VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Done by

Pratham Ganapathy
1BM22CS206

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 Pratham Ganapathy **(1BM22CS206)**, 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 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.

Dr. Sneha S Bagalkot	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	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.

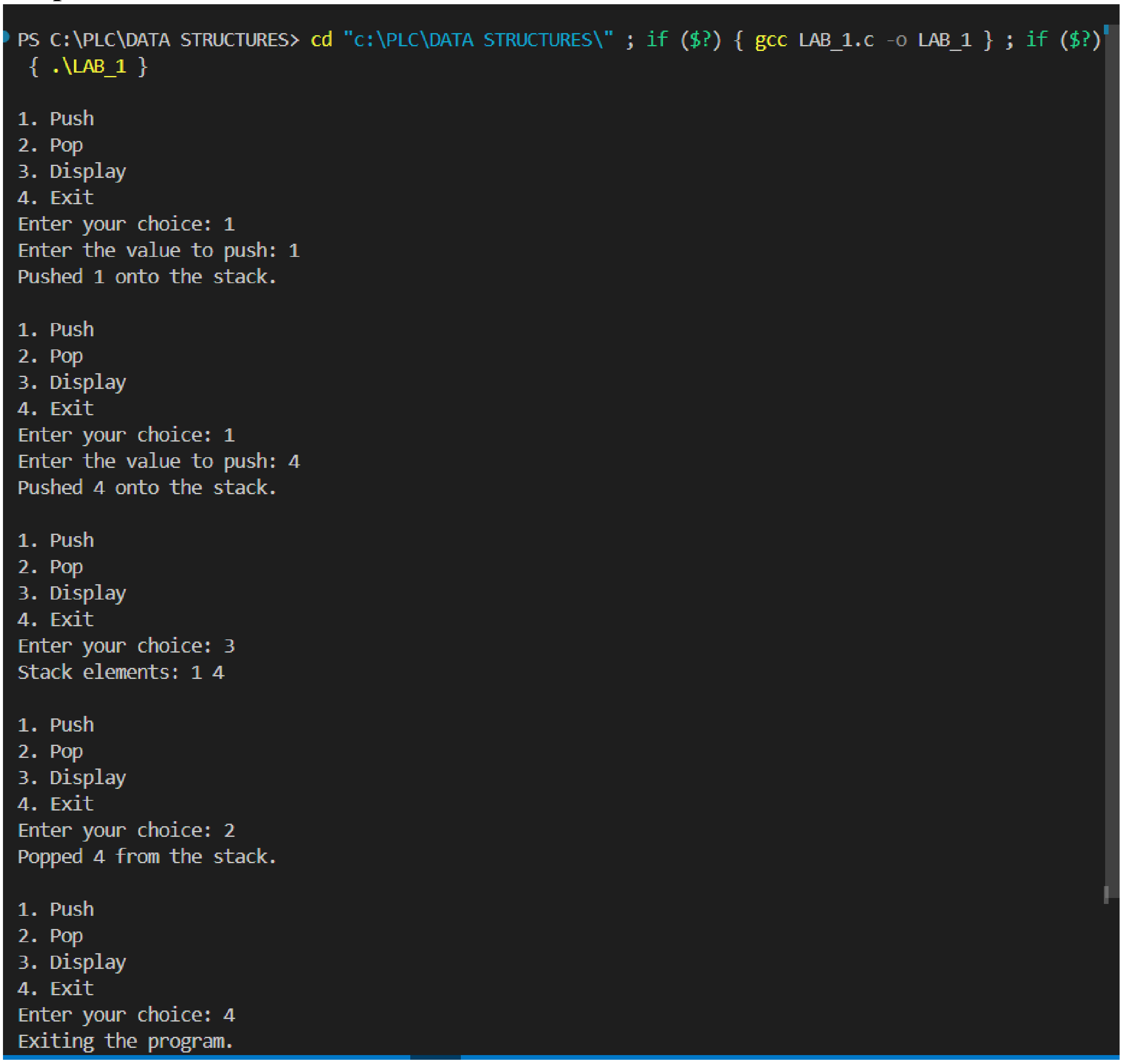
Lab program 1:

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

#define MAX_SIZE 5

// Structure to represent the stack
struct Stack {
   int arr[MAX_SIZE];
   int top;
};
struct Stack stack={{0,0,0,0,0},-1};
// Function to initialize the stack



// Function to check if the stack is empty
int isEmpty() {
   if(stack.top ==-1){
      return 1;
   }
   else{
      return 0;
   }
}

// Function to check if the stack is full
int isFull() {
   return stack.top == MAX_SIZE - 1;
}

// Function to push an element onto the stack
void push(int value) {
   if (isFull(stack)) {
      printf("Stack Overflow! Cannot push %d.\n", value);
   } else {
      stack.arr[++(stack.top)] = value;
      printf("Pushed %d onto the stack.\n", value);
   }

}

// Function to pop an element from the stack
```

```
void pop() {
    if (isEmpty(stack)) {
        printf("Stack Underflow! Cannot pop from an empty stack.\n");
    } else {
        printf("Popped %d from the stack.\n", stack.arr[stack.top--]);
    }

}

// Function to display the elements of the stack
void display() {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements: ");
        for (int i = 0; i <= stack.top; i++) {
            printf("%d ", stack.arr[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Stack stack;


    int choice, value;

    do {
        printf("\n1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
               push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
      }
  } while (choice != 4);
  getchar();
  return 0;
}
```

Output:



```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc LAB_1.c -o LAB_1 } ; if ($?)
 { .\LAB_1 }

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 1
Pushed 1 onto the stack.

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 4
Pushed 4 onto the stack.

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 1 4

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 4 from the stack.

1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

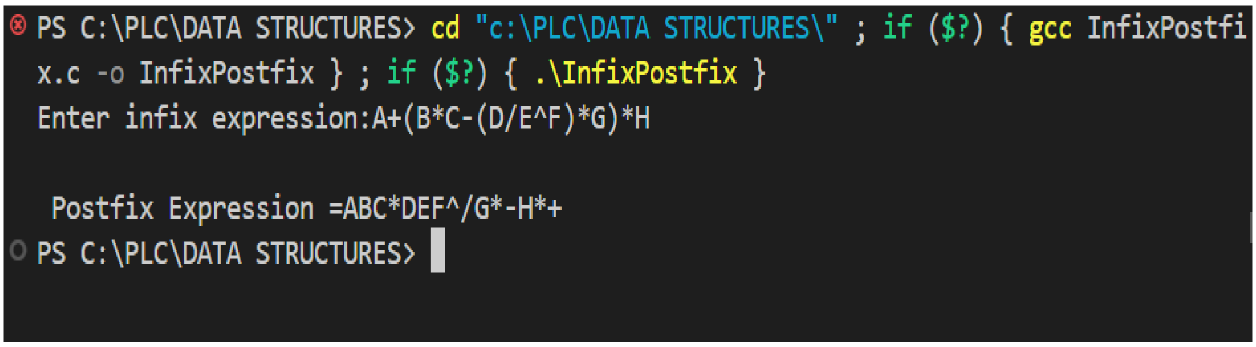
Lab Program 2:

Write a program 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), / (divide) and ^ (power).

```
#include<stdio.h>
#include<ctype.h>
#define SIZE 50
char stack[SIZE];
int top=-1;
void push(char x){
   top++;
   stack[top]=x;
}
char pop(){
   char x;
   return(stack[top--]);
}
int pr(char symbol){
   if(symbol=='^'){
      return (3);
   }
   else if(symbol=='*' || symbol=='/'){
      return(2);
   }
   else if(symbol=='+' || symbol=='-'){
      return(1);
   }
   else{
      return(0);
   }
}
void main(){
   char infix[50],postfix[50],ch,elem;
   int i=0,k=0;
   printf("Enter infix expression:");
   scanf("%s",infix);
   push('#');
   while((ch=infix[i++])!='\0'){
      if(ch=='(')
      push (ch);
      else
      if(isalnum(ch))
      postfix[k++]=ch;
      else
      if(ch==')')
      {
         while(stack[top]!='('){
            postfix[k++]=pop();
         }
```

```
            elem=pop();
        }
        else{
            while(pr(stack[top])>=pr(ch)){
                postfix[k++]=pop();
            }
            push(ch);
        }
    }
    while(stack[top]!='#'){
        postfix[k++]=pop();
    }
    postfix[k]='\0';
    printf("\n Postfix Expression =%s \n",postfix);

}
```

Output:



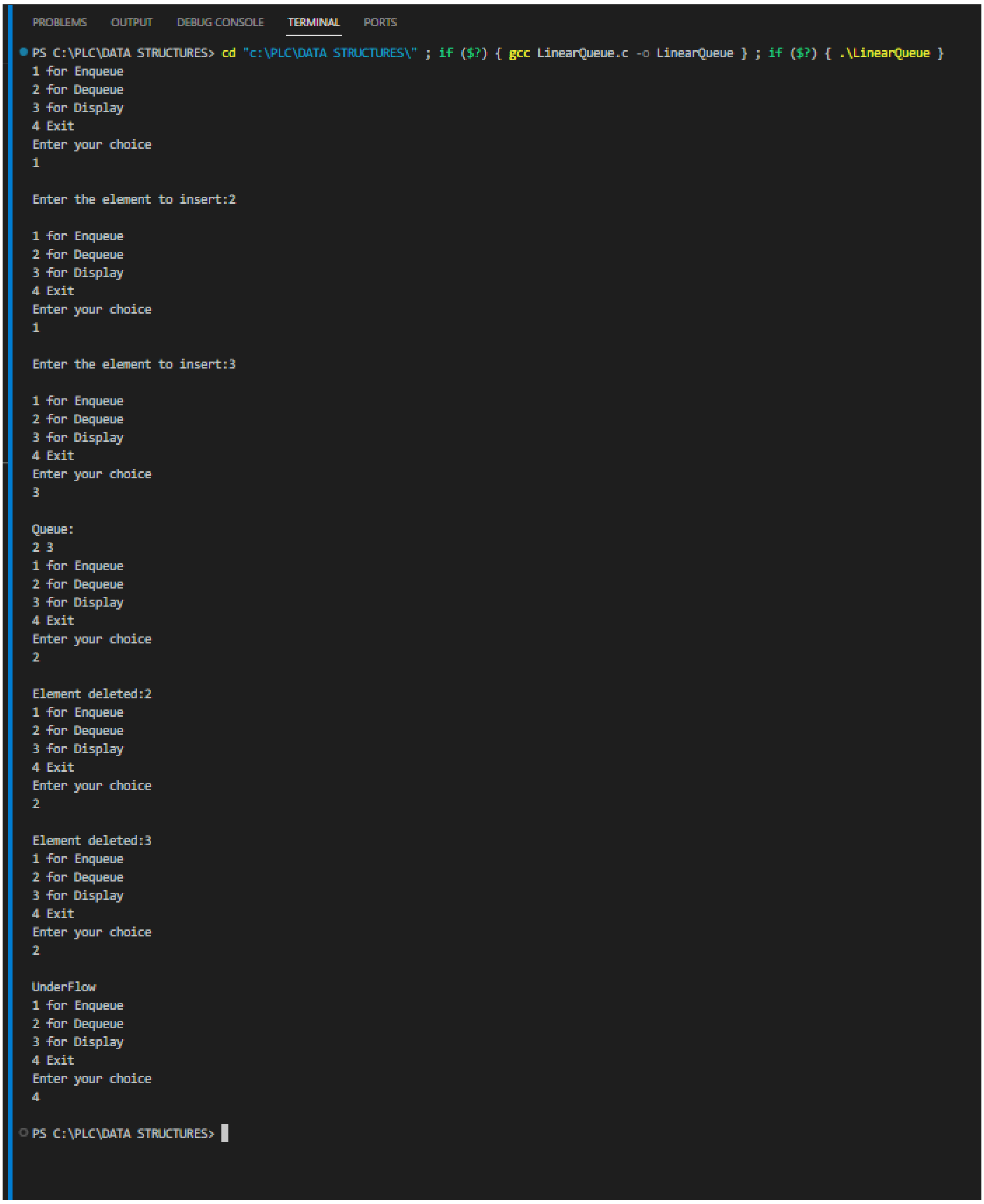
```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc InfixPostfi
x.c -o InfixPostfix } ; if ($?) { .\InfixPostfix }
Enter infix expression:A+(B*C-(D/E^F)*G)*H

 Postfix Expression =ABC*DEF^/G*-H*+
PS C:\PLC\DATA STRUCTURES>
```

