VISVESVARAYATECHNOLOGICALUNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT

On

DATA STRUCTURES (23CS3PCDST)

Submitted by

SUJAY PRASAD P V (2023BMS02634)

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 **SUJAY PRASAD P V (2023BMS02634)**, who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

Prof. Lakshmi Neelima
Assistant 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:

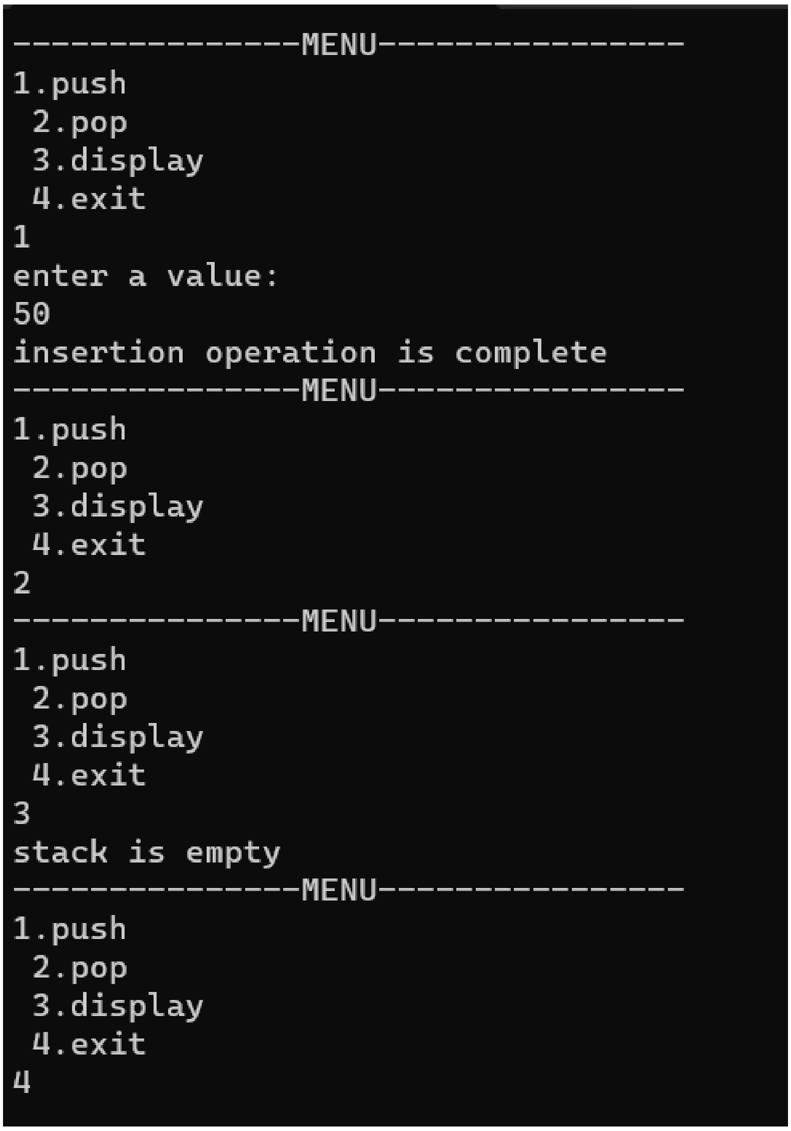
Write a program to simulate the working of stack using an array with the following:
a) **Push**
b) **Pop**
c) **Display**
The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>

#include <stdlib.h>

int size = 5;

int top = -1, stack[5];

void push(int a)

{ if (top == size)

      printf("stack overflow\n");

   else

   {

      top = top + 1;

      stack[top] = a;

      printf("insertion operation is complete\n");

   }

}

void pop()

{

   if (top == -1)

   {

      printf("stack is empty\n");

   }

   else

   {

      top--;

   }

}

void display()
```

```
{
   if (top == -1)
      printf("stack is empty\n");
   else
   {
      for (int i = top; i >= 0; i--)
      {
         printf("%d\n", stack[i]);
      }
   }
}
int main()
{
   int value, choice, t = 0;
   while (1)
   {
      printf("---------------MENU----------------\n");
      printf("1.push\n 2.pop\n 3.display\n 4.exit\n");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         printf("enter a value:\n");
         scanf("%d", &value);
         push(value);
         break;
      case 2:
         pop();
         break;
      case 3:
         display();
```

```
            break;
          case 4:
            exit(0);
            break;
          default:
            printf("wrong input!\n");
            break;
          }
        }
      }
```

OUTPUT:



```
---------------MENU----------------
1.push
 2.pop
 3.display
 4.exit
1
enter a value:
50
insertion operation is complete
---------------MENU----------------
1.push
 2.pop
 3.display
 4.exit
2
---------------MENU----------------
1.push
 2.pop
 3.display
 4.exit
3
stack is empty
---------------MENU----------------
1.push
 2.pop
 3.display
 4.exit
4
```

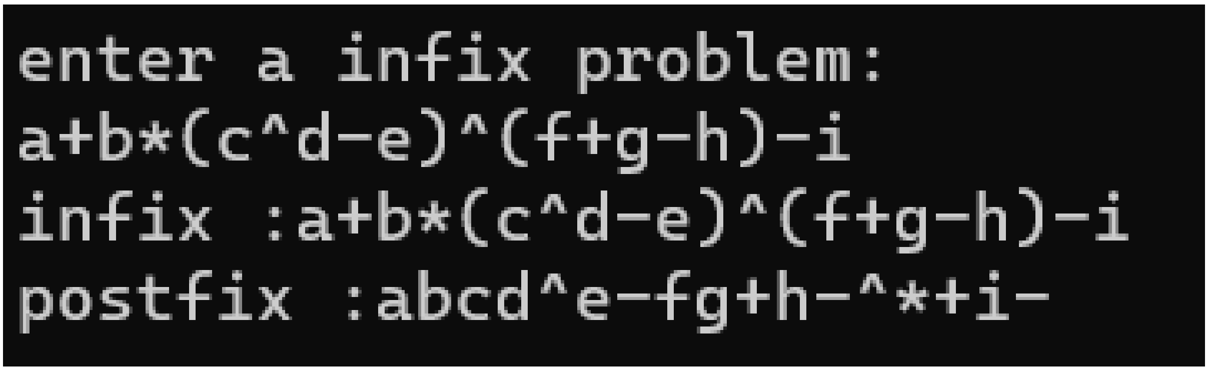
LAB PROGRAM 2:

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) and / (divide).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int top = -1, pos = 0;
char temp, stack[25], infix[25], postfix[25];
void push(char s)
{
    stack[++top] = s;
}
int precedence(char s)
{
    switch (s)
    {
    case '^':
        return 3;
    case '+':
    case '-':
        return 1;
    case '*':
    case '/':
        return 2;
    case '(':
        return 0;
    }
}
char pop()
{  char symb = stack[top];
    top--;
    return symb;
}
```

```
void infixtopostfix()
{
    int len = strlen(infix);
    int i = 0;
    char symbol;
    while (i < len)
    {
        symbol = infix[i];
        switch (symbol)
        {
        case '(':
            push(symbol);
            break;
        case ')':
            temp = pop();
            while (temp != '(')
            {
                postfix[pos++] = temp;
                temp = pop();
            }
            break;
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while (precedence(stack[top]) >= precedence(symbol))
            {
                postfix[pos++] = pop();
            }
            push(symbol);
            break;
        default:
            postfix[pos++] = symbol;
        }
        i++;
    }
    while (top != -1)
    {
        postfix[pos++] = stack[top];
        top--;
    }
    return;
}
int main()
{
    printf("enter a infix problem:\n");
    scanf("%s", infix);
    infixtopostfix();
    printf("infix :%s\n", infix);
    printf("postfix :%s\n", postfix);
}
```

OUTPUT:



```
enter a infix problem:
a+b*(c^d-e)^(f+g-h)-i
infix :a+b*(c^d-e)^(f+g-h)-i
postfix :abcd^e-fg+h-^*+i-
```

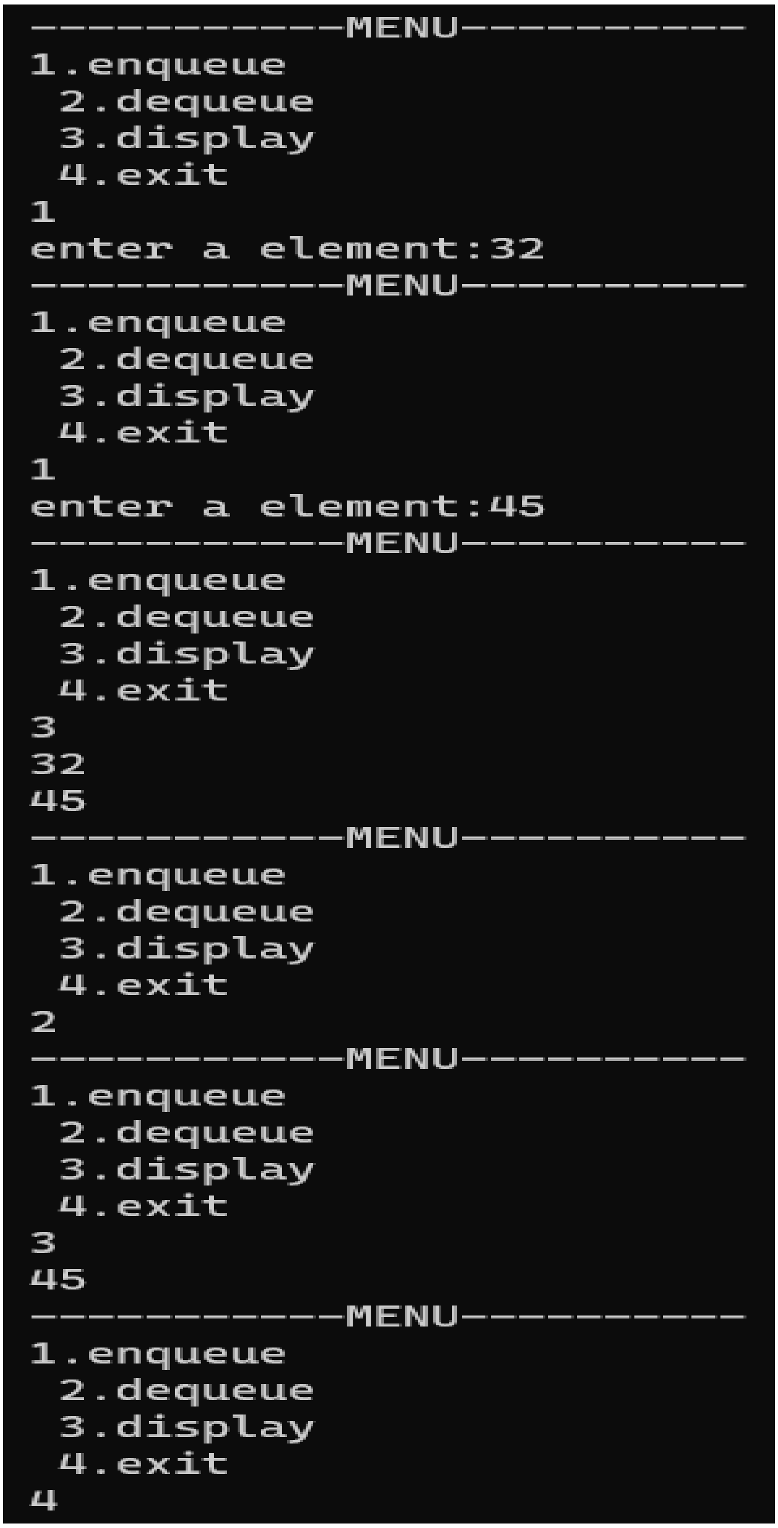
LAB PROGRAM 3:

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

```
#include<stdio.h>
#include<stdlib.h>
# define size 5
int front=-1,rear=-1,queue[size];
void enqueue()
{
   int a;
   if (rear==size-1)
   {
      printf("overflow\n");
   }
   else
   {
      if(front==-1)
         front=0;
      printf("enter a element:");
      scanf("%d",&a);
      queue[++rear]=a;
   }
}

int dequeue()
{
   if(front==-1||front>rear)
   {
      printf("underflow\n");
      return;
   }
   else
   {
      if (front==rear)
      {
         front=-1;
         rear=-1;
         return;
      }
      int s = queue[front++];
      return s;
   }
}
```

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

int main()
{
    int choice;

    while(1)
    {
        printf("-----------MENU----------\n");
        printf("1.enqueue\n 2.dequeue\n 3.display\n 4.exit\n");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:enqueue();
                    break;
            case 2:dequeue();
                    break;
            case 3:display();
                    break;
            case 4: exit(0);
                    break;
            default:printf("wrong input");
        }
    }
}
```

OUTPUT:



```
-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
1
enter a element:32
-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
1
enter a element:45
-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
3
32
45
-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
2
-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
3
45
-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
4
```

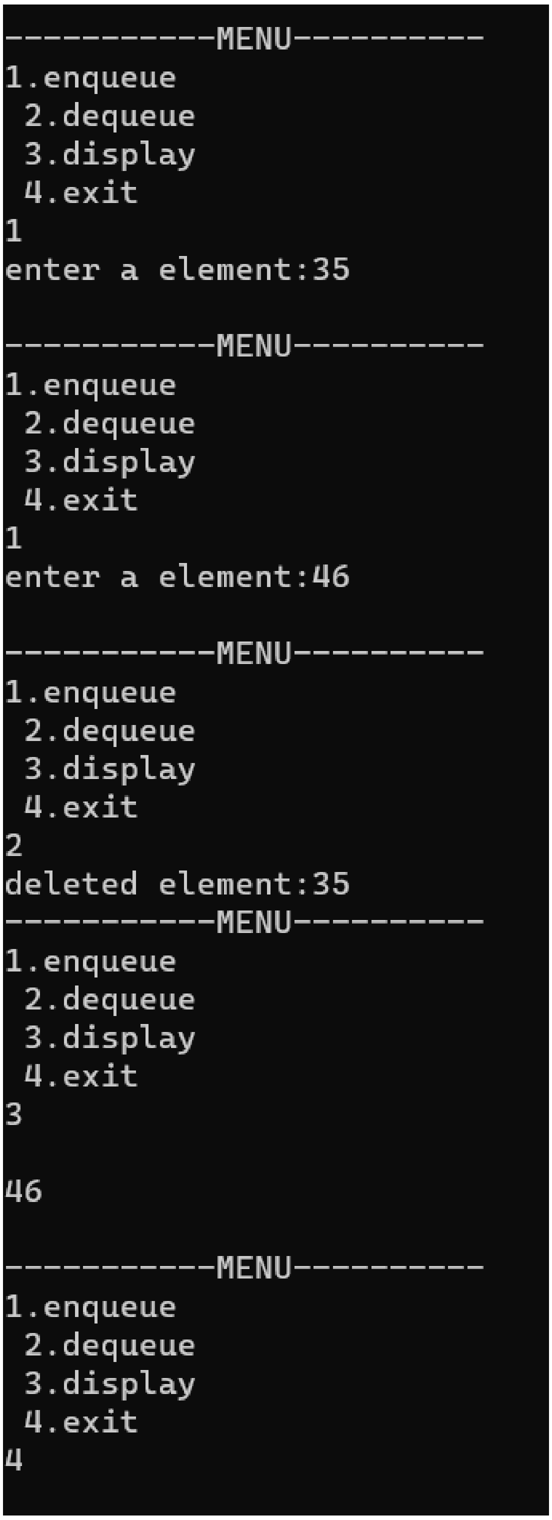
LAB PROGRAM 4:

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

```
#include <stdio.h>
#include <stdlib.h>
#define size 5
int front = -1, rear = -1, queue[size];

void enqueue()
{
   int a;
   if (front == rear + 1 || front == 0 && rear == size - 1)
   {
      printf("overflow\n");
   }
   else
   {
      if (front == -1)
         front = 0;
      printf("enter a element:");
      scanf("%d", &a);
      rear = (rear + 1) % size;
      queue[rear] = a;
   }
}

int dequeue()
{
   if (front == -1)
   {
      printf("underflow\n");
   }
   else
   {

      int s = queue[front];
      if (front == rear)
      {
         front = -1;
         rear = -1;
      }
      else
      {
         front = (front + 1) % size;
      }
```

```
        printf("deleted element:%d", s);
        return s;
    }
}
void display()
{
    int i;
    if (front == -1)
    {
        printf("overflow\n");
    }
    else
    {
        for (i = front; i != rear; i = (i + 1) % size)
        {
            printf("\n%d\n", queue[i]);
        }
        printf("\n%d\n", queue[i]);
    }
}

int main()
{
    int choice;

    while (1)
    {
        printf("\n-----------MENU----------\n");
        printf("1.enqueue\n 2.dequeue\n 3.display\n
4.exit\n");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("wrong input");
        }
    }
}
```

OUTPUT:



```
-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
1
enter a element:35

-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
1
enter a element:46

-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
2
deleted element:35
-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
3

46

-----------MENU----------
1.enqueue
 2.dequeue
 3.display
 4.exit
4
```

LAB PROGRAM 5:

Write a program 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.**

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

Display the contents of the linked list.

```
#includ<stdio.h>

#include<stdlib.>

struct node

{

int data;

structnode*next;

};

struct node *head= NULL;


struct node *create_ll(struct node *head)

{

struct node *new_node, *ptr;

int num;

printf("Enter -1 to exit.. \n");

printf("Enter the num: ");

scanf("%d", &num);


while (num != -1)

{

new_node = (struct node *)malloc(sizeof(struct node));

new_node->data = num;

if (head == NULL)

{
```

```
head = new_node;
new_node->next = NULL;
}
else
{
ptr = head;
while (ptr->next != NULL)
{
ptr = ptr->next;
}
ptr->next = new_node;
new_node->next = NULL;
}
printf("Enter the num: ");
scanf("%d", &num);
}
return head;
}

struct node *insert_beg(struct node *head)
{
struct node *new_node;
int num;
printf("Enter the num: ");
scanf("%d", &num);
new_node = (struct node *)malloc(sizeof(struct node));
new_node->data = num;
if (head == NULL)
{
```

```
head = new_node;
new_node->next = NULL;
}
else
{
new_node->next = head;
head = new_node;
}
return head;
}

struct node *insert_end(struct node *head)
{
struct node *ptr, *new_node;
int num;
printf("Enter the num: ");
scanf("%d", &num);
new_node = (struct node *)malloc(sizeof(struct node));
new_node->data = num;
new_node->next = NULL;
if (head == NULL)
{
head = new_node;
}
else
{
ptr = head;
while (ptr->next != NULL)
{
```

```
ptr = ptr->next;
}
ptr->next = new_node;
}
return head;
}

struct node *insert_before(struct node *head)
{
struct node *new_node, *ptr, *prevptr;
int num, val;
printf("Enter the num: ");
scanf("%d", &num);
printf("Enter the value before which number has to be inserted: ");
scanf("%d", &val);
new_node = (struct node *)malloc(sizeof(struct node));
new_node->data = num;
ptr = head;
while (ptr->next != NULL)
{
prevptr = ptr;
ptr = ptr->next;
}
prevptr->next = new_node;
new_node->next = ptr;
return head;
}
struct node *insert_after(struct node *head)
{
```

