VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

APRAMEYA S J (1BM23CS048)

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 Aprameya S J **(1BM23CS048)**, 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 202425. 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.

K Sunil Kumar
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
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.

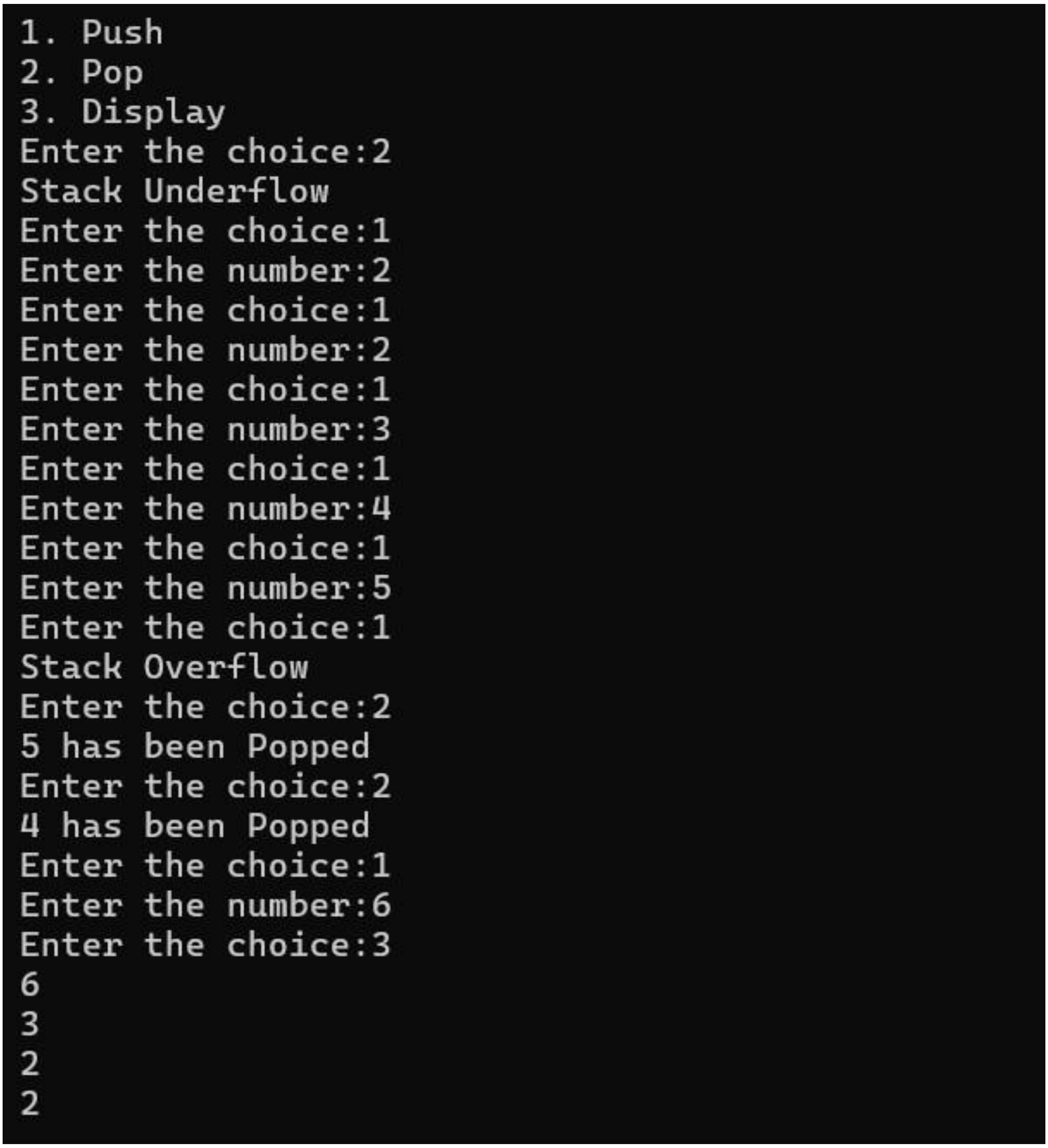
Lab-1

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>

#define  LEN  5  int

stack[5],top=-1;

int arrlen = sizeof(stack)/sizeof(stack[0]);

void  pop();  void  push();  void  display();

void    main(){    int    n,choice;    printf("1.

Push\n");  printf("2.  Pop\n");  printf("3.

Display\n");  printf("Enter  the  choice:");

scanf("%d",&choice);


while(choice == 1 || choice == 2 || choice ==
   3){ switch(choice){ case 1:
        push();
break; case 2:

        pop(); break;

     case 3:

        display();

        exit(0);

  }

  printf("Enter the choice:");

  scanf("%d",&choice);

  }


  }


  void push(){ if(top

  == arrlen-1){

     printf("Stack Overflow\n");

  }else{ int num;

     printf("Enter the

     number:");

     scanf("%d",&num);

     top = top+1;

  stack[top] = num;

  } }
```

```
void pop(){

if(top == -1)
{printf("Stack Underflow\n");
}
else{
   int ele; ele = stack[top]; printf("%d
   has been Popped\n",ele); top = top-
   1;
}
}
void display(){
if(top == -1){
   printf("Stack Underflow\n");
}
else{
   for(int i=top;i>=0;i--){
      printf("%d\n",stack[i]);
   }
}
}
```

Output



```
1. Push
2. Pop
3. Display
Enter the choice:2
Stack Underflow
Enter the choice:1
Enter the number:2
Enter the choice:1
Enter the number:2
Enter the choice:1
Enter the number:3
Enter the choice:1
Enter the number:4
Enter the choice:1
Enter the number:5
Enter the choice:1
Stack Overflow
Enter the choice:2
5 has been Popped
Enter the choice:2
4 has been Popped
Enter the choice:1
Enter the number:6
Enter the choice:3
6
3
2
2
```

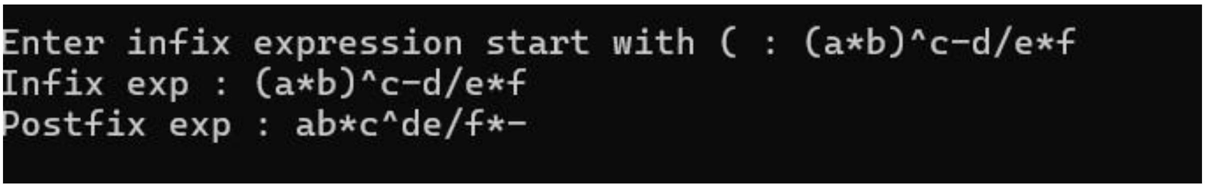
Lab-2

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

```
#include <stdio.h> #include
<string.h>
int index1=0, pos = 0, top = -1, length; char symbol,
temp,   infix[20],   postfix[20],   stack[20];   void
infixpostfix(); void push(char symbol); char pop();
int pred(char symbol);



int main()
{
   printf("Enter infix expression start with ( :
   "); scanf("%s",&infix); infixpostfix();
   printf("Infix exp : %s \n",infix);
   printf("Postfix exp : %s \n",postfix); return
   0;
}



void infixpostfix()
{
   length = strlen(infix);
   push('#');
   while(index1 < length)
   {
      symbol = infix[index1];
      switch(symbol)
      {
      case '(':
         push(symbol)
      ; break; case ')':
         temp = pop();
         while(temp != '(')
            {
                postfix[pos++] = temp;
                temp = pop();
            }
            break;

        case '+' :
        case '-':
        case '*':
        case '/':
        case '^':
            while(pred(stack[top]) >= pred(symbol))
            {
                temp = pop();
                postfix[pos++]=temp;
            }
            push(symbol);
            break;
        default: postfix[pos++] =
            symbol;
        }
        index1++;
    }
    while(top > 0)
    {
        temp = pop();
        postfix[pos++] = temp;
    }

