# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT**
**on**

# Data Structures using C

***Submitted by***


**SAKSHI B R**
**(1BM22CS233)**


***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**


**June-2023 to September-2023**


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

## Department of Computer Science and Engineering

# CERTIFICATE

This is to certify that the Lab work entitled "**Data Structures using C**" carried out by **SAKSHI B R (1BM22CS233),** 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 academic semester December- 2023 to March-2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **Data Structures using C (23CS3PCDST)** work prescribed for the said degree.


Radhika A D :
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

# Index Sheet

# Index Sheet

| | | |
|---|---|---|
| 5 | 5a) WAP to Implement Singly Linked List with following opera,ons<br>Create a linked list.<br>Dele,on of first element, specified element and last element in the list.<br>Display the contents of the linked list.<br><br>5b) Program - Leetcode pla]orm | 24-30 |
| 6 | 6a) WAP to Implement Single Link List with following opera,ons: Sort the linked list, Reverse the linked list, Concatena,on of two linked lists.<br><br>6b) WAP to Implement Single Link List to simulate Stack & Queue Opera,ons. | 31-38 |
| 7 | 7a) WAP to Implement doubly link list with primi,ve opera,ons<br>Create a doubly linked list.<br>Insert a new node to the leh of the node. Delete<br>the node based on a specific value<br>Display the contents of the list<br><br>7b) Program - Leetcode pla]orm | 39-43 |
| 8 | 8a) Write a program<br>To construct a binary Search tree.<br>To traverse the tree using all the methods i.e., in-order, preorder and post order<br>To display the elements in the tree.<br><br>8b) Program - Leetcode pla]orm | 44-48 |
| 9 | 9a) Write a program to traverse a graph using BFS method.<br>9b) Write a program to check whether given graph is connected or not using DFS method. | 49-52 |

## Course Outcome

| | |
|---|---|
| CO1 | Apply the concept of linear and nonlinear data structures. |
| CO2 | Analyse data structure opera,ons for a given problem. |
| CO3 | CO3 Design and implement opera,ons of linear and nonlinear data structure. |
| CO4 | Conduct prac,cal experiments for demonstra,ng the opera,ons of different data structures and sor,ng techniques. |

## Course Outcome

| | |
|---|---|
| CO1 | Apply the concept of linear and nonlinear data structures. |
| CO2 | Analyse data structure operations for a given problem. |
| CO3 | CO3 Design and implement operations of linear and nonlinear data structure. |
| CO4 | Conduct practical experiments for demonstrating the operations of different data structures and sorting techniques. |

1a). 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.

Code:

```
stack underflow #include <stdio.h> int top=-
1; void push(int arr[],int value){ if(top>4){
printf("\nstack overflow cant insert the
element");
}
else{ ++top; arr[top]=value;
printf("\nSuccessfully pushed the element
%d",arr[top]);
}}
void pop(int arr[]){ if(top==-1){
printf("\nstack underflow no element to
pop\n");
}
else{ printf("\none Element was popped
%d\n",arr[top]);
   top--;
}}
void display(int arr[]){
printf("The elements are
```

```
\n"); for(int i=top;i>=0;i--){
printf("%d\n",arr[i]);}} void
main(){
   int stack[5];    void operations(){    int choice;    int value;    printf("\n
Enter your choice \n");    printf("Enter 1 to push\n Enter 2 to pop\n
Enter 3 to display\n Enter 4 for exit\n");    scanf("%d",&choice);
switch(choice){ case 1:
   printf("Enter the element to
push ");    scanf("%d",&value);
push(stack,value);
operations();    break; case 2:

pop(stack);
operations();
break; case
3:
   display(stack);
operations();    break;
case 4:    printf("You
have exited");
   }}
   operations();}
```

Output:

Overflow:

```
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 10

Successfully pushed the element 10
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 60

Successfully pushed the element 60
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 70

Successfully pushed the element 70
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 66

Successfully pushed the element 66
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 55

Successfully pushed the element 55
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 99

stack overflow cant insert the element
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
```

Underflow:

```
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 4

Successfully pushed the element 4
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
3
The elements are
4

 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
2

one Element was popped 4

 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
2

stack underflow no element to pop

 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 9

Successfully pushed the element 9
 Enter your choice
Enter 1 to push
 Enter 2 to pop
 Enter 3 to display
 Enter 4 for exit
1
Enter the element to push 10

Successfully pushed the element 10
 Enter your choice
Enter 1 to push
 Enter 2 to pop
```

2a). 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)

Code:

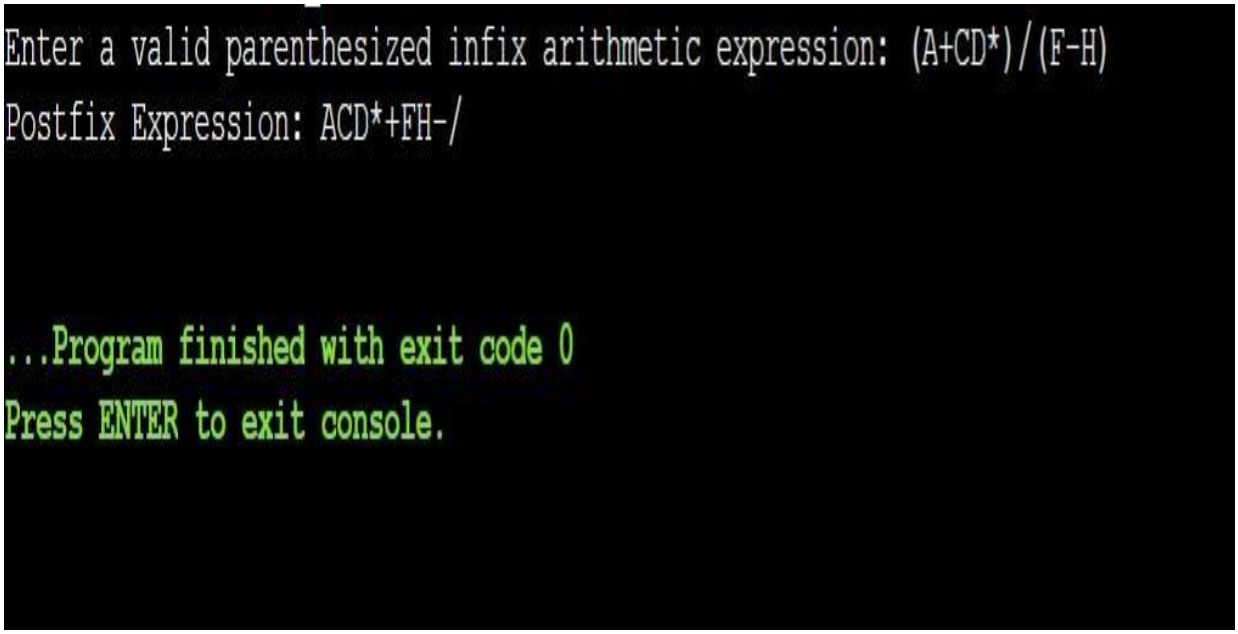
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
char stack[MAX_SIZE];
int top = -1; void push(char
item) {     if (top ==
MAX_SIZE - 1) {
printf("Stack Overflow\n");
        exit(1);
    }
    stack[++top] = item;}
char pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
        exit(1);
    }
    return stack[top--];} int isOperator(char ch) {     return
(ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch ==
'^');
}
int precedence(char ch) {
    switch (ch) {
```

```
      case '^':
return 3;
case '*':
case '/':
return 2;
case '+':
case '-':
return 1;
default:
return -1;
   }}

void infixToPostfix(char infix[])
{    char postfix[MAX_SIZE];
   int i, j = 0;    for (i = 0;
infix[i] != '\0'; i++) {        if
(isalnum(infix[i])) {
postfix[j++] = infix[i];        }
else if (infix[i] == '(') {
push(infix[i]);        } else if
(infix[i] == ')') {
          while (top != -1 && stack[top]
!= '(') {              postfix[j++] = pop();
        }
          if (top != -1 && stack[top] == '(')
{
```