```
struct node *new_node, *ptr, *prevptr;
int num, val;
printf("Enter the num: ");
scanf("%d", &num);
printf("Enter the value before which number has to be inserted: ");
scanf("%d", &val);
new_node = (struct node *)malloc(sizeof(struct node));
new_node->data = num;
ptr = head;
while (ptr->next != NULL)
{
prevptr = ptr;
ptr = ptr->next;
}
prevptr = ptr;
ptr = ptr->next;
prevptr->next = new_node;
new_node->next = ptr;
return head;
}
struct node *display(struct node *head)
{
struct node *ptr;
if (head == NULL)
{
printf("Linked List is empty...\n");
}
else
{
```

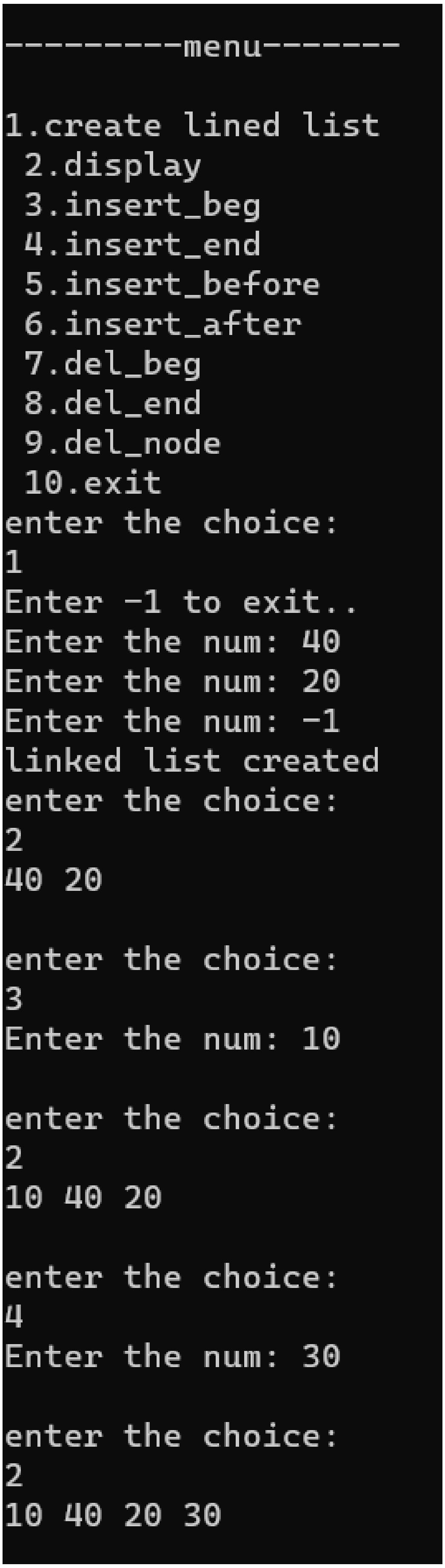
```
ptr = head;
while (ptr != NULL)
{
printf("%d ", ptr->data);
ptr = ptr->next;
}
printf("\n");
}
return head;
}

struct node *delete_beg(struct node *head)
{
struct node *ptr;
if (head == NULL)
{
printf("Nothing to delete.. \n");
}
else
{
ptr = head;
head = ptr->next;
free(ptr);
}
return head;
}

struct node *delete_end(struct node *head)
{
```

```
struct node *ptr, *prevptr;
ptr = head;
while (ptr->next != NULL)
{
prevptr = ptr;
ptr = ptr->next;
}
prevptr->next = NULL;
free(ptr);
return head;
}

struct node *delete_node(struct node *head)
{
struct node *ptr, *prevptr;
int val;
printf("Enter the value that has to be deleted: ");
scanf("%d", &val);
ptr = head;
if (ptr->data == val)
{
head = delete_beg(head);
return head;
}
else
{
while (ptr->data != val)
{
prevptr = ptr;
```

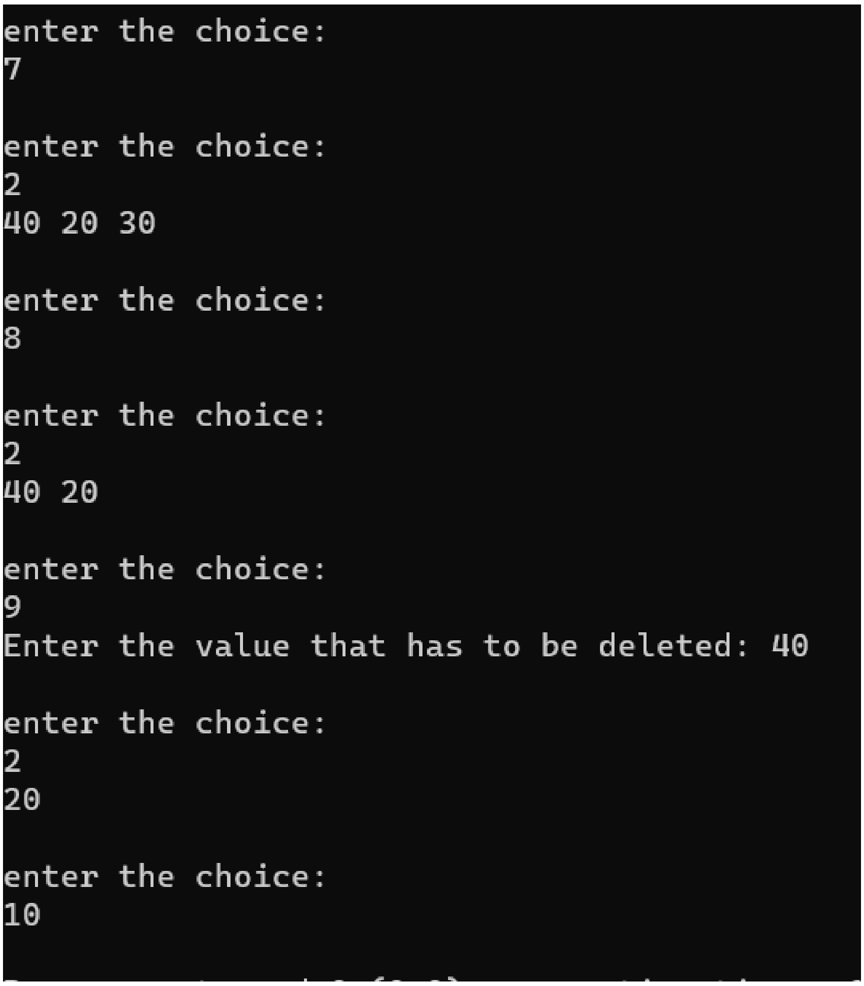
```
ptr = ptr->next;
}
prevptr->next = ptr->next;
free(ptr);
return head;
}
}

int main()
{
int choice;
printf("\n---------menu-------\n");
printf("\n1.create lined list\n 2.display\n 3.insert_beg\n 4.insert_end\n 5.insert_before\n 6.insert_after\n
7.del_beg\n 8.del_end\n 9.del_node\n 10.exit");
do
{
printf("\nenter the choice:\n");
scanf("%d", &choice);
switch (choice)
{
case 1:
head = create_ll(head);
printf("linked list created");
break;
case 2:
head = display(head);
break;
case 3:
head = insert_beg(head);
```

```
break;
case 4:
head = insert_end(head);
break;
case 5:
head = insert_before(head);
break;
case 6:
head = insert_after(head);
break;
case 7:
head = delete_beg(head);
break;
case 8:
head = delete_end(head);
break;
case 9:
head = delete_node(head);
break;
}
} while (choice != 10);
}
```

OUTPUT:



```
---------menu-------

1.create lined list
 2.display
 3.insert_beg
 4.insert_end
 5.insert_before
 6.insert_after
 7.del_beg
 8.del_end
 9.del_node
 10.exit
enter the choice:
1
Enter -1 to exit..
Enter the num: 40
Enter the num: 20
Enter the num: -1
linked list created
enter the choice:
2
40 20