}



void push(char symbol)
{
    stack[++top] = symbol;
}
```

```
char pop()
{
   char symb; symb =
   stack[top--] ;
   return symb;
}

int pred(char symbol)
{
   int p ;
   switch(symbol)
   {
   case '^':
      p=3;
      break;
   case '*':
   case '/':
      p=2;
      break;
   case '+':
   case   '-':
   p=1;
   break;
   case   '(':
   p   =   0;
   break;
   case  '#':
   p   =   -1;
   break;
   }
   return p;
}
```

Output



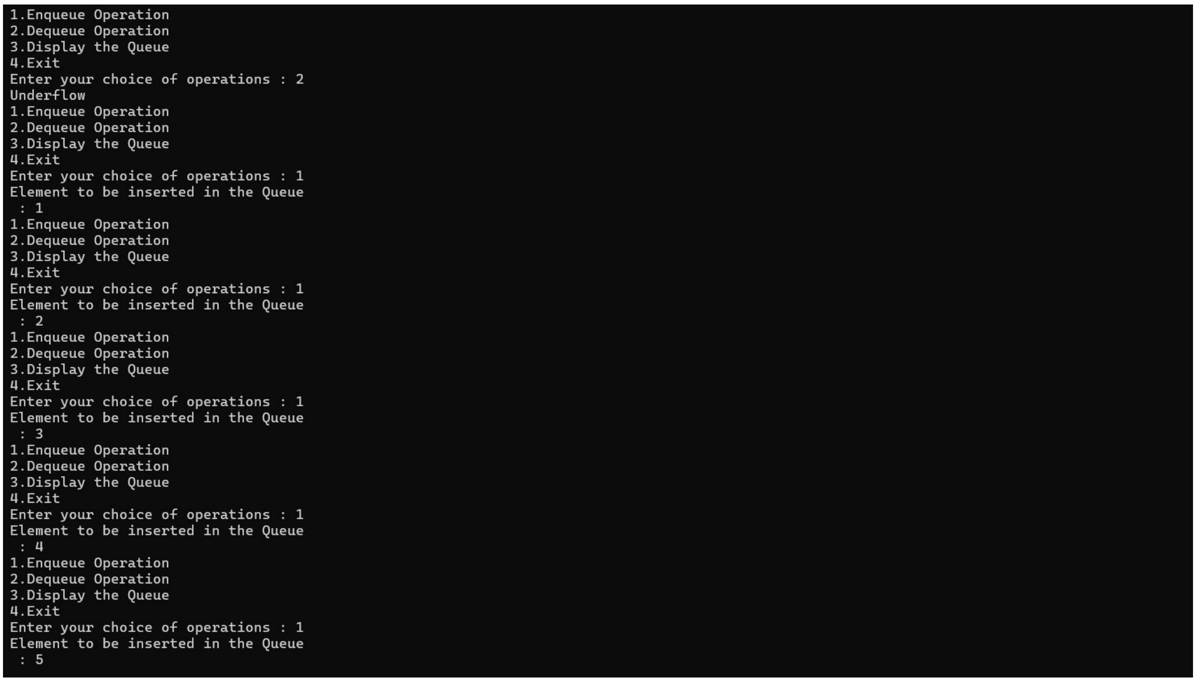
```
Enter infix expression start with ( : (a*b)^c-d/e*f
Infix exp : (a*b)^c-d/e*f
Postfix exp : ab*c^de/f*-
```

Lab-3

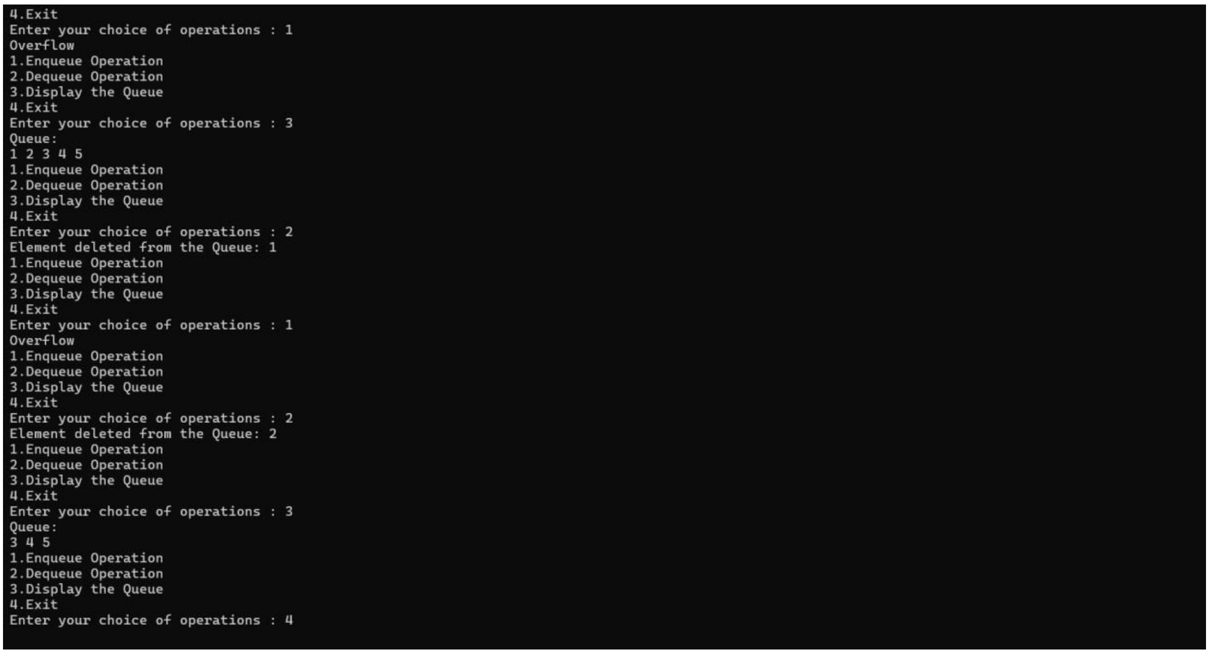
3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>

#  define   SIZE   3

void     enqueue();

void     dequeue();

void   show();   int

inp_arr[SIZE];   int

Rear   =   -   1;   int

Front = - 1; main()

{

   int ch; while

   (1)

   {

      printf("1.Enqueue  Operation\n");

      printf("2.Dequeue Operation\n");

      printf("3.Display  the  Queue\n");

      printf("4.Exit\n");

      printf("Enter your choice of operations : ");

      scanf("%d", &ch); switch (ch)

      {
          case 1:
          enqueue();
          break;
          case 2:
          dequeue();
          break;
          case 3:
          show();
          break;
          case 4:
          exit(0);
          default:
          printf("Incorrect choice \n");
        }
    }
}

void enqueue()
{
    int  insert_item;  if
    (Rear == SIZE - 1)
      printf("Overflow \n"); else
    {
        if (Front == - 1)


        Front = 0;
        printf("Element to be inserted in the Queue\n : ");
        scanf("%d",   &insert_item);   Rear   =   Rear   +   1;
        inp_arr[Rear] = insert_item;
    }
}

void dequeue()
{
    if (Front == - 1 || Front > Rear)
    {
        printf("Underflow \n"); return
        ;
    }
    else
    {
        printf("Element deleted from the Queue: %d\n", inp_arr[Front]);
        Front = Front + 1;
    }
}
void show()
{
    if (Front == - 1)
        printf("Empty Queue \n");
    else
    {
        printf("Queue: \n"); for (int i =
        Front; i <= Rear; i++)
            printf("%d ", inp_arr[i]);
        printf("\n");
    }
}
```

Output



```
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 2
Underflow
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Element to be inserted in the Queue
 : 1
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Element to be inserted in the Queue
 : 2
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Element to be inserted in the Queue
 : 3
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Element to be inserted in the Queue
 : 4
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Element to be inserted in the Queue
 : 5
```



```
4.Exit
Enter your choice of operations : 1
Overflow
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 3
Queue:
1 2 3 4 5
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 2
Element deleted from the Queue: 1
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Overflow
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 2
Element deleted from the Queue: 2
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 3
Queue:
3 4 5
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 4
```

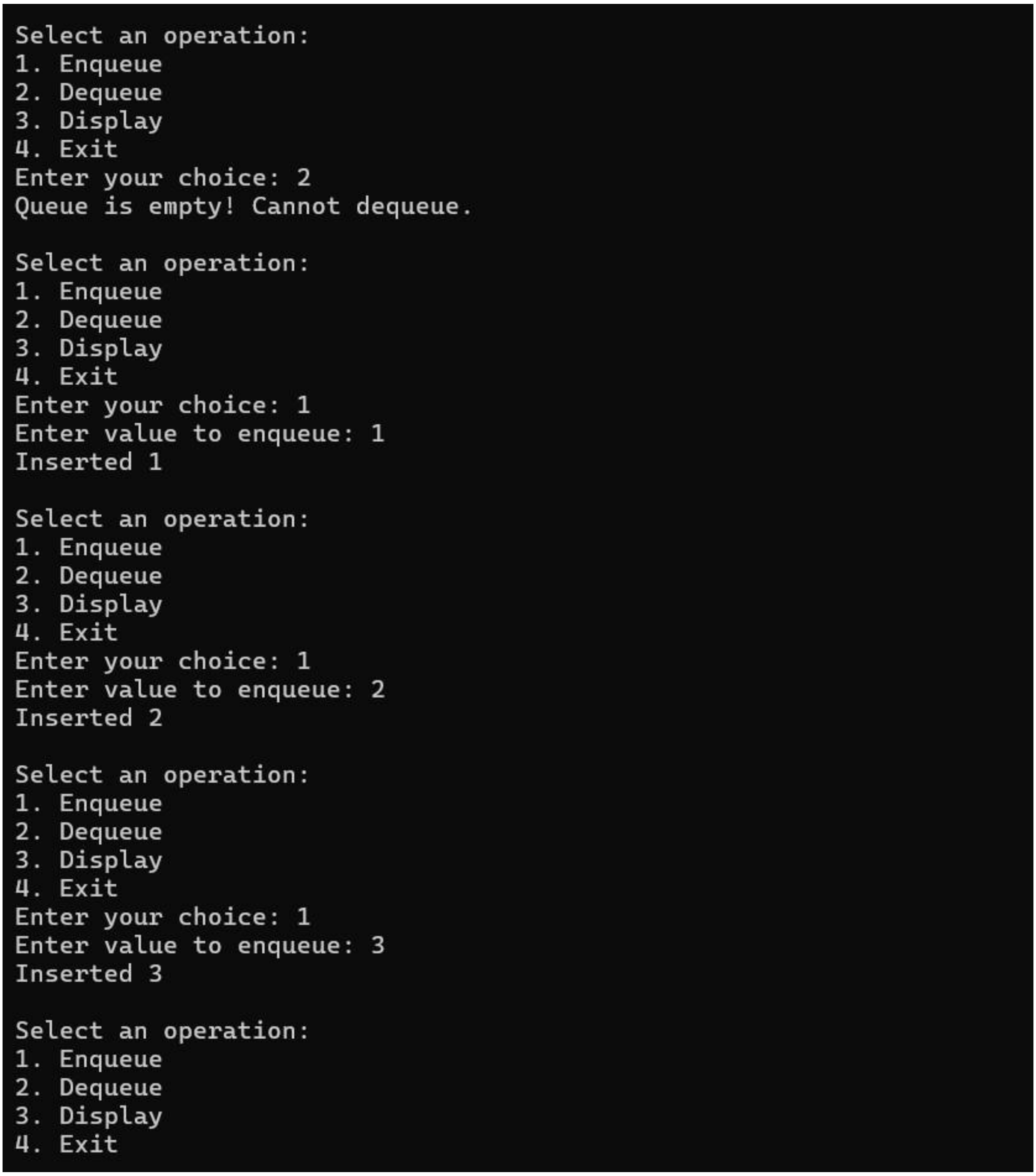
3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display

