VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

BHOOMI UDEDH (1BM23CS066)

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

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

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

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by BHOOMI UDEDH**(1BM23CS066)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.

Dr. Selva kumar S
Associate Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course outcomes:

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

Lab Program 1

Write a program to simulate the working of stack using an array with the following:

a) Push

b) Pop

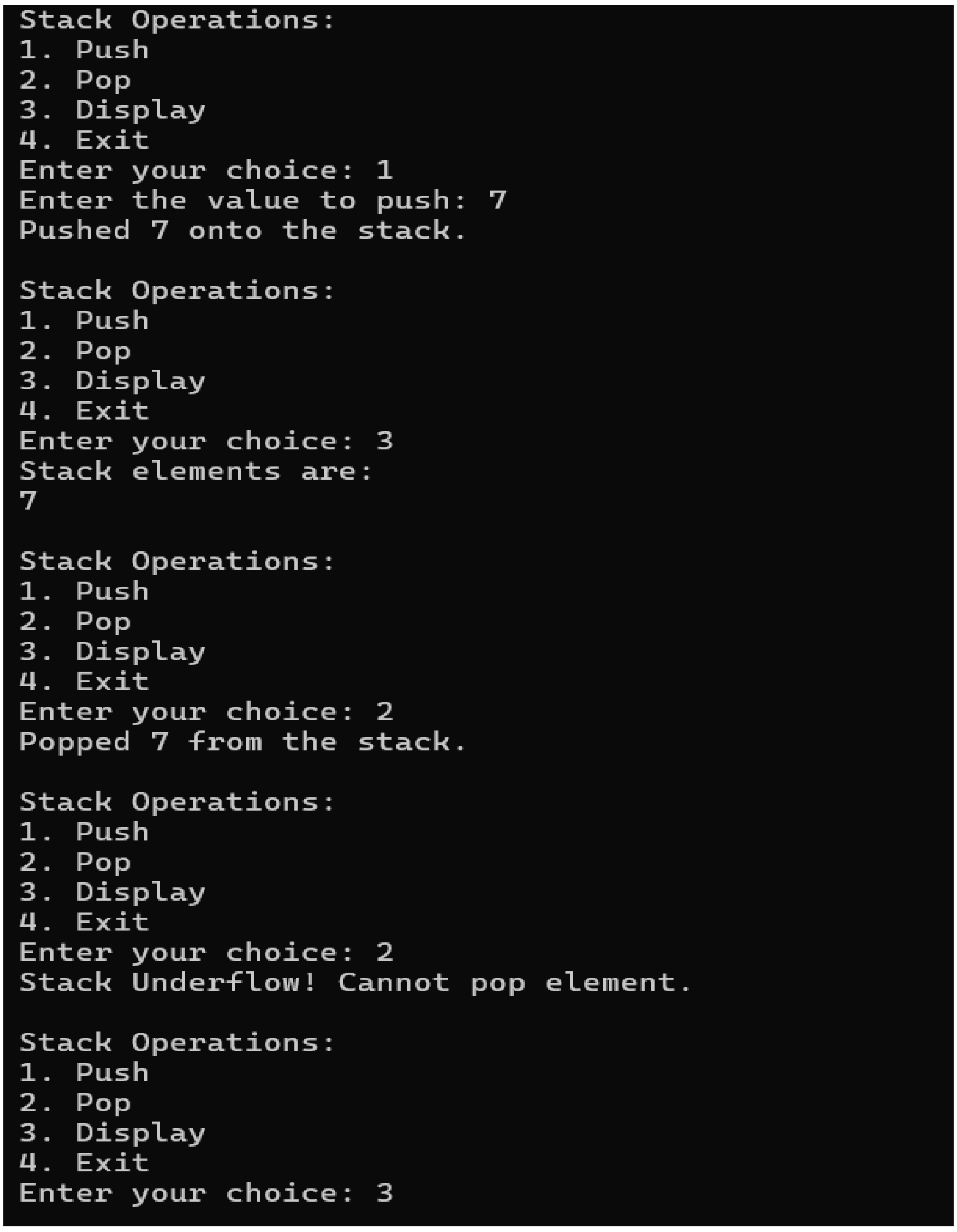
c) Display

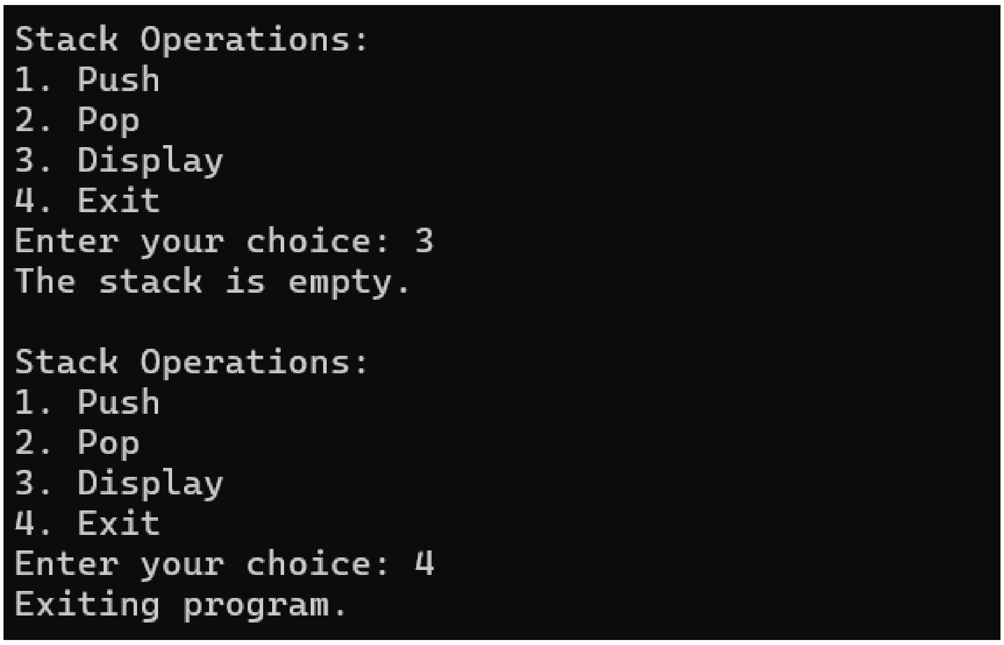
The program should print appropriate messages for stack overflow, stack underflow.

Code:

```
#include <stdio.h>
#define MAX 100
int stack[MAX];
int top = -1;

void push(int item) {
    if (top == MAX - 1) {
        printf("Stack Overflow! Cannot push element.\n");
    } else {
        stack[++top] = item;
        printf("Pushed %d onto the stack.\n", item);
    }
}
void pop() {
    if (top == -1) {
        printf("Stack Underflow! Cannot pop element.\n");
    } else {
        int item = stack[top--];
        printf("Popped %d from the stack.\n", item);
    }
}
void display() {
    if (top == -1) {
        printf("The stack is empty.\n");
    } else {
        printf("Stack elements are:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d\n", stack[i]);
        }
    }
}
int main() {
    int choice, value;

    while (1) {
        printf("\nStack Operations:\n");
        printf("1. 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 program.\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
}
```

Output:


```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 7
Pushed 7 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements are:
7

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 7 from the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Stack Underflow! Cannot pop element.

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


```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
The stack is empty.

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

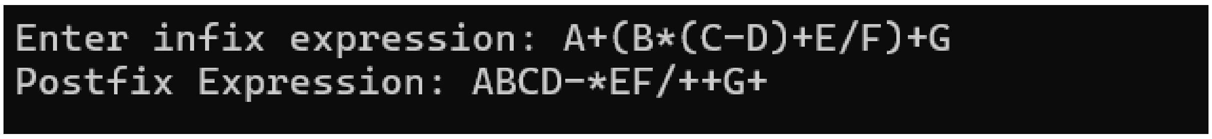
Lab Program 2

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 <ctype.h>
#define MAX 100
char stack[MAX];
int top = -1;
void push(char c) {
   stack[++top] = c;
}
char pop() {
   return stack[top--];
}
int precedence(char op) {
   if (op == '+' || op == '-')
```

```
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return 0;
}
void infixToPostfix(char* exp) {
    int i = 0, k = 0;
    char postfix[MAX];
    while (exp[i]) {
        if (isalpha(exp[i])) {
            postfix[k++] = exp[i];
        }
        else if (exp[i] == '(') {
            push(exp[i]);
        }
        else if (exp[i] == ')') {
            while (stack[top] != '(') {
                postfix[k++] = pop();
            }
            pop();
        }
        else {
            while (top != -1 && precedence(stack[top]) >= precedence(exp[i])) {
                postfix[k++] = pop();
            }
            push(exp[i]);
        }
        i++;
    }
    while (top != -1) {
        postfix[k++] = pop();
    }
    postfix[k] = '\0';
    printf("Postfix Expression: %s\n", postfix);
}
int main() {
    char exp[MAX];
    printf("Enter infix expression: ");
    scanf("%s", exp);
    infixToPostfix(exp);
    return 0;
}
```

Output:



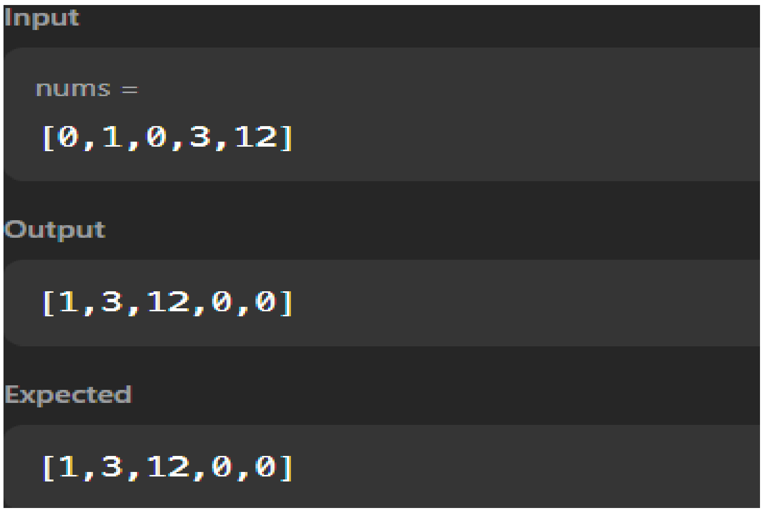
```
Enter infix expression: A+(B*(C-D)+E/F)+G
Postfix Expression: ABCD-*EF/++G+
```

Demonstration of account creation on LeetCode platform

Program - Leetcode platform