Lab Progarm 3:

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

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 5
void enqueue();
void dequeue();
void show();
struct queue{
   int arr[SIZE];
   int top;
   int rear;
};
struct queue q={{0,0,0,0,0},-1,-1};
void enqueue(){
   int item;
   if(q.rear == SIZE-1){
      printf("OverFlow \n");
   }
   else{
      if(q.top ==-1 || q.top>=0){
         q.top=0;
         printf("Enter the element to insert:");
         scanf("%d",&item);
         printf("\n");
         q.rear+=1;
         q.arr[q.rear]=item;
      }
   }
}
void dequeue(){
   if(q.top==-1 || q.top>q.rear){
      printf("UnderFlow \n");
      return;
   }
   else{
      printf("Element deleted:%d \n",q.arr[q.top]);
      q.top=q.top+1;
   }
}
void show(){
   if(q.top == -1){
        printf("Empty Queue \n");
    }
    else{
        printf("Queue: \n");
        for(int i=q.top;i<=q.rear;i++){
            printf("%d ",q.arr[i]);
        }
        printf("\n");
    }
}
int main(){
    int ch;
    while(1){
        printf("1 for Enqueue \n");
        printf("2 for Dequeue \n");
        printf("3 for Display \n");
        printf("4 Exit \n");
        printf("Enter your choice \n");
        scanf("%d",&ch);
        printf("\n");
        switch(ch){
            case 1:
            enqueue();
            break;
            case 2:
            dequeue();
            break;
            case 3:
            show();
            break;
            case 4:
            exit(0);
            default:
            printf("Wrong Choice \n");
            break;
        }
    }
}
```

Output:



```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc LinearQueue.c -o LinearQueue } ; if ($?) { .\LinearQueue }
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
1

Enter the element to insert:2

1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
1

Enter the element to insert:3

1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
3

Queue:
2 3
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
2

Element deleted:2
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
2

Element deleted:3
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
2

UnderFlow
1 for Enqueue
2 for Dequeue
3 for Display
4 Exit
Enter your choice
4

PS C:\PLC\DATA STRUCTURES>
```

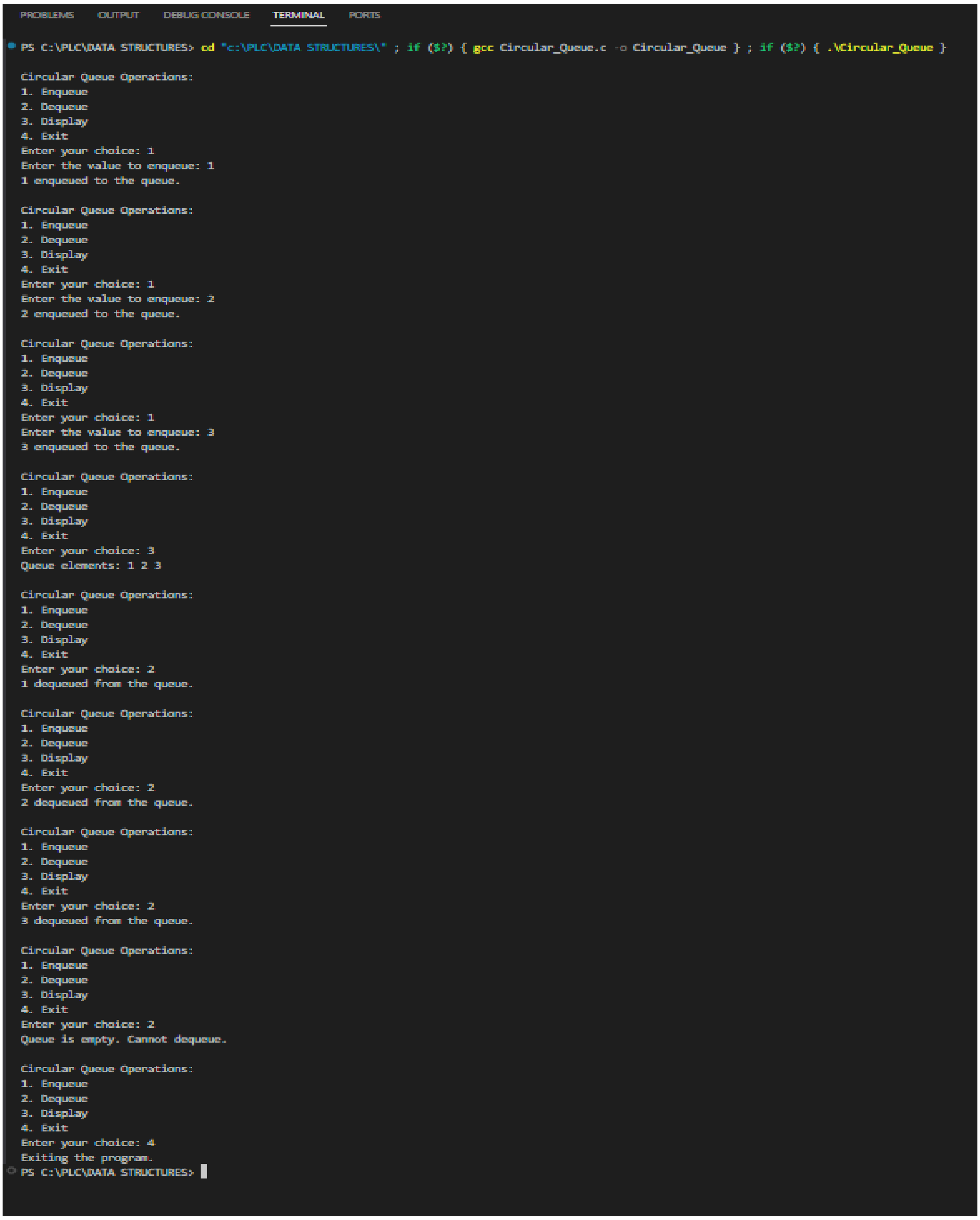
Lab Program 4:

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

```
#include <stdio.h>

#define MAX_SIZE 5

// Circular Queue variables
int items[MAX_SIZE];
int front = -1, rear = -1;

// Function to check if the queue is empty
int isEmpty() {
  return (front == -1 && rear == -1);
}

// Function to check if the queue is full
int isFull() {
  return ((rear + 1) % MAX_SIZE == front);
}

// Function to enqueue an element into the circular queue
void enqueue(int value) {
  if (isFull()) {
    printf("Queue is full. Cannot enqueue %d.\n", value);
    return;
  }

  if (isEmpty()) {
    front = 0;
    rear = 0;
  } else {
    rear = (rear + 1) % MAX_SIZE;
  }

  items[rear] = value;
  printf("%d enqueued to the queue.\n", value);
}

// Function to dequeue an element from the circular queue
int dequeue() {
  int dequeuedItem;

  if (isEmpty()) {
```

```
printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }

    dequeuedItem = items[front];

    if (front == rear) {
        // If there was only one element in the queue
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % MAX_SIZE;
    }

    printf("%d dequeued from the queue.\n", dequeuedItem);
    return dequeuedItem;
}

// Function to display the elements of the circular queue
void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements: ");
    int i = front;
    do {
        printf("%d ", items[i]);
        i = (i + 1) % MAX_SIZE;
    } while (i != (rear + 1) % MAX_SIZE);
    printf("\n");
}

int main() {
    int choice, value;

    do {
        printf("\nCircular Queue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;

            case 2:
                dequeue();
                break;

            case 3:
                display();
                break;

            case 4:
                printf("Exiting the program.\n");
                break;

            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }

    } while (choice != 4);