```
            pop();
        } else {
            printf("Invalid expression\n");
            exit(1);
        }
    } else if (isOperator(infix[i])) {
        while (top != -1 && precedence(stack[top]) >= precedence(infix[i])) {
            postfix[j++] = pop();
        }
        push(infix[i]);
    } else {
        printf("Invalid character in expression\n");
        exit(1);
    }}
  while (top != -1) {       if
(stack[top] == '(') {
printf("Invalid expression\n");
        exit(1);
    }
    postfix[j++] = pop(); }
postfix[j] = '\0';
  printf("Postfix Expression: %s\n",
postfix);
}
```

```
int main() {     char infix[MAX_SIZE];     printf("Enter a
valid parenthesized infix arithmetic expression: ");
scanf("%s", infix);     infixToPostfix(infix);     return 0;
}
```

Output:



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>
int rear = -1;
int front = -1;
int max = 5;

void Enqueue(int arr[], int
*value) {     if (rear == -1 ||
front == -1) {         rear++;
```

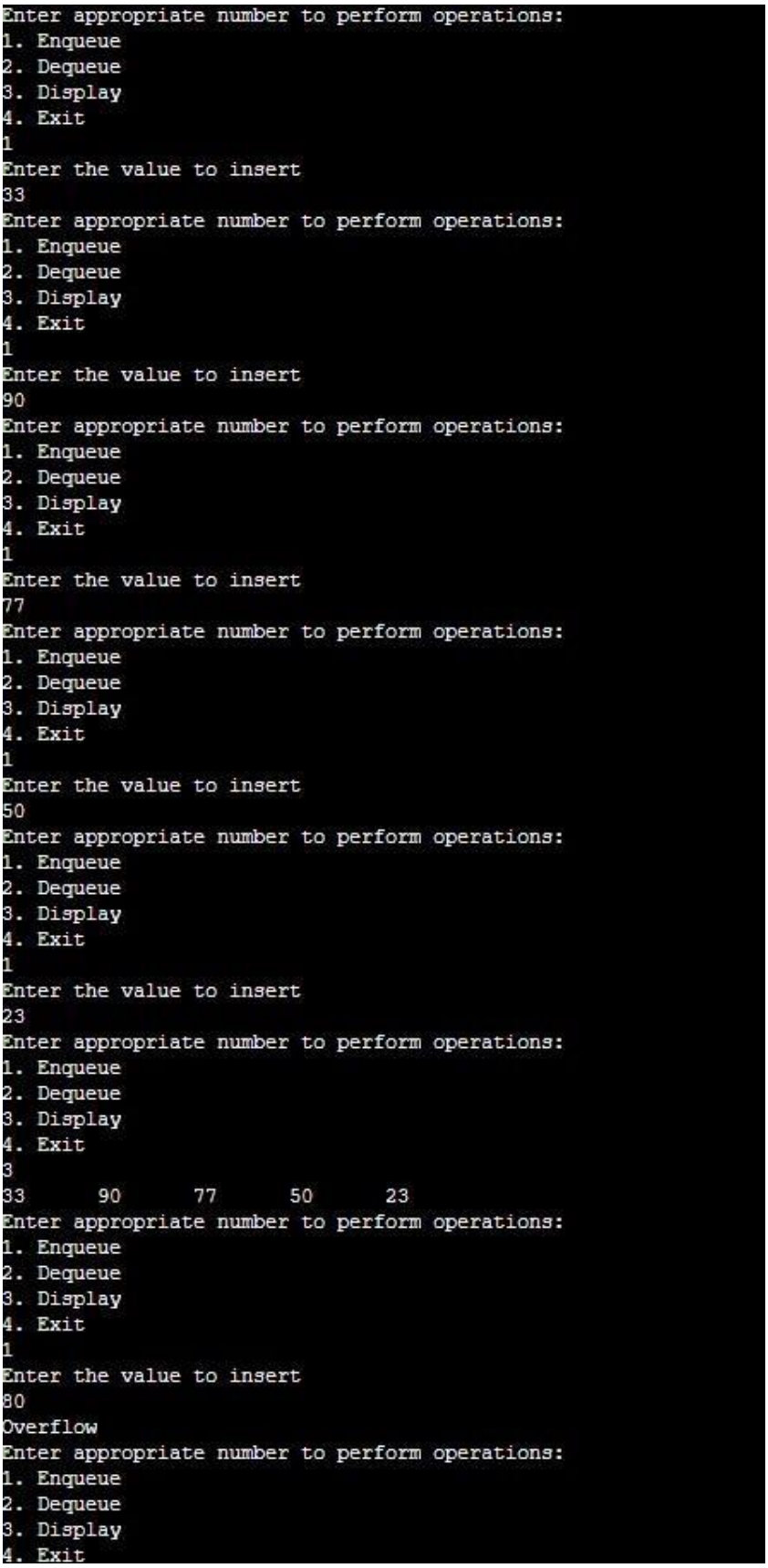
```
front++;        arr[rear] =
*value;        rear++;
   } else if (rear == max) {
printf("Overflow\n");
   } else {
arr[rear] = *value;
rear++;
   }}

void Dequeue(int arr[]) {
   if (front == -1 || rear == -1) {
printf("Underflow\n");
   } else if (front == (rear - 1)) {
      printf("Deleted element = %d\n", arr[front]);
      rear = -1;
front = -1;    }
else {
      int temp = arr[front];
front++;
      printf("Deleted element = %d\n",
temp);
   }
}
```

```
void display(int arr[]) {     for
(front; front <rear; front++) {
printf("%d\t", arr[front]);
   }
   printf("\n");
}

int main() {
int choice;
int arr[5];
int value;

   void operations() {
      printf("Enter appropriate number to perform operations: \n1. Enqueue \n2.
Dequeue
\n3.      Display      \n4.
Exit\n");
scanf("%d",   &choice);
switch (choice) {
          case 1:
             printf("Enter the value to
insert\n");              scanf("%d",
&value);

             Enqueue(arr,
&value);
operations();              break;
```