```
void moveZeroes(int* nums, int numsSize) {
    int j=0;
    for(int i=0;i<numsSize;i++)
    {
        if(nums[i]!=0)
        {
            int temp=nums[i];
            nums[i]=nums[j];
            nums[j]=temp;
            j++;
        }
    }
}
```

Output:



```
Input
nums =
[0,1,0,3,12]
Output
[1,3,12,0,0]
Expected
[1,3,12,0,0]
```

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

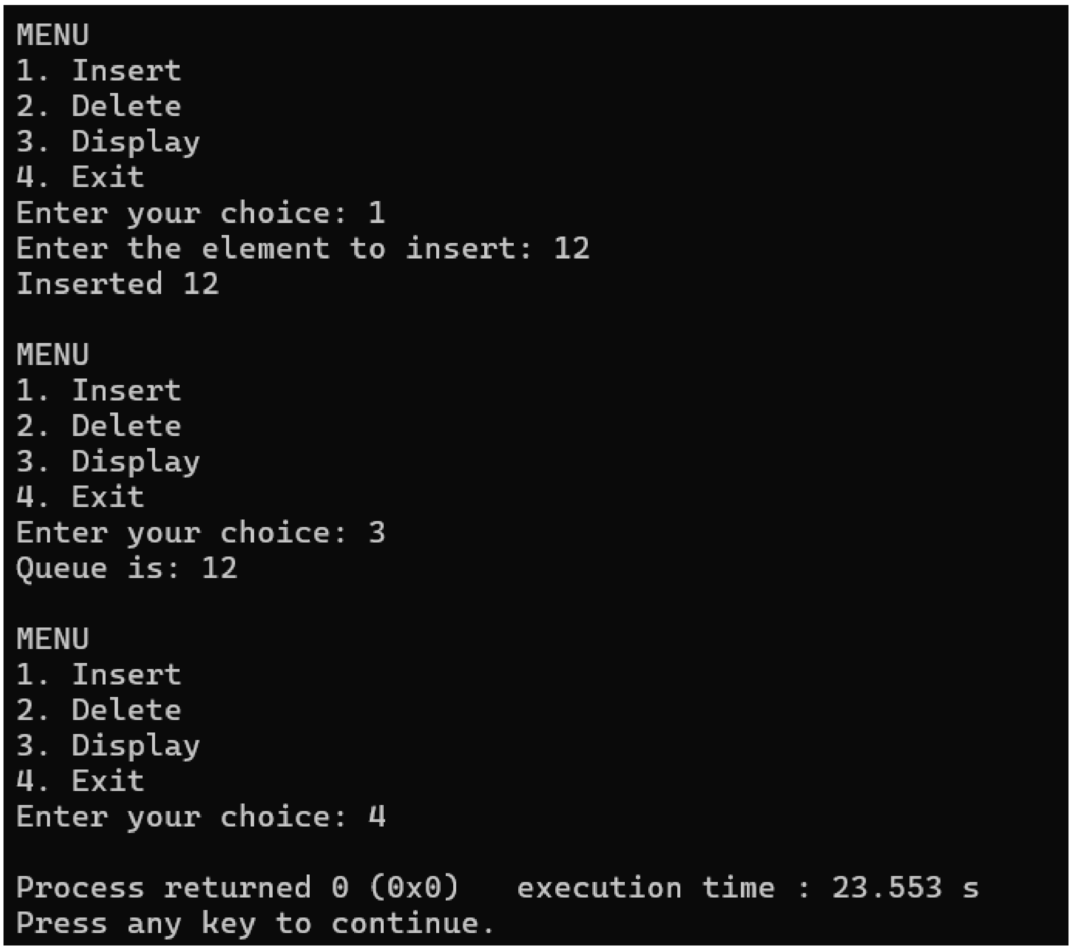
Code

```
#include<stdio.h>
#define Max 5
int queue[Max];
int front=-1;
int rear=-1;
void insert(int item);
void delete();
void display();
void main()
```

```
{
   int choice, item;
   while(1)
   {
      printf("\nMENU\n");
      printf("1. Insert\n");
      printf("2. Delete\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch(choice)
      {
         case 1:
            printf("Enter the element to insert: ");
            scanf("%d", &item);
            insert(item);
            break;
         case 2:
            delete();
            break;
         case 3:
            display();
            break;
         case 4:
            exit(0);
         default:
            printf("Invalid choice\n");
      }
```

```
    }
}

void insert(int add_item)
{
    if(rear == Max-1)
    {
        printf("Queue overflow\n");
    }
    else
    {
        if(front == -1)
        {
            front = 0;
        }
        rear = rear + 1;
        queue[rear] = add_item;
        printf("Inserted %d\n", add_item);
    }
}

void delete()
{
    if(front == -1 || front > rear)
    {
        printf("Queue underflow\n");
        return;
    }
    else
    {
        printf("Deleted item is %d\n", queue[front]);
        front = front + 1;
    }
}

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

Output:


```
MENU
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to insert: 12
Inserted 12

MENU
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue is: 12

MENU
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 23.553 s
Press any key to continue.
```

3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete &amp; Display

The program should print appropriate messages for queue empty and queue overflow conditions

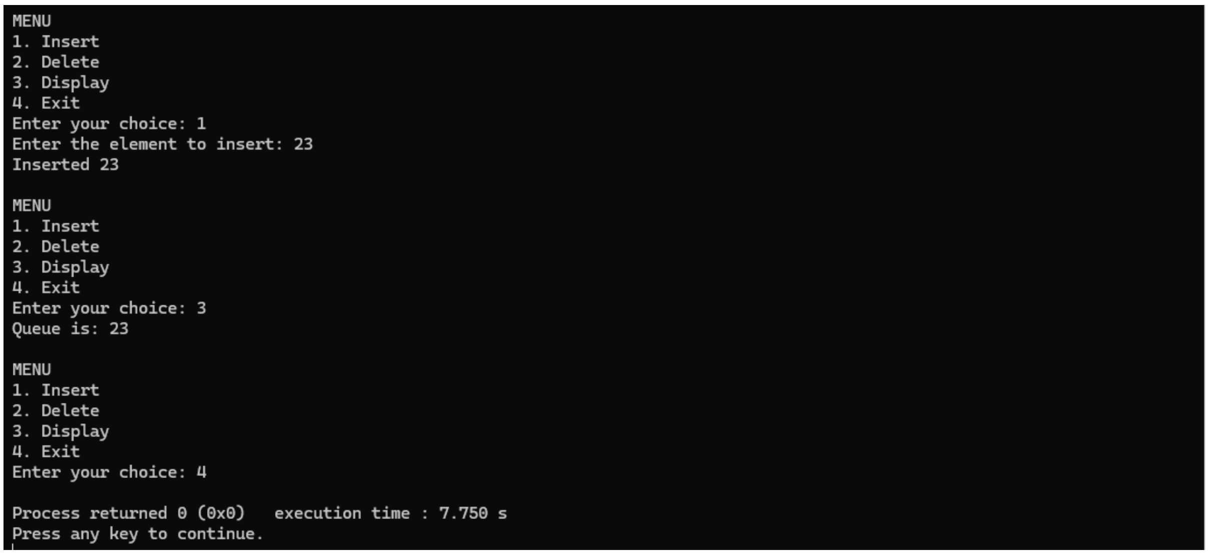
Code

```
#include<stdio.h>
#define Max 5
int queue[Max];
int front = -1;
int rear = -1;
void insert(int item);
void delete();
void display();
void main() {
   int choice, item;
   while(1) {
      printf("\nMENU\n");
      printf("1. Insert\n");
      printf("2. Delete\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch(choice) {
         case 1:
            printf("Enter the element to insert: ");
            scanf("%d", &item);
            insert(item);
            break;
         case 2:
            delete();
            break;
         case 3:
            display();
            break;
         case 4:
            exit(0);
         default:
            printf("Invalid choice\n");
      }
   }
}
```

```
void insert(int item)
{
   if ((front == 0 && rear == Max - 1) || (rear == (front - 1) % (Max - 1)))
   {
      printf("Queue overflow\n");
      return;
   }
   else if (front == -1)
   {
      front = rear = 0;
      queue[rear] = item;
   }
   else if (rear == Max - 1 && front != 0)
   {
      rear = 0;
      queue[rear] = item;
   }
   else
   {
      rear++;
      queue[rear] = item;
   }
   printf("Inserted %d\n", item);
}
void delete()
{
   if (front == -1) {
      printf("Queue underflow\n");
      return;
```

```
    }
    printf("Deleted item is %d\n", queue[front]);
    if (front == rear)
    {
        front = rear = -1;
    }
    else if (front == Max - 1)
    {
        front = 0;
    }
    else
    {
        front++;
    }
}
void display() {
    int i;
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue is: ");
    if (rear >= front)
    {
        for(i = front; i <= rear; i++)
        {
            printf("%d ", queue[i]);
        }
    }
    else
    {
        for(i = front; i < Max; i++)
        {
            printf("%d ", queue[i]);
        }
        for(i = 0; i <= rear; i++)
        {
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
}
```

Output



```
MENU
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to insert: 23
Inserted 23

MENU
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue is: 23

MENU
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 7.750 s
Press any key to continue.
```

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.

