left);
                printf("%d ",ptr->data);
                inorder(ptr->right);
        }
}
void main()
{
        int n;
        printf("Enter no of nodes to be inserted:");
        scanf("%d",&n);
        for(int i=0;i<n;i++)
        {
                create();
        }
        printf("\nInorder:\n");
        inorder(root);
```

```
        printf("\nPreorder:\n");
        preorder(root);
        printf("\nPostorder:\n");
        postorder(root);
}
```

OUTPUT:

```
Enter no of nodes to be inserted:5
Enter data:5
Enter data:7
Enter data:1
Enter data:3
Enter data:9

Inorder:
1 3 5 7 9
Preorder:
5 1 3 7 9
Postorder:
3 1 9 7 5
```

LEETCODE 4: Rotate list

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* rotateRight(struct ListNode* head, int k) {
   struct ListNode *ptr=head;
    int count=1;
    if(head == NULL || head->next==NULL ||k==0)
        return head;
    else
    {
        while(ptr->next!=NULL)
        {
            ptr=ptr->next;
            count++;
        }
        if(k%count==0)
            return head;
        ptr->next=head;
        for(int i=k%count ; i<count ; i++)
        {
            ptr=ptr->next;
        }
        head=ptr->next;
        ptr->next=NULL;
```

```
        return head;
    }
}
```

OUTPUT:

Lab program 9:
9a) Write a program to traverse a graph using BFS method.

```
#include <stdio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
	for(i=1; i<=n; i++)
	{
		if(a[v][i] && !visited[i])
			q[++r]=i;
	}
	if(f<=r)
	{
		visited[q[f]]=1;
		bfs(q[f++]);
	}
}
int main()
{
	int v;
	printf("\n Enter the number of vertices:");
	scanf("%d", &n);
	for(i=1; i<=n; i++)
	{
		q[i]=0;
		visited[i]=0;
	}
	printf("Enter graph data in matrix form:\n");
	for(i=1; i<=n; i++)
	{	for(j=1; j<=n; j++)
		{
			scanf("%d", &a[i][j]);
		}
	}
	printf("\n Enter the starting vertex:");
	scanf("%d", &v);
	bfs(v);
	printf("The nodes which are reachable are:\n");
	for(i=1; i<=n; i++)
	{
		if(visited[i])
		printf("%d\t", i);
	}
	return 0;
}
```

OUTPUT:

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

 Enter the starting vertex:1
The nodes which are reachable are:
1   2   4
```

9b) Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>
int a[20][20], s[20], n;
void dfs(int v)
{
        int i;
        s[v]=1;
        for(i=1; i<=n; i++)
        {
                if(a[v][i] && !s[i])
                {
                        printf("\n %d->%d",v,i);
                        dfs(i);
                }
        }
}
int main()
{
        int i, j, count=0;
        printf("\n Enter number of vertices:");
        scanf("%d", &n);
        for(i=1; i<=n; i++)
        {
                s[i]=0;
                for(j=1; j<=n; j++)
                a[i][j]=0;
        }
        printf("Enter the adjacency matrix:\n");
        for(i=1; i<=n; i++)
        {
                for(j=1; j<=n; j++)
                scanf("%d", &a[i][j]);
        }
```

```
        dfs(1);
        printf("\n");
        for(i=1; i<=n; i++)
        {
                if(s[i])
                        count++;
        }
        if(count==n)
                printf("Graph is connected");
        else
                printf("Graph is not connected");
        return 0;
}
```

OUTPUT:

```
 Enter number of vertices:4
Enter the adjacency matrix:
0 1 0 1
1 0 0 1
0 0 0 0
1 1 0 0

 1->2
 2->4
Graph is not connected
```