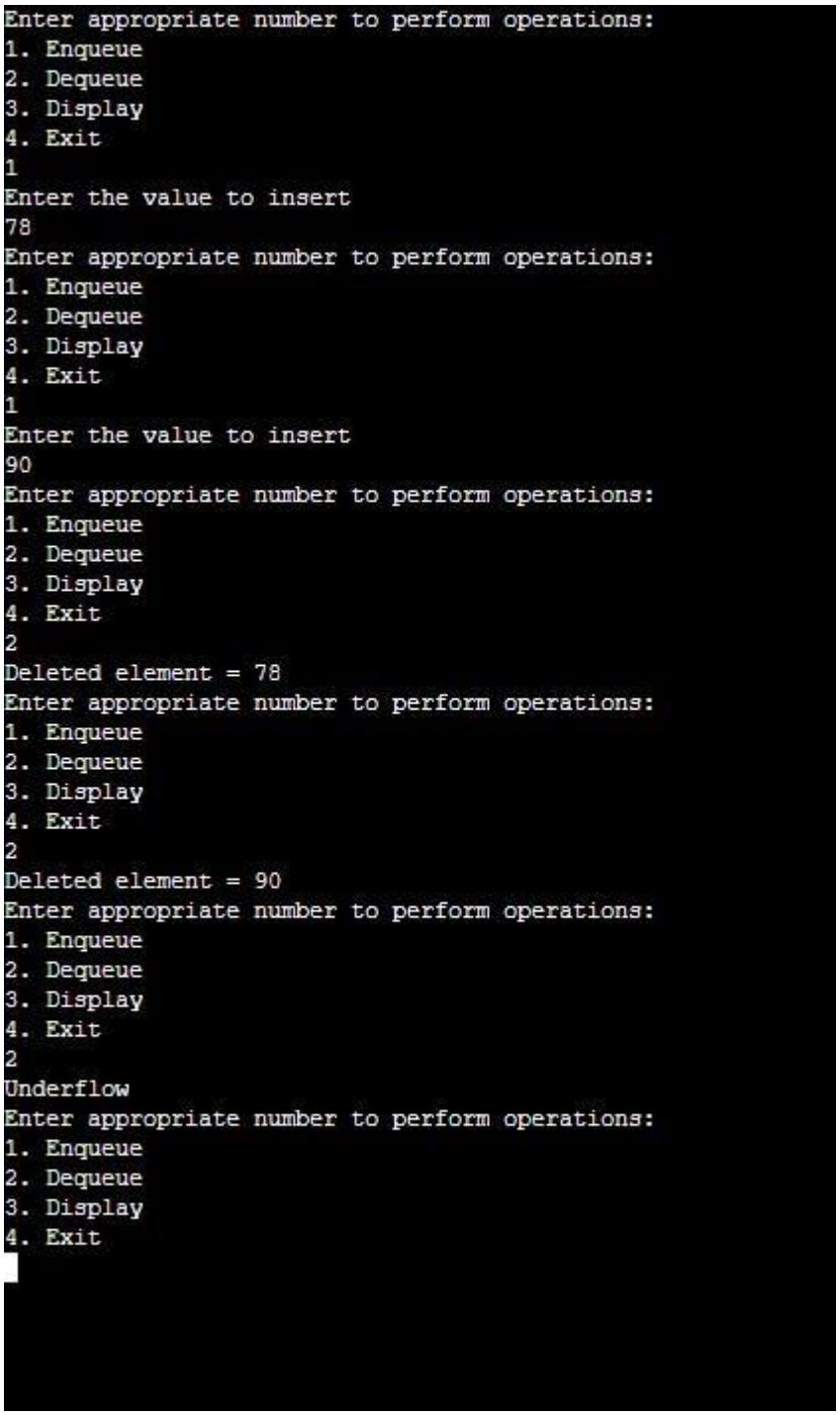
```
case 2:
Dequeue(arr);
operations();              break;
case 3:              display(arr);
operations();              break;
case 4:
printf("Exited\n");
break;          default:
printf("Invalid choice\n");
operations();              break;
      }}
   operations();

   return 0;
}
```

Output:

Overflow:


Enter appropriate number to perform operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
1
Enter the value to insert
33
Enter appropriate number to perform operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
1
Enter the value to insert
90
Enter appropriate number to perform operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
1
Enter the value to insert
77
Enter appropriate number to perform operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
1
Enter the value to insert
50
Enter appropriate number to perform operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
1
Enter the value to insert
23
Enter appropriate number to perform operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
3
33 90 77 50 23
Enter appropriate number to perform operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
1
Enter the value to insert
80
Overflow
Enter appropriate number to perform operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit

**Underflow:**




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

program should print appropriate messages for queue empty and queue overflow conditions Code:

Circular queue:

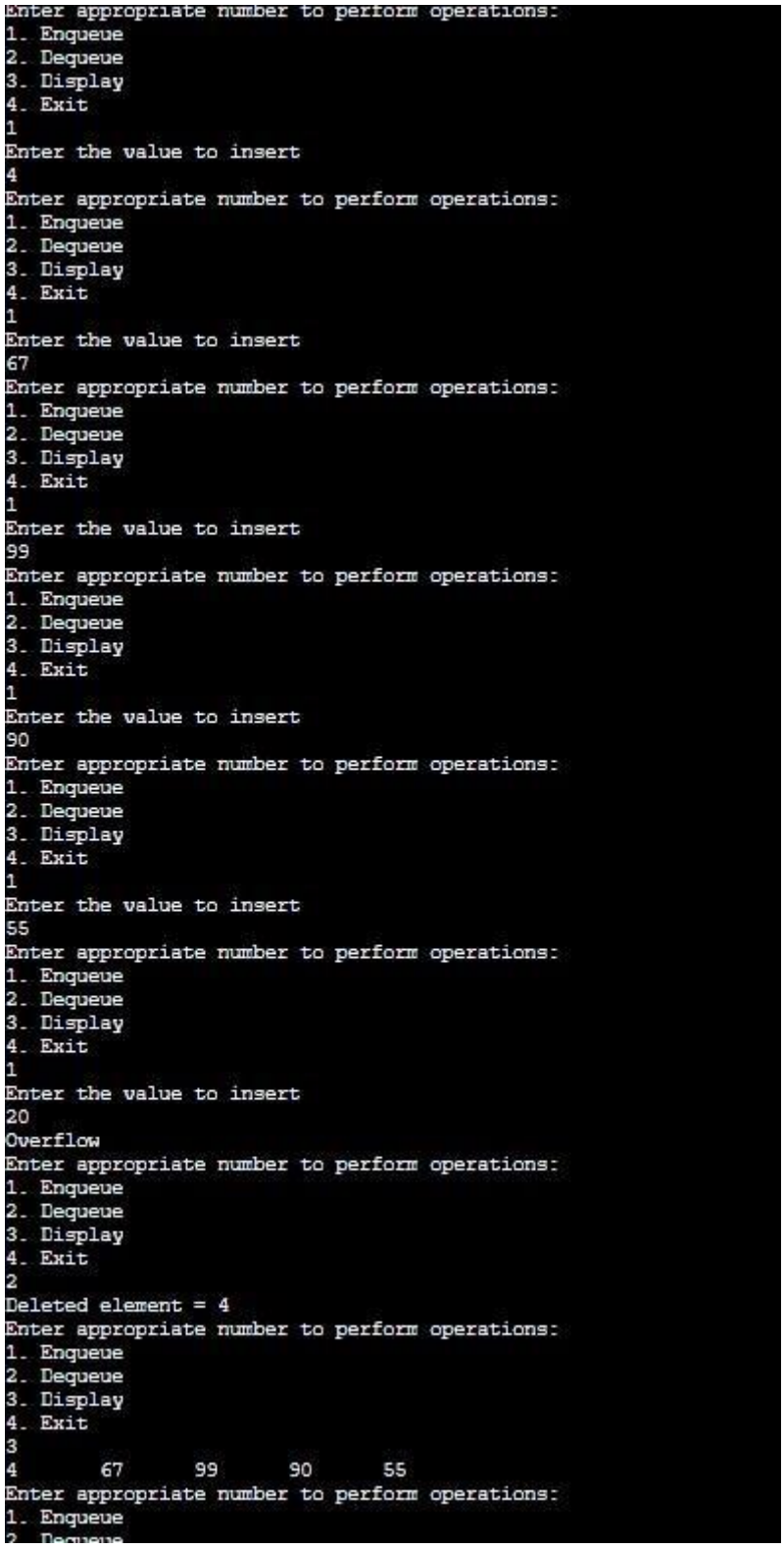
```
#include <stdio.h>
int rear = -1;
int front = -1;
int max = 5;

void Enqueue(int arr[], int
value) {    if (rear == -1 ||
front == -1) {       rear++;
front++;       arr[rear] =
value;       rear++;
   }
   else if (rear ==
max) {
if(front!=0){
rear=0;
arr[rear]=value;
rear++;
      }
      else{
      printf("Overflow\n");
      }}
```

```
   else if(rear==(front)){
printf("overflow");
          }
   else {
arr[rear] = value;
rear++;
   }}

void Dequeue(int arr[]) {
   if (front == -1 || rear ==
-1) {
printf("Underflow\n");    }
else if (front == (rear - 1))
{
      printf("Deleted element = %d\n", arr[front]);
      rear = -1;
front = -1;    }
else {
      int temp = arr[front];
front++;
      printf("Deleted element = %d\n",
temp);
   }}

void display(int arr[]) {
for (int i=0; i <max;
```

```
i++) {
printf("%d\t", arr[i]);
   }
   printf("\n");
}
int main() {    int
choice;    int
arr[5];    int
value;    void
operations() {
printf("Enter
appropriate
number to
perform
operations: \n1.
Enqueue \n2.
Dequeue
\n3.     Display     \n4.
Exit\n");
scanf("%d",  &choice);
switch (choice) {
          case 1:
             printf("Enter the value to
insert\n");              scanf("%d",
&value);              Enqueue(arr, value);
```

```
operations();               break;
case 2:

Dequeue(arr);
operations();
break;          case
3:
            display(arr);
operations();
break;          case 4:
printf("Exited\n");
break;          default:
            printf("Invalid
choice\n");
operations();               break;
      } }

   operations();

   return 0;}
```