enter the choice:
3
Enter the num: 10

enter the choice:
2
10 40 20

enter the choice:
4
Enter the num: 30

enter the choice:
2
10 40 20 30
```



```
enter the choice:
7

enter the choice:
2
40 20 30

enter the choice:
8

enter the choice:
2
40 20

enter the choice:
9
Enter the value that has to be deleted: 40

enter the choice:
2
20

enter the choice:
10
```

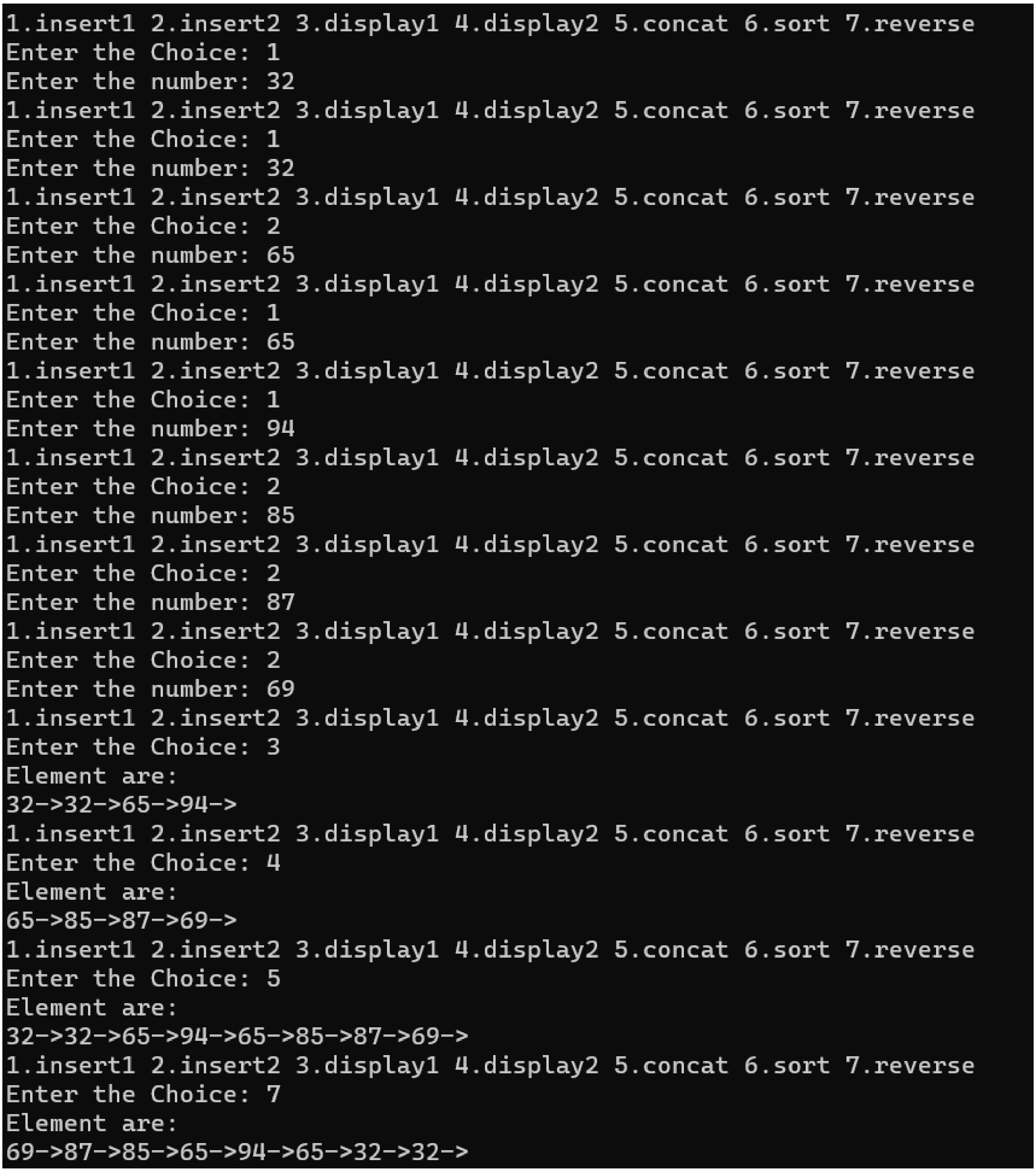
LAB PROGRAM 7:

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

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head1;
struct node *head2;
struct node *insert(struct node *head)
{
   struct node *new_node, *ptr;
   int num;
   printf("Enter the number: ");
   scanf("%d", &num);
   new_node = (struct node *)malloc(sizeof(struct node));
   new_node->next = NULL;
   new_node->data = num;
   if (head == NULL)
   {
      head = new_node;
   }
   else
   {
      ptr = head;
      while (ptr->next != NULL)
      {
         ptr = ptr->next;
      }
      ptr->next = new_node;
   }
   return head;
}
void display(struct node *head)
{
   struct node *ptr;

   ptr = head;
   printf("Element are: \n");
   while (ptr != NULL)
   {
      printf("%d->", ptr->data);
      ptr = ptr->next;
   }
   printf("\n");
}
void concat(struct node *head1, struct node *head2)
{
   struct node *ptr;

   if (head1 != NULL && head2 != NULL)
   {
      ptr = head1;
      while (ptr->next != NULL)
      {
         ptr = ptr->next;
      }
      ptr->next = head2;
      head2=head1;
   }
   else
   {
      printf("either of the Linked List is Empty");
   }
   display(head1);
}
void reverse(struct node *head1)
{
   struct node *ptr = NULL, *prev = NULL;
   while (head1 != NULL)
   {
      ptr = head1->next;
      head1->next = prev;
      prev = head1;
      head1 = ptr;
   }
   head1 = prev;
   display(head1);
}

void sort(struct node *head1)
{ struct node *current, *nextNode;
   int temp;
   current = head1;
    while (current != NULL)
      {
         nextNode = current->next;
         while (nextNode != NULL)
         {
             if (current->data > nextNode->data)
```

```
                {
                    temp = current->data;
                    current->data = nextNode->data;
                    nextNode->data = temp;
                }
            nextNode = nextNode->next;
            }
        current = current->next;
        }
        display(head1);
}
int main()
{
    int choice;
    while (1)
    {
        printf("1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse\n");
            printf("Enter the Choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            head1 = insert(head1);
            break;
        case 2:
            head2 = insert(head2);
            break;
        case 3:
            display(head1);
            break;
        case 4:
            display(head2);
            break;
        case 5:
            concat(head1, head2);
            break;
        case 6:
            sort(head1);
            break;
        case 7:
            reverse(head1);
            break;
        }
    }
}
```

OUTPUT:



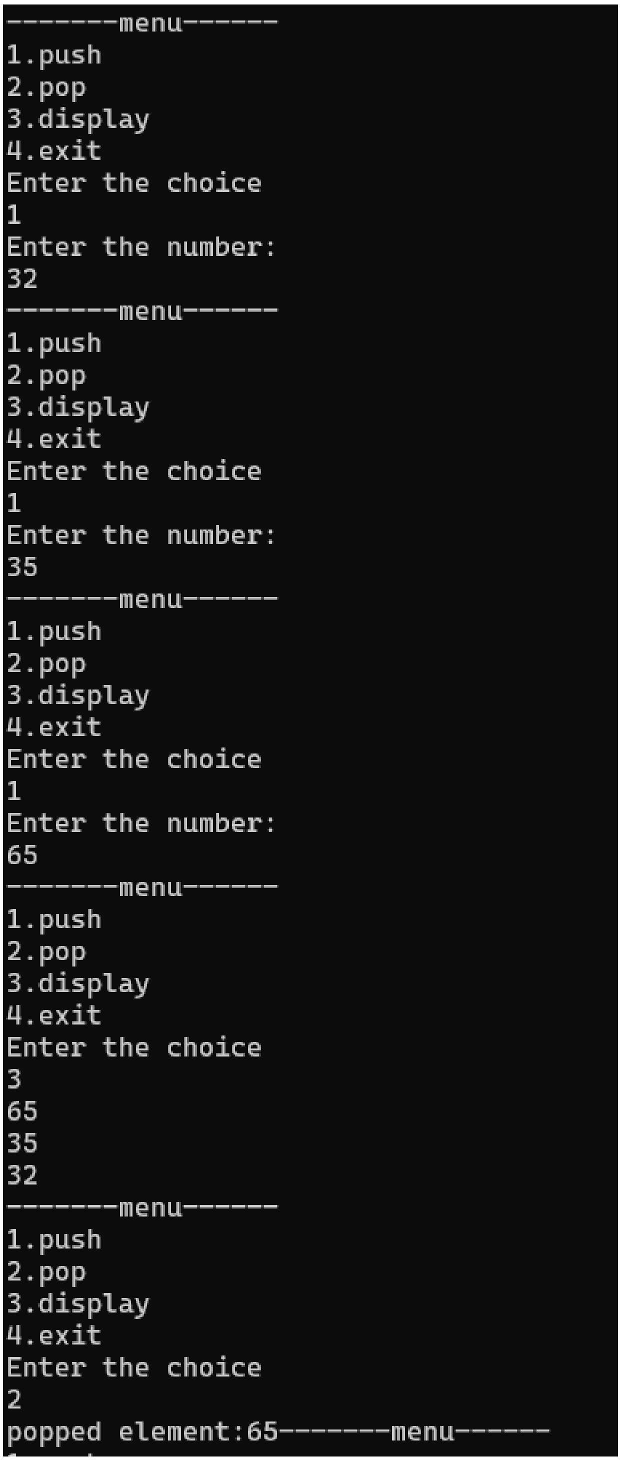
```
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 1
Enter the number: 32
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 1
Enter the number: 32
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 2
Enter the number: 65
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 1
Enter the number: 65
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 1
Enter the number: 94
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 2
Enter the number: 85
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 2
Enter the number: 87
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 2
Enter the number: 69
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 3
Element are:
32->32->65->94->
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 4
Element are:
65->85->87->69->
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 5
Element are:
32->32->65->94->65->85->87->69->
1.insert1 2.insert2 3.display1 4.display2 5.concat 6.sort 7.reverse
Enter the Choice: 7
Element are:
69->87->85->65->94->65->32->32->
```

LAB PROGRAM 8 :

Write a program to Implement Single Link List to simulate Stack & Queue Operations.

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

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

struct node *head = NULL;
void push()
{
   struct node *new_node;
   int num;
   printf("Enter the number:\n");
   scanf("%d", &num);
   new_node = (struct node *)malloc(sizeof(struct node));
   new_node->data = num;
   if (head == NULL)
   {
      head = new_node;
      new_node->next = NULL;
   }
   else
   {
      new_node->next = head;
      head = new_node;
   }
}

void pop()
{
   struct node *ptr;
   if (head == NULL)
   {
      printf("underflow\n");
   }
   else
   {
      ptr = head;
      head = ptr->next;
      printf("popped element:%d", ptr->data);
      free(ptr);
   }
}