HACKER RANK: Swap nodes

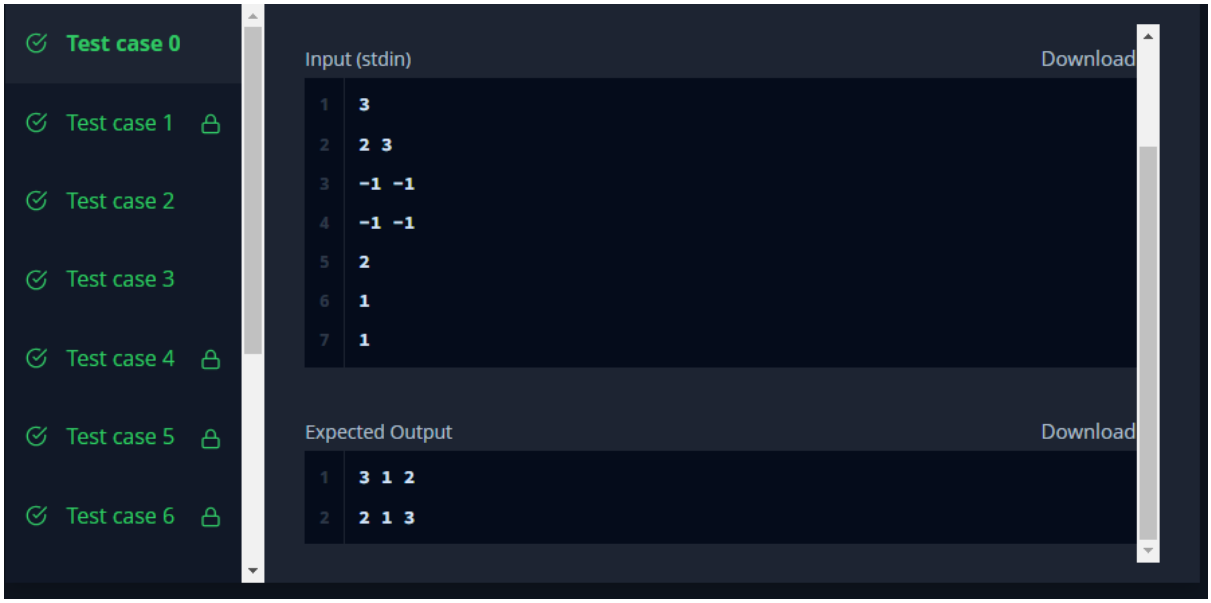
```
typedef struct TreeNode {
        int data;
        struct TreeNode* left;
        struct TreeNode* right;
} TreeNode;
void inOrderTraversal(TreeNode* root, int* result, int* index) {
        if (root == NULL)
                return;
        inOrderTraversal(root->left, result, index);
        result[(*index)++] = root->data;
        inOrderTraversal(root->right, result, index);
}
void swapSubtrees(TreeNode* root, int k, int depth) {
        if (root == NULL)
                return;
        if (depth % k == 0) {
                TreeNode* temp = root->left;
```

```
            root->left = root->right;
            root->right = temp;
        }
        swapSubtrees(root->left, k, depth + 1);
        swapSubtrees(root->right, k, depth + 1);
}
TreeNode* buildTree(int indexes_rows, int indexes_columns, int** indexes)
{
        TreeNode* root = (TreeNode*)malloc(sizeof(TreeNode));
        root->data = 1;
        root->left = NULL;
        root->right = NULL;
        TreeNode* nodes[indexes_rows + 1];
        nodes[1] = root;
        for (int i = 0; i < indexes_rows; i++)
        {
                TreeNode* curr = nodes[i + 1];
                if (indexes[i][0] != -1)
                {
                        curr->left = (TreeNode*)malloc(sizeof(TreeNode));
                        curr->left->data = indexes[i][0];
                        curr->left->left = NULL;
                        curr->left->right = NULL;
                        nodes[indexes[i][0]] = curr->left;
                }
                if (indexes[i][1] != -1)
                {
                        curr->right = (TreeNode*)malloc(sizeof(TreeNode));
                        curr->right->data = indexes[i][1];
                        curr->right->left = NULL;
                        curr->right->right = NULL;
                        nodes[indexes[i][1]] = curr->right;
                }
        }
        return root;
}

int** swapNodes(int indexes_rows, int indexes_columns, int** indexes, int queries_count,
int* queries, int* result_rows, int* result_columns)
{
        int** result = (int**)malloc(queries_count * sizeof(int*));
        *result_rows = queries_count;
        *result_columns = indexes_rows;
        TreeNode* root = buildTree(indexes_rows, indexes_columns, indexes);
        for (int i = 0; i < queries_count; i++)
        {
                int k = queries[i];
                swapSubtrees(root, k, 1);
                int* traversal = (int*)malloc(indexes_rows * sizeof(int));
                int index = 0;
```

```
            inOrderTraversal(root, traversal, &index);
            result[i] = traversal;
        }
        return result;
}
```

OUTPUT:



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_MEMORY_LOCATIONS 10
int hashTable[MAX_MEMORY_LOCATIONS] = {0};
int status[MAX_MEMORY_LOCATIONS] = {0};
void initializeHashTable(int size)
{
	for (int i=0; i < size; i++) {
		status[i]=0;
	}
}
int hashFunction(int key, int size) {
	return key%size;
}
int linearProbe(int hashValue, int attempt, int size)
{
	return ((hashValue + attempt)%size);
}
void insertEmployee(int size, int key)
{
	int hashValue = hashFunction (key, size);
	int index = hashValue;
	int attempt=1;
	while (status[index] == 1)
	{
		index = linearProbe(hashValue, attempt ,size);
		attempt ++;
	}
	hashTable[index] = key;
	status[index] = 1;
}
void displayHashTable(int size)
{
	printf("\n Hash Table: \n");
	printf("Index\tKey\n");
	for(int i=0;i<size;i++)
	{
		printf("%d\t",i);
		if(status[i]==1)
			printf("%d\n",hashTable[i]);
```

```
                else
                        printf("Empty\n");
        }
}
int main()
{
        int m=MAX_MEMORY_LOCATIONS;
        initializeHashTable(m);
        int n;
        printf("Enter the number of employees:");
        scanf("%d",&n);
        int key[n];
        printf("Enter the employee keys:\n");
        for(int i=0;i<n;i++)
        {
                printf("Employee %d: ",i+1);
                scanf("%d",&key[i]);
        }
        for(int i=0;i<n;i++)
        {
                insertEmployee(m,key[i]);
        }
        displayHashTable(m);
        return 0;
}
```

OUTPUT:

```
Enter the number of employees:6
Enter the employee keys:
Employee 1: 85
Employee 2: 56
Employee 3: 23
Employee 4: 35
Employee 5: 43
Employee 6: 11

 Hash Table:
Index   Key
0   Empty
1   11
2   Empty
3   23
4   43
5   85
6   56
7   35
8   Empty
9   Empty
```