    return 0;
}
```

Output:



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Circular_Queue.c -o Circular_Queue } ; if ($?) { .\Circular_Queue }

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 1
1 enqueued to the queue.

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 2
2 enqueued to the queue.

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 3
3 enqueued to the queue.

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 1 2 3

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
1 dequeued from the queue.

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
2 dequeued from the queue.

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
3 dequeued from the queue.

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue is empty. Cannot dequeue.

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 5:

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

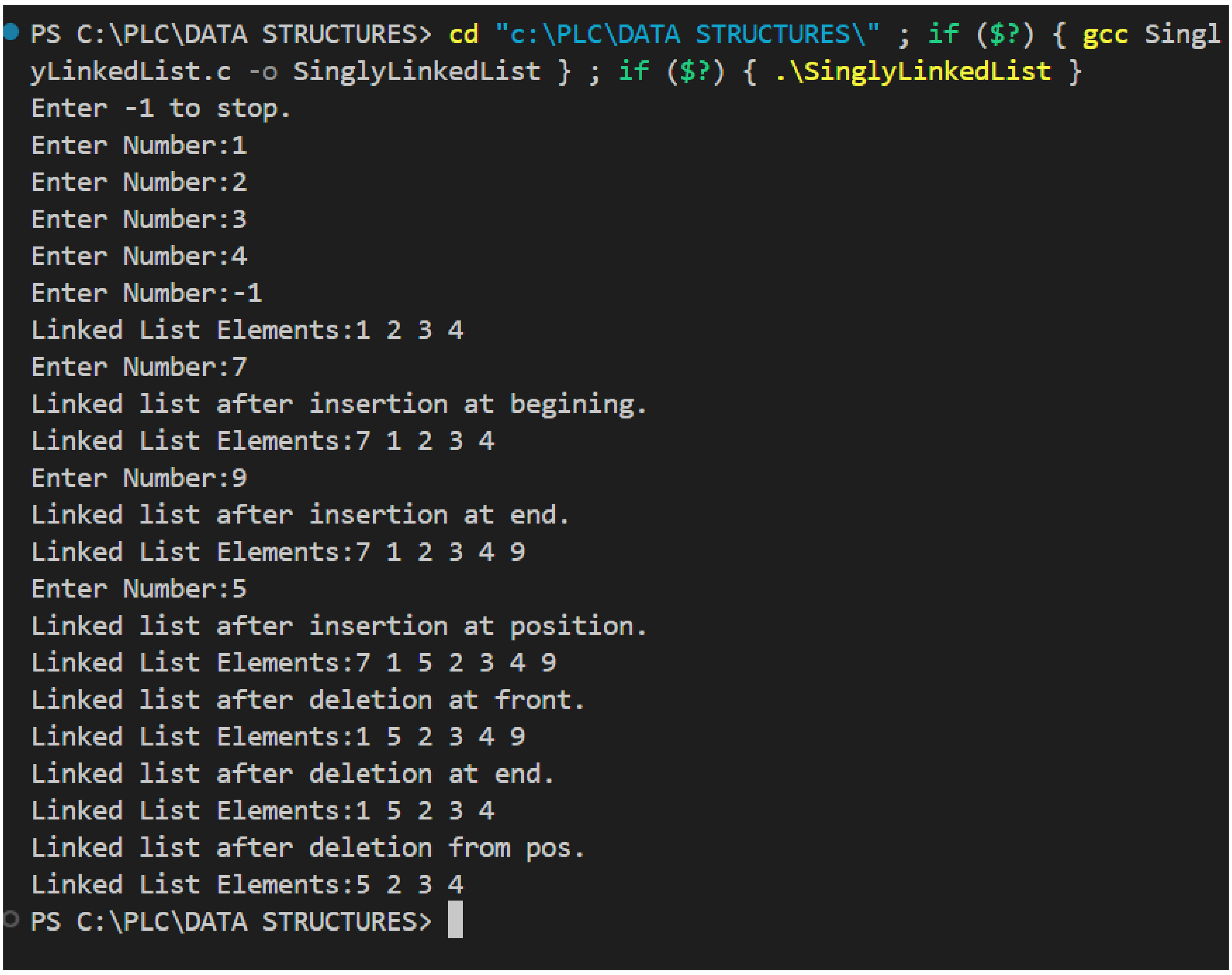
```
#include<stdio.h>
#include<stdlib.h>
struct Node{
   int data;
   struct Node *next;
};
//Create Linked List
struct Node* createLL(struct Node* head){
   int num;

   printf("Enter -1 to stop.\n");
   printf("Enter Number:");
   scanf("%d",&num);
   while(num!=-1){
      struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
      struct Node* p;
      p=head;
      if(head==NULL){
         newNode->data=num;
         newNode->next=NULL;
         head=newNode;
      }
      else{

         while(p->next!=NULL){
            p=p->next;
         }
         newNode->data=num;
         p->next=newNode;
         newNode->next=NULL;
      }
   printf("Enter Number:");
   scanf("%d",&num);
   }
   return head;
}
//Display Linked List
struct Node* displayLL(struct Node* head){
   struct Node*p;
   p=head;
   printf("Linked List Elements:");
   while(p !=NULL){
```

```
 printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
    return head;
}
//Insert a node at First Position
struct Node* insertAtBeg(struct Node* head){
    int num;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    newNode->data=num;
    newNode->next=head;
    head=newNode;
    return head;
}
//Insert a node at End Position
struct Node* insertAtEnd(struct Node* head){
    int num;
    struct Node *p;
    p=head;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    while(p->next!=NULL){
        p=p->next;
    }
    newNode->data=num;
    p->next=newNode;
    newNode->next=NULL;
    return head;
}
//Insert a node at any Position
struct Node* insertAtPos(struct Node* head,int pos){
    int num,i=0;
    struct Node *p;
    p=head;
    printf("Enter Number:");
    scanf("%d",&num);
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    while(i!=pos-1){
        p=p->next;
        i++;
    }
    newNode->data=num;
    newNode->next=p->next;
    p->next=newNode;
    return head;
}
//Delete a node at front
```

```
struct Node* delAtFront(struct Node* head){
   if(head==NULL){
      printf("Linked List alredy empty.\n");
      return head;
   }
   else{
   struct Node* p;
   p=head->next;
   free(head);
   head=p;
   return head;
   }
}
//Delete a node at end
struct Node* delAtEnd(struct Node* head){
   struct Node *p,*preNode;
   p=head;
   while(p->next!=NULL){
      preNode=p;
      p=p->next;
   }
   preNode->next=NULL;
   free(p);
   return head;
}
//Delete a node at any position
struct Node* delAtPos(struct Node* head, int pos){
   struct Node*p,*preNode;
   int i=0;
   p=head;
   if(pos==0){
      head=delAtFront(head);
      return head;
   }
   while(i!=pos){
      preNode=p;
      p=p->next;
      i++;
   }
   preNode->next=p->next;
   free(p);
   return head;
}
int main(){
   struct Node* head=NULL;
   head=createLL(head);
   head=displayLL(head);

   head=insertAtBeg(head);
   printf("Linked list after insertion at begining.\n");
   head=displayLL(head);
```

```
    head=insertAtEnd(head);
    printf("Linked list after insertion at end.\n");
    head=displayLL(head);

    head=insertAtPos(head,2);
    printf("Linked list after insertion at position.\n");
    head=displayLL(head);

    head=delAtFront(head);
    printf("Linked list after deletion at front.\n");
    head=displayLL(head);

    head=delAtEnd(head);
    printf("Linked list after deletion at end.\n");
    head=displayLL(head);

    head=delAtPos(head,0);
    printf("Linked list after deletion from pos.\n");
    head=displayLL(head);
}
```

Output:



```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Singl
yLinkedList.c -o SinglyLinkedList } ; if ($?) { .\SinglyLinkedList }
Enter -1 to stop.
Enter Number:1
Enter Number:2
Enter Number:3
Enter Number:4
Enter Number:-1
Linked List Elements:1 2 3 4
Enter Number:7
Linked list after insertion at begining.
Linked List Elements:7 1 2 3 4
Enter Number:9
Linked list after insertion at end.
Linked List Elements:7 1 2 3 4 9
Enter Number:5
Linked list after insertion at position.
Linked List Elements:7 1 5 2 3 4 9
Linked list after deletion at front.
Linked List Elements:1 5 2 3 4 9
Linked list after deletion at end.
Linked List Elements:1 5 2 3 4
Linked list after deletion from pos.
Linked List Elements:5 2 3 4
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 6:

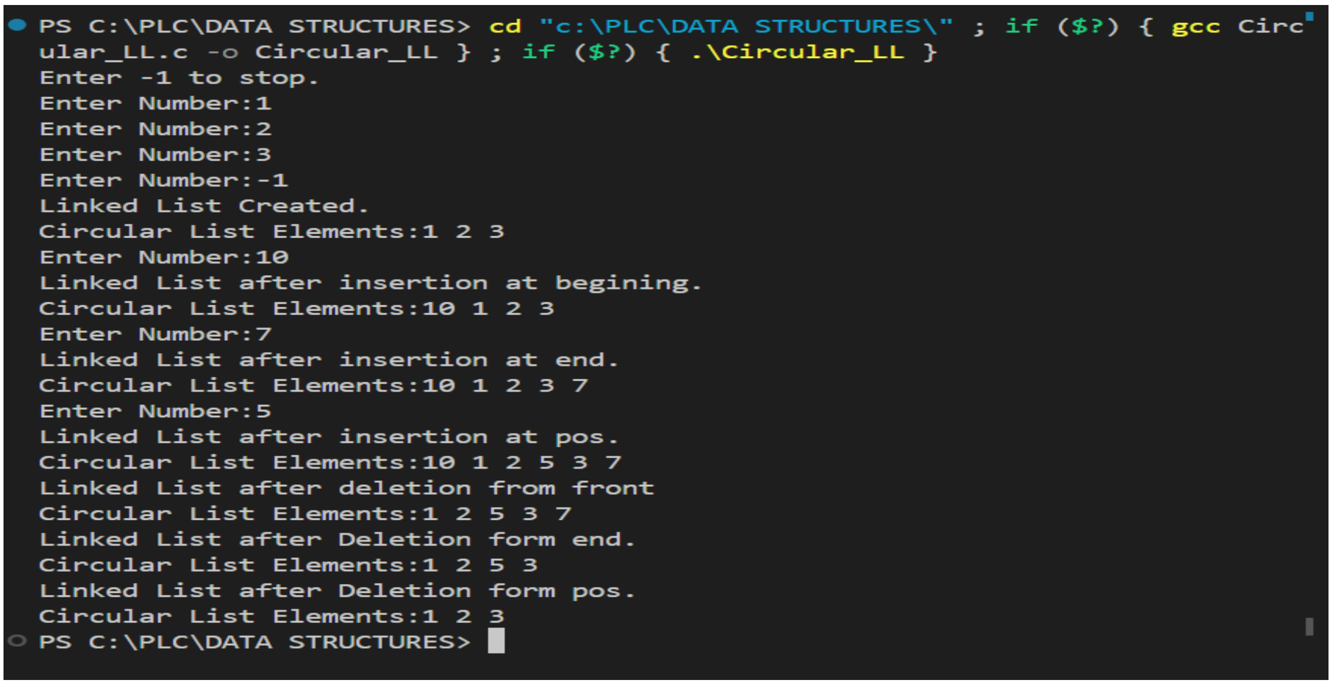
WAP to Implement Circular Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at first position, and at end of list and at any position
c) Delete a node at front and at the end of the list and at any position
d) Display the contents of the linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
   int data;
   struct Node *next;
};
//Create Circular Linked List
struct Node* createCircularLL(struct Node* head){
   int data;
   struct Node*p;
   printf("Enter -1 to stop.\n");
   printf("Enter Number:");
   scanf("%d",&data);
   while(data!=-1){

      struct Node *newNode=(struct Node*)malloc(sizeof(struct Node));
      newNode->data=data;
      if(head==NULL){
         newNode->next=newNode;
         head=newNode;
      }
      else{
         p=head;
         while(p->next!=head){
            p=p->next;
         }
         p->next=newNode;
         newNode->next=head;

      }
   printf("Enter Number:");
   scanf("%d",&data);
   }
   return head;
}
//Display Circular Linked List
struct Node* displayCircularLL(struct Node* head){
   struct Node* p;
   p=head;
   printf("Circular List Elements:");
   while(p->next !=head){
      printf("%d ",p->data);
      p=p->next;
   }
```