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

int main()
{
   int ch;
   while (1)
   {
      printf("-------menu------\n1.push\n2.pop\n3.display\n4.exit\n");
      printf("Enter the choice\n");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
          push();
          break;
      case 2:
          pop();
          break;
      case 3:
          display();
          break;
      case 4:
          exit(0);
          break;
      default:
          printf("invalid number!");
      }
   }
}
```

OUTPUT:

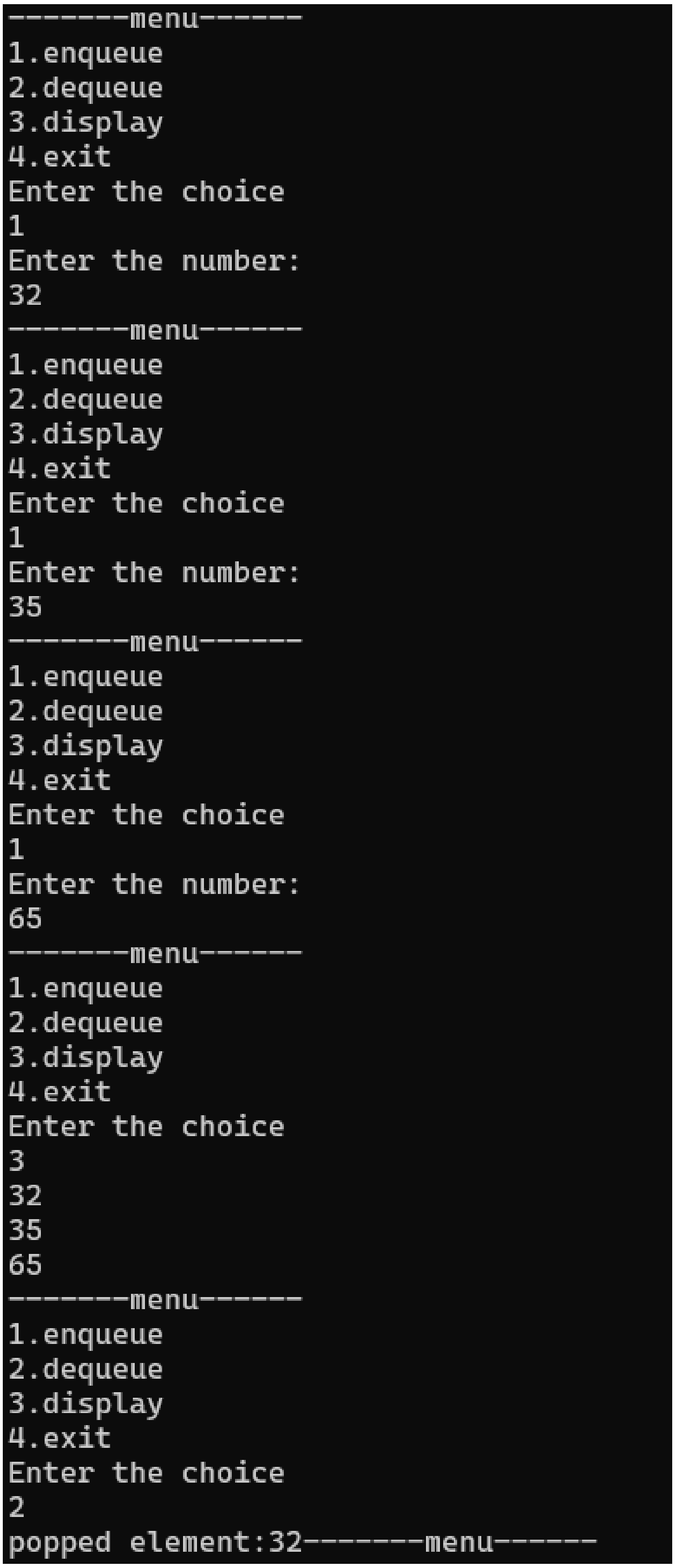


```
-------menu------
1.push
2.pop
3.display
4.exit
Enter the choice
1
Enter the number:
32
-------menu------
1.push
2.pop
3.display
4.exit
Enter the choice
1
Enter the number:
35
-------menu------
1.push
2.pop
3.display
4.exit
Enter the choice
1
Enter the number:
65
-------menu------
1.push
2.pop
3.display
4.exit
Enter the choice
3
65
35
32
-------menu------
1.push
2.pop
3.display
4.exit
Enter the choice
2
popped element:65-------menu------
```

Queue Implementation:

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

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

struct node *head = NULL;
void enqueue()
{
    struct node *new_node;
    int num;
    printf("Enter the number:\n");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    if (head == NULL)
    {
        head = new_node;
        new_node->next = NULL;
    }
    else
    {
        struct node *ptr=head;
        while(ptr->next!=NULL)
        {
            ptr=ptr->next;

        }
        ptr->next=new_node;
        new_node->next = NULL;
    }
}
void dequeue()
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("underflow\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        printf("popped element:%d", ptr->data);
        free(ptr);
    }
}
void display()
{
    struct node *ptr;
    ptr = head;
    if (head == NULL)
    {
        printf("stack is empty\n");
    }
    else
    {
        while (ptr != NULL)
        {
            printf("%d\n", ptr->data);
            ptr = ptr->next;
        }
    }
}
int main()
{  int ch;
    while (1)
    {
        printf("-------menu------\n1.enqueue\n2.dequeue\n3.display\n4.exit\n");
        printf("Enter the choice\n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("invalid number!");
        }
}
}
```

OUTPUT:



```
-------menu------
1.enqueue
2.dequeue
3.display
4.exit
Enter the choice
1
Enter the number:
32
-------menu------
1.enqueue
2.dequeue
3.display
4.exit
Enter the choice
1
Enter the number:
35
-------menu------
1.enqueue
2.dequeue
3.display
4.exit
Enter the choice
1
Enter the number:
65
-------menu------
1.enqueue
2.dequeue
3.display
4.exit
Enter the choice
3
32
35
65
-------menu------
1.enqueue
2.dequeue
3.display
4.exit
Enter the choice
2
popped element:32-------menu------
```

LAB PROGRAM 9:

Write a program 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.**

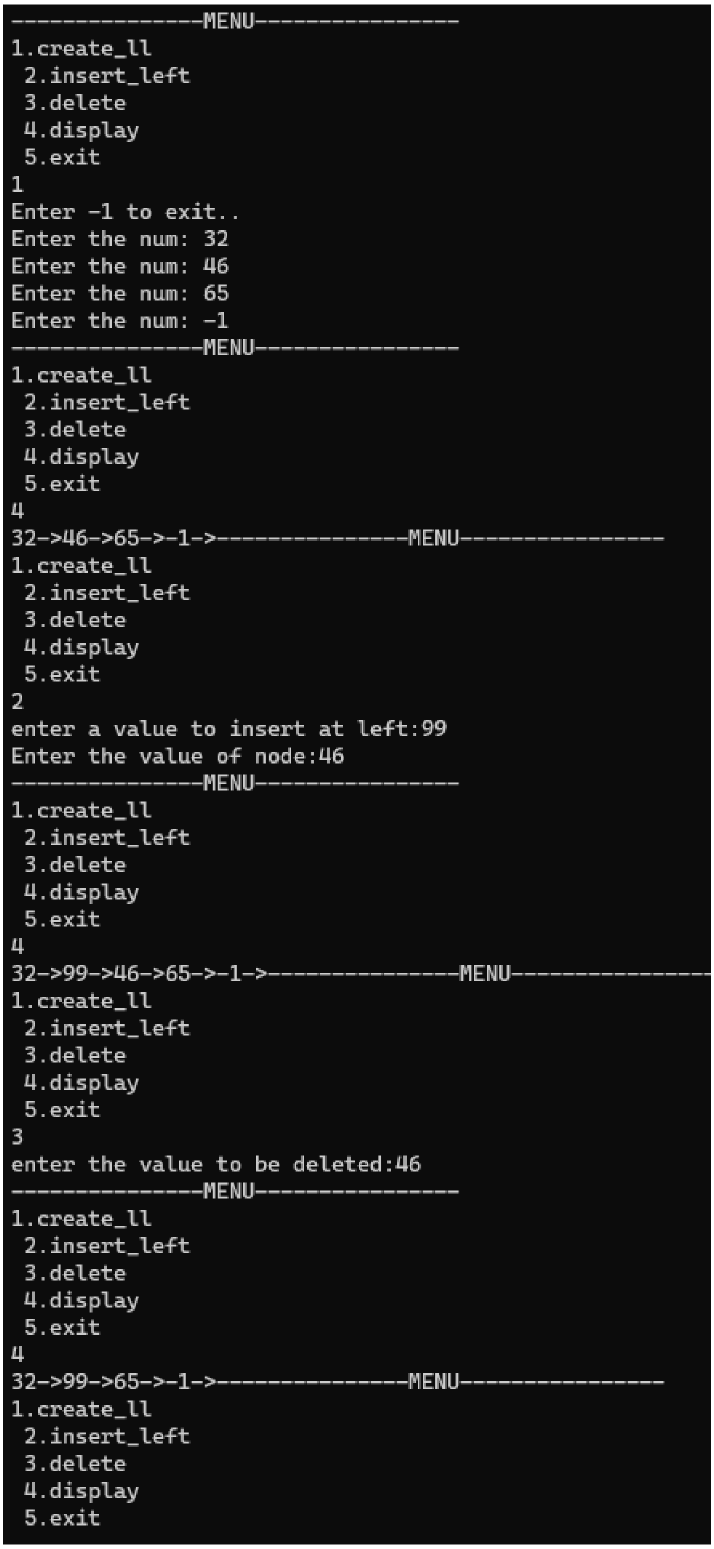
c) **Delete the node based on a specific value**

d) **Display the contents of the list**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{  int data;
   struct node *next;
   struct node *prev;
};
struct node *head;
void create_ll()
{  struct node *new_node, *ptr;
   int num;
   printf("Enter -1 to exit.. \n");
   while (num != -1)
   {  printf("Enter the num: ");
      scanf("%d", &num);
      new_node = (struct node *)malloc(sizeof(struct node));
      new_node->data = num;
      if (head == NULL)
      {  head = new_node;
         new_node->next = NULL;
         new_node->prev = NULL;
      }
      else
      {  ptr = head;
         while (ptr->next != NULL)
         {ptr = ptr->next;}
         ptr->next = new_node;
         new_node->prev = ptr;
         new_node->next = NULL;
      }
   }
}
void insert_left()
{  struct node *new_node, *ptr;
   int val, num;
   new_node = (struct node *)malloc(sizeof(struct node));
   printf("enter a value to insert at left:");
```