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

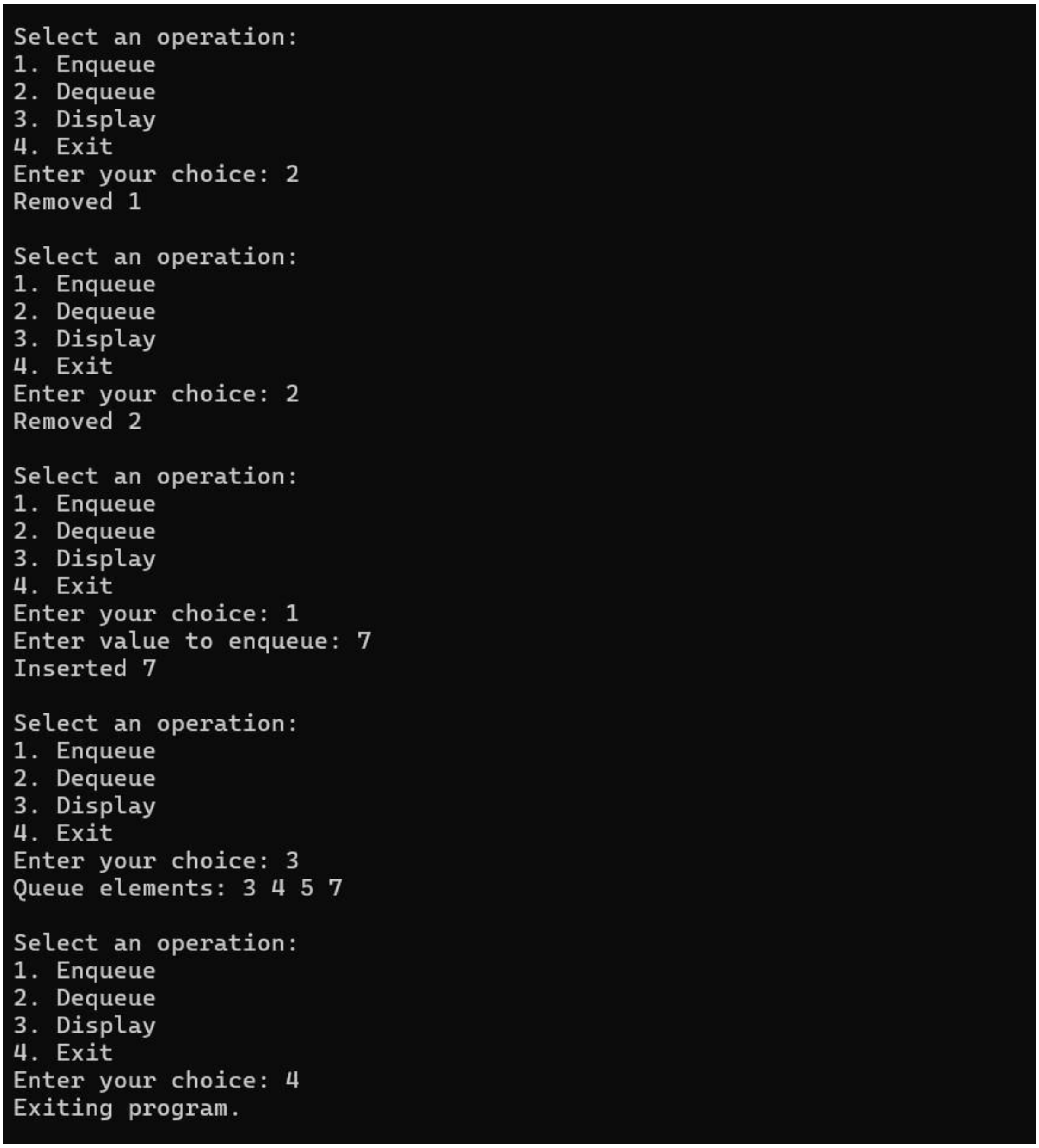
```
#include <stdio.h>


#define MAX 5 // Define the maximum capacity of the queue


int queue[MAX];

int front = -1; int

rear = -1;


// Check if the queue is full int isFull()

{ return ((rear + 1) % MAX == front);
}

// Check if the queue is empty
int isEmpty() { return (front ==
-1);
}

// Add an element to the queue
void enqueue(int value) { if
   (isFull()) {
      printf("Queue is full! Cannot insert %d.\n", value);
   }    else    {    if
      (isEmpty())    {
      front = 0;
      }
      rear  =  (rear  +  1)  %  MAX;
      queue[rear]        =        value;
      printf("Inserted            %d\n",
      value);
   }
}
```

```
// Remove an element from the queue int
dequeue() { if (isEmpty()) { printf("Queue is empty!
Cannot dequeue.\n"); return -1;
  } else { int element =
    queue[front]; if (front ==
    rear) {
      // Queue has only one element, so we reset it after dequeueing
      front = -1; rear = -1;
    } else { front = (front + 1) %
      MAX;
    }
    printf("Removed %d\n", element); return
    element;
  }
}

// Display the elements in the queue void
display() {
  if (isEmpty()) {
    printf("Queue is empty!\n");
  } else { printf("Queue
    elements: "); int i = front;
    while (1) {
            printf("%d ", queue[i]);
            if (i == rear) break; i =
            (i + 1) % MAX;
        }
        printf("\n");
    }
}
int   main()   {   int
    choice,      value;
    while (1) {
        printf("\nSelect an operation:\n");
        printf("1.  Enqueue\n");  printf("2.
        Dequeue\n");                  printf("3.
        Display\n");   printf("4.   Exit\n");
        printf("Enter    your    choice:    ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d",                    &value);
                enqueue(value); break;
            case 2:
                dequeue();
                break;
            case 3:
            display();
            break; case
            4:
            printf("Exitin
            g
            program.\n")
            ; return 0;
            default:
               printf("Invalid choice. Please try again.\n");
        }
    }
}
```

Output



```
Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue is empty! Cannot dequeue.

Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 1
Inserted 1

Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 2
Inserted 2

Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 3
Inserted 3

Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
```



```
Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Removed 1

Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Removed 2

Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 7
Inserted 7

Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 3 4 5 7

Select an operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program.
```

Lab-4 and Lab-5 // Since we did Both lab programs In one lab

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.

5) WAP to Implement Singly Linked List with following operations

a) Create a linked list.

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

c) Display the contents of the linked list.

```
#include<stdio.h>

#include<stdlib.h>


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


struct  node  *start  =  NULL;  struct
node *create_ll(struct node*); struct
node  *display(struct  node*);  struct
node    *insert_beg(struct    node*);
struct      node      *insert_end(struct
node*);             struct             node
*insert_atPos(struct  node*);  struct
node    *delete_beg(struct    node*);
```

```
struct     node     *delete_end(struct
node*);            struct            node
*delete_atPos(struct node*);

int main() { int
   choice;
   printf("\n1: Create LL\n2: Display\n3: Insert at Beginning\n4: Insert
at End\n5: Insert at Position\n6: Delete from Beginning\n7: Delete
from End\n8: Delete from Position\n9: Exit\n"); int flag = 1; while
(flag) { printf("\nEnter your choice: "); scanf("%d", &choice); switch
(choice) { case 1: start = create_ll(start); break; case 2: start =
display(start); break; case 3: start = insert_beg(start); break; case
4: start = insert_end(start); break; case 5: start =
insert_atPos(start); break; case 6: start = delete_beg(start); break;
case 7: start = delete_end(start); break;
         case 8: start = delete_atPos(start); break; case
         9: flag = 0; break;
         default: printf("Invalid choice. Try again.\n");
      }
   }
   return 0;
}

struct node *create_ll(struct node *start) {
    struct node *new_node, *ptr;
    int num; printf("Enter num:
    "); scanf("%d", &num);
    while(num != -1) {
        new_node = (struct node*)malloc(sizeof(struct node));
        new_node->data = num; if(start == NULL) { new_node-
        >next = NULL; start = new_node;
        } else { ptr =
            start;
            while(ptr->next != NULL) ptr = ptr->next;
            ptr->next = new_node; new_node->next
            = NULL;

        }
        printf("Enter num: "); scanf("%d",
        &num);
    }
    return start;
}

struct node *display(struct node *start) {
    struct node *ptr; ptr = start;
    while (ptr != NULL) {
```

```
    printf("\t   %d",   ptr->data);
    ptr = ptr->next;
    }
    return start;
}

struct node *insert_beg(struct node *start) {
    struct node *new_node;
    int   num;   printf("Enter
    num:    ");    scanf("%d",
    &num);
    new_node = (struct node*)malloc(sizeof(struct node)); new_node-
    >data = num;
    new_node->next = start;
    start      =      new_node;
    return start;
}

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

struct node *insert_atPos(struct node *start) {
    struct node *new_node, *ptr, *preptr;
    int num, indx = 0, pos; printf("Enter
    num: "); scanf("%d", &num);
    printf("Enter position: "); scanf("%d",
    &pos);

    if(pos < 0) {
        printf("Invalid position.\n"); return
        start;
    }
```