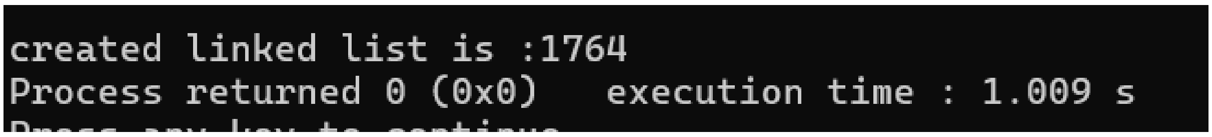
Code

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

struct Node
{
   int data;
   struct Node*next;
};

void push(struct Node**head_ref,int new_data)
{
   struct Node*new_node=(struct Node*)malloc(sizeof(struct Node));
   new_node->data=new_data;
   new_node->next=(*head_ref);
   (*head_ref)=new_node;
}

void append(struct Node**head_ref,int new_data)
{
struct Node*new_node=(struct Node*)malloc(sizeof(struct Node));
struct Node*last=*head_ref;
new_node->data=new_data;
new_node->next=NULL;
if(*head_ref==NULL)
{
```

```
*head_ref=new_node;
return;
}
while(last->next!=NULL)
last=last->next;
last->next=new_node;
return;
}

void printList(struct Node*node)


{
   while(node!=NULL)
   {
      printf("%d",node->data);
      node=node->next;
   }
}

int main()
{
   struct Node*head=NULL;
   append(&head,6);
   push(&head,7);
   push(&head,1);
   append(&head,4);
   printf("\ncreated linked list is :");
   printList(head);
```

```
    return 0;
}
```

Output



```
created linked list is :1764
Process returned 0 (0x0)   execution time : 1.009 s
```

Program - Leetcode platform

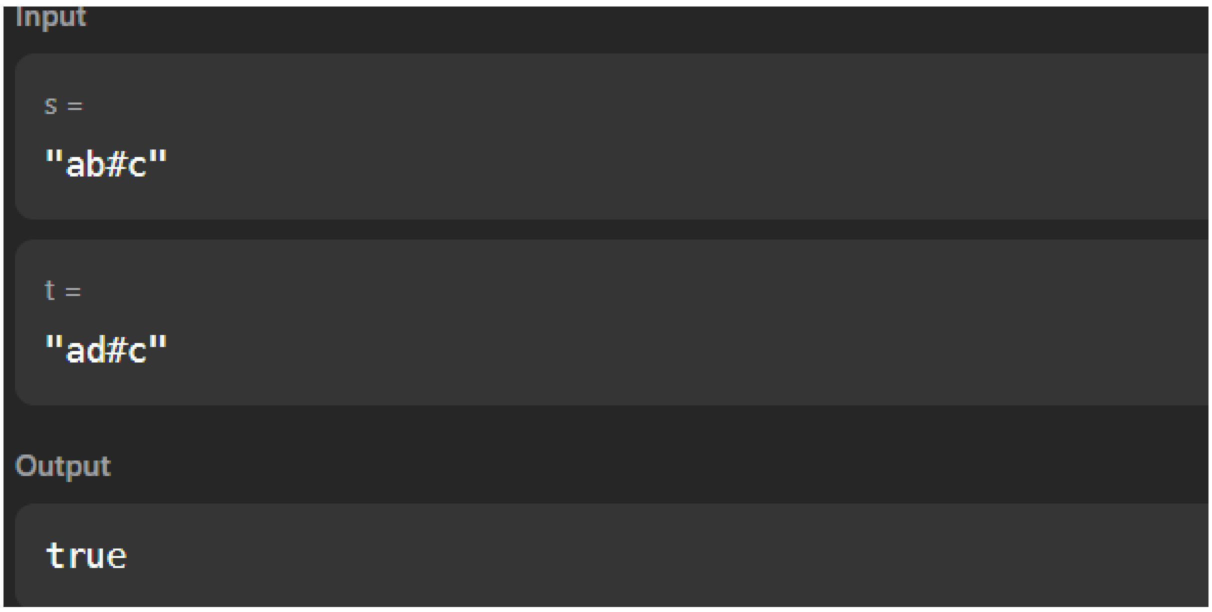
Backspace String Compare

Code

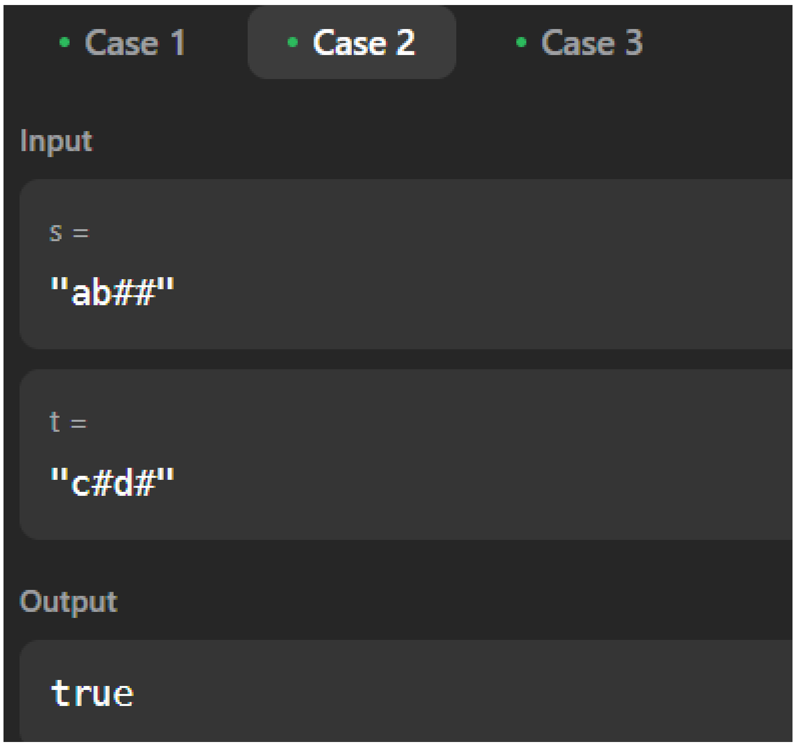
```
bool backspaceCompare(char* s, char* t) {
    int i, top = 0, len1 = strlen(s), len2 = strlen(t);

    for (i = 0;  i < len1; i++) {
        if (s[i] != '#')
            s[top++] = s[i];
        else {
            if (top > 0)
                top--;
        }
    }

    s[top] = '\0';

    top = 0;

    for (i = 0;  i < len2; i++) {
        if (t[i] != '#')
```

```
            t[top++] = t[i];
        else {
            if (top > 0)
                top--;
        }
    }

    t[top] = '\0';

    return strcmp(s, t) == 0;
}
```

Output



```
Input
s =
"ab#c"
t =
"ad#c"
Output
true
```

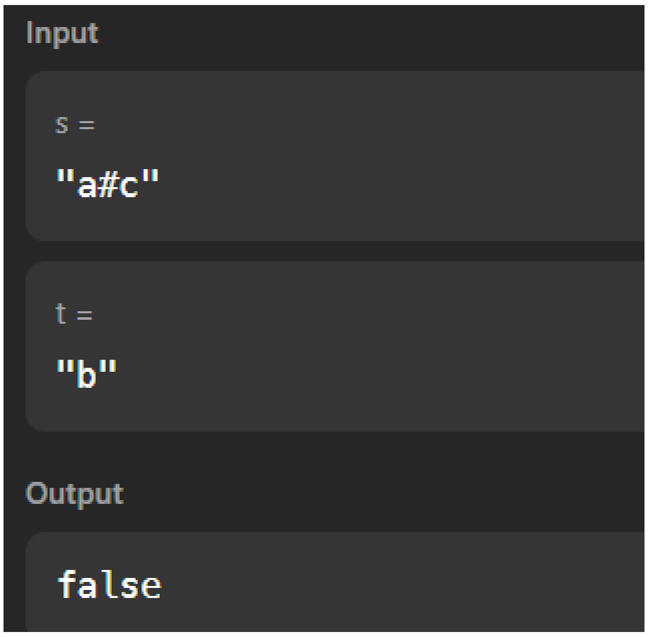


Input

```
s =
"ab##"
t =
"c#d#"
```

Output

```
true
```


Input

```
s =
"a#c"
t =
"b"
```

Output

```
false
```

Remove digit from Number to maximize result

Code

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

char* removeDigit(char* number, char digit) {
    int n = strlen(number);
int max_index = -1;


for (int i = 0; i < n; i++) {
    if (number[i] == digit) {


        if (i + 1 < n && number[i] < number[i + 1]) {
            max_index = i;
            break;
        }
        max_index = i;
    }
}



char* result = (char*)malloc(n * sizeof(char));
if (!result) {
    return NULL;
}



int j = 0;
for (int i = 0; i < n; i++) {
    if (i != max_index) {
        result[j++] = number[i];
    }
}
result[j] = '\0';


return result;
```

```
}
```

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.

Code

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

struct node *head=NULL;

void push();
void delete_at_front();
void delete_at_end();
void delete_at_pos();
void display();