```
    scanf("%d", &val);
    printf("Enter the value of node:");
    scanf("%d", &num);
    new_node->data = val;
    ptr = head;
    if (head == NULL)
    {printf("list is empty!");}
    else
    {while (ptr->data != num)
        { ptr = ptr->next;}
        ptr->prev->next = new_node;
        new_node->prev = ptr->prev;
        new_node->next = ptr;
        ptr->prev = new_node;
    }
}
void display()
{  struct node *ptr;
    if (head == NULL)
    {
        printf("Linked list is empty!");
    }
    else
    {
        ptr = head;
        while (ptr != NULL)
        {
            printf("%d->", ptr->data);
            ptr = ptr->next;
        }
    }
}

void del()
{  struct node *ptr;
    int val;
    printf("enter the value to be deleted:");
    scanf("%d", &val);
    ptr = head;
    if (head->data == val)
    {
        ptr = ptr->next;
        head = ptr;
    }
    else
    {
        while (ptr->data != val)
        {
            ptr = ptr->next;
        }
        ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
        free(ptr);
    }
}

int main()
{
    int value, choice;
    while (1)
    {
        printf("---------------MENU----------------\n");
        printf("1.create_ll\n 2.insert_left\n 3.delete\n 4.display\n 5.exit\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            create_ll();
            break;

        case 2:
            insert_left();
            break;

        case 3:
            del();
            break;

        case 4:
            display();
            break;

        case 5:
            exit(0);
            break;

        default:
            printf("wrong input!\n");
            break;
        }
    }
}
```

OUTPUT:



```
---------------MENU----------------
1.create_ll
 2.insert_left
 3.delete
 4.display
 5.exit
1
Enter -1 to exit..
Enter the num: 32
Enter the num: 46
Enter the num: 65
Enter the num: -1
---------------MENU----------------
1.create_ll
 2.insert_left
 3.delete
 4.display
 5.exit
4
32->46->65->-1->---------------MENU----------------
1.create_ll
 2.insert_left
 3.delete
 4.display
 5.exit
2
enter a value to insert at left:99
Enter the value of node:46
---------------MENU----------------
1.create_ll
 2.insert_left
 3.delete
 4.display
 5.exit
4
32->99->46->65->-1->---------------MENU---------------
1.create_ll
 2.insert_left
 3.delete
 4.display
 5.exit
3
enter the value to be deleted:46
---------------MENU----------------
1.create_ll
 2.insert_left
 3.delete
 4.display
 5.exit
4
32->99->65->-1->---------------MENU----------------
1.create_ll
 2.insert_left
 3.delete
 4.display
 5.exit
```

LAB PROGRAM 10:

Write a program

a) **To construct a binary Search tree.**

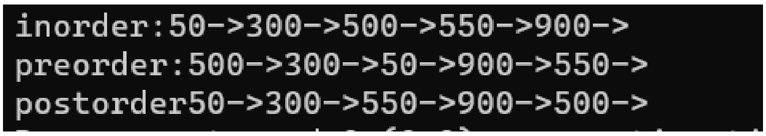
b) **To traverse the tree using all the methods i.e., in-order, preorder and post order**

c) **To display the elements in the tree.**

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node* left;
   struct node* right;
};
struct node* create_tree(int data)
{struct node * new_node;
   new_node=(struct node*)malloc(sizeof(struct node));
   new_node->data=data;
   new_node->left=new_node->right=NULL;
   return new_node;
}

struct node* insert(struct node* root,int data)
{if(root==NULL)
      return create_tree(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 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);
   }
}
void display(struct node *root)
{if (root!=NULL)
   {printf("inorder:");
      inorder(root);
      printf("\npreorder:");
      preorder(root);
      printf("\npostorder");
      postorder(root);

   }
}
int main()
{struct node *root=NULL;
   root=insert(root,500);
   root=insert(root,300);
   root=insert(root,900);
   root=insert(root,550);
   root=insert(root,50);
   display(root);
}
```

OUTPUT:



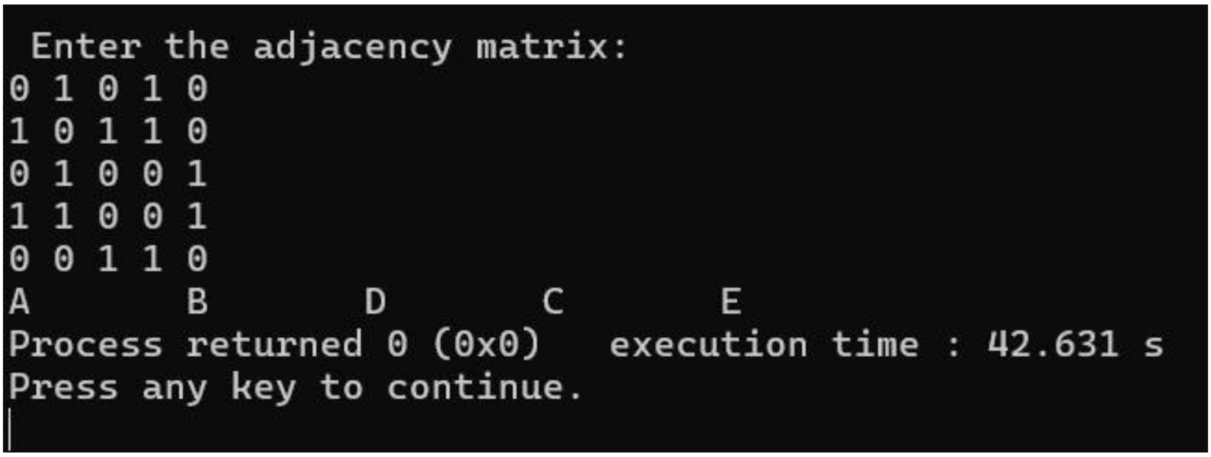
```
inorder:50->300->500->550->900->
preorder:500->300->50->900->550->
postorder50->300->550->900->500->
```

LAB PROGRAM 9:

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

```
#include <stdio.h>
#define MAX 5
void breadth_first_search(int adj[][MAX],int visited[],int start){
int queue[MAX],rear = -1,front = -1, i;
queue[++rear] = start;
visited[start] = 1;
while(rear != front)
{ start = queue[++front];
if(start == 4)
printf("%c\t",start+65);
else {printf("%c \t",start + 65);
for(i = 0; i < MAX; i++) {
    if(adj[start][i] == 1 && visited[i] ==0)
    { queue[++rear] = i;
    visited[i] = 1;
    } }}
}
int main()
{ int visited[MAX] = {0};
 int adj[MAX][MAX], i, j;
 printf("\n Enter the adjacency matrix:");
for(i = 0; i < MAX; i++)
for(j=0;j<MAX; j++)
scanf("%d", &adj[i][j]);
breadth_first_search(adj,visited,0);
return 0;
}
```

OUTPUT:

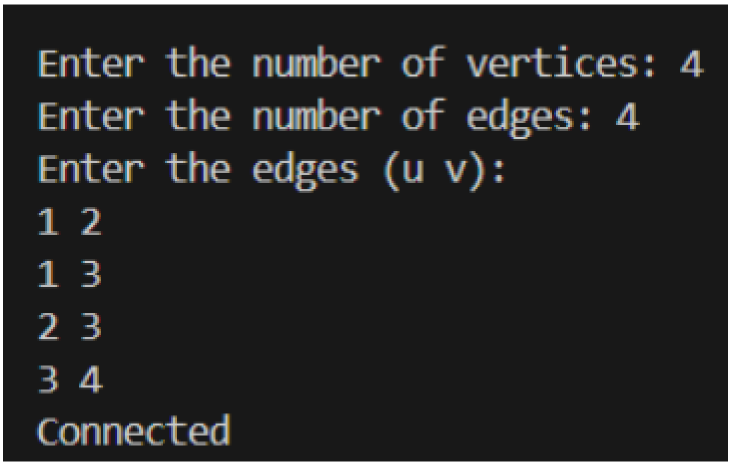


```
 Enter the adjacency matrix:
0 1 0 1 0
1 0 1 1 0
0 1 0 0 1
1 1 0 0 1
0 0 1 1 0
A       B       D       C       E
Process returned 0 (0x0)   execution time : 42.631 s
Press any key to continue.
```

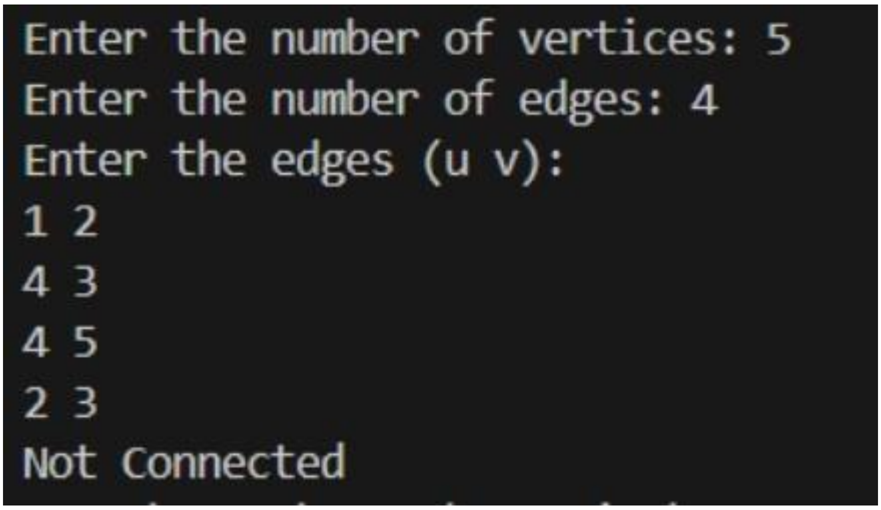
b) Write a program to check whether a given graph is connected or not using the DFS method.

```
#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#define N 50 int gr[N][N];

 bool vis[N];

void Add_edge(int u, int v)

{ gr[u][v] = 1;}

 void dfs(int x)

{ vis[x] = true;

for (int i = 1; i <= N; i++)

{ if (gr[x][i] && !vis[i])

                    dfs(i);

}

bool Is_Connected(int n)

{ memset(vis, false, sizeof vis);

dfs(1);

for (int i = 1; i <= n; i++){

if (!vis[i])

return false; }
return true;}
int main()
{ int n, u, v;
printf("Enter the number of vertices: ");
scanf("%d", &n);
printf("Enter the number of edges: ");
int m; scanf("%d", &m);
printf("Enter the edges (u v):\n");
for (int i = 0; i < m; ++i)
{ scanf("%d %d", &u, &v);
Add_edge(u, v);
}
if (Is_Connected(n))
printf("Connected\n");
else
printf("Not Connected\n");
return 0;
}
```