Output:




4a). 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:

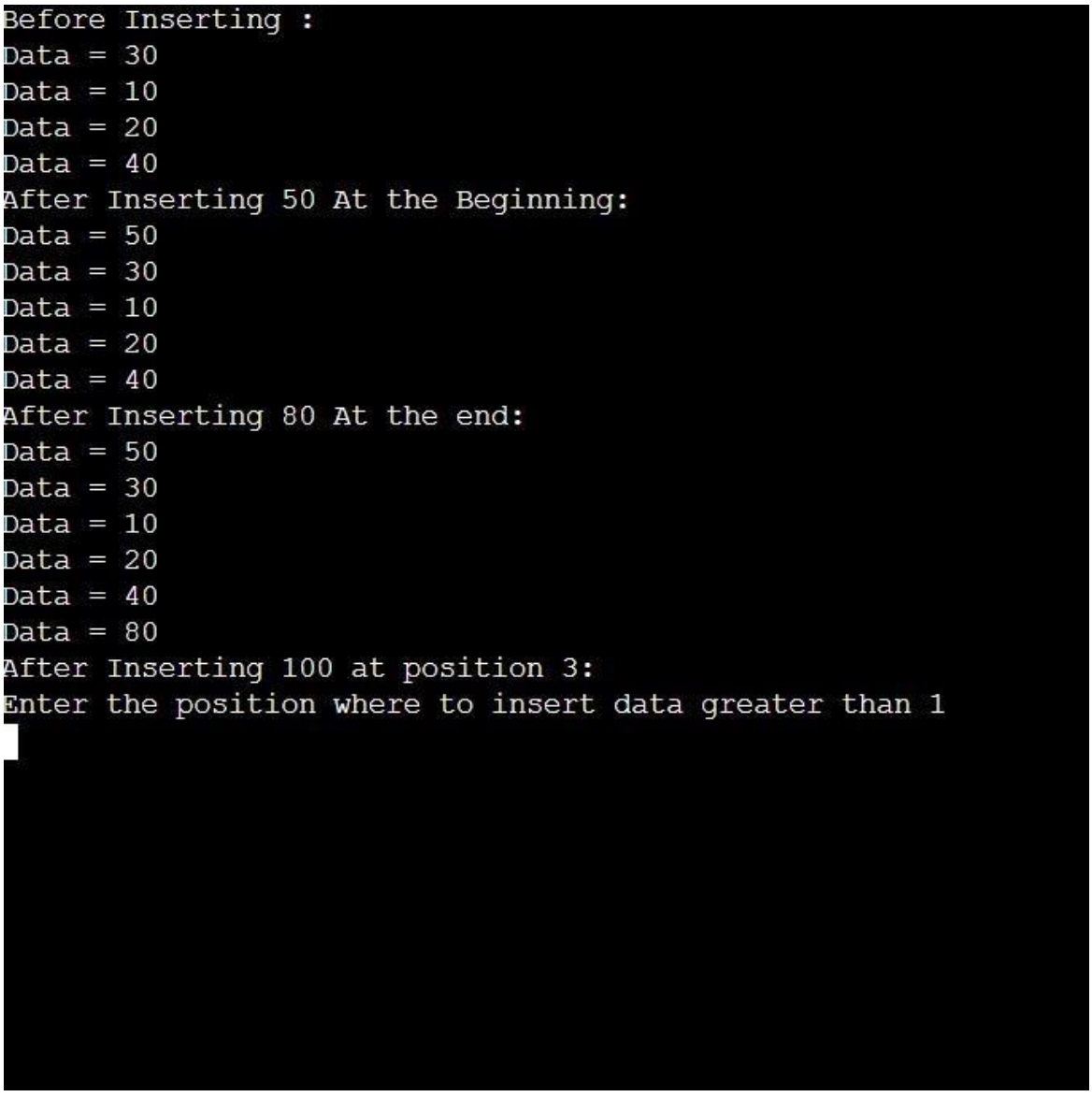
```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node* next;
};

void insert_at_beg(struct node** head, int data) {     struct node* new_node = (struct node*)malloc(sizeof(struct node));     new_node->data = data;
   if (*head == NULL) {
*head = new_node;
new_node->next = NULL;
   } else {
      new_node->next = *head;
      *head = new_node;
   }}

void insert_in_between(struct node** head,int data){
int pos; printf("Enter the position where to insert data greater than 1\n"); scanf("%d",&pos); int count=1;
```

```
struct node* ptr=*head;
if(*head==NULL){
   printf("No Nodes cant insert at
position");
}
while(ptr->next!=NULL){
   if(count==(pos-1)){        struct node* new_node=(struct
node*)malloc(sizeof(struct node));        new_node-
>data=data;        new_node->next=ptr->next;        ptr-
>next=new_node;
       return;
   }
   count++; ptr=ptr->next; if(ptr->next==NULL){
printf("reached the end node cant insert at specified
node \n");
   return;
}}}
void insert_at_end(struct node** head,int data){     struct
node* new_node=(struct node*)malloc(sizeof(struct
node));     new_node->data=data;     new_node-
>next=NULL; if(*head==NULL){    *head=new_node;
  return;
}
struct node* ptr=*head; while(ptr-
>next!=NULL){
   ptr=ptr->next;}
```

```
ptr->next=new_node;} void
display(struct node** head)
{    if(*head==NULL){
printf("No nodes\n");
      return;}
   struct node* ptr = *head;
while (ptr!= NULL) {
printf("Data = %d\n", ptr-
>data);        ptr = ptr->next;
   }}
int main() {    struct node* head = NULL;
insert_at_beg(&head,10);
insert_at_end(&head,20);
insert_at_beg(&head,30);
insert_at_end(&head,40);
printf("Before Inserting :\n");
display(&head);     printf("After Inserting
50 At the Beginning:\n");
insert_at_beg(&head,50);
display(&head);    printf("After Inserting
80 At the end:\n");
insert_at_end(&head,80);
display(&head);    printf("After Inserting
100 at position 3:\n");
```

```
insert_in_between(&head,100);

display(&head);

   return 0;

}
```

Output:



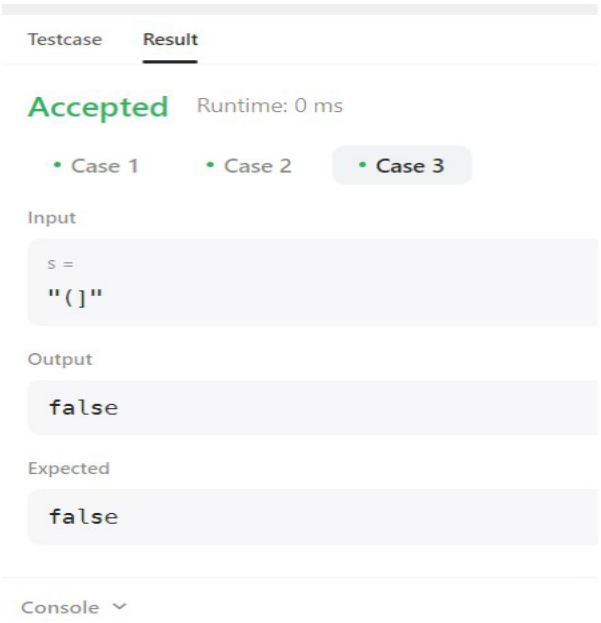

b)LEETCODE:WAP for Valid Parenthesis

CODE:

```
#define MAX 10 char
stack[MAX]; int top = -1;
```

```
void push(char ); char pop(void
); char peek(void ); bool
isValid(char* s) {
   int i = 0;     while(s[i] != '\0')
{
      if(s[i] == '(' || s[i] == '[' || s[i] == '{') {            push(s[i]);
      }        else if(s[i] == ')'){
char ch = pop();
         if(ch != '(') return false;
      }        else if(s[i] == ']'){
char ch = pop();
         if(ch != '[') return false;
      }        else if(s[i] == '}'){
char ch = pop();
         if(ch != '{') return false;
      }        i++;
   }
   if(top != -1) return false;
   return true;
}

void push(char ch) {     if(top == MAX
- 1) return;     stack[++top] = ch;
}

char pop() {     if(top == -1) return
'n';
   return stack[top--];
}
```

OUTPUT :




5a). WAP to Implement Singly Linked List with following operations

a) Create a linked list.

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

c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node* next;
};
void insert_at_beg(struct node** head, int data) {     struct
node* new_node = (struct node*)malloc(sizeof(struct
node));     new_node->data = data;
   if (*head == NULL) {
*head = new_node;
new_node->next = NULL;
   } else {
```

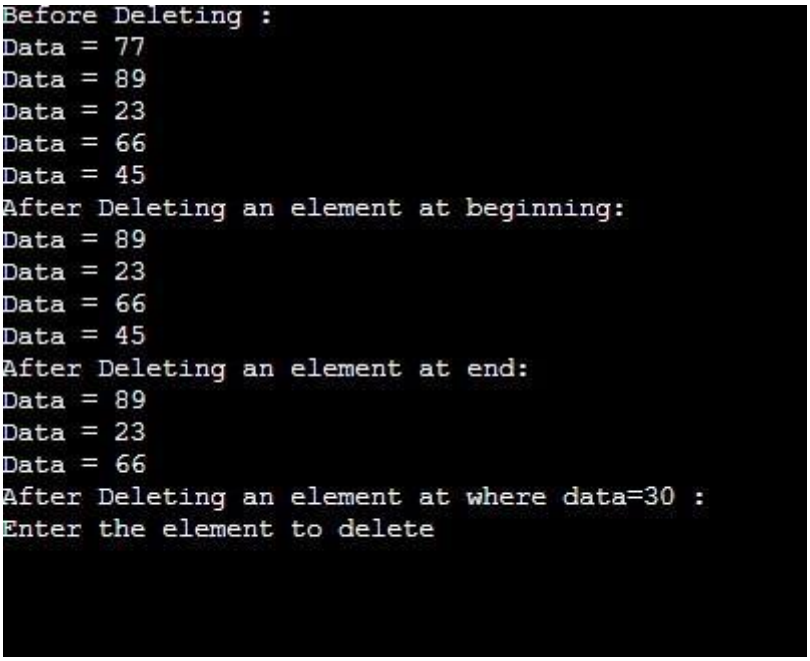
```
        new_node->next = *head;
        *head = new_node;
    }}
void display(struct node**
head) {     if(*head==NULL){
printf("No nodes\n");
        return;}
    struct node* ptr = *head;
while (ptr!= NULL) {
printf("Data = %d\n", ptr-
>data);         ptr = ptr->next;
    }}

void delete_at_beg(struct node**
head){     if(*head==NULL){
printf("underflow\n");
    }
   else{
    struct node* temp=*head;
*head=temp->next;
    free(temp);
   }}

void delete_at_end(struct node**
head){ if(*head==NULL){
```

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

struct node* ptr=*head;
struct node* prev; if(ptr-
>next==NULL){
*head=NULL;
    free(ptr);
return;
}
while(ptr->next!=NULL){
    prev=ptr;     ptr=ptr-
>next;
}
prev->next=NULL;
free(ptr);
}

void delete_element(struct node**
head){ int element; printf("Enter the
element to delete\n");
scanf("%d",&element);
if(*head==NULL){
    printf("Underflow\n");
    return;
```

```
}
struct node* ptr=*head;
struct node* prev; if(ptr-
>next==NULL){    if(ptr-
>data==element){
      free(ptr);
      *head=NULL;
   }}
else{
while(ptr!=NULL){

   if(ptr->data==element){       if(ptr-
>next==NULL){          prev-
>next=NULL;
         free(ptr);
return;
      }

else{          prev-
>next=ptr->next;
         free(ptr);
return;
      }
   }
   prev=ptr;    ptr=ptr->next;
if(ptr==NULL){
```

```
printf("element not
found\n");
   }
}
}}
int main() {    struct node* head = NULL;
printf("Before Deleting :\n");
insert_at_beg(&head,45);
insert_at_beg(&head,66);
insert_at_beg(&head,23);
insert_at_beg(&head,89);
insert_at_beg(&head,77);     display(&head);
printf("After Deleting an element at
beginning:\n");    delete_at_beg(&head);
display(&head);    printf("After Deleting an
element at end:\n");     delete_at_end(&head);
display(&head);    printf("After Deleting an
element at where data=30 :\n");
delete_element(&head);    display(&head);
   return 0;
}
```

Ouput:




b)LEETCODE: Reverse the LinkedList CODE:

```
struct ListNode* reverse(struct ListNode* node); struct
ListNode* reverseList(struct ListNode* head) {     if(head ==
NULL) return NULL;
   return reverse(head);
}

struct ListNode* reverse(struct ListNode *node) {     if(node->next ==
NULL) {
      return node;
   }
   struct ListNode* newHead = reverse(node->next);     node-
>next->next = node;     node->next = NULL;
   return newHead;
}
```

OUTPUT:


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
<stdlib.h> struct
node { int data;
struct node*
next;
};
void reverse(struct node**
head){ struct node*
nextNode; struct node*
prev=NULL; struct node*
```

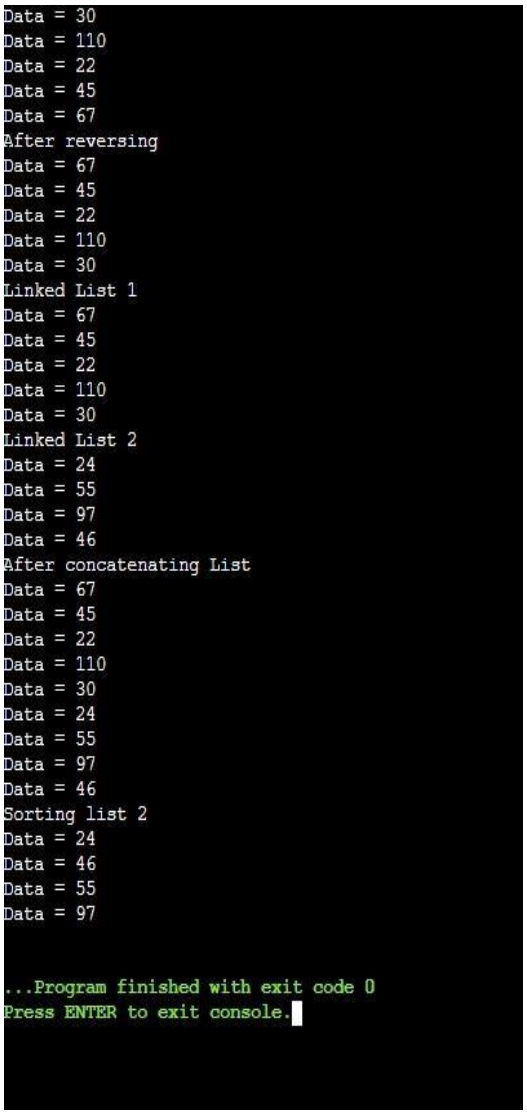
```
ptr=*head;
while(ptr!=NULL){
nextNode=ptr->next;     ptr-
>next=prev;     prev=ptr;
ptr=nextNode;
}
*head=prev;
}
void concat(struct node** head1,struct node**
head2){ struct node* ptr=*head1; while(ptr-
>next!=NULL){
   ptr=ptr->next;
}
ptr->next=*head2;
}