void push()
{
```

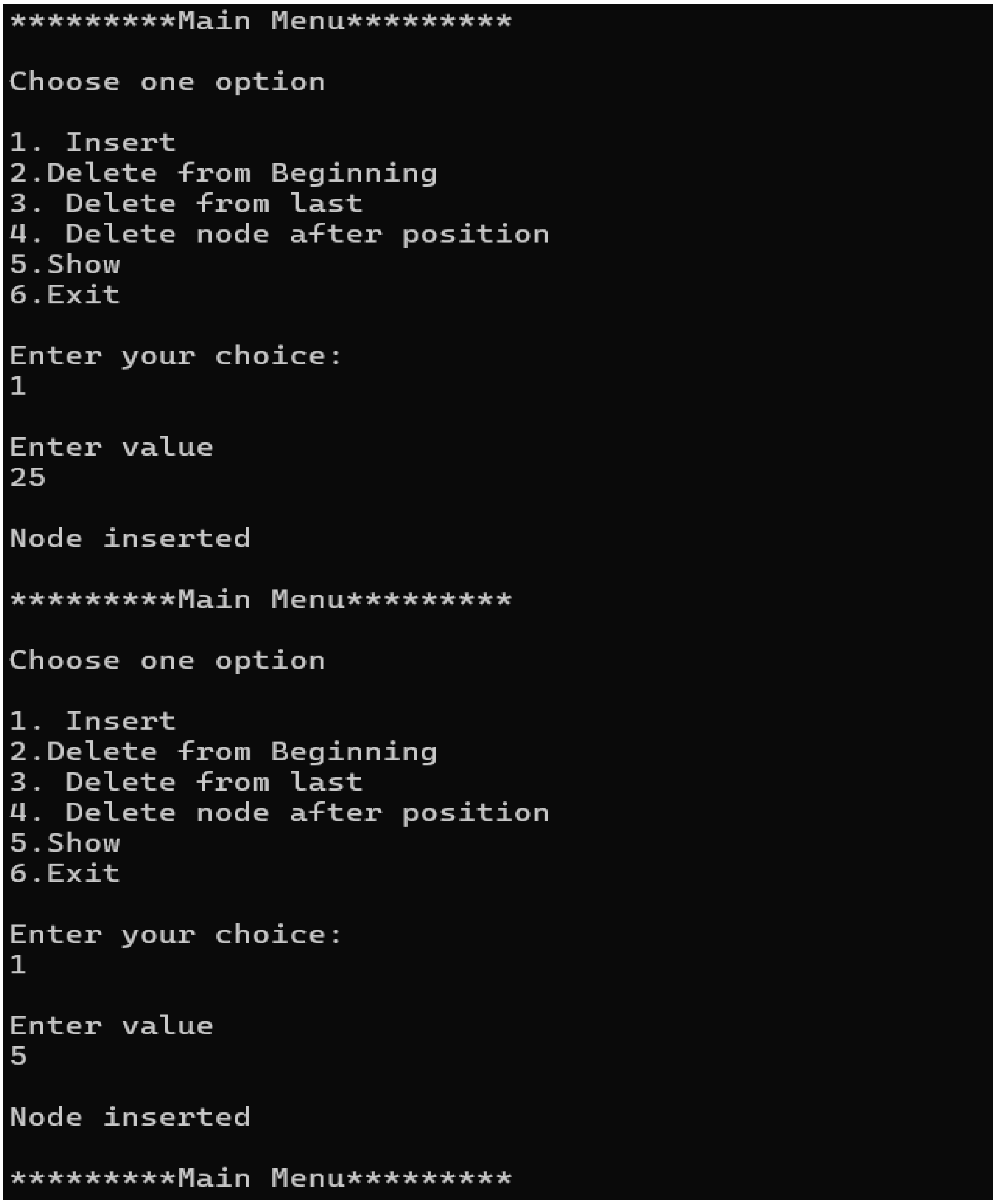
```
struct node *ptr;
int item;
ptr = (struct node *) malloc(sizeof(struct node *));
 if(ptr == NULL)
{
  printf("\nOverflow");
}
else
{
  printf("\nEnter value\n");
   scanf("%d",&item);
  ptr->data = item;
  ptr->next = head;
  head = ptr;
  printf("\nNode inserted");
}
}

void delete_at_front()
{
struct node *ptr;
if (head == NULL)
{
printf("\nList is empty\n");
}
else
{
ptr = head;
head = ptr->next;
```

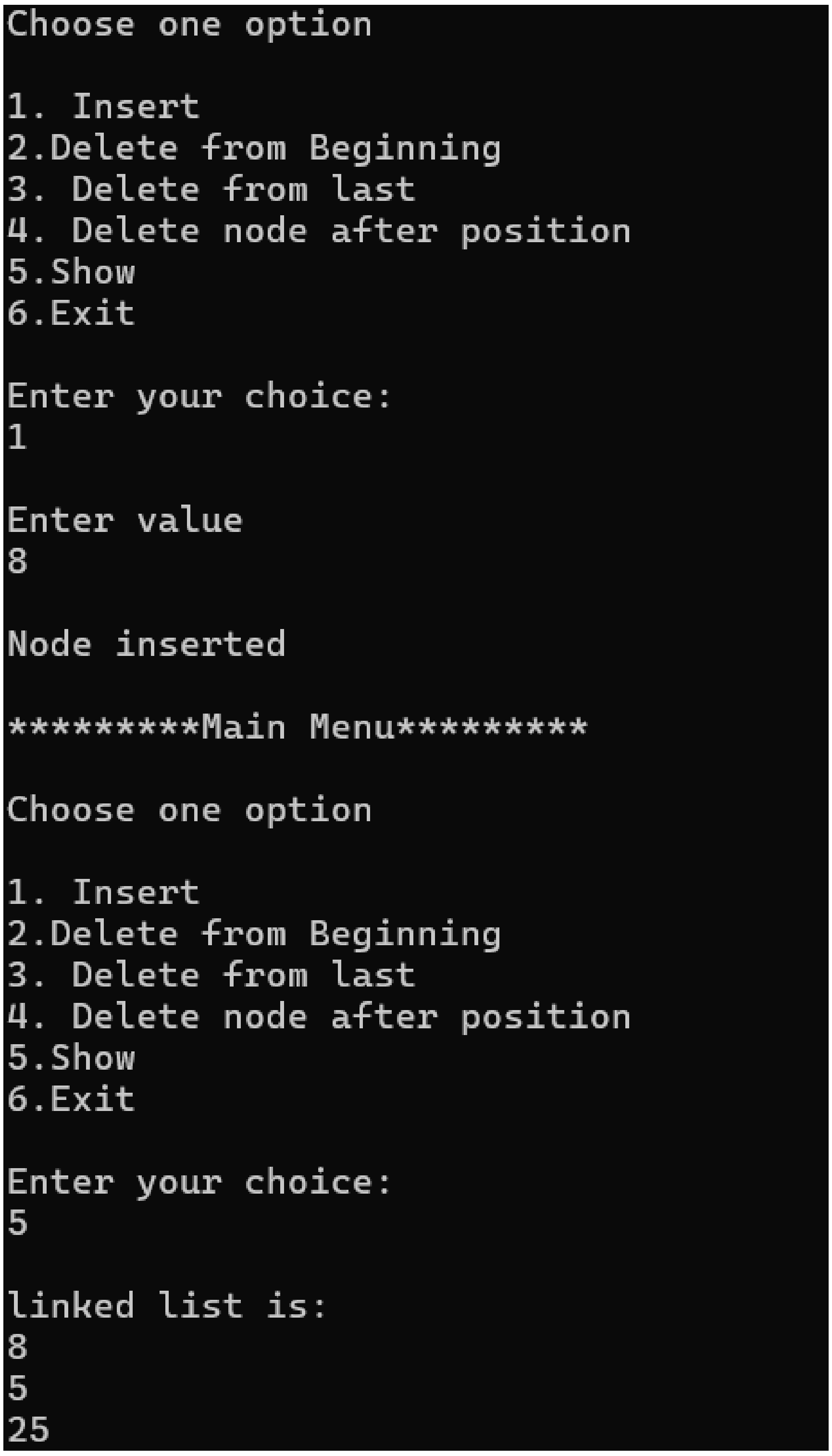
```
free(ptr);
printf("\nNode deleted from the beginning\n");
}
}

void delete_at_end()
{
struct node *ptr, *ptr1;
if(head == NULL)
{
printf("\nlist is empty");
}
else if (head -> next == NULL)
{
head = NULL;
free (head);
printf("\nOnly node deleted ...\n");
}
else
{
ptr = head;
while(ptr->next != NULL)
{
ptr1 = ptr;
ptr=ptr ->next;
}
ptr1->next = NULL;
free(ptr);
printf("\nDeleted Node from the last ...\n");
```

```
}
}

void delete_at_pos()
{
struct node *ptr, *ptr1;
int loc,i;
printf("\n Enter the location: \n");
scanf("%d", &loc);
if(head==NULL)
printf("empty list\n");
else{
   ptr=head;
   while(ptr->next!=loc)
   {
      ptr1=ptr;
      ptr=ptr->next;
   }
   ptr1->next=ptr->next;
   free(ptr);
}

}


void display()
{
struct node *ptr;
ptr = head;
```

```
if(ptr == NULL)
{

printf("empty");
}
else
{
printf("\nlinked list is:\n");
while(ptr!=NULL){
   printf("%d\n",ptr->data);
   ptr=ptr->next;
}
}
}
void main()
{
int choice =0;
while(choice != 6) {
printf("\n\n*********Main Menu*********\n");
printf("\nChoose one option\n");
printf("\n1. Insert \n2.Delete from Beginning\n3. Delete from last\n4. Delete node after
position\n5.Show\n6.Exit\n");
printf("\nEnter your choice:\n");
scanf("\n%d",&choice);
switch(choice){
case 1:
push();
break;
case 2:
delete_at_front();
break;
case 3:
delete_at_end();
break;
case 4:
delete_at_pos();
break;
case 5:
display();
break;
case 6:
exit(0);
break;
default:
   printf("\ninvalid choice.try again.");
}
}
}
```

Output


```
*********Main Menu*********

Choose one option

1. Insert
2.Delete from Beginning
3. Delete from last
4. Delete node after position
5.Show
6.Exit

Enter your choice:
1

Enter value
25

Node inserted

*********Main Menu*********

Choose one option

1. Insert
2.Delete from Beginning
3. Delete from last
4. Delete node after position
5.Show
6.Exit

Enter your choice:
1

Enter value
5

Node inserted

*********Main Menu*********
```



```
Choose one option

1. Insert
2.Delete from Beginning
3. Delete from last
4. Delete node after position
5.Show
6.Exit

Enter your choice:
1

Enter value
8

Node inserted

*********Main Menu*********

Choose one option

1. Insert
2.Delete from Beginning
3. Delete from last
4. Delete node after position
5.Show
6.Exit

Enter your choice:
5

linked list is:
8
5
25
```

Program - Leetcode platform

Remove Duplicates from Sorted List

Code

```
struct ListNode* deleteDuplicates(struct ListNode* head) {
    struct ListNode* ptr = head;
    while (ptr != NULL && ptr->next != NULL) {
        if (ptr->val == ptr->next->val) {

            struct ListNode* ptr1 = ptr->next;
            ptr->next = ptr->next->next;