OUTPUT:



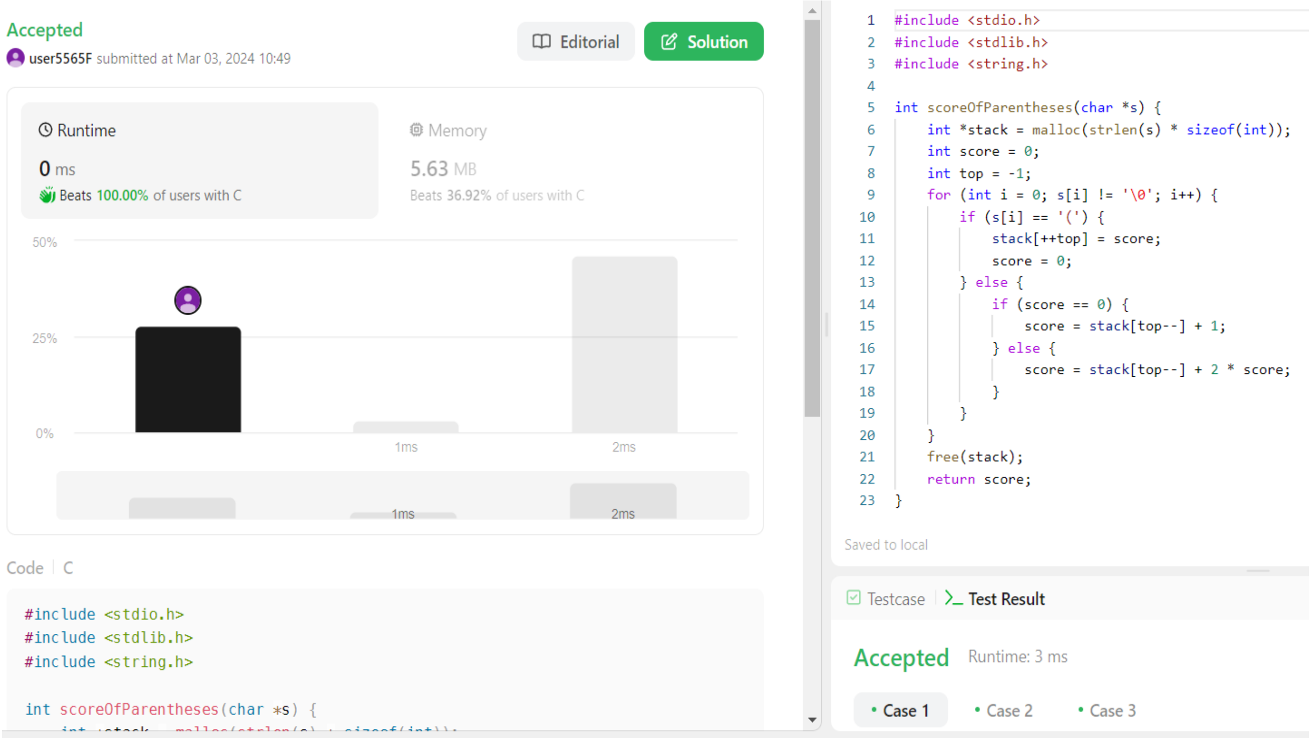
```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter the edges (u v):
1 2
1 3
2 3
3 4
Connected
```



```
Enter the number of vertices: 5
Enter the number of edges: 4
Enter the edges (u v):
1 2
4 3
4 5
2 3
Not Connected
```

LeetCode Programs:

1.Score of Parentheses(LP:856)



Accepted

user5565F submitted at Mar 03, 2024 10:49

Runtime

0 ms

Beats 100.00% of users with C

Memory

5.63 MB

Beats 36.92% of users with C

Code | C

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

int scoreOfParentheses(char *s) {
```

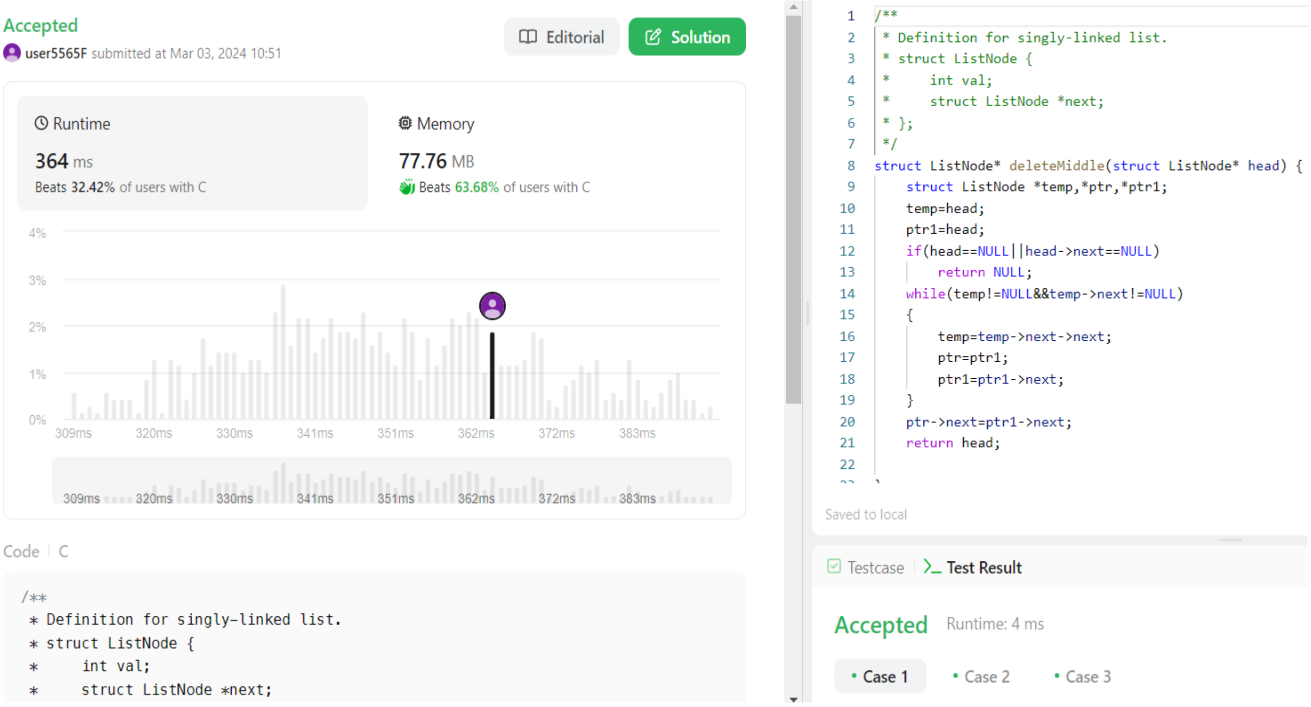
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int scoreOfParentheses(char *s) {
    int *stack = malloc(strlen(s) * sizeof(int));
    int score = 0;
    int top = -1;
    for (int i = 0; s[i] != '\0'; i++) {
        if (s[i] == '(') {
            stack[++top] = score;
            score = 0;
        } else {
            if (score == 0) {
                score = stack[top--] + 1;
            } else {
                score = stack[top--] + 2 * score;
            }
        }
    }
    free(stack);
    return score;
}
```

Saved to local

Accepted Runtime: 3 ms

• Case 1 • Case 2 • Case 3

2.Delete middle node of linked list.(LP:2095)



Accepted

user5565F submitted at Mar 03, 2024 10:51

Runtime

364 ms

Beats 32.42% of users with C

Memory

77.76 MB

Beats 63.68% of users with C

Code | C

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
```

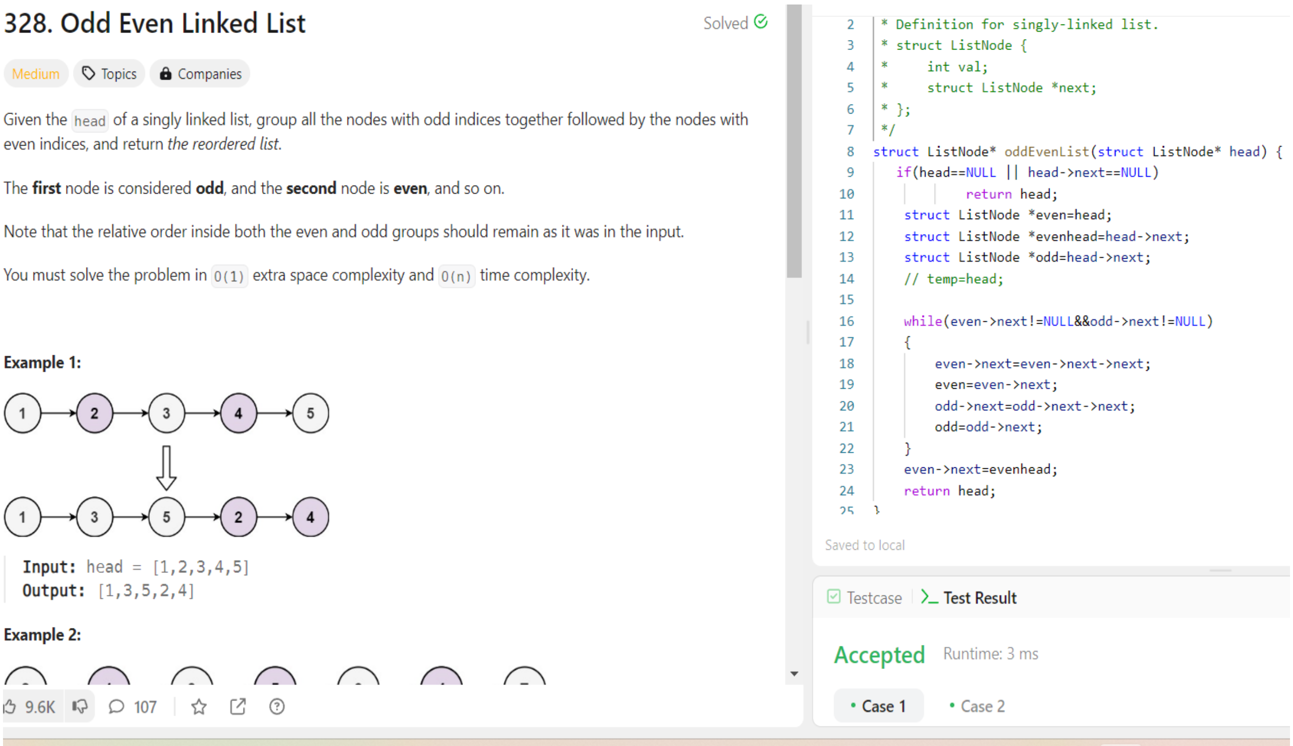
```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* deleteMiddle(struct ListNode* head) {
    struct ListNode *temp,*ptr,*ptr1;
    temp=head;
    ptr1=head;
    if(head==NULL||head->next==NULL)
        return NULL;
    while(temp!=NULL&&temp->next!=NULL)
    {
        temp=temp->next->next;
        ptr=ptr1;
        ptr1=ptr1->next;
    }
    ptr->next=ptr1->next;
    return head;