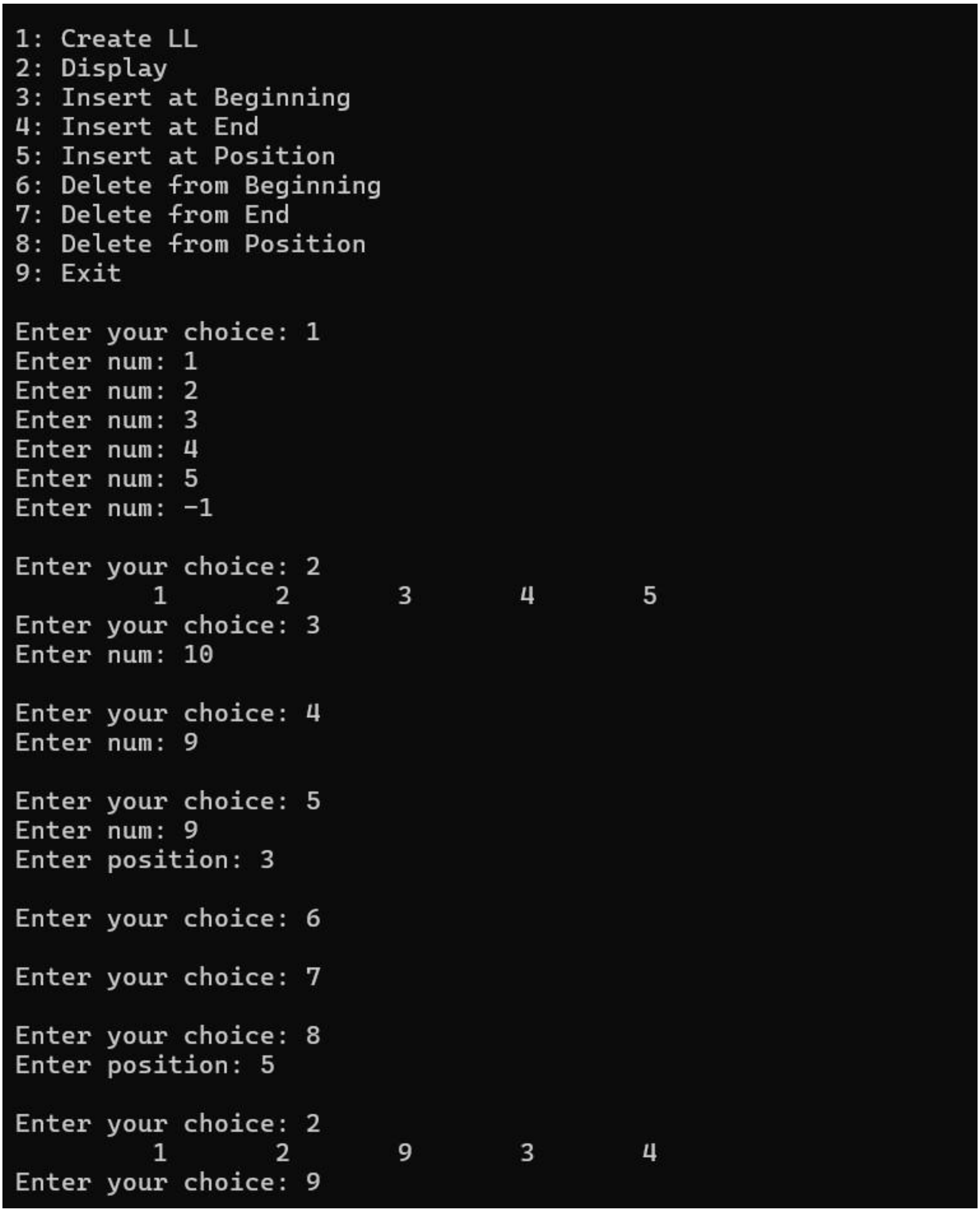
```
new_node = (struct node*)malloc(sizeof(struct node)); new_node-
>data = num;
ptr = start;

if(pos == 0) {
   new_node->next = start;
   start      =      new_node;
   return start;
}

while(ptr != NULL && indx < pos) {
   preptr = ptr;
   ptr = ptr->next; indx++;
}

if(ptr == NULL && indx < pos) {
   printf("Position is greater than the length of the list.\n");
   free(new_node); return start;
}

preptr->next = new_node; new_node->next
= ptr;
```

```
    return start;
}

struct node *delete_beg(struct node *start) {
    struct   node   *ptr;
    ptr = start; start =
    start->next;
    free(ptr);     return
    start;
}
struct node *delete_end(struct node *start) {
    struct node *ptr, *preptr;
    ptr   =   start;   while(ptr-
    >next != NULL) {
        preptr = ptr; ptr
        = ptr->next;
    }
    preptr->next = NULL;
    free(ptr);        return
    start;
}

struct node *delete_atPos(struct node *start) {
```

```
struct node *ptr, *preptr;
int    indx    =    0,    pos;
printf("Enter position: ");
scanf("%d", &pos);

if(pos < 0 || start == NULL) {
   printf("Invalid position or empty list.\n"); return
   start;
}

ptr = start;
preptr = NULL;

if(pos == 0) { start =
   start->next;
   free(ptr);    return
   start;
}

while(ptr != NULL && indx < pos) {
   preptr   =   ptr;
   ptr = ptr->next;
   indx++;
```

```
    }

    if(ptr == NULL) { printf("Position is greater than the length
      of the list.\n"); return start;
    }

    preptr->next = ptr->next; free(ptr);

    return start;
}
```

Output



```
1: Create LL
2: Display
3: Insert at Beginning
4: Insert at End
5: Insert at Position
6: Delete from Beginning
7: Delete from End
8: Delete from Position
9: Exit

Enter your choice: 1
Enter num: 1
Enter num: 2
Enter num: 3
Enter num: 4
Enter num: 5
Enter num: -1

Enter your choice: 2
         1       2       3       4       5
Enter your choice: 3
Enter num: 10

Enter your choice: 4
Enter num: 9

Enter your choice: 5
Enter num: 9
Enter position: 3

Enter your choice: 6

Enter your choice: 7

Enter your choice: 8
Enter position: 5

Enter your choice: 2
         1       2       9       3       4
Enter your choice: 9
```