void sort(struct node** head){
struct node* current=*head; int
temp; while(current!=NULL){
struct node* next=current-
>next;     while(next!=NULL){
if(current->data>=next->data){
temp=current->data;
current->data=next->data;
next->data=temp;
```

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

}}

void insert(struct node** head,int data){     struct node*
NewNode=(struct node*)malloc(sizeof(struct node));
NewNode->data=data; if(*head==NULL){
    *head=NewNode;
    NewNode->next=NULL;
}
else{
    NewNode->next=*head;
    *head=NewNode;
}}
void display(struct node**
head){ struct node*
ptr=*head;
while(ptr!=NULL){
printf("Data = %d\n",ptr-
>data);     ptr=ptr->next;

}}
int main(){
```

```
struct node*
head1=NULL;
insert(&head1,67);
insert(&head1,45);
insert(&head1,22);
insert(&head1,110);
insert(&head1,30);
display(&head1);
printf("After
reversing\n");
reverse(&head1);
display(&head1);

struct node* head2=NULL;
insert(&head2,46);
insert(&head2,97);
insert(&head2,55);
insert(&head2,24);
printf("Linked List 1\n");
display(&head1);
printf("Linked List 2\n");
display(&head2); printf("After
concatenating List\n");
concat(&head1,&head2);
display(&head1);
```

```
printf("Sorting list 2\n");
sort(&head2);
display(&head2);
return 0;
}
```

Output:




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

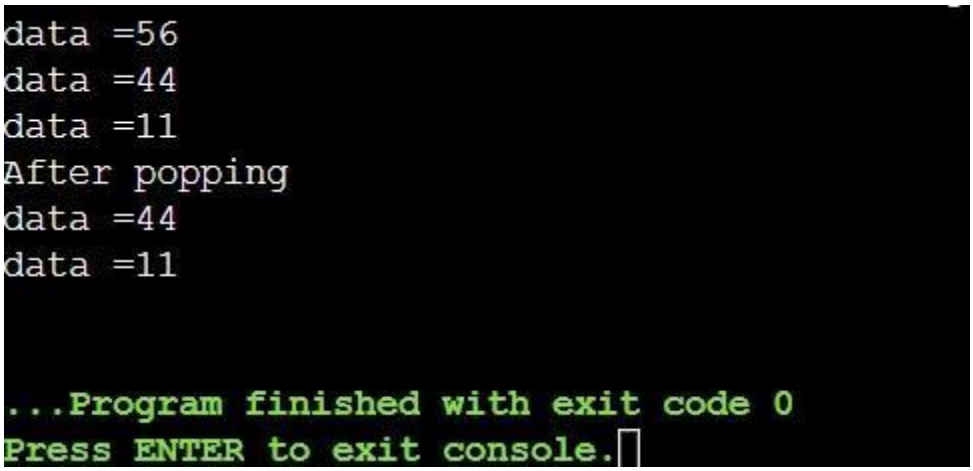
Stack:

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

```
node{ int data;
struct node*
next;
};
void push(struct node** head,int data){     struct node*
new_node=(struct node*)malloc(sizeof(struct node));
new_node->data=data; if(*head==NULL){
*head=new_node;     new_node->next=NULL;
}
else{
   new_node->next=*head;
   *head=new_node;
}}

void pop(struct node** head){
if(*head==NULL){
printf("Underflow\n");
   }
   else{

      struct node* ptr=*head;
*head=ptr->next;
      free(ptr);
   }}
```

```
void display(struct node**
head){    struct node*
ptr=*head;
while(ptr!=NULL){
printf("data =%d\n",ptr-
>data);       ptr=ptr->next;
   }
}
int main(){
struct node*
head=NULL;
push(&head,11);
push(&head,44);
push(&head,56);
display(&head);
printf("After
popping\n");
pop(&head);
display(&head);
return 0;
}
```

Output:



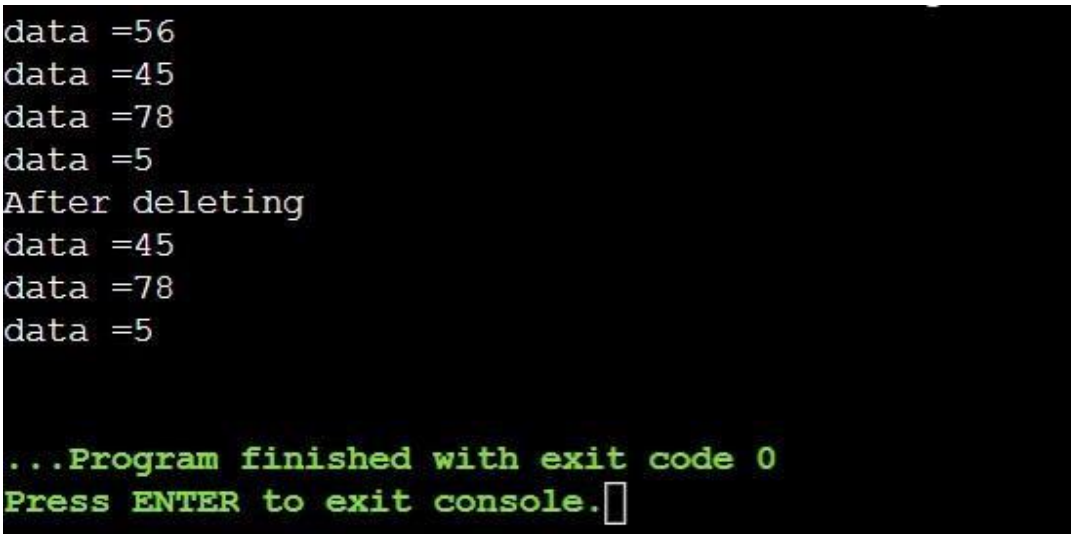

Queue:

```
#include
<stdio.h> struct
node{ int data;
struct node*
next;
};
void enqueue(struct node** head,struct node** tail, int
data){    struct node* new_node=(struct
node*)malloc(sizeof(struct node));    new_node-
>data=data; if(*head==NULL){
  *head=new_node;
*tail=new_node;    new_node-
>next=NULL;

}
else{       struct node*
ptr=*tail;       ptr-
>next=new_node;
```

```
*tail=new_node;
new_node-
>next=NULL;
}}
void dequeue(struct node** head,struct node** tail){
if(head==NULL){
   printf("Underflow\n");
}
else if(*head==*tail){
   *head=NULL;
   *tail=NULL;
}
else{    struct node*
ptr=*head;
*head=ptr->next;
   free(ptr);
}}

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