        } else {

            ptr = ptr->next;
        }
    }

    return head;
}
```

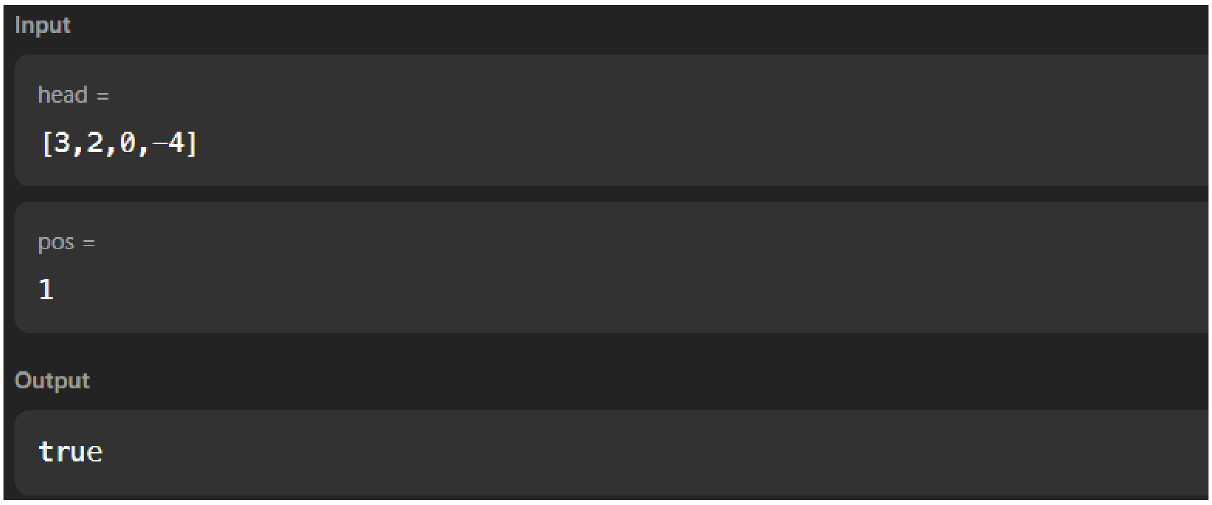
Linked List Cycle

Code

```
bool hasCycle(struct ListNode *head) {
    if (head == NULL || head->next == NULL)
        return false;

    struct ListNode* slow = head;
    struct ListNode* fast = head;

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;

        if (slow == fast)
```

```
            return true;
        }

    return false;
}
```

Output:



Input

head =

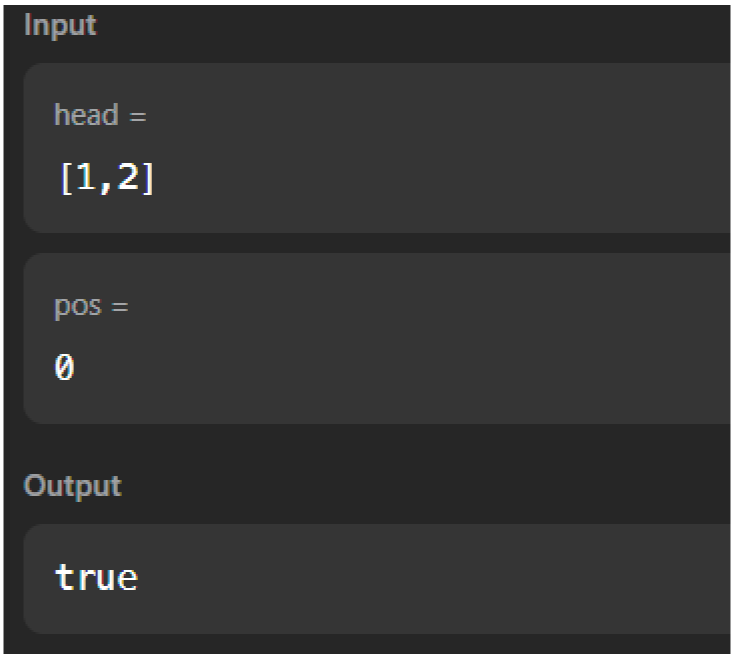
```
[3,2,0,-4]
```

pos =

```
1
```

Output

```
true
```



Input

head =

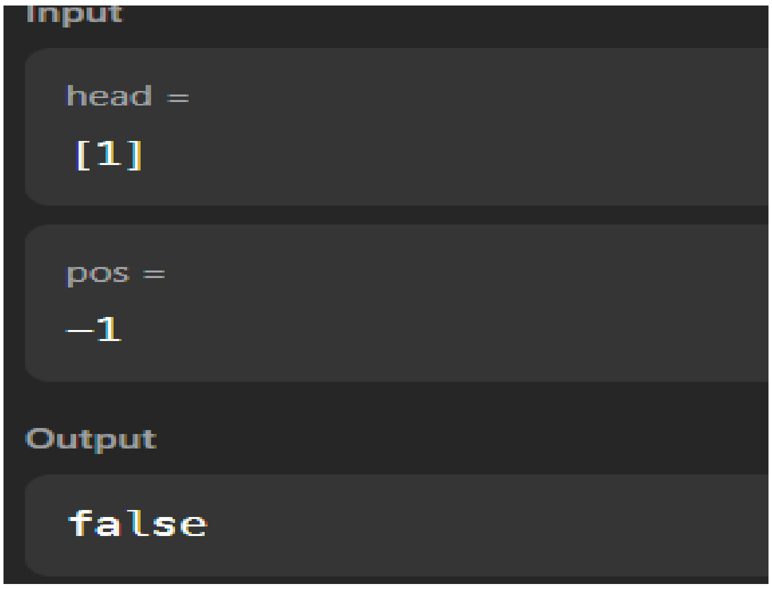
```
[1,2]
```

pos =

```
0
```

Output

```
true
```



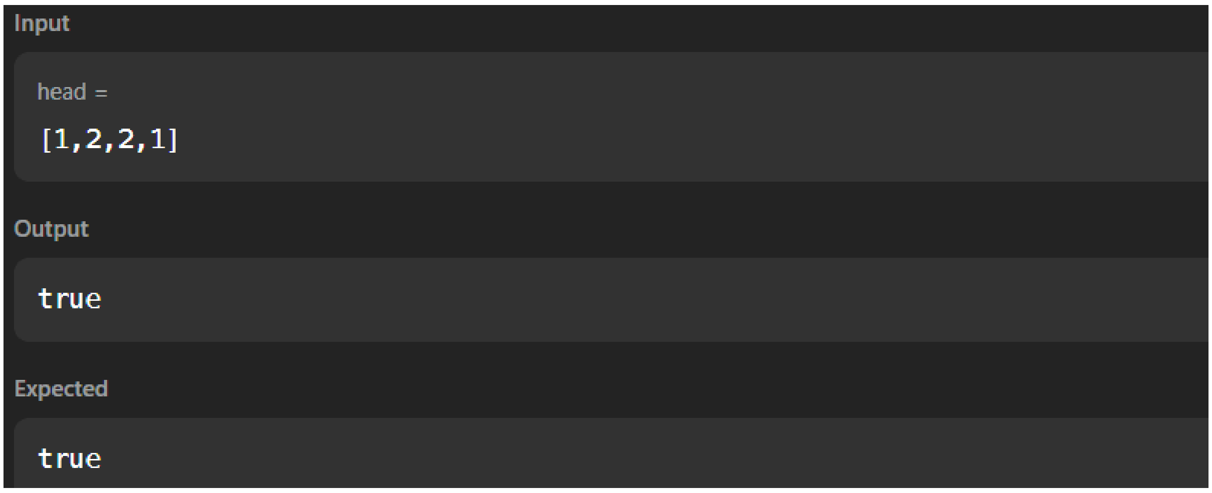
```
Input
head =
[1]
pos =
-1
Output
false
```

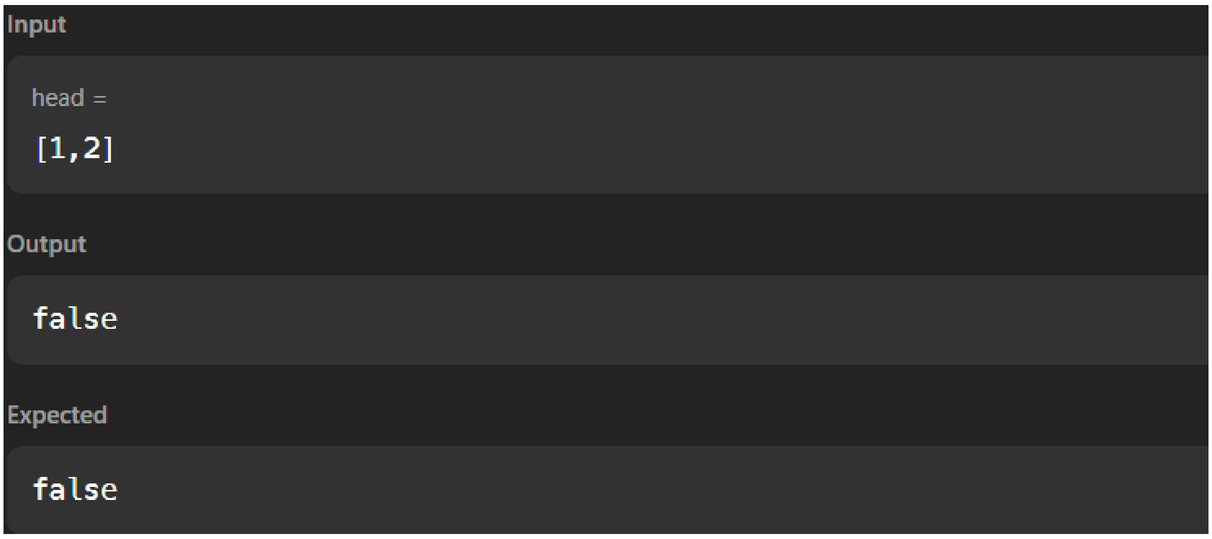
Leetcode Program

Palindrome Linked List Program

Code

```
bool isPalindrome(struct ListNode* head) {
    if (head == NULL || head->next == NULL)
        return true;

    struct ListNode *slow, *fast;
    slow = head;
    fast = head;

    while (fast->next != NULL && fast->next->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }

    struct ListNode *prev, *curr, *nxt;
    prev = NULL;
    curr = slow->next;
    nxt = slow->next;


    while (curr != NULL) {
        nxt = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nxt;
    }


    slow->next = NULL;


    struct ListNode *firstHalf = head;
    struct ListNode *secondHalf = prev;


    while (firstHalf != NULL && secondHalf != NULL) {
        if (firstHalf->val != secondHalf->val)
            return false;


        firstHalf = firstHalf->next;
        secondHalf = secondHalf->next;
    }


    return true;
}
```
Output:


```
Input
head =
[1,2,2,1]
Output
true
Expected
true
```


```
Input
head =
[1,2]
Output
false
Expected
false
```

Lab Program 06

6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

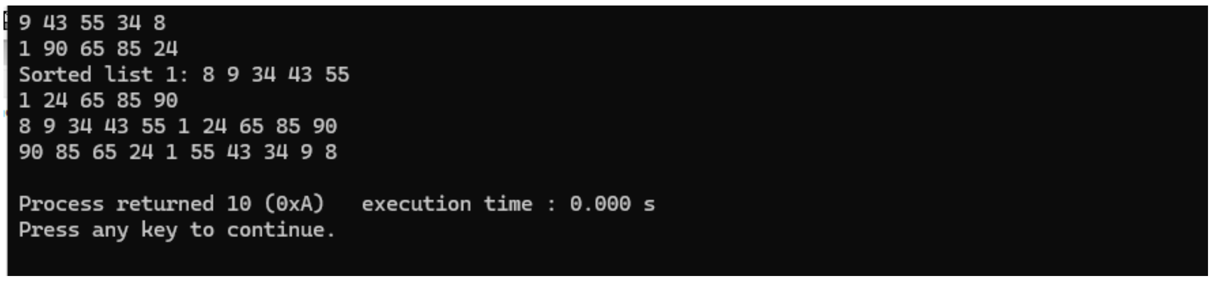
Code

```
#include<stdio.h>
#include<conio.h>
struct node{
    int data;
    struct node *next;
};
struct node *head = NULL;
struct node *head1 = NULL;
```

```
void insert(int x){
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = x;
    new_node -> next = NULL;
    if(head==NULL)
    {
        head = new_node;
    }
    else{
        struct node *temp = head;
        while(temp -> next != NULL)
            temp = temp -> next;
        temp -> next = new_node;
    }
}
void insert2(int x){
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = x;
    new_node -> next = NULL;
    if(head1==NULL)
    {
        head1 = new_node;
    }
    else{
        struct node *temp = head1;
        while(temp -> next != NULL)
            temp = temp -> next;
        temp -> next = new_node;
    }
```

```
}
struct node *sort(struct node *head)
{
        struct node *ptr = head;
        struct node *nextptr = head;
        int item;
        if(ptr == NULL)
        {
            printf("Empty");
            return;
        }
        for(ptr=head; ptr!=NULL; ptr=ptr->next)
        {
            for(nextptr=ptr->next; nextptr!=NULL; nextptr=nextptr->next)
            {
                if(ptr->data > nextptr -> data)
                {
                    item = ptr->data;
                    ptr -> data = nextptr -> data;
                    nextptr -> data = item;
                }
            }
        }
        return(head);
}
struct node *concatenate(struct node *head, struct node *head1)
{
        if(head1!=NULL && head1!=NULL)
        {
            if(head->next==NULL)
            {
               head->next=head1;
            }
            else concatenate(head->next, head1);
        }
        else printf("Either of them is empty");
        return(head);
}
struct node *reverse(struct node *head)
{
        struct node *prev = NULL;
        struct node *next = NULL;
        struct node *current = head;
        while(current!=NULL)
        {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }
        head = prev;
        return(head);
}
void display(struct node *head)
{
    struct node *ptr = head;
    while(ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
void main(){
    insert(9);
    insert(43);
    insert(55);
    insert(34);
    insert(8);
    display(head);
    insert2(1);
    insert2(90);
    insert2(65);
    insert2(85);
    insert2(24);
    display(head1);
    struct node *a = sort(head);
    printf("Sorted list 1: ");
    display(a);
    struct node *b=sort(head1);
    display(b);
    struct node *c=concatenate(a, b);
    display(c);
    struct node *d=reverse(c);
    display(d);
}
```

Output



```
9 43 55 34 8
1 90 65 85 24
Sorted list 1: 8 9 34 43 55
1 24 65 85 90
8 9 34 43 55 1 24 65 85 90
90 85 65 24 1 55 43 34 9 8

Process returned 10 (0xA)   execution time : 0.000 s
Press any key to continue.
```

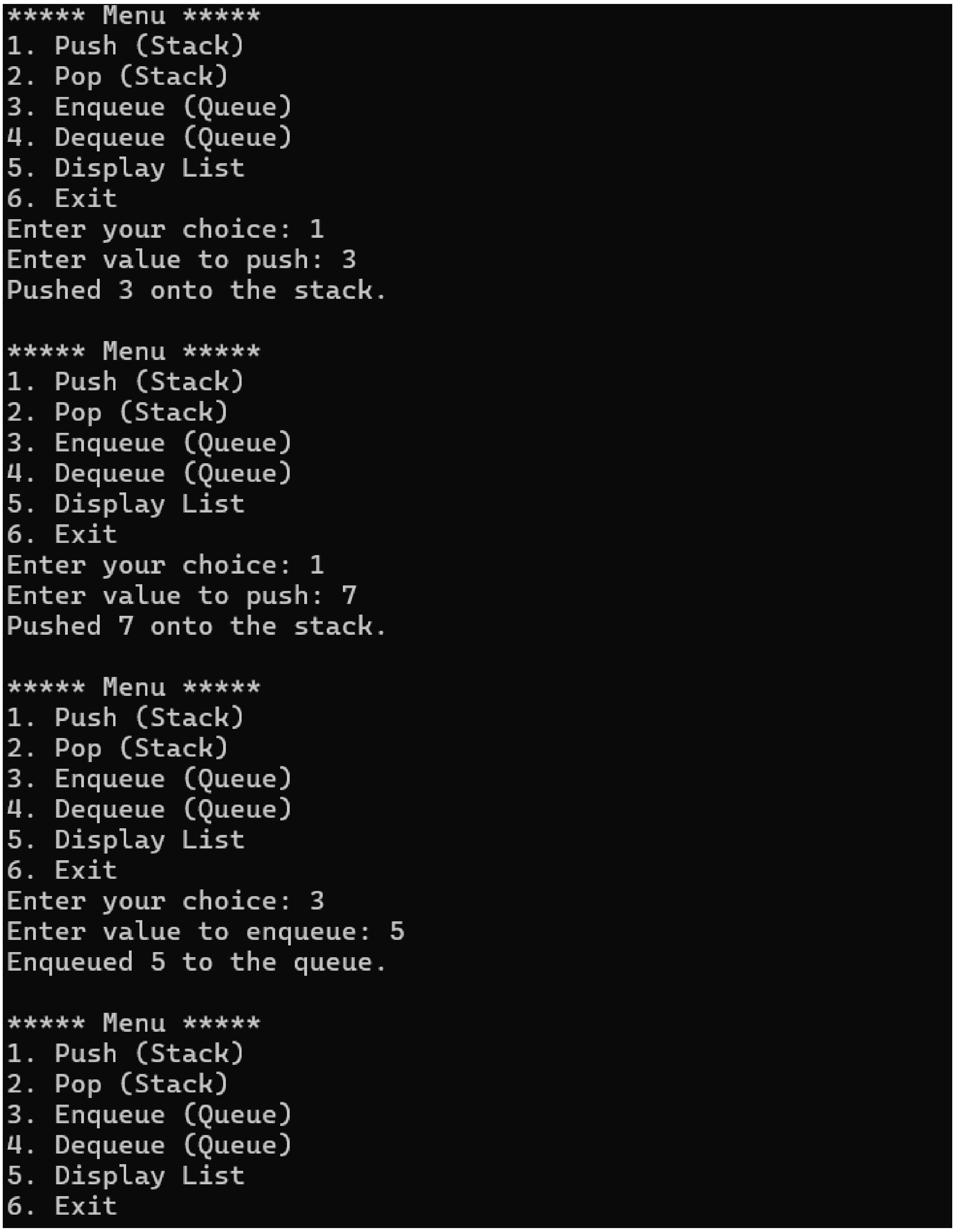
6b) WAP to Implement Single Link List to simulate Stack &amp; Queue Operations.

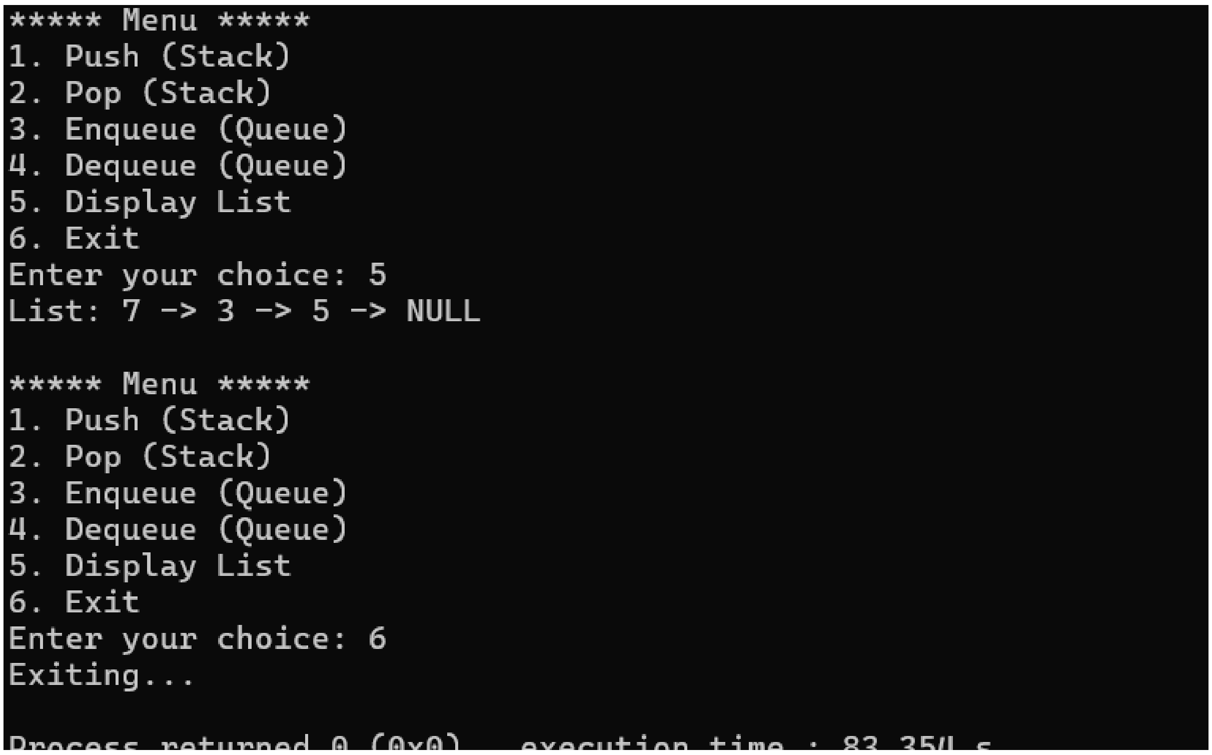
Code

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



struct node {
   int data;
   struct node* next;
};



struct node* head = NULL;



void push_stack(int value);
void pop_stack();



void enqueue_queue(int value);
void dequeue_queue();
void display();

int main() {
    int choice, value;

    while (1) {
        printf("\n***** Menu *****\n");
        printf("1. Push (Stack)\n");
        printf("2. Pop (Stack)\n");
        printf("3. Enqueue (Queue)\n");
        printf("4. Dequeue (Queue)\n");
        printf("5. Display List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push_stack(value);
                break;
            case 2:
                pop_stack();
                break;
            case 3:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue_queue(value);
                break;
            case 4:
                dequeue_queue();
                break;
            case 5:
                display();
                break;
            case 6:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}


void push_stack(int value) {
    struct node* new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = value;
    new_node->next = head;
    head = new_node;
    printf("Pushed %d onto the stack.\n", value);
}

void pop_stack() {
    if (head == NULL) {
        printf("Stack is empty.\n");
    } else {
        struct node* temp = head;
        head = head->next;
        printf("Popped %d from the stack.\n", temp->data);
        free(temp);
    }
}


void enqueue_queue(int value) {
    struct node* new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = value;
    new_node->next = NULL;

    if (head == NULL) {
        head = new_node;
    } else {
        struct node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = new_node;
    }
    printf("Enqueued %d to the queue.\n", value);
}
```

```
void dequeue_queue() {
   if (head == NULL) {
      printf("Queue is empty.\n");
   } else {
      struct node* temp = head;
      head = head->next;
      printf("Dequeued %d from the queue.\n", temp->data);
      free(temp);
   }
}


void display() {
   if (head == NULL) {
      printf("List is empty.\n");
   } else {
      struct node* temp = head;
      printf("List: ");
      while (temp != NULL) {
         printf("%d -> ", temp->data);
         temp = temp->next;
      }
      printf("NULL\n");
   }
}
```