Lab-6

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

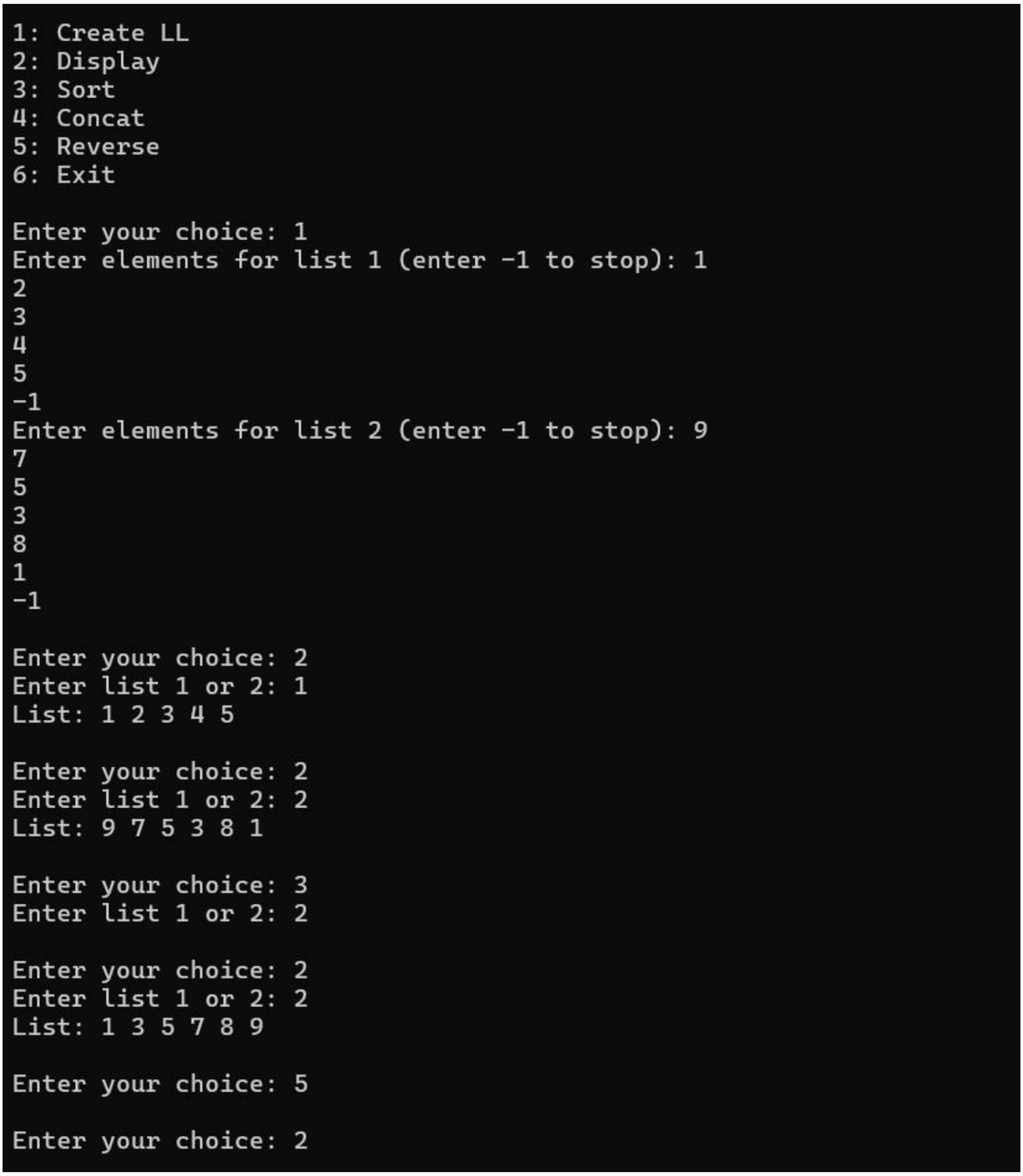
#include<stdio.h>

```
#include<stdlib.h>

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

struct node *head1 = NULL;
struct node *head2 = NULL;
void create_ll();
struct node *display(struct node*); struct
node *sort(struct node*);
struct node *concat(struct node*, struct node*); struct
node *reverse(struct node*);

int main() { int choice; printf("\n1: Create LL\n2: Display\n3:
    Sort\n4: Concat\n5:
Reverse\n6: Exit\n");
    int flag = 1; while
    (flag) {
        printf("\nEnter your choice: ");
        scanf("%d",  &choice);  switch
        (choice) {
```

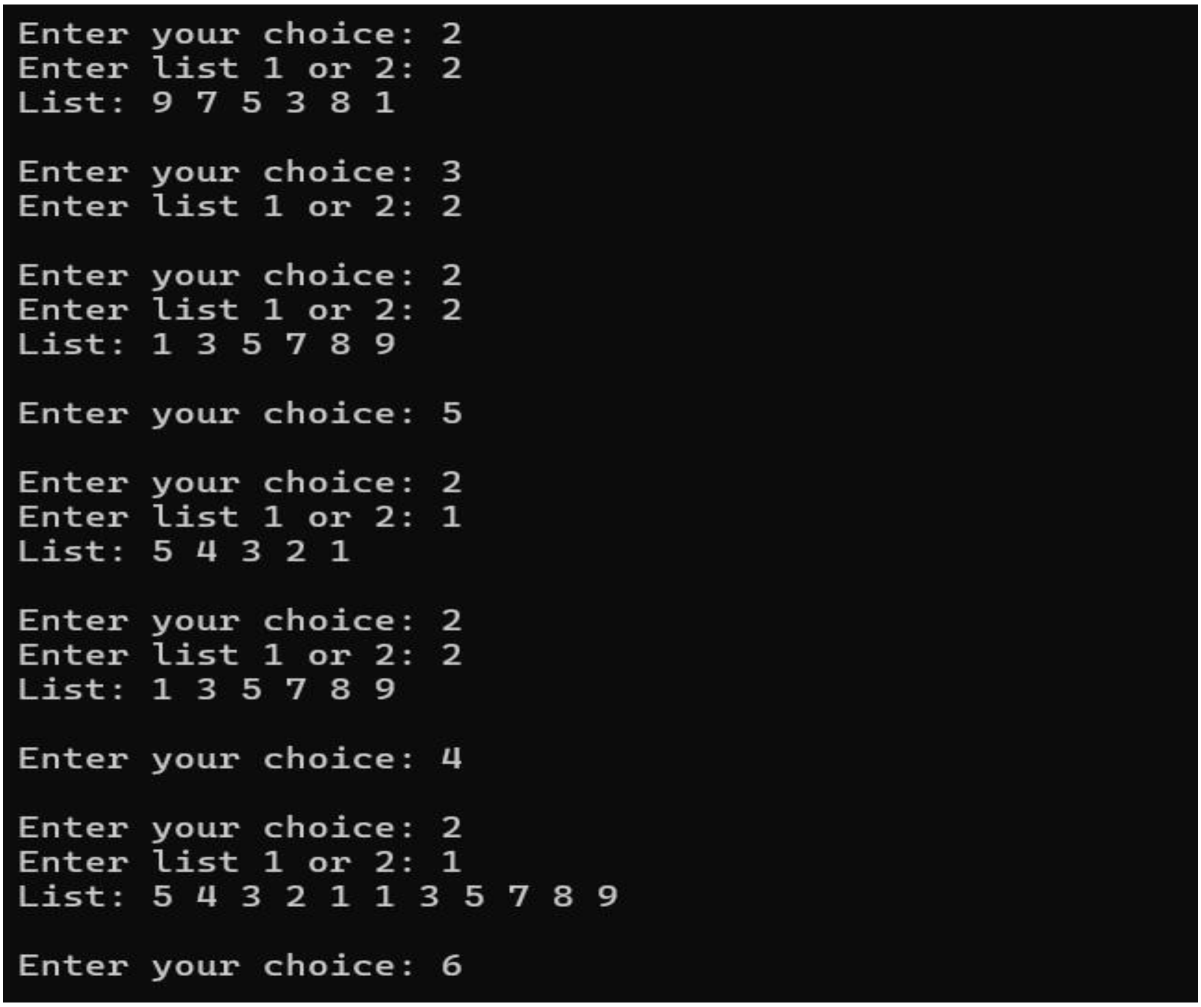
```
case 1: create_ll(); // Create both
   lists break;
case 2: {
   int c; printf("Enter list 1 or
   2: "); scanf("%d", &c);
   if (c == 1) {
      head1 = display(head1);
   } else if (c == 2) {
      head2 = display(head2);
   }   else   {   printf("Invalid   list
      choice.\n");
   }
   break;
}
case 3: {
   int c; printf("Enter list 1 or
   2: "); scanf("%d", &c); if (c
   == 1) {
      head1 = sort(head1);
   } else if (c == 2) { head2
      = sort(head2);
   }   else   {   printf("Invalid   list
      choice.\n");
   }
```

```
            break;
          }
          case 4: head1 = concat(head1, head2); break;
          case 5: head1 = reverse(head1); break; case
          6: flag = 0; break;
          default: printf("Invalid choice. Try again.\n");
        }
    }
    return 0;
}

void create_ll() {
    struct node *new_node, *ptr; int
    num;

   printf("Enter elements for list 1 (enter -1 to stop): ");
                                                         while (1) {
        scanf("%d", &num); if
        (num == -1) break;
        new_node = (struct node*)malloc(sizeof(struct node));
        new_node->data = num; new_node->next = NULL; if
        (head1 == NULL) {
            head1 = new_node;
        }  else  {  ptr  =
            head1;
            while (ptr->next != NULL) ptr = ptr->next; ptr->next
            = new_node;
        }
    }

    printf("Enter elements for list 2 (enter -1 to stop): "); while
    (1) {
        scanf("%d", &num); if
        (num == -1) break;
        new_node = (struct node*)malloc(sizeof(struct node));
        new_node->data = num; new_node->next = NULL; if
        (head2 == NULL) { head2 = new_node;
        }  else  {  ptr  =
            head2;
            while (ptr->next != NULL) ptr = ptr->next; ptr->next
            = new_node;
        }
    }
}

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

struct node *sort(struct node *head) {
    struct node *ptr, *cptr; int
    temp;
    for (ptr = head; ptr != NULL; ptr = ptr->next) {
        for (cptr = ptr->next; cptr != NULL; cptr = cptr->next) {
            if (ptr->data > cptr->data) {
                temp   =   ptr->data;   ptr-
                >data = cptr->data; cptr-
                >data = temp;
            }
        }
    }
    return head;
}

struct node *concat(struct node *head1, struct node *head2) {
    struct   node   *ptr   =   head1;   if
    (head1  ==  NULL)  return  head2;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }
    ptr->next = head2; return
    head1;
}
struct node *reverse(struct node *head) {
    struct node *prev = NULL, *current = head, *next = NULL;
    while  (current  !=  NULL)  {  next  =  current->next;  current-
    >next = prev; prev = current; current = next;
    }
    return prev;
}
```