```
int main(){
struct node*
head=NULL; struct
node* tail=NULL;
enqueue(&head,&tail,5
6);
enqueue(&head,&tail,4
5);
enqueue(&head,&tail,7
8);
enqueue(&head,&tail,5)
; display(&head);
printf("After
deleting\n");
dequeue(&head,&t
ail);
display(&head);
return 0;
}
```

Output:




7a). 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:

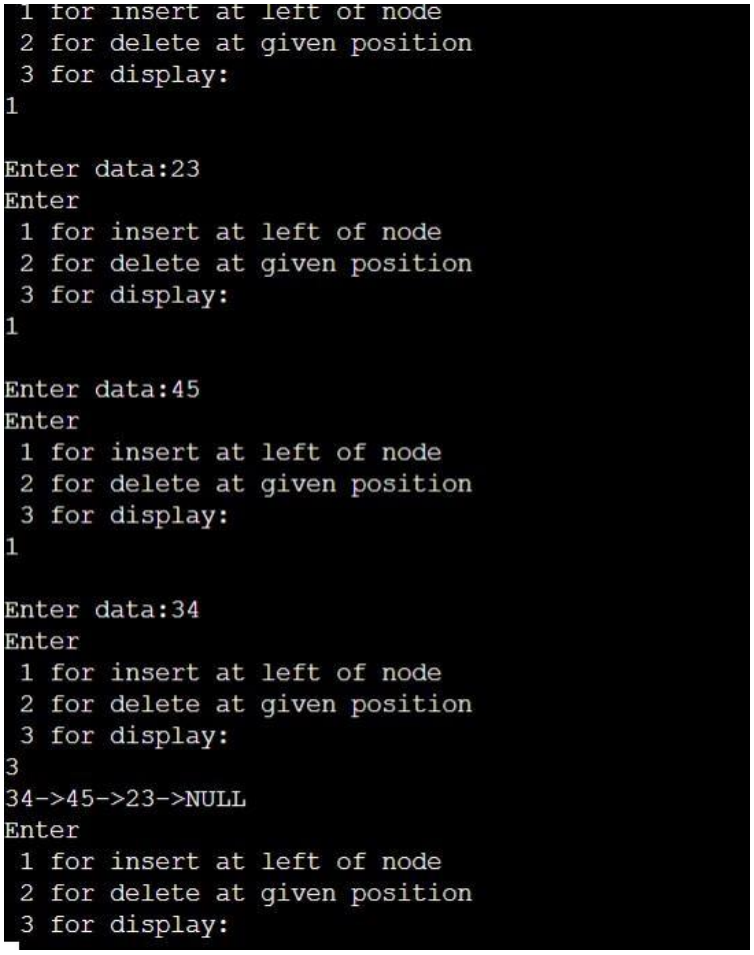
```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct node* next;
struct node* prev;
};
void insertAtBeginning(struct node** head,int
value){     struct node*
newNode=malloc(sizeof(struct node));
newNode->data=value;     newNode-
```

```
>prev=NULL;     newNode->next=NULL;
if(*head==NULL){        *head=newNode;
       return;}
   (*head)->prev=newNode;
newNode->next=*head;
   *head=newNode;
}
void deleteNode(struct node** head, int key)
{    struct node* temp = *head;

   while (temp != NULL && temp->data != key)
       temp = temp-
>next;     if (temp ==
NULL)
       return;
   if (temp->prev != NULL)
temp->prev->next = temp-
>next;     if (temp->next !=
NULL)        temp->next->prev =
temp->prev;     if (temp ==
*head)        *head = temp-
>next;     free(temp);
}
void displayList(struct node *head){
struct node *temp=head;
```

```
while(temp!=NULL){
printf("%d->",temp->data);
temp=temp->next;
   }
   printf("NULL\n");} int
main(){    struct node*
head=NULL;
   int data;
int pos;
int choice;
do{
      printf("Enter \n 1 for insert at left of node \n 2 for delete at given
position\n 3 for display:\n");       scanf("%d",&choice);
switch(choice){
         case 1:
            printf("\nEnter data:");
            scanf("%d",&data);
insertAtBeginning(&head,data);
            break;
case 2:
            printf("Enter the key:");
scanf("%d",&pos);
deleteNode(&head,pos);
```

```
            break;
case 3:
displayList(head);
            break;
      }}while(choice!=0);
}
```

Output:

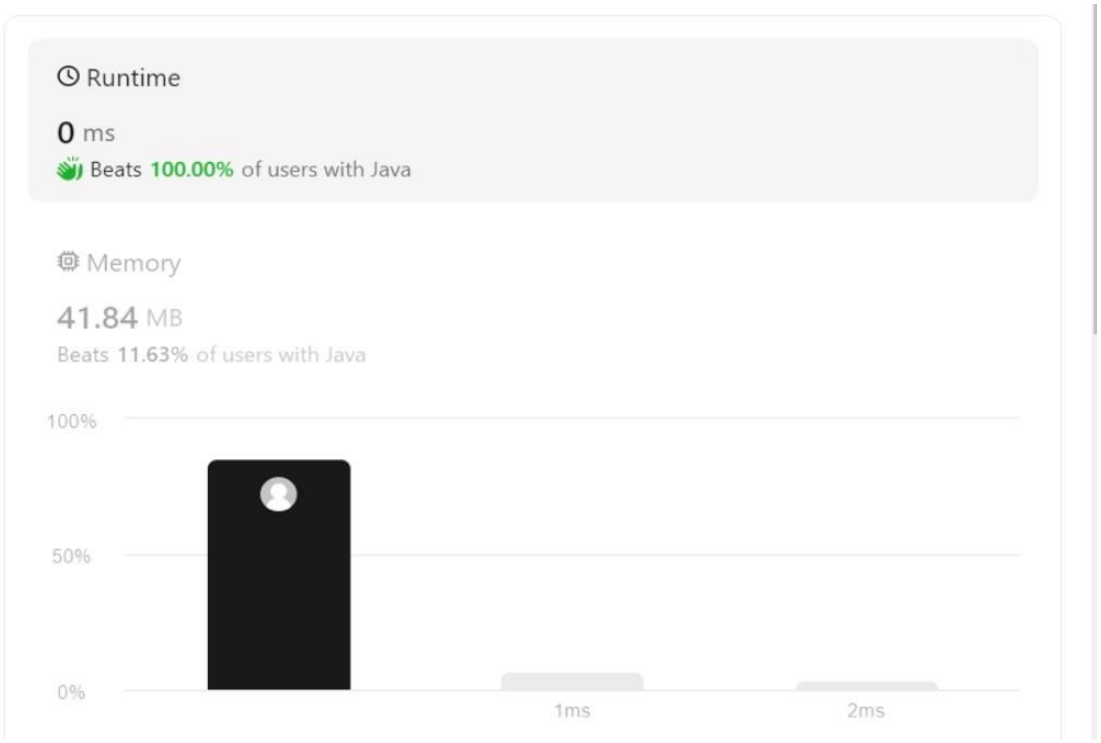



b)LEETCODE: Leaf similar trees

```
int sum1 = 0; int sum2 =
0;
```

```
void preOrder1(struct TreeNode* node); void
preOrder2(struct TreeNode* node);

bool leafSimilar(struct TreeNode* root1, struct TreeNode* root2) {
preOrder1(root1);     preOrder2(root2);


   return sum1 == sum2;

}

void preOrder1(struct TreeNode* node) {

   if (node == NULL) return;


   if (node->leh == NULL && node->right == NULL) {        sum1 =
(sum1*10) + node->val;

      return;

   }


   preOrder1(node->leh);

   preOrder1(node->right);

}


void preOrder2(struct TreeNode* node) {

   if (node == NULL) return;


   if (node->leh == NULL && node->right == NULL) {        sum2 =
(sum2*10) + node->val;

      return;

   }


   preOrder2(node->leh);

   preOrder2(node->right);

}
```

OUTPUT:




8a). 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.

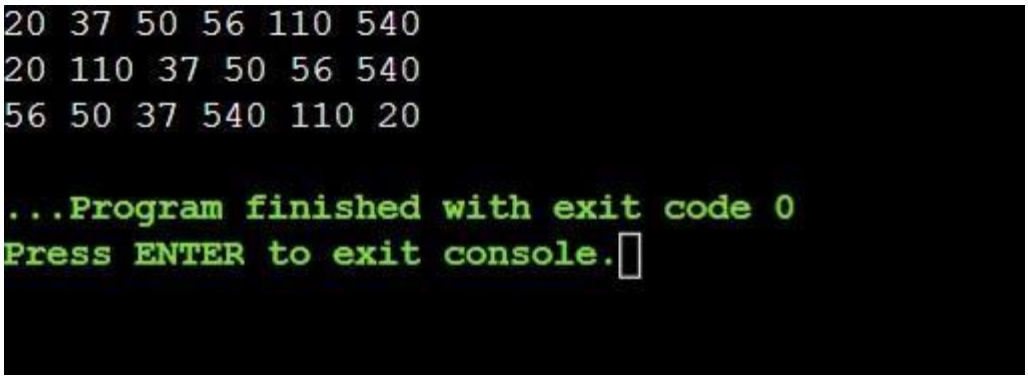
Code:

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