}
```

Saved to local

Accepted Runtime: 4 ms

• Case 1 • Case 2 • Case 3

3.Odd Even Linked List(LP:328)



328. Odd Even Linked List

Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

Example 1:

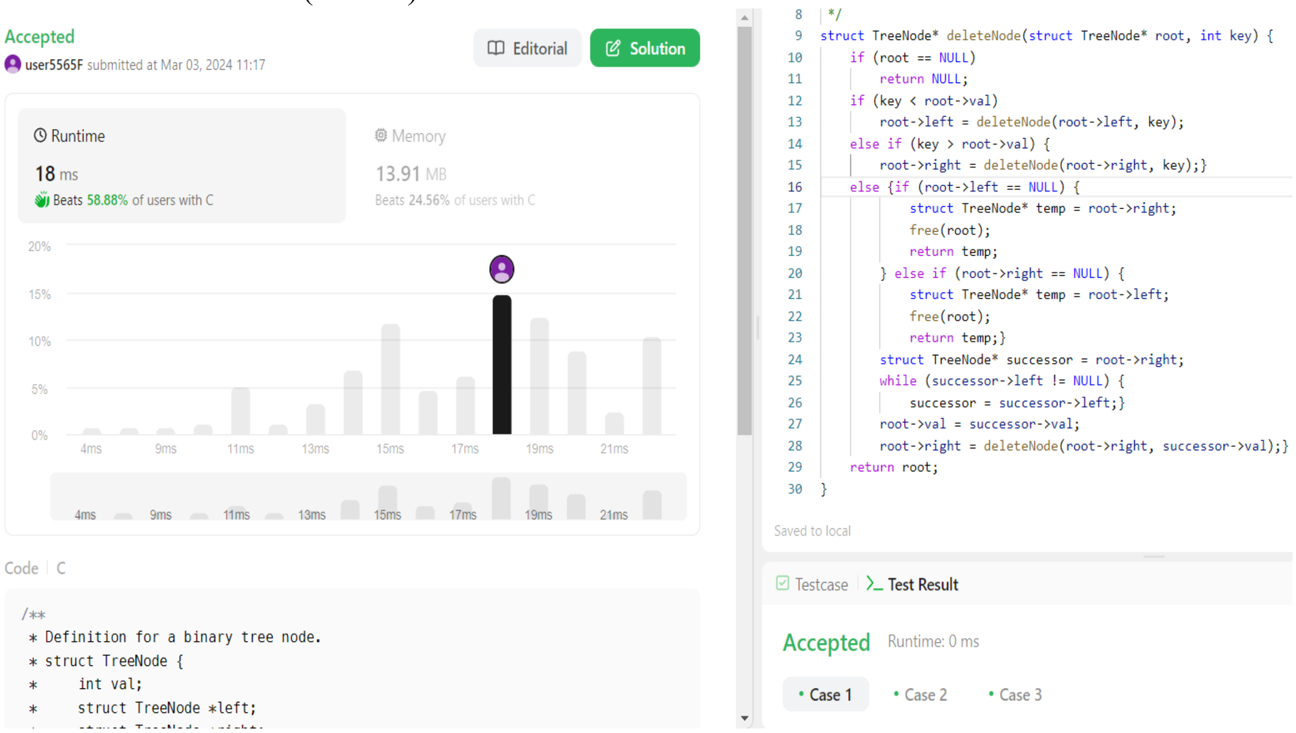
```
Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]
```

Example 2:

```
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* oddEvenList(struct ListNode* head) {
  if(head==NULL || head->next==NULL)
           return head;
   struct ListNode *even=head;
   struct ListNode *evenhead=head->next;
   struct ListNode *odd=head->next;
   // temp=head;

   while(even->next!=NULL&&odd->next!=NULL)
   {
       even->next=even->next->next;
       even=even->next;
       odd->next=odd->next->next;
       odd=odd->next;
   }
   even->next=evenhead;
   return head;
}
```

Accepted Runtime: 3 ms

4.Delete a node in BST.(LP:450)



Accepted

user5565F submitted at Mar 03, 2024 11:17

Runtime

18 ms

Beats 58.88% of users with C

Memory

13.91 MB

Beats 24.56% of users with C

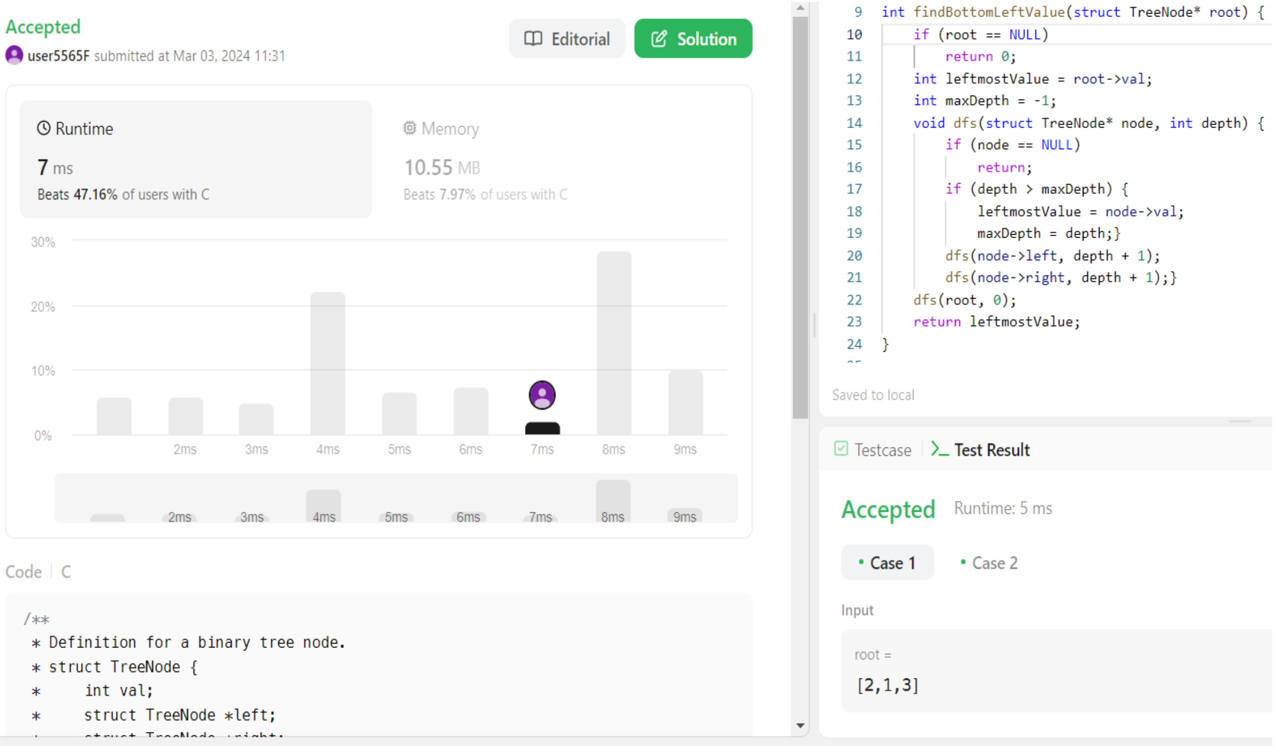
Code | C

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
```

```
 */
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL)
        return NULL;
    if (key < root->val)
        root->left = deleteNode(root->left, key);
    else if (key > root->val) {
        root->right = deleteNode(root->right, key);}
    else {if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;}
        struct TreeNode* successor = root->right;
        while (successor->left != NULL) {
            successor = successor->left;}
        root->val = successor->val;
        root->right = deleteNode(root->right, successor->val);}
    return root;
}
```

Accepted Runtime: 0 ms

5.Bottom Left Tree Value.(LP:513)



Accepted

user5565F submitted at Mar 03, 2024 11:31

Runtime

7 ms

Beats 47.16% of users with C

Memory

10.55 MB

Beats 7.97% of users with C

Code | C

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
```

```
int findBottomLeftValue(struct TreeNode* root) {
    if (root == NULL)
        return 0;
    int leftmostValue = root->val;
    int maxDepth = -1;
    void dfs(struct TreeNode* node, int depth) {
        if (node == NULL)
            return;
        if (depth > maxDepth) {
            leftmostValue = node->val;
            maxDepth = depth;}
        dfs(node->left, depth + 1);
        dfs(node->right, depth + 1);}
    dfs(root, 0);
    return leftmostValue;
}
```

Saved to local

Accepted Runtime: 5 ms

Input

root =

[2,1,3]