Output



```
1: Create LL
2: Display
3: Sort
4: Concat
5: Reverse
6: Exit

Enter your choice: 1
Enter elements for list 1 (enter -1 to stop): 1
2
3
4
5
-1
Enter elements for list 2 (enter -1 to stop): 9
7
5
3
8
1
-1

Enter your choice: 2
Enter list 1 or 2: 1
List: 1 2 3 4 5

Enter your choice: 2
Enter list 1 or 2: 2
List: 9 7 5 3 8 1

Enter your choice: 3
Enter list 1 or 2: 2

Enter your choice: 2
Enter list 1 or 2: 2
List: 1 3 5 7 8 9

Enter your choice: 5

Enter your choice: 2
```



```
Enter your choice: 2
Enter list 1 or 2: 2
List: 9 7 5 3 8 1

Enter your choice: 3
Enter list 1 or 2: 2

Enter your choice: 2
Enter list 1 or 2: 2
List: 1 3 5 7 8 9

Enter your choice: 5

Enter your choice: 2
Enter list 1 or 2: 1
List: 5 4 3 2 1

Enter your choice: 2
Enter list 1 or 2: 2
List: 1 3 5 7 8 9

Enter your choice: 4

Enter your choice: 2
Enter list 1 or 2: 1
List: 5 4 3 2 1 1 3 5 7 8 9

Enter your choice: 6
```

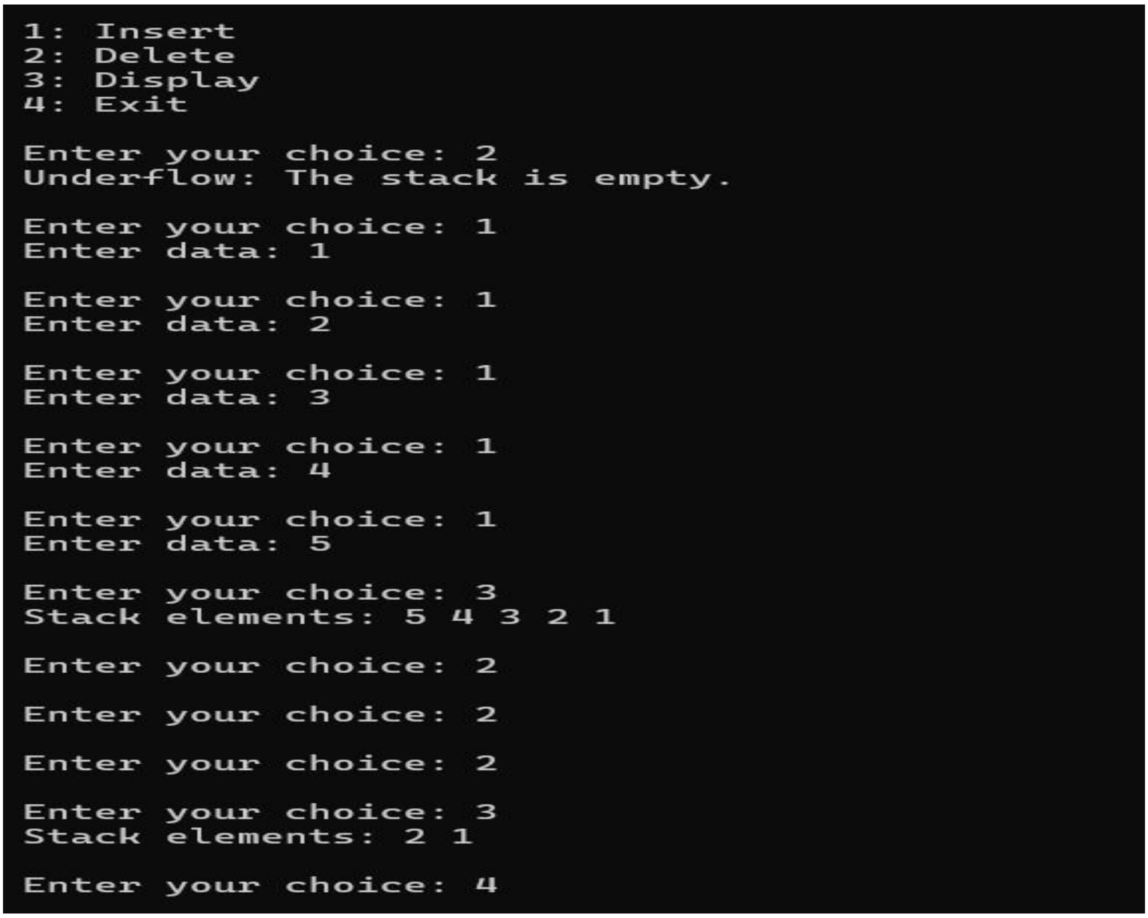
6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

Stacks using Linked list

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

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

struct node *top = NULL;
struct node* push(struct node*, int); struct
node* delete(struct node*);
void display(struct node*);

int main() { int
   choice;
   printf("\n1: Insert\n2: Delete\n3: Display\n4: Exit\n");
   int flag = 1; while (flag) { printf("\nEnter your choice:
   "); scanf("%d", &choice); switch (choice) { case 1: { int
   data; printf("Enter data: "); scanf("%d", &data); top =
   push(top, data); break;
         }
         case     2:     top     =
            delete(top);
            break;
         case 3:
            display(top);
            break;
         case      4:
            flag = 0;
            break;
         default:    printf("Invalid    choice.    Try
            again.\n");
      }
   }
   return 0;
}

struct node* push(struct node* top, int data) {
   struct node* new_node = (struct node*)malloc(sizeof(struct node));
   if (!new_node) { printf("Memory allocation failed!\n"); return top;
   }
```

```
    new_node->data = data;
    new_node->next  =  top;
    top = new_node;
    return top;
}

struct node* delete(struct node* top) { if (top ==
    NULL)  {  printf("Underflow:  The  stack  is
    empty.\n"); return top;
    }
    struct node* temp = top;
    top        =        top->next;
    free(temp); return top;

}

void display(struct node* top) {
    if (top == NULL) {
        printf("The stack is empty.\n"); return;
    }
    struct  node*  ptr  =  top;
    printf("Stack elements: ");
    while  (ptr  !=  NULL)  {
    printf("%d  ",  ptr->data);
    ptr = ptr->next;
    }
    printf("\n");
}
```

Output



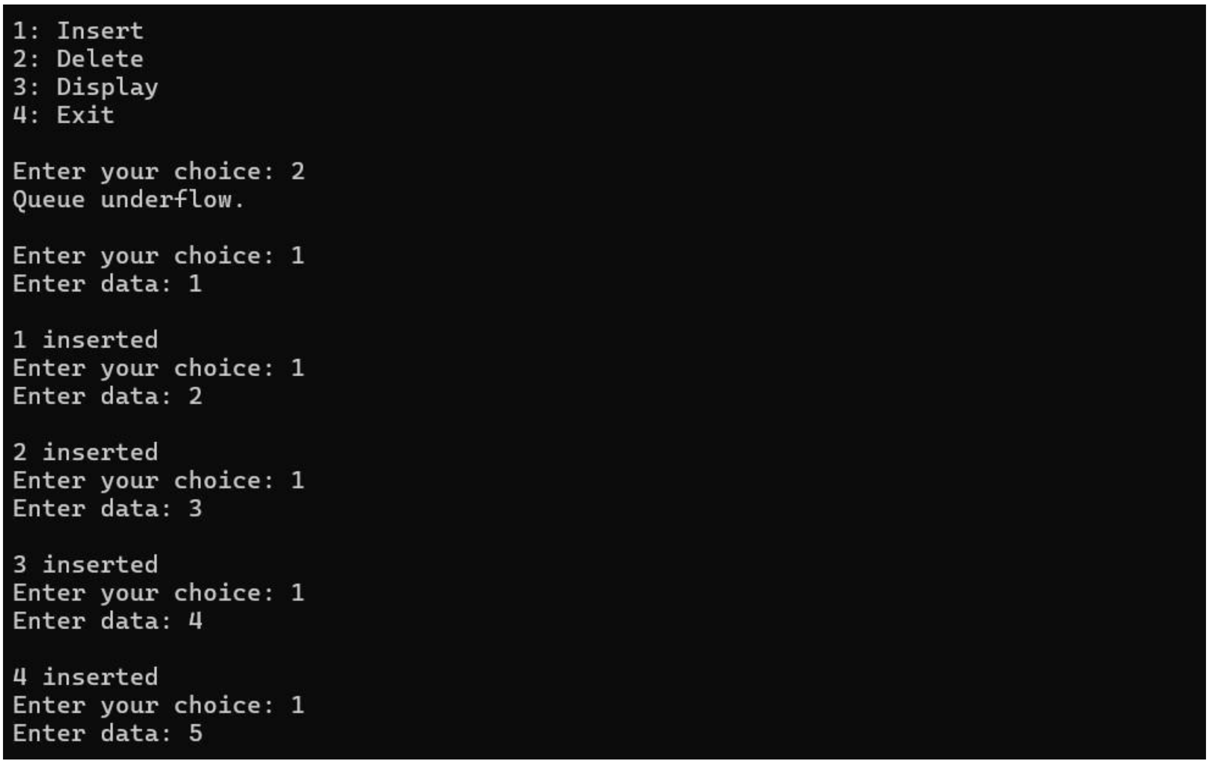
```
1: Insert
2: Delete
3: Display
4: Exit

Enter your choice: 2
Underflow: The stack is empty.