Output:


```
***** Menu *****
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display List
6. Exit
Enter your choice: 1
Enter value to push: 3
Pushed 3 onto the stack.

***** Menu *****
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display List
6. Exit
Enter your choice: 1
Enter value to push: 7
Pushed 7 onto the stack.

***** Menu *****
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display List
6. Exit
Enter your choice: 3
Enter value to enqueue: 5
Enqueued 5 to the queue.

***** Menu *****
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display List
6. Exit
```



```
***** Menu *****
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display List
6. Exit
Enter your choice: 5
List: 7 -> 3 -> 5 -> NULL

***** Menu *****
1. Push (Stack)
2. Pop (Stack)
3. Enqueue (Queue)
4. Dequeue (Queue)
5. Display List
6. Exit
Enter your choice: 6
Exiting...

Process returned 0 (0x0)   execution time : 83.354 s
```

Lab Program 07

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

Code

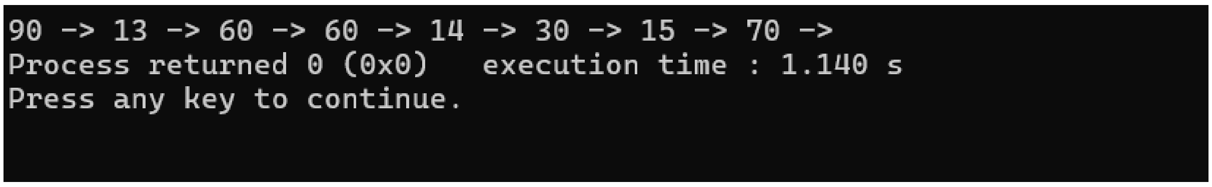
```
#include<stdio.h>

#include <stdlib.h>

#include <conio.h>


struct node

{

   int data;
```

```
    struct node *prev;
    struct node *next;
};

struct node *head=NULL;

struct node *create_node(int data){
    struct node *obj1 = (struct node*)malloc(sizeof(struct node));
    obj1-> data = data;
    obj1->prev = NULL;
    obj1->next = NULL;
    return(obj1);
}

void insert_node(int data)
{
     struct node *obj = create_node(data);
     if(head==NULL)
     {
         head = obj;
     }
     else{
         obj->next = head;
         head->prev = obj;
         head = obj;
     }
}

void append_node(int pos, int data)
```

```
{
   struct node *ptr = head;
   struct node *obj = create_node(data);
   for(int i=1;i<pos;i++)
   {
      ptr=ptr->next;
   }
   ptr->prev->next = obj;
   obj->prev=ptr->prev;
   ptr->prev=obj;
   obj->next=ptr;

}

void delete_node(int data)
{
   struct node *ptr = head;
   while(ptr->data != data && ptr!=NULL)
   {
      ptr=ptr->next;
   }
   ptr->prev->next = ptr->next;
   ptr->next->prev=ptr->prev;
   free(ptr);
}

void display(struct node *head)
{
   struct node *ptr = head;
```

```
    while(ptr!=NULL)
    {
        printf("%d -> ", ptr->data);
        ptr=ptr->next;
    }
}

void main()
{
    insert_node(70);
    insert_node(80);
    insert_node(30);
    insert_node(60);
    insert_node(60);
    insert_node(90);
    append_node(6,15);
    append_node(4,14);
    append_node(2,13);
    delete_node(80);
    display(head);
}
```

Output



```
90 -> 13 -> 60 -> 60 -> 14 -> 30 -> 15 -> 70 ->
Process returned 0 (0x0)   execution time : 1.140 s
Press any key to continue.
```

Lab Program 08

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

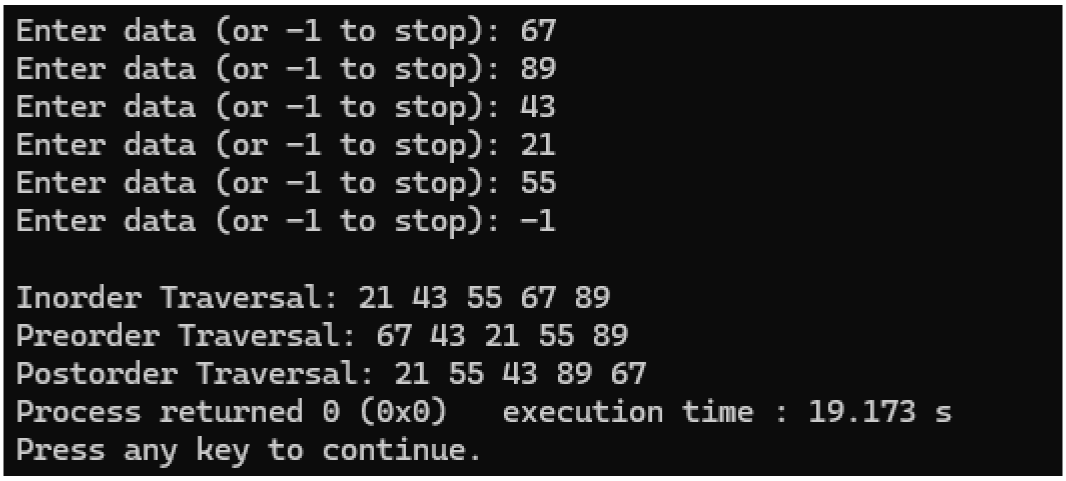
c) To display the elements in the tree.

Code

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

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

struct BST* createNode(int data) {
   struct BST* newNode = (struct BST*)malloc(sizeof(struct BST));
   newNode->data = data;
   newNode->left = newNode->right = NULL;
   return newNode;
}

struct BST* insert(struct BST* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }

   if (data < root->data) {
      root->left = insert(root->left, data);
   } else if (data > root->data) {
        root->right = insert(root->right, data);
    }

    return root;
}



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

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

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

```
}
int main() {
    struct BST* root = NULL;
    int data;

    while (1) {
        printf("Enter data (or -1 to stop): ");
        scanf("%d", &data);

        if (data == -1) {
            break;
        }

        root = insert(root, data);
    }

    printf("\nInorder Traversal: ");
    inorderTraversal(root);

    printf("\nPreorder Traversal: ");
    preorderTraversal(root);

    printf("\nPostorder Traversal: ");
    postorderTraversal(root);

    return 0;
}
```

Output:


```
Enter data (or -1 to stop): 67
Enter data (or -1 to stop): 89
Enter data (or -1 to stop): 43
Enter data (or -1 to stop): 21
Enter data (or -1 to stop): 55
Enter data (or -1 to stop): -1

Inorder Traversal: 21 43 55 67 89
Preorder Traversal: 67 43 21 55 89
Postorder Traversal: 21 55 43 89 67
Process returned 0 (0x0)   execution time : 19.173 s
Press any key to continue.
```

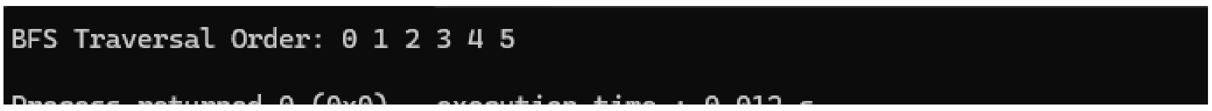
Lab Program 09

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

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 100




typedef struct {

    int items[MAX];

    int front;

    int rear;

} Queue;


typedef struct {

    int vertices;

    int adjMatrix[MAX][MAX];