```
printf("%d ",p->data);
   printf("\n");
   return head;
}
//Insert At Begining
struct Node* insertFirst(struct Node* head){
   int num;
   struct Node *p;
   p=head;
   printf("Enter Number:");
   scanf("%d",&num);
   struct Node *newNode=(struct Node*)malloc(sizeof(struct Node));
   newNode->data=num;
   while(p->next!=head){
      p=p->next;
   }
   p->next=newNode;
   newNode->next=head;
   head=newNode;
   return head;
}
//Insert At End
struct Node* insertEnd(struct Node* head){
   int num;
   struct Node *p;
   p=head;
   printf("Enter Number:");
   scanf("%d",&num);
   struct Node *newNode=(struct Node*)malloc(sizeof(struct Node));
   newNode->data=num;
   while(p->next!=head){
      p=p->next;
   }
   p->next=newNode;
   newNode->next=head;
   return head;
}
//Insert At Any Position
struct Node* insertPosition(struct Node* head, int pos){
   int num,i=0;
   struct Node *p;
   p=head;
   printf("Enter Number:");
   scanf("%d",&num);
   struct Node *newNode=(struct Node*)malloc(sizeof(struct Node));
   newNode->data=num;
   if(pos==0){
      head=insertFirst(head);
      return head;
   }
   else{
```

```
 while(i!=pos-1){
           p=p->next;
           i++;
       }
   newNode->next=p->next;
   p->next=newNode;
   return head;
   }

}
//Delete From Front
struct Node* DelFromFront(struct Node* head){
   struct Node *p;
   p=head;
   while(p->next!=head){
      p=p->next;
   }
   p->next=head->next;
   free(head);
   head=p->next;
}
//Delete From End
struct Node* DelFromEnd(struct Node* head){
   struct Node *p,*preNode;
   p=head;
   while(p->next!=head){
      preNode=p;
      p=p->next;
   }
   preNode->next=p->next;
   free(p);
   return head;
}
//Delete From Any Position
struct Node* DelFromPos(struct Node* head,int pos){
   int i=0;
   struct Node* p,*preNode;
   p=head;
   if(pos==0){
      head=DelFromFront(head);
      return head;
   }
   else{
      while(i!=pos){
         preNode=p;
         p=p->next;
         i++;
      }
   preNode->next=p->next;
   free(p);
   return head;
```

```
}
}
int main(){
    struct Node* head=NULL;
    head=createCircularLL(head);
    printf("Linked List Created. \n");
    head=displayCircularLL(head);

    head=insertFirst(head);
    printf("Linked List after insertion at begining.\n");
    head=displayCircularLL(head);

    head=insertEnd(head);
    printf("Linked List after insertion at end.\n");
    head=displayCircularLL(head);

    head=insertPosition(head, 3);
    printf("Linked List after insertion at pos.\n");
    head=displayCircularLL(head);

    head=DelFromFront(head);
    printf("Linked List after deletion from front\n");
    head=displayCircularLL(head);

    head=DelFromEnd(head);
    printf("Linked List after Deletion form end.\n");
    head=displayCircularLL(head);

    head=DelFromPos(head,2);
    printf("Linked List after Deletion form pos.\n");
    head=displayCircularLL(head);
}
```
Output:



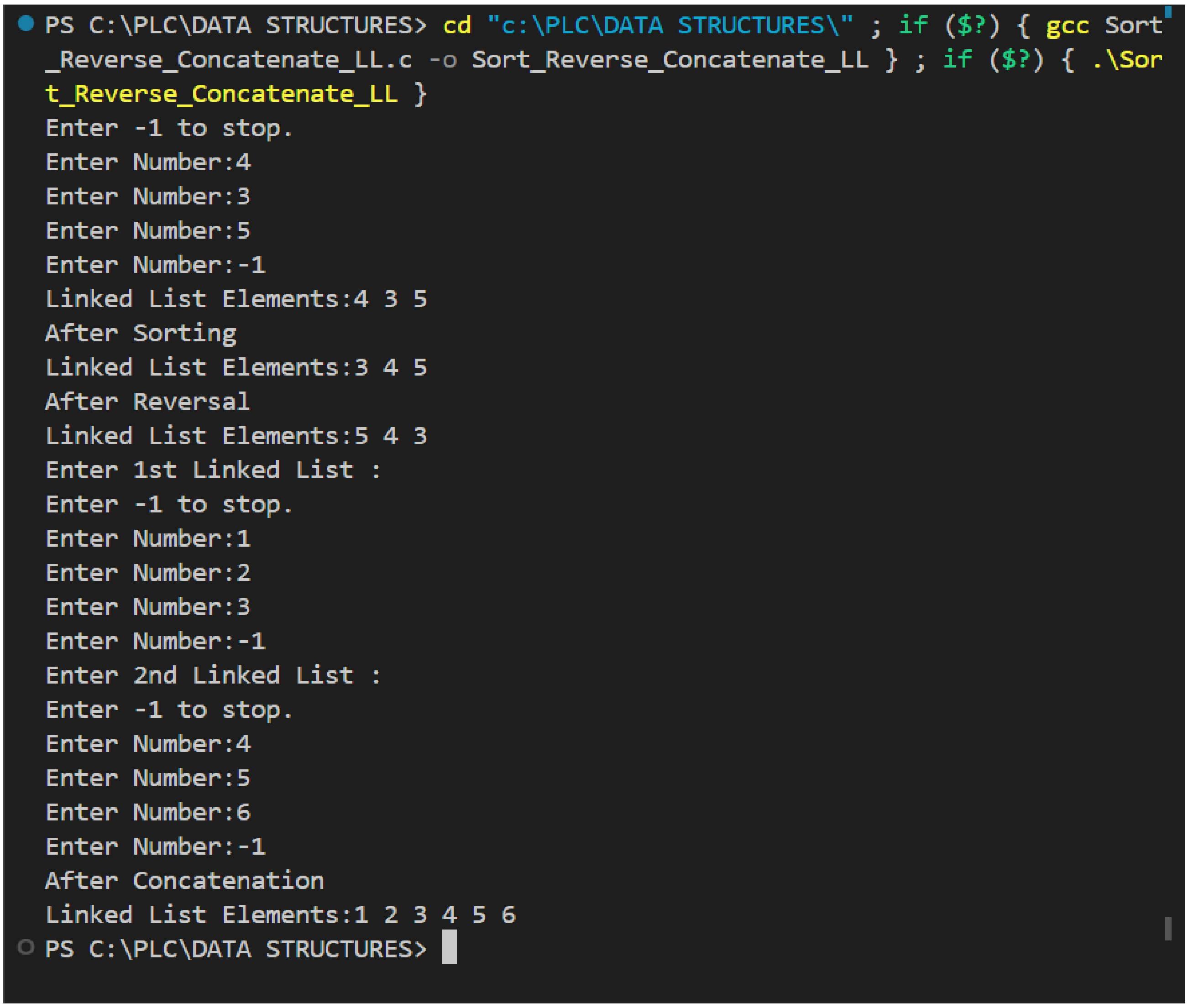
```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Circ
ular_LL.c -o Circular_LL } ; if ($?) { .\Circular_LL }
Enter -1 to stop.
Enter Number:1
Enter Number:2
Enter Number:3
Enter Number:-1
Linked List Created.
Circular List Elements:1 2 3
Enter Number:10
Linked List after insertion at begining.
Circular List Elements:10 1 2 3
Enter Number:7
Linked List after insertion at end.
Circular List Elements:10 1 2 3 7
Enter Number:5
Linked List after insertion at pos.
Circular List Elements:10 1 2 5 3 7
Linked List after deletion from front
Circular List Elements:1 2 5 3 7
Linked List after Deletion form end.
Circular List Elements:1 2 5 3
Linked List after Deletion form pos.
Circular List Elements:1 2 3
PS C:\PLC\DATA STRUCTURES> 
```

Lab Program 7:

7(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 data;
    struct Node *next;
};
//Create Linked List
struct Node* createLL(struct Node* head){
    int num;

    printf("Enter -1 to stop.\n");
    printf("Enter Number:");
    scanf("%d",&num);
    while(num!=-1){
        struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
        struct Node* p;
        p=head;
        if(head==NULL){
            newNode->data=num;
            newNode->next=NULL;
            head=newNode;
        }
        else{

            while(p->next!=NULL){
                p=p->next;
            }
            newNode->data=num;
            p->next=newNode;
            newNode->next=NULL;
        }
    printf("Enter Number:");
    scanf("%d",&num);
    }
    return head;
}
//Display Linked List
struct Node* displayLL(struct Node* head){
    struct Node*p;
    p=head;
    printf("Linked List Elements:");
    while(p !=NULL){
        printf("%d ",p->data);
        p=p->next;

}

 printf("\n");
    return head;
}
//Sort Linked List
struct Node* sortLL(struct Node* head){
    struct Node* ptr,*trav;
    int temp;
    ptr=head;
    while(ptr->next != NULL){
        trav=ptr->next;
        while(trav!=NULL){
            if(ptr->data > trav->data){
                temp=ptr->data;
                ptr->data=trav->data;
                trav->data=temp;
            }
            trav=trav->next;
        }
        ptr=ptr->next;
    }
    return head;
}
struct Node* LLRev(struct Node* head){
    struct Node* temp;
    struct Node* prev=NULL;
    struct Node* cur=head;
    while(cur!=NULL){
        temp=cur->next;
        cur->next=prev;
        prev=cur;
        cur=temp;
    }
    head=prev;
    return head;
}
struct Node* ConcatLL(struct Node* head1,struct Node* head2){
    struct Node*ptr;
    ptr=head1;
    while(ptr->next!=NULL){
        ptr= ptr->next;
    }
    ptr->next=head2;
    return head1;

}
int main(){
    struct Node* head=NULL;
    struct Node* head1=NULL;
    struct Node* head2=NULL;
    head=createLL(head);


head=displayLL(head);
    printf("After Sorting \n");
    head=sortLL(head);
    head=displayLL(head);
    printf("After Reversal \n");
    head=LLRev(head);
    head=displayLL(head);
    printf("Enter 1st Linked List : \n");
    head1=createLL(head1);
    printf("Enter 2nd Linked List : \n");
    head2=createLL(head2);
    printf("After Concatenation \n");
    head1=ConcatLL(head1,head2);
    head1=displayLL(head1);
}
```

Output:



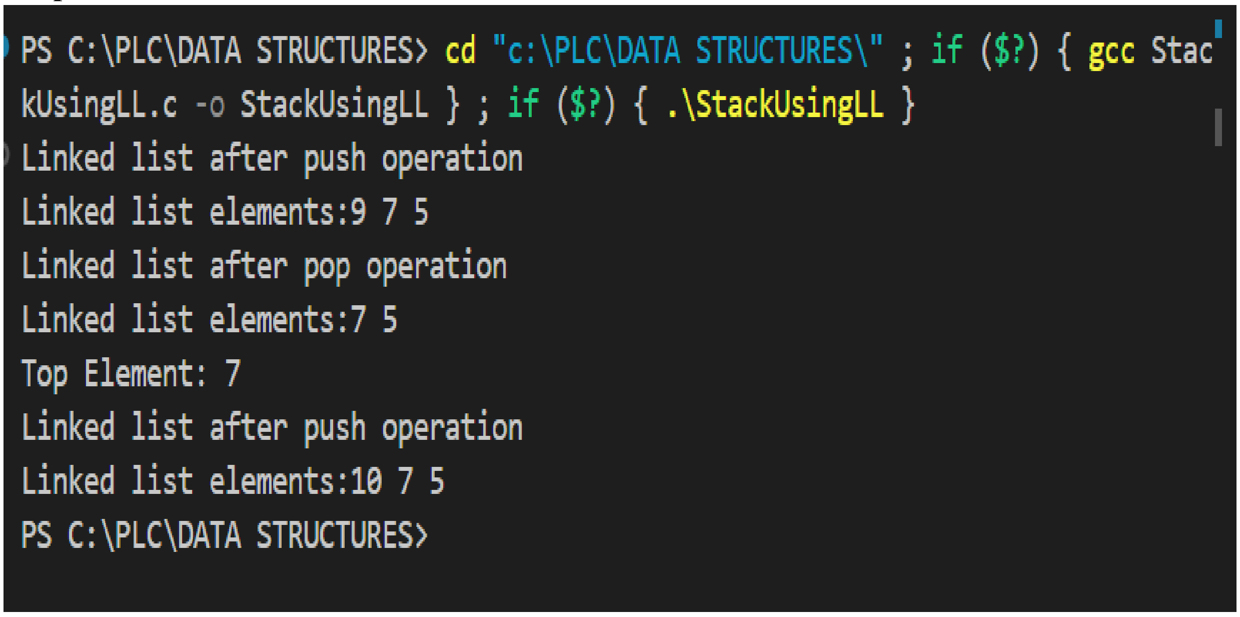
```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Sort
_Reverse_Concatenate_LL.c -o Sort_Reverse_Concatenate_LL } ; if ($?) { .\Sor
t_Reverse_Concatenate_LL }
Enter -1 to stop.
Enter Number:4
Enter Number:3
Enter Number:5
Enter Number:-1
Linked List Elements:4 3 5
After Sorting
Linked List Elements:3 4 5
After Reversal
Linked List Elements:5 4 3
Enter 1st Linked List :
Enter -1 to stop.
Enter Number:1
Enter Number:2
Enter Number:3
Enter Number:-1
Enter 2nd Linked List :
Enter -1 to stop.
Enter Number:4
Enter Number:5
Enter Number:6
Enter Number:-1
After Concatenation
Linked List Elements:1 2 3 4 5 6
PS C:\PLC\DATA STRUCTURES>
```

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

Stack Implementation :-

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int data;
    struct Node* next;
};
int isEmpty(struct Node* top){
    if(top == NULL){
        return 1;
    }
    return 0;
}
struct Node* displayLL(struct Node* top){
    if(isEmpty(top)){
        printf("No elements to print.\n");
        return top;
    }
    printf("Linked list elements:");
    struct Node* p =top;
    while(p!=NULL){
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
    return top;
}
struct Node* push(struct Node* top,int data){
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    if(top==NULL){
        newNode->data=data;
        top=newNode;
        newNode->next=NULL;
        return top;
    }
    else{
        newNode->data=data;
        newNode->next=top;
        top=newNode;
        return top;
    }
}
struct Node* pop(struct Node* top){
    if(isEmpty(top)){
        printf("Stack is empty.\n");
    }
    else{
        struct Node* p =top;

top=p->next;
        free(p);
        return top;
    }
}
int peek(struct Node* top){
    struct Node* p =top;
    return p->data;
}
int main(){
    struct Node* top=NULL;
    top=push(top,5);
    top=push(top,7);
    top=push(top,9);
    printf("Linked list after push operation \n");
    top=displayLL(top);
    printf("Linked list after pop operation \n");
    top=pop(top);
    top=displayLL(top);
    int x=peek(top);
    printf("Top Element: %d \n",x);
    top=push(top,10);
    printf("Linked list after push operation \n");
    top=displayLL(top);
}
```

Output:



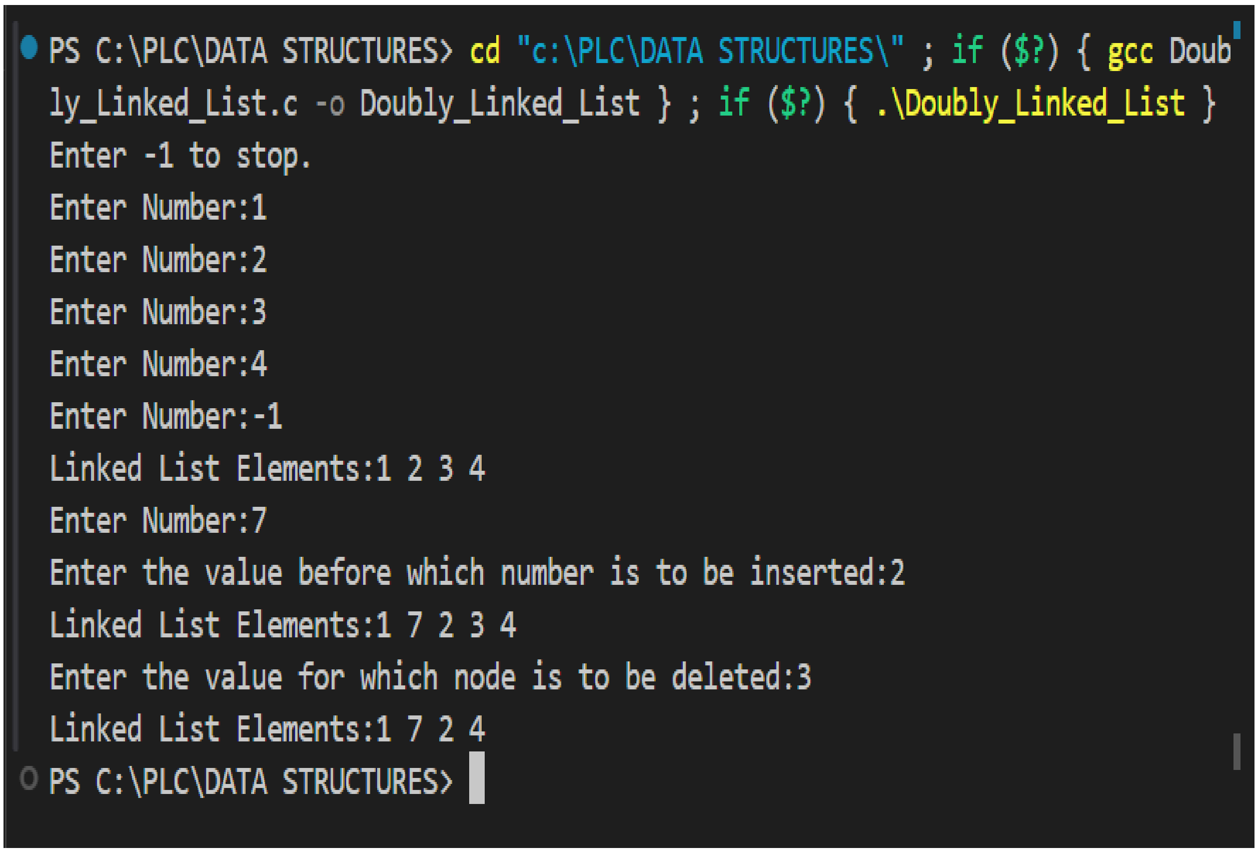
```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc StackUsingLL.c -o StackUsingLL } ; if ($?) { .\StackUsingLL }
Linked list after push operation
Linked list elements:9 7 5
Linked list after pop operation
Linked list elements:7 5
Top Element: 7
Linked list after push operation
Linked list elements:10 7 5
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 8:

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;
};
//Create a DLL
struct Node* createDLL(struct Node* head){
    int num;
    printf("Enter -1 to stop.\n");
    printf("Enter Number:");
    scanf("%d",&num);
    while(num!=-1){
    struct Node* ptr;
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if(head==NULL){
        newNode->data=num;
        newNode->prev=NULL;
        newNode->next=NULL;
        head=newNode;
    }
    else{
        ptr=head;
        while(ptr->next!=NULL){
            ptr=ptr->next;
        }
        newNode->data=num;
        newNode->prev=ptr;
        ptr->next=newNode;
        newNode->next=NULL;
    }
    printf("Enter Number:");
    scanf("%d",&num);
    }
    return head;

}
//Display Linked List
struct Node* displayLL(struct Node* head){
    struct Node*p;
    p=head;
    printf("Linked List Elements:");

while(p !=NULL){
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
    return head;
}

//Insert a new node to the left of the node
struct Node* insertLeft(struct Node* head){
    struct Node*ptr;
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    int n,val;
    printf("Enter Number:");
    scanf("%d",&n);
    printf("Enter the value before which number is to be inserted:");
    scanf("%d",&val);
    ptr=head;
    while(ptr->data!=val){
        ptr=ptr->next;
    }
    newNode->data=n;
    newNode->next=ptr;
    newNode->prev=ptr->prev;
    ptr->prev->next=newNode;
    ptr->prev=newNode;
    return head;

}
//Delete the node based on a specific value
struct Node* deleteNode(struct Node* head){
    int val;
    struct Node* ptr;
    printf("Enter the value for which node is to be deleted:");
    scanf("%d",&val);
    ptr=head;
    while(ptr->data!=val){
        ptr=ptr->next;
    }
    ptr->prev->next=ptr->next;
    ptr->next->prev=ptr->prev;
    return head;
}
int main(){
    struct Node* head=NULL;
    head=createDLL(head);
    head=displayLL(head);
    head=insertLeft(head);
    head=displayLL(head);
    head=deleteNode(head);
    head=displayLL(head);
}
```

Output:



```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Doub
ly_Linked_List.c -o Doubly_Linked_List } ; if ($?) { .\Doubly_Linked_List }
Enter -1 to stop.
Enter Number:1
Enter Number:2
Enter Number:3
Enter Number:4
Enter Number:-1
Linked List Elements:1 2 3 4
Enter Number:7
Enter the value before which number is to be inserted:2
Linked List Elements:1 7 2 3 4
Enter the value for which node is to be deleted:3
Linked List Elements:1 7 2 4
PS C:\PLC\DATA STRUCTURES>
```

Lab Program 9:

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.

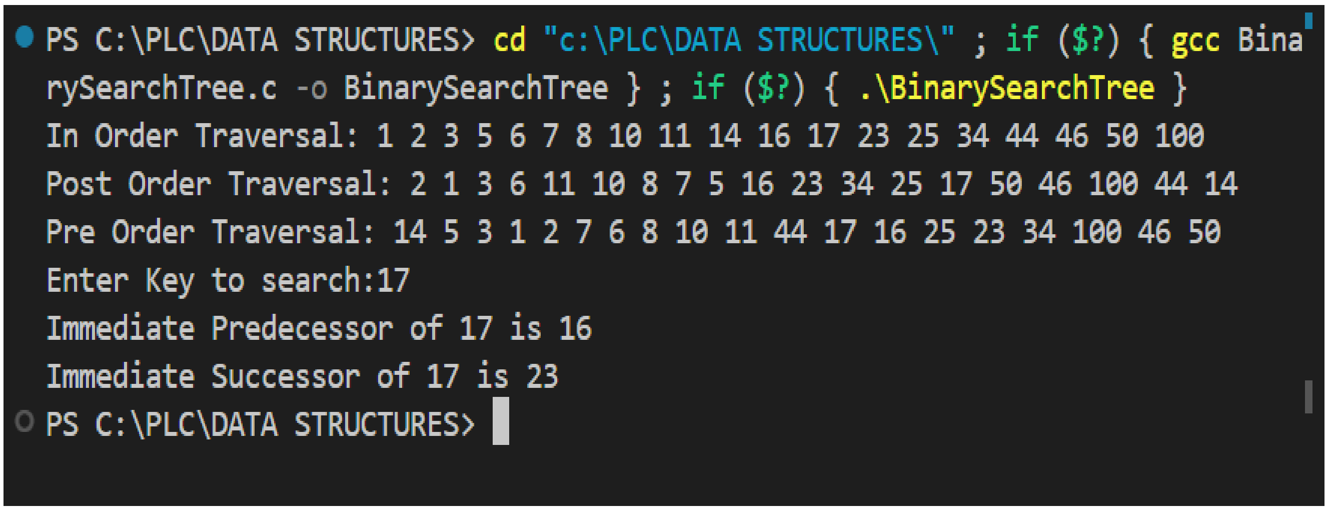
Also perform finding the immediate predecessor and immediate successor in inorder traversal using BST.

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

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

struct node* CreateNode(int ele) {
   struct node* nn = (struct node*)malloc(sizeof(struct node));
   if (nn == NULL) {
      printf("Memory Can't be allocated");
   }
   else {
      nn->data = ele;
      nn->left = NULL;
      nn->right = NULL;
      return nn;
   }
}


struct node* insert(struct node* root, int data) {
   if (root == NULL) {
      root = CreateNode(data);
   }
   else if (data >= root->data) {
      root->right = insert(root->right, data);
   }
   else if (data < root->data) {
      root->left = insert(root->left, data);
   }
   return root;
}
void inordertrav(struct node* root) {
   if (root == NULL) {
      return;
   }
   inordertrav(root->left);
   printf("%d ", root->data);
   inordertrav(root->right);
}
```

```
void postordertrav(struct node* root) {
   if (root == NULL) {
      return;
   }
   postordertrav(root->left);
   postordertrav(root->right);
   printf("%d ", root->data);
}
void preordertrav(struct node* root) {
   if (root == NULL) {
      return;
   }
   printf("%d ", root->data);
   preordertrav(root->left);
   preordertrav(root->right);
}
struct node* findImmediatePredecessor(struct node* root, int key) {
   struct node* pre = NULL;
   while (root) {
      if (root->data < key) {
         pre = root;
         root = root->right;
      }
      else if (root->data >= key) {
         root = root->left;
      }
   }
   return pre;
}
struct node* findImmediateSuccessor(struct node* root, int key) {
   struct node* suc = NULL;
   while (root) {
      if (root->data > key) {
         suc = root;
         root = root->left;
      }
      else if (root->data <= key) {
         root = root->right;
      }
   }
   return suc;
}

int main() {
   struct node* root = NULL;
   int data,key;
   root = insert(root, 14);
   root = insert(root, 5);
   root = insert(root, 44);
```

```
    root = insert(root, 3);
    root = insert(root, 7);
    root = insert(root, 100);
    root = insert(root, 46);
    root = insert(root, 8);
    root = insert(root, 10);
    root = insert(root, 11);
    root = insert(root, 17);
    root = insert(root, 25);
    root = insert(root, 23);
    root = insert(root, 34);
    root = insert(root, 16);
    root = insert(root, 50);
    root = insert(root, 1);
    root = insert(root, 6);
    root = insert(root, 2);

    printf("In Order Traversal: ");
    inordertrav(root);
    printf("\n");

    printf("Post Order Traversal: ");
    postordertrav(root);
    printf("\n");

    printf("Pre Order Traversal: ");
    preordertrav(root);
    printf("\n");

    printf("Enter Key to search:");
    scanf("%d",&key);
    struct node* pre = findImmediatePredecessor(root, key);
    struct node* suc = findImmediateSuccessor(root, key);

    if (pre)
        printf("Immediate Predecessor of %d is %d\n", key, pre->data);
    else
        printf("No Immediate Predecessor of %d\n", key);

    if (suc)
        printf("Immediate Successor of %d is %d\n", key, suc->data);
    else
        printf("No Immediate Successor of %d\n", key);

    return 0;
}
```

Output:

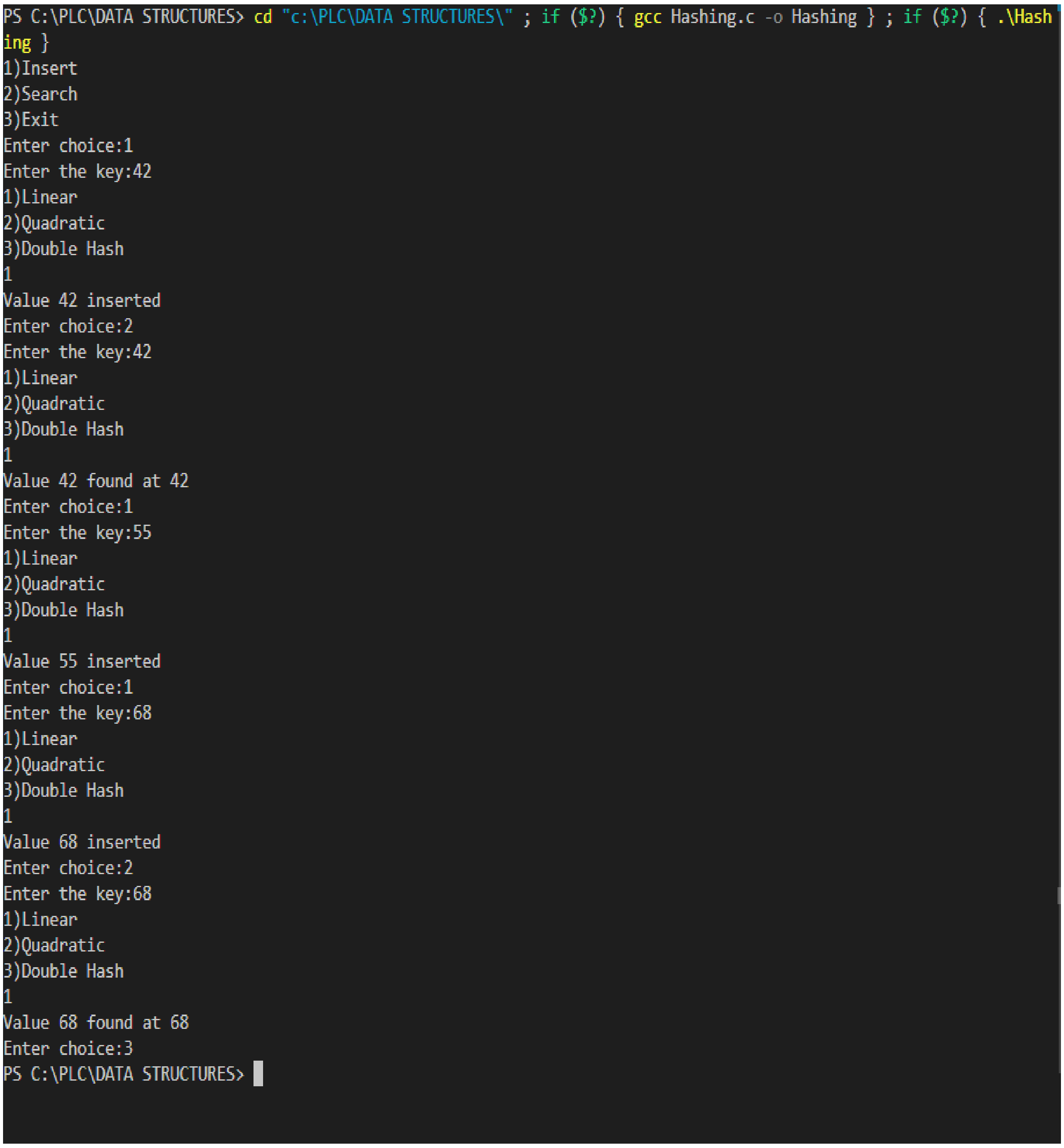


```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc BinarySearchTree.c -o BinarySearchTree } ; if ($?) { .\BinarySearchTree }
In Order Traversal: 1 2 3 5 6 7 8 10 11 14 16 17 23 25 34 44 46 50 100
Post Order Traversal: 2 1 3 6 11 10 8 7 5 16 23 34 25 17 50 46 100 44 14
Pre Order Traversal: 14 5 3 1 2 7 6 8 10 11 44 17 16 25 23 34 100 46 50
Enter Key to search:17
Immediate Predecessor of 17 is 16
Immediate Successor of 17 is 23
PS C:\PLC\DATA STRUCTURES>
```

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 -&gt; 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
i. linear probing
ii. Quadratic Probing
iii. Double Hashing

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 100
int L[MAX_SIZE];
int count = 0;
int hash_lprobe(int key) {
    int i = 0;
    while (L[(key + i) % MAX_SIZE] != 0) {
        i++;
        if (count == MAX_SIZE) {
            printf("Array is full\n");
            return -1;
        }
    }
    count++;
    return (key + i) % MAX_SIZE;
}
int hash_qprobe(int key) {
    int i = 0;
    while (L[(key + i * i) % MAX_SIZE] != 0) {
        i++;
        if (i == MAX_SIZE)
            return -1;
        if (count == MAX_SIZE) {
            printf("Array is full\n");
            return -1;
        }
    }
    count++;
    return (key + i * i) % MAX_SIZE;
}
int double_hash(int key) {
    int i = 0;
    while (L[(key % MAX_SIZE + 97 - key % 97) % MAX_SIZE] != 0) {
        i++;
        if (count == MAX_SIZE) {
            printf("Array is full\n");
            return -1;
        }
    }
    count++;
    return (key % MAX_SIZE + 97 - key % 97) % MAX_SIZE;
}
int search_lp(int key) {
    int i = 0;
    while (L[(key + i) % MAX_SIZE] != key) {
        if (i == MAX_SIZE) {
            printf("Value doesn't exist\n");
            return -1;
        }
        i++;
    }
    return (key + i) % MAX_SIZE;
}
int search_qp(int key) {
    int i = 0;
    while (L[(key + i * i) % MAX_SIZE] != key) {
        if (i == MAX_SIZE) {
            printf("Value doesn't exist\n");
            return -1;
        }
        i++;
    }
    return (key + i * i) % MAX_SIZE;
}

int search_db(int key) {
    int i = 0;
    while (L[(key % MAX_SIZE + 97 - key % 97) % MAX_SIZE] != key) {
        if (i == MAX_SIZE) {
            printf("Value doesn't exist\n");
            return -1;
        }
        i++;
    }
    return (key % MAX_SIZE + 97 - key % 97) % MAX_SIZE;
}

int main() {
    printf("1)Insert\n2)Search\n3)Exit\n");

    while (1) {
        int choice, subChoice, key;
        printf("Enter choice:");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the key:");
                scanf("%d", &key);
                printf("1)Linear\n2)Quadratic\n3)Double Hash\n");
                scanf("%d", &subChoice);

                if (subChoice == 1)
                    key = hash_lprobe(key);
                else if (subChoice == 2)
                    key = hash_qprobe(key);
                else
                    key = double_hash(key);