Enter your choice: 1
Enter data: 1

Enter your choice: 1
Enter data: 2

Enter your choice: 1
Enter data: 3

Enter your choice: 1
Enter data: 4

Enter your choice: 1
Enter data: 5

Enter your choice: 3
Stack elements: 5 4 3 2 1

Enter your choice: 2

Enter your choice: 2

Enter your choice: 2

Enter your choice: 3
Stack elements: 2 1

Enter your choice: 4
```

Queues using Linked list

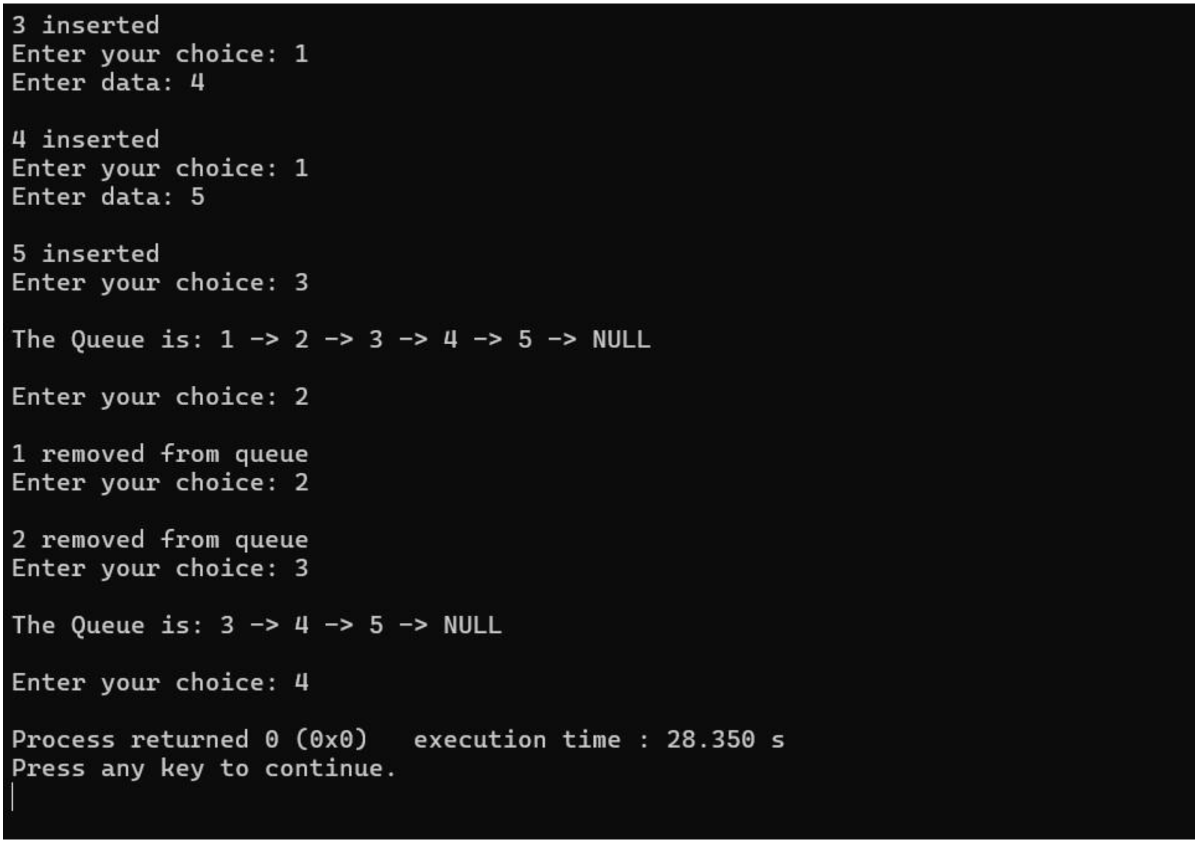
```
#include<stdio.h>

#include<stdlib.h>

struct Node

{

   int    data;    struct

   Node *Next;

};

struct Node * newNode(int data){

   struct Node* new;

   new= (struct Node *)malloc(sizeof(struct Node));

   new->data=data; new->Next=NULL; return new;
```

```
}
struct Node *enqueue(struct Node *front,struct Node **rear,int data){
    struct Node
    *newN=newNode(data);
    if(front==NULL){ front=newN;
        *rear=newN;
    }
    else{
        (*rear)->Next=newN;
        *rear=newN;
    }
    printf("\n %d inserted",data); return
    front;
}
struct Node *dequeue(struct Node *front){
if(front==NULL){           printf("underflow");
return NULL;
}
struct Node *ptr=front; front=front->Next;
printf("\n %d removed from queue",ptr->data);
free(ptr); return front;
}
void display(struct Node *front){
    struct Node *ptr; ptr=front;
    printf("\n The Queue is: ");
    while(ptr!=NULL){
    printf("%d  ->  ",ptr->data);
    ptr=ptr->Next;
    }
    printf("NULL");
}
int main() {
    int choice;
    printf("\n1: Insert\n2: Delete\n3: Display\n4: Exit\n");
    int flag = 1; while (flag) {
        printf("\nEnter your choice: ");
        scanf("%d",  &choice);  switch
        (choice) {
            case 1: {
                int data; printf("Enter
                data:  ");  scanf("%d",
                &data);      top      =
                push(top,        data);
                break;
            }
            case    2:    top    =
              delete(top);
              break;
            case 3:
            display(top);
            break; case 4:
              flag = 0; break;
            default:
              printf("Invalid choice. Try again.\n");
          }
        }
        return 0;
      }
```

Output



```
1: Insert
2: Delete
3: Display
4: Exit

Enter your choice: 2
Queue underflow.

Enter your choice: 1
Enter data: 1

1 inserted
Enter your choice: 1
Enter data: 2

2 inserted
Enter your choice: 1
Enter data: 3

3 inserted
Enter your choice: 1
Enter data: 4

4 inserted
Enter your choice: 1
Enter data: 5
```



```
3 inserted
Enter your choice: 1
Enter data: 4

4 inserted
Enter your choice: 1
Enter data: 5

5 inserted
Enter your choice: 3

The Queue is: 1 -> 2 -> 3 -> 4 -> 5 -> NULL

Enter your choice: 2

1 removed from queue
Enter your choice: 2

2 removed from queue
Enter your choice: 3

The Queue is: 3 -> 4 -> 5 -> NULL

Enter your choice: 4

Process returned 0 (0x0)   execution time : 28.350 s
Press any key to continue.
```

Lab-7

WAP to Implement doubly link list with primitive operations a) Create a doubly linked list.

b) Insert a new node to the left of the node.

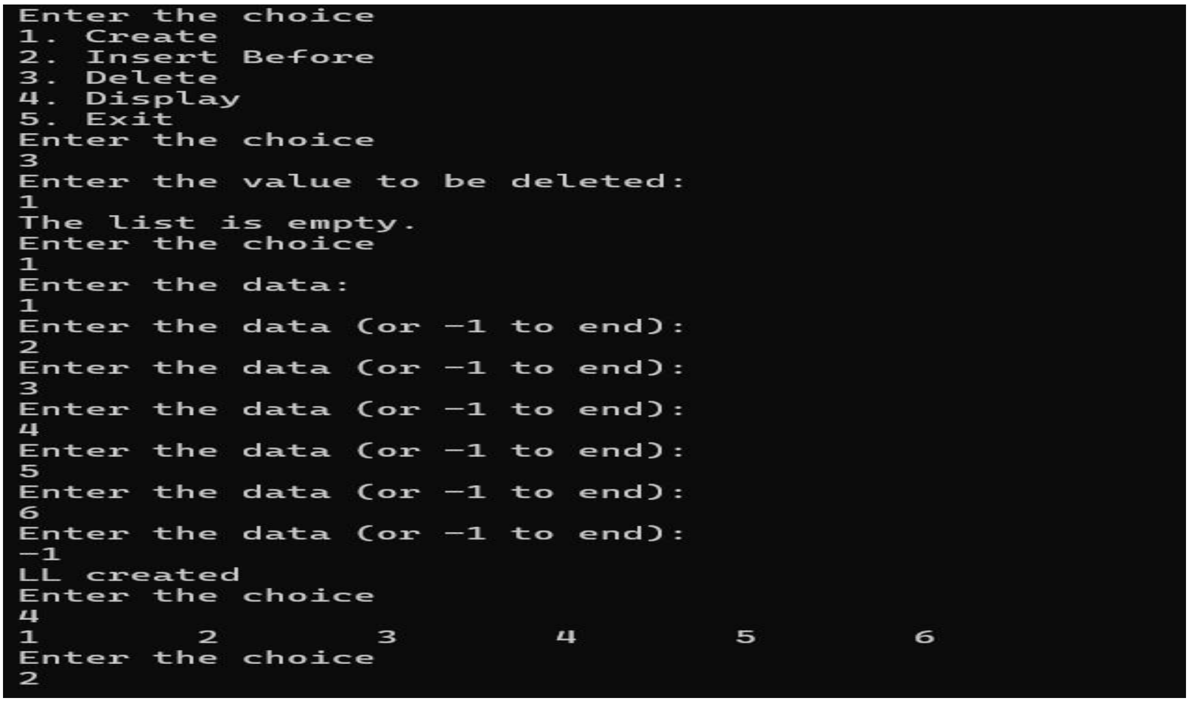
c) Delete the node based on a specific value

d) Display the contents of the list

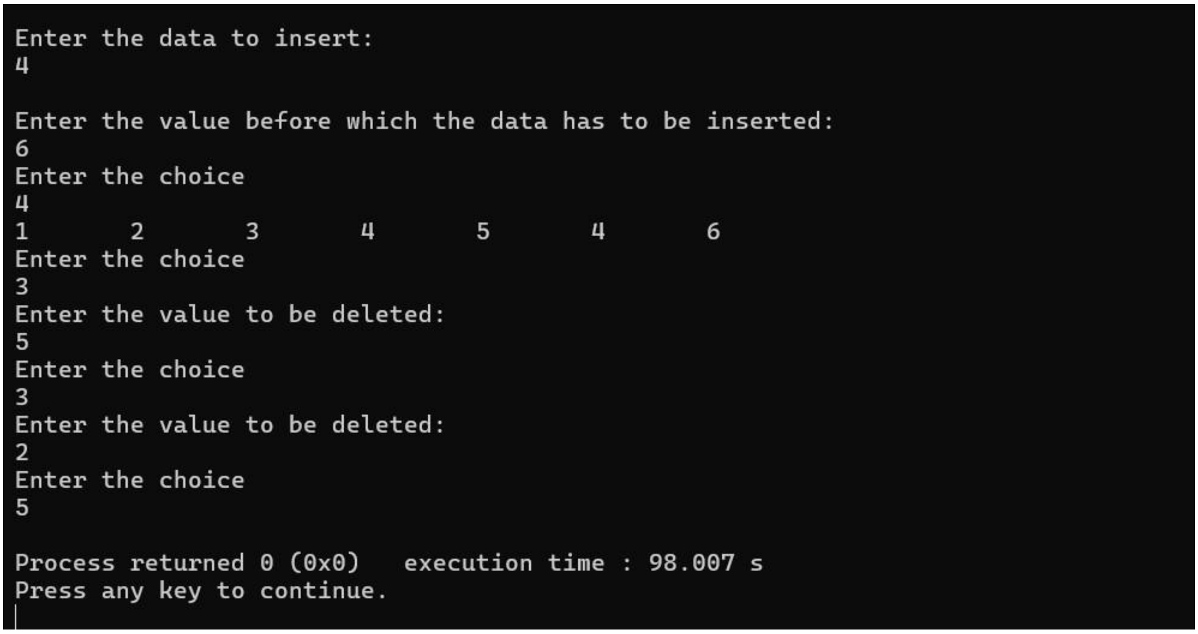
```
#include <stdio.h> #include
<stdlib.h>