} Graph;




void initializeQueue(Queue* q) {
    q->front = -1;
    q->rear = -1;
}


int isEmpty(Queue* q) {
    return q->front == -1;
}


void enqueue(Queue* q, int value) {
    if (q->rear == MAX - 1) {
        printf("Queue overflow\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->items[q->rear] = value;
}


int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue underflow\n");
        return -1;
    }
    int item = q->items[q->front];
    if (q->front == q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return item;
}



void bfs(Graph* graph, int startVertex) {
    int visited[MAX] = {0};
    Queue q;
    initializeQueue(&q);
    printf("BFS Traversal Order: ");
    enqueue(&q, startVertex);
    visited[startVertex] = 1;

    while (!isEmpty(&q)) {
        int currentVertex = dequeue(&q);
        printf("%d ", currentVertex);



        for (int i = 0; i < graph->vertices; i++) {
            if (graph->adjMatrix[currentVertex][i] == 1 && !visited[i]) {
                enqueue(&q, i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}
```

```
int main() {
    Graph graph;
    graph.vertices = 6;
    for (int i = 0; i < graph.vertices; i++) {
        for (int j = 0; j < graph.vertices; j++) {
            graph.adjMatrix[i][j] = 0;
        }
    }



    graph.adjMatrix[0][1] = 1;
    graph.adjMatrix[0][2] = 1;
    graph.adjMatrix[1][0] = 1;
    graph.adjMatrix[1][3] = 1;
    graph.adjMatrix[1][4] = 1;
    graph.adjMatrix[2][0] = 1;
    graph.adjMatrix[2][5] = 1;
    graph.adjMatrix[3][1] = 1;
    graph.adjMatrix[4][1] = 1;
    graph.adjMatrix[4][5] = 1;
    graph.adjMatrix[5][2] = 1;
    graph.adjMatrix[5][4] = 1;



    bfs(&graph, 0);


    return 0;
}
```

Output:



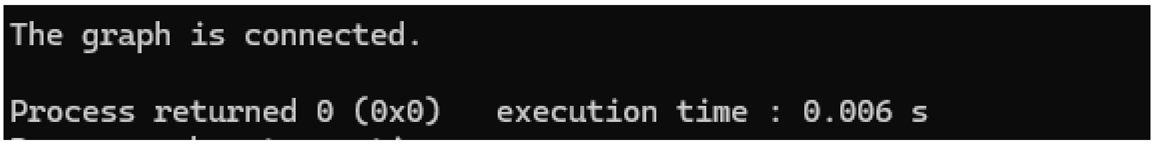
```
BFS Traversal Order: 0 1 2 3 4 5

Process returned 0 (0x0)   execution time : 0.012 s
```

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

Code:

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

#define MAX 100

int adjMatrix[MAX][MAX];
bool visited[MAX];

void dfs(int vertex, int vertices) {
    visited[vertex] = true;

    for (int i = 0; i < vertices; i++) {
        if (adjMatrix[vertex][i] == 1 && !visited[i]) {
            dfs(i, vertices);
        }
    }
}

bool isConnected(int vertices) {
    for (int i = 0; i < vertices; i++) {
        visited[i] = false;
    }

    dfs(0, vertices);

    for (int i = 0; i < vertices; i++) {
        if (!visited[i]) {
            return false;
        }
    }
    return true;
}

int main() {
    int vertices = 6;
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            adjMatrix[i][j] = 0;
        }
    }
    adjMatrix[0][1] = 1;
    adjMatrix[1][0] = 1;
    adjMatrix[1][2] = 1;
    adjMatrix[2][1] = 1;
    adjMatrix[2][3] = 1;
    adjMatrix[3][2] = 1;
    adjMatrix[3][4] = 1;
    adjMatrix[4][3] = 1;
    adjMatrix[4][5] = 1;
    adjMatrix[5][4] = 1;
    if (isConnected(vertices)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

Output:



```
The graph is connected.

Process returned 0 (0x0)   execution time : 0.006 s
```

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

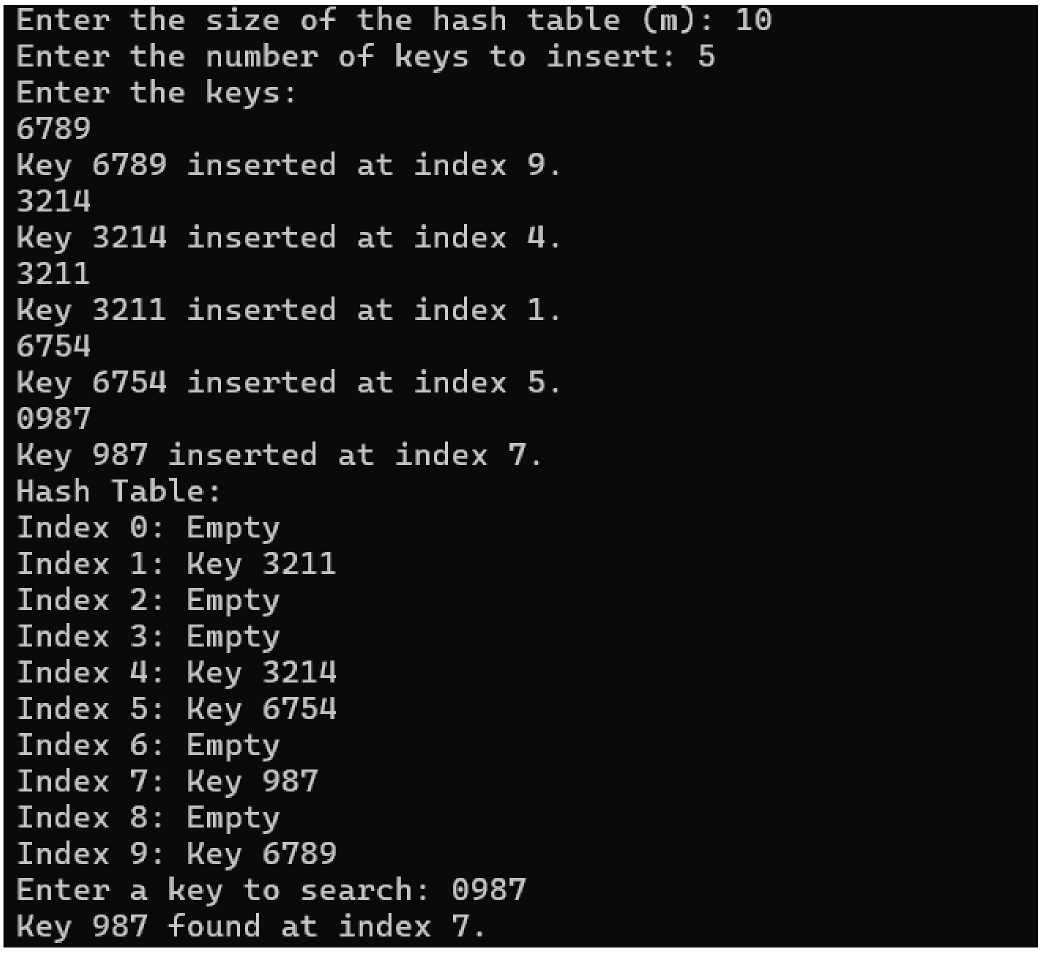
map a given key K to the address space L.

Resolve the collision (if any) using linear probing.

Code:

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

#define MAX 100

typedef struct {
   int key;
   int isOccupied;
} HashTable;
void initializeTable(HashTable table[], int m) {
   for (int i = 0; i < m; i++) {
        table[i].key = -1;
        table[i].isOccupied = 0;
    }
}
int hashFunction(int key, int m) {
    return key % m;
}
void insertKey(HashTable table[], int m, int key) {
    int index = hashFunction(key, m);
    int originalIndex = index;
    while (table[index].isOccupied) {
        index = (index + 1) % m;
        if (index == originalIndex) {
            printf("Hash table is full. Cannot insert key %d.\n", key);
            return;
        }
    }

    table[index].key = key;
    table[index].isOccupied = 1;
    printf("Key %d inserted at index %d.\n", key, index);
}
int searchKey(HashTable table[], int m, int key) {
    int index = hashFunction(key, m);
    int originalIndex = index;
    while (table[index].isOccupied) {
        if (table[index].key == key) {
            return index;
        }
        index = (index + 1) % m;
        if (index == originalIndex) {
            break;
        }
    }

    return -1;
}

void displayTable(HashTable table[], int m) {
    printf("Hash Table:\n");
    for (int i = 0; i < m; i++) {
        if (table[i].isOccupied) {
            printf("Index %d: Key %d\n", i, table[i].key);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}
int main() {
    int m, n;
    printf("Enter the size of the hash table (m): ");
    scanf("%d", &m);

    if (m > MAX) {
        printf("Size exceeds the maximum allowed size of %d.\n", MAX);
        return 1;
    }

    HashTable table[m];
    initializeTable(table, m);

    printf("Enter the number of keys to insert: ");
    scanf("%d", &n);


    printf("Enter the keys:\n");
    for (int i = 0; i < n; i++) {
        int key;
        scanf("%d", &key);
        insertKey(table, m, key);
    }


    displayTable(table, m);


    printf("Enter a key to search: ");
    int searchKeyInput;
    scanf("%d", &searchKeyInput);


    int foundIndex = searchKey(table, m, searchKeyInput);
    if (foundIndex != -1) {
        printf("Key %d found at index %d.\n", searchKeyInput, foundIndex);
    } else {
        printf("Key %d not found in the hash table.\n", searchKeyInput);
    }


    return 0;
}
```

Output:



```
Enter the size of the hash table (m): 10
Enter the number of keys to insert: 5
Enter the keys:
6789
Key 6789 inserted at index 9.
3214
Key 3214 inserted at index 4.
3211
Key 3211 inserted at index 1.
6754
Key 6754 inserted at index 5.
0987
Key 987 inserted at index 7.
Hash Table:
Index 0: Empty
Index 1: Key 3211
Index 2: Empty
Index 3: Empty
Index 4: Key 3214
Index 5: Key 6754
Index 6: Empty
Index 7: Key 987
Index 8: Empty
Index 9: Key 6789
Enter a key to search: 0987
Key 987 found at index 7.
```