                L[key] = key;
                if (key != -1)
                    printf("Value %d inserted\n", key);
                break;

            case 2:
                printf("Enter the key:");
                scanf("%d", &key);
                printf("1)Linear\n2)Quadratic\n3)Double Hash\n");
                scanf("%d", &subChoice);

                if (subChoice == 1)
                    key = search_lp(key);
                else if (subChoice == 2)
                    key = search_qp(key);
                else
                    key = search_db(key);

                if (key != -1)
                    printf("Value %d found at %d\n", L[key], key);
                break;

            case 3:
                exit(0);

            default:
                printf("Invalid choice\n");
                break;
        }
    }

    return 0;
}
```
Output:



```
PS C:\PLC\DATA STRUCTURES> cd "c:\PLC\DATA STRUCTURES\" ; if ($?) { gcc Hashing.c -o Hashing } ; if ($?) { .\Hash
ing }
1)Insert
2)Search
3)Exit
Enter choice:1
Enter the key:42
1)Linear
2)Quadratic
3)Double Hash
1
Value 42 inserted
Enter choice:2
Enter the key:42
1)Linear
2)Quadratic
3)Double Hash
1
Value 42 found at 42
Enter choice:1
Enter the key:55
1)Linear
2)Quadratic
3)Double Hash
1
Value 55 inserted
Enter choice:1
Enter the key:68
1)Linear
2)Quadratic
3)Double Hash
1
Value 68 inserted
Enter choice:2
Enter the key:68
1)Linear
2)Quadratic
3)Double Hash
1
Value 68 found at 68
Enter choice:3
PS C:\PLC\DATA STRUCTURES>
```

<u>**Hacker Rank Questions**</u>

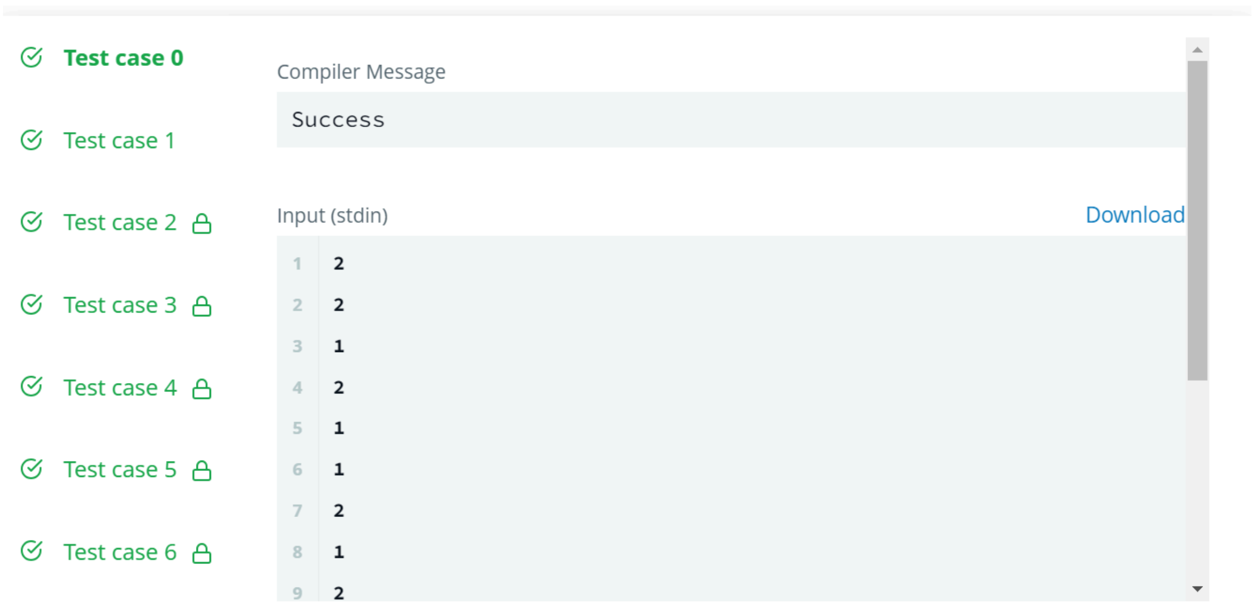
Question 1:

You're given the pointer to the head nodes of two linked lists. Compare the data in the nodes of the linked lists to check if they are equal. If all data attributes are equal and the lists are the same length, return 1. Otherwise, return 0.

Solution :

```
bool compare_lists(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2) {
  while (head1 != NULL && head2 != NULL) {
        if (head1->data != head2->data)
            return 0;

        head1 = head1->next;
        head2 = head2->next;
    }

    if (head1 == NULL && head2 == NULL)
        return 1;
    else
        return 0;


}
```

Output:



Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin) Download

```
2
2
1
2
1
1
2
1
2
```

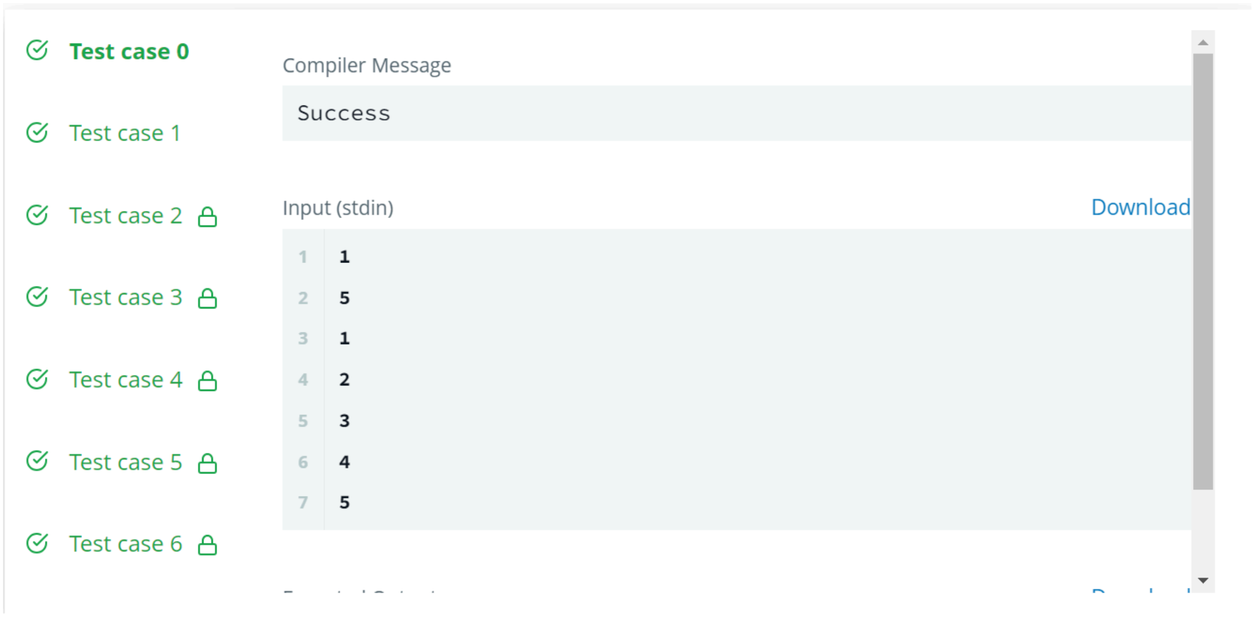
Question 2:

Given the pointer to the head node of a linked list, change the next pointers of the nodes so that their order is reversed. The head pointer given may be null meaning that the initial list is empty.

Solution:

```
SinglyLinkedListNode* reverse(SinglyLinkedListNode* llist) {
SinglyLinkedListNode* temp;
SinglyLinkedListNode* prev=NULL;
SinglyLinkedListNode* cur=llist;
   while(cur!=NULL){
      temp=cur->next;
      cur->next=prev;
      prev=cur;
      cur=temp;
   }
   llist=prev;
   return llist;
}
```

Output:



Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin) Download

```
1
5
1
2
3
4
5
```

Question 3:

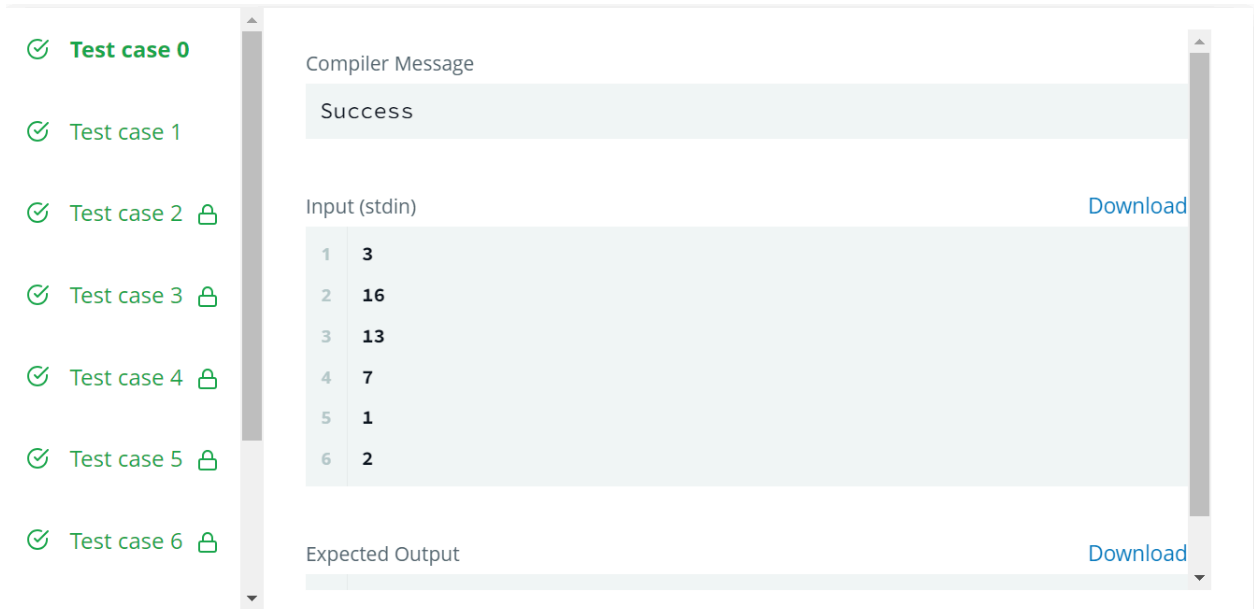
Given the pointer to the head node of a linked list and an integer to insert at a certain position, create a new node with the given integer as its data attribute, insert this node at the desired position and return the head node.

A position of 0 indicates head, a position of 1 indicates one node away from the head and so on. The head pointer given may be null meaning that the initial list is empty.

Solution:

```
SinglyLinkedListNode* insertNodeAtPosition(SinglyLinkedListNode* llist, int data, int
position) {
int i=0;

SinglyLinkedListNode* p;
   p=llist;


SinglyLinkedListNode* newNode=(struct Node*)malloc(sizeof(
SinglyLinkedListNode*));
   while(i!=position-1){
      p=p->next;
      i++;
   }
   newNode->data=data;
   newNode->next=p->next;
   p->next=newNode;
   return llist;
}
```

Output:



Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin) Download

```
3
16
13
7
1
2
```

Expected Output Download

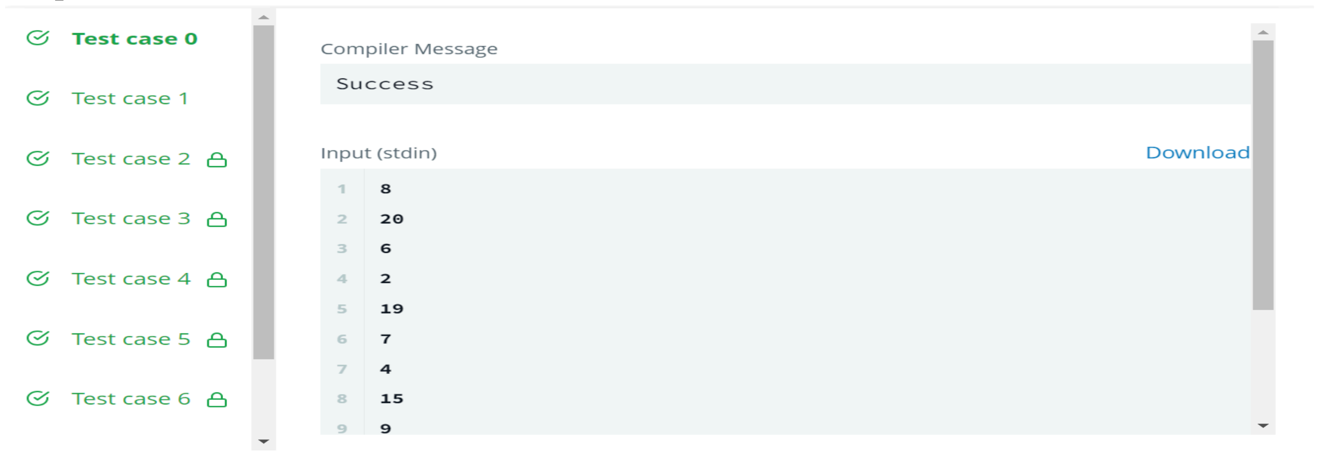
Question 4:

Delete the node at a given position in a linked list and return a reference to the head node. The head is at position 0. The list may be empty after you delete the node. In that case, return a null value.

Solution:

```
SinglyLinkedListNode* delAtFront(SinglyLinkedListNode* llist){
   if(llist==NULL){
      printf("Linked List alredy empty.\n");
      return llist;
   }
   else{
   SinglyLinkedListNode* p;
   p=llist->next;
   free(llist);
   llist=p;
   return llist;
   }
}
SinglyLinkedListNode* deleteNode(SinglyLinkedListNode* llist, int position) {
SinglyLinkedListNode*p,*preNode;
   int i=0;
   p=llist;
   if(position==0){
      llist=delAtFront(llist);
      return llist;
   }
   while(i!=position){
      preNode=p;
      p=p->next;
      i++;
   }
   preNode->next=p->next;
   free(p);
   return llist;
}
```

Output:



Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin) Download

```
8
20
6
2
19
7
4
15
9
```

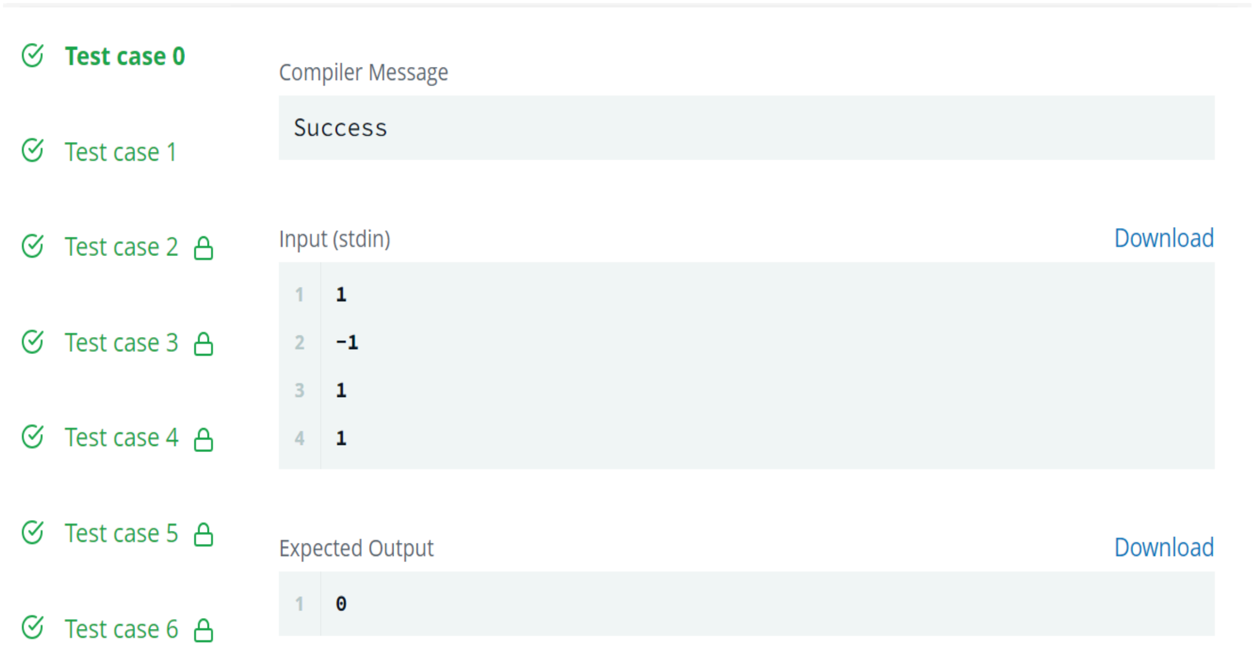
Question 5:

A linked list is said to contain a cycle if any node is visited more than once while traversing the list. Given a pointer to the head of a linked list, determine if it contains a cycle. If it does, return 1. Otherwise, return 0.

Solution:

```
bool has_cycle(SinglyLinkedListNode* head) {
    if (head == NULL || head->next == NULL) {
        return 0; // No cycle if the list is empty or has only one node
    }
    SinglyLinkedListNode* ptr1 = head;
    SinglyLinkedListNode* ptr2 = head->next;
    while (ptr2 != NULL && ptr2->next != NULL) {
        if (ptr1 == ptr2) {
            return 1; // Cycle detected
        }
        ptr1 = ptr1->next;
        ptr2 = ptr2->next->next;
    }
    return 0;
}
```

Output:



Compiler Message

```
Success
```

Input (stdin)

```
1
-1
1
1
```

Expected Output

```
0
```

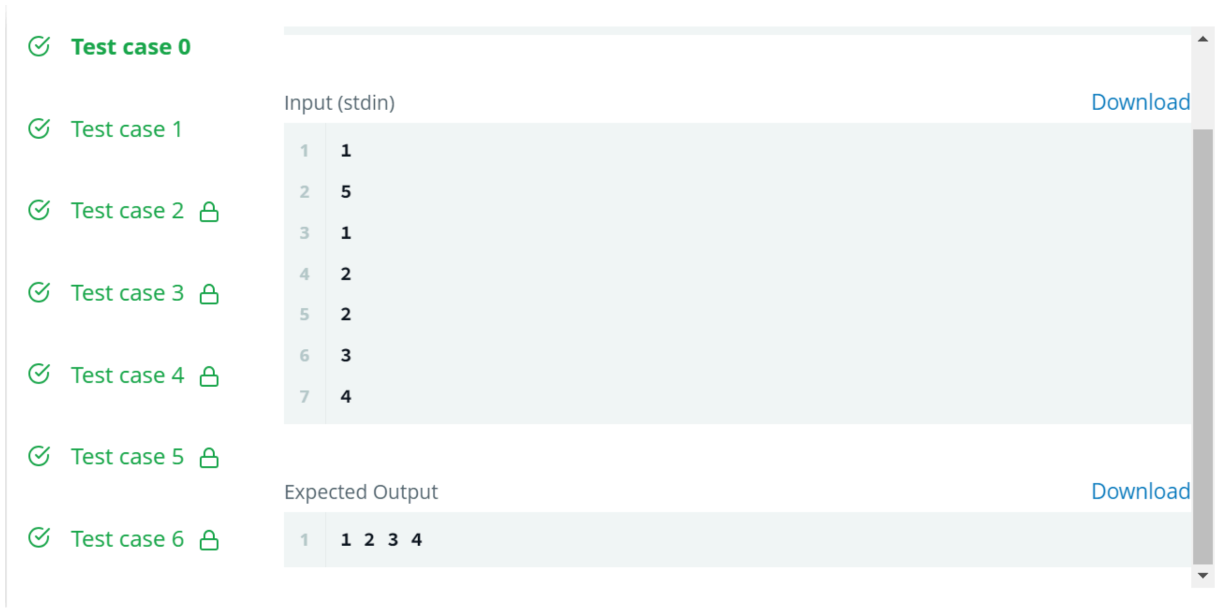
Question 6:

You are given the pointer to the head node of a sorted linked list, where the data in the nodes is in ascending order. Delete nodes and return a sorted list with each distinct value in the original list. The given head pointer may be null indicating that the list is empty.

Solution:

```
SinglyLinkedListNode* removeDuplicates(SinglyLinkedListNode* llist) {
SinglyLinkedListNode* current = llist;
   while (current != NULL && current->next != NULL)
      if (current->data == current->next->data) {
         // Delete the next node
         SinglyLinkedListNode* temp = current->next;
         current->next = current->next->next;
         free(temp);
      } else {

         current = current->next;
      }
   }
return llist;
}
```

Output:



```
1
5
1
2
2
3
4
```

```
1 2 3 4
```