```
struct node* createNode(int data){ struct node*
temp=(struct node*)malloc(sizeof(struct node));
temp->left=NULL; temp->right=NULL; temp-
>data=data; return temp;
};
struct node* insert(struct node*
root,int data){ if(root==NULL){     return
createNode(data);
}
else if(data > root->data){     root-
>right=insert(root->right,data);
}
else{     root->left=insert(root-
>left,data); }
return root;
};

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

```
printf("%d ",root->data);
preorder(root->left);
preorder(root->right);
}}


void postorder(struct node* root){
if(root!=NULL){

   postorder(root->left);
postorder(root->right);
printf("%d ",root->data);

}}
int main(){
struct node* root=NULL;
root=insert(root,20); insert(root,110);
insert(root,37); insert(root,540);
insert(root,50); insert(root,56);
inorder(root); printf("\n");
preorder(root); printf("\n");
postorder(root); return 0;
}:
```

Output:




b)HACKER RANK: Reverse a double linked list

CODE:

```
DoublyLinkedListNode* reverse(DoublyLinkedListNode* llist) {
   DoublyLinkedListNode* current = llist;     DoublyLinkedListNode* temp = NULL;
   // Traverse the list and swap prev and next pointers for each node     while (current != NULL) {         temp = current->prev;         current->prev = current->next;         current->next = temp;
      // Move to the next node
      current = current->prev;
   }
   // Update the head pointer to the last node (previous head becomes the new tail)     if (temp != NULL) {
      llist = temp->prev;
   }
   return llist;
}
```

OUTPUT:

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

Code:

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 50
typedef struct Graph_t
{    int
V;
   bool
adj[MAX_VERTICES][MAX_VERTICES];
} Graph;
Graph* Graph_create(int V)
{
   Graph* g = malloc(sizeof(Graph));
   g->V = V;

   for (int i = 0; i < V; i++)
   {
      for (int j = 0; j < V; j++)
      {
         g->adj[i][j] = false;
      }
   }

   return g;
}
void Graph_destroy(Graph* g)
```

```
{
   free(g);
}

void Graph_addEdge(Graph* g, int v,
int w)
{
   g->adj[v][w] = true;
}

void Graph_BFS(Graph* g, int s)
{
   bool visited[MAX_VERTICES];
   for (int i = 0; i < g->V; i++)
   {
      visited[i] = false;
   }


   int queue[MAX_VERTICES];
   int front = 0, rear = 0;

   visited[s] = true;
queue[rear++] = s;

   while (front != rear)
   {
```

```
        s = queue[front++];
printf("%d ", s);

        for (int adjacent = 0; adjacent < g-
>V;              adjacent++)
        {
           if (g->adj[s][adjacent] && !visited[adjacent])
           {
              visited[adjacent] = true;
queue[rear++] = adjacent;
           }}}}

int main()
{

   Graph* g = Graph_create(4);
   Graph_addEdge(g, 0, 1);
   Graph_addEdge(g, 0, 2);
   Graph_addEdge(g, 1, 2);
   Graph_addEdge(g, 2, 0);
   Graph_addEdge(g, 2, 3);
Graph_addEdge(g, 3, 3);

   printf("Following is Breadth First Traversal (starting from vertex 2) \n");
   Graph_BFS(g, 2);
```

```
    Graph_destroy(g);

    return 0;
}
```

Output:


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

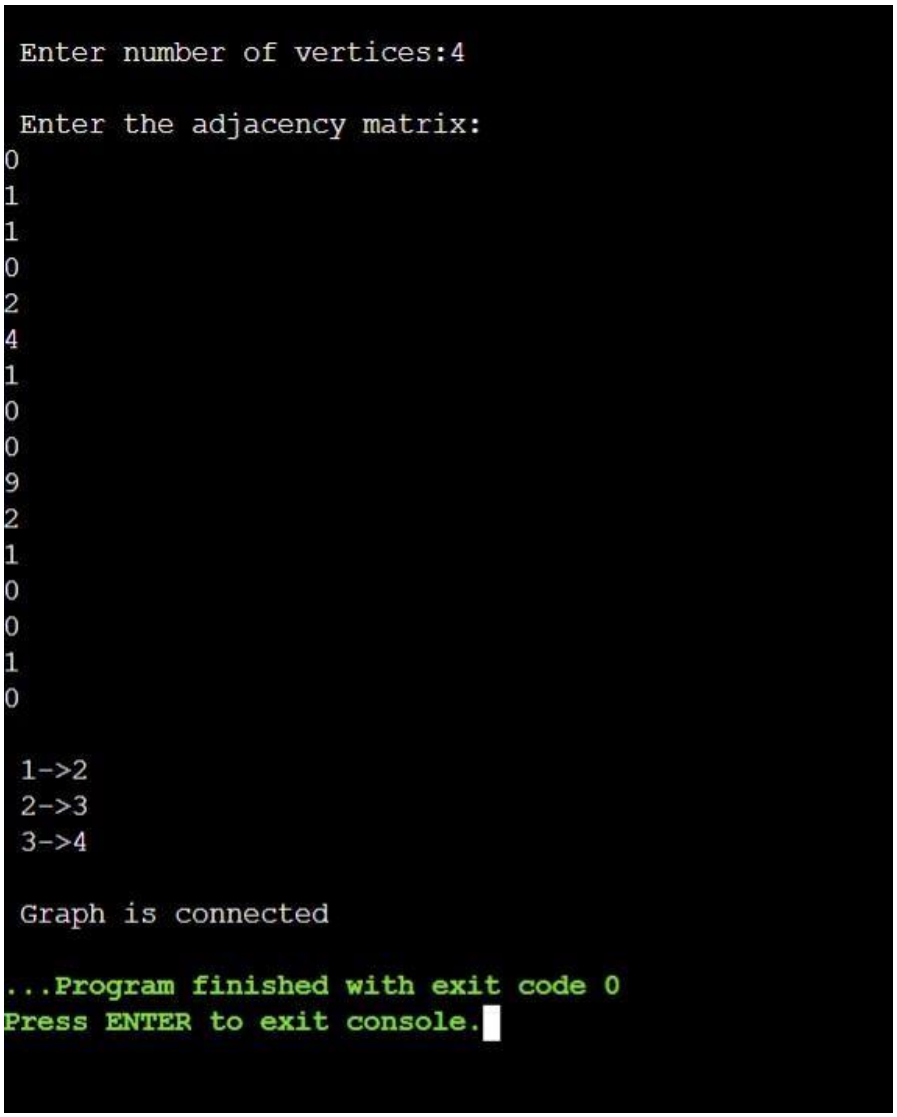
Code:

```
#include<stdio.h> int
a[20][20], reach[20],
n; void dfs(int v) {
   int i;
   reach[v] = 1;     for (i = 1;
i <= n; i++)        if (a[v][i]
&& !reach[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++) {       reach[i] =
0;       for (j = 1; j <=
n; j++)
          a[i][j] = 0;
   }
   printf("\n 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 (reach[i])
count++;
   }
   if (count == n)       printf("\n
Graph is connected");
   else
      printf("\n Graph is not
connected");    return 0;
```

}

**Output:**


Enter number of vertices:4
Enter the adjacency matrix:
0
1
1
0
2
4
1
0
0
9
2
1
0
0
1
0
1->2
2->3
3->4
Graph is connected
...Program finished with exit code 0
Press ENTER to exit console.