struct node {
   struct node *prev;
   struct node *next;
   int data;
};
struct node *start = NULL;
struct node *create_ll(struct node *start) {
   struct node *new_node, *ptr;
   int   num;   printf("Enter   the
   data:\n");            scanf("%d",
   &num);

   while (num != -1) {
      new_node = (struct node*)malloc(sizeof(struct node));
      new_node->data = num;
      new_node->next = NULL;

      if    (start    ==    NULL)    {
         new_node->prev = NULL;
         start = new_node;
      } else { ptr =
         start;
         while (ptr->next != NULL) {
                ptr = ptr->next;
            }
            ptr->next = new_node; new_node->prev
            = ptr;
        }

        printf("Enter the data (or -1 to end):\n");
        scanf("%d", &num);
    }
    return start;
};
struct node *insert_before(struct node *start) {
    struct node *ptr, *newnode;
    int num, val;
    printf("\nEnter the data to insert:\n");
    scanf("%d", &num);
    printf("\nEnter the value before which the data has to be
inserted:\n");
    scanf("%d", &val);

    newnode = (struct node *)malloc(sizeof(struct node)); newnode-
    >data = num;

    ptr = start;
    while (ptr != NULL && ptr->data != val) {
        ptr = ptr->next;
    }

    if (ptr == NULL) { printf("Value %d not found in
        the list.\n", val); return start;
    }
    newnode->next      =      ptr;

    newnode->prev = ptr->prev;

    if (ptr->prev != NULL) { ptr-

    >prev->next = newnode;

    } else {
        start = newnode;
    }


    ptr->prev = newnode; return
    start;
};
void display(struct node *start) {
    struct node *ptr = start; if
    (ptr == NULL) {
        printf("The list is empty.\n"); return;
    }
    while    (ptr    !=    NULL)    {
        printf("%d\t", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
struct node *delete_node(struct node *start) {
    struct node *ptr, *temp; int
    val;
    printf("Enter the value to be deleted:\n"); scanf("%d",
    &val);

    if (start == NULL) {
        printf("The list is empty.\n"); return
        start;
    }

    ptr = start; while (ptr != NULL && ptr-
    >data != val) {
        ptr = ptr->next;
    }
    if (ptr == NULL) { printf("Value %d not found in
        the list.\n", val); return start;
    }

    if (ptr->prev != NULL) {
        ptr->prev->next = ptr->next;
    } else { start = ptr-
        >next;
    }

    if (ptr->next != NULL) {
        ptr->next->prev = ptr->prev;
    }

    temp = ptr;
    free(temp);
    return start;
};
int main() {
    int ch; printf("Enter the
    choice\n");          printf("1.
    Create\n"); printf("2. Insert
    Before\n");          printf("3.
    Delete\n");          printf("4.
    Display\n");         printf("5.
    Exit\n");
```

```
   while (1) {
      printf("Enter the choice\n");
      scanf("%d",    &ch);    switch
      (ch) { case 1:
         start = create_ll(start);
         printf("LL created\n"); break;
      case         2:         start         =
         insert_before(start); break;
      case         3:         start         =
         delete_node(start); break;
      case 4:
         display(start);
         break;
      case 5:
         exit(0);
      default:   printf("Invalid   choice!   Please   try
         again.\n");
      }
   }
   return 0;
}
```

Output:



```
Enter the choice
1. Create
2. Insert Before
3. Delete
4. Display
5. Exit
Enter the choice
3
Enter the value to be deleted:
1
The list is empty.
Enter the choice
1
Enter the data:
1
Enter the data (or -1 to end):
2
Enter the data (or -1 to end):
3
Enter the data (or -1 to end):
4
Enter the data (or -1 to end):
5
Enter the data (or -1 to end):
6
Enter the data (or -1 to end):
-1
LL created
Enter the choice
4
1       2       3       4       5       6
Enter the choice
2
```



```
Enter the data to insert:
4

Enter the value before which the data has to be inserted:
6
Enter the choice
4
1       2       3       4       5       4       6
Enter the choice
3
Enter the value to be deleted:
5
Enter the choice
3
Enter the value to be deleted:
2
Enter the choice
5

Process returned 0 (0x0)   execution time : 98.007 s
Press any key to continue.
```

Lab-8

Write a program

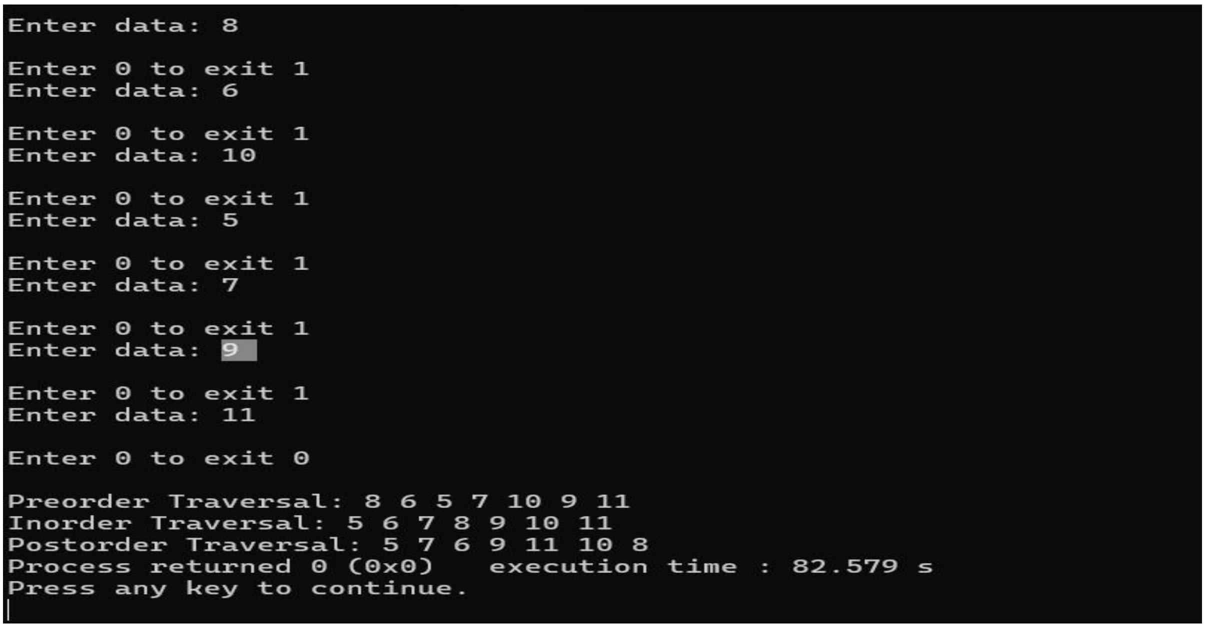
a) To construct a binary Search tree.

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

c) To display the elements in the tree.

```
#include <stdio.h>

#include <malloc.h>


typedef struct BST {

   int   data;   struct

   BST  *left;  struct

   BST *right;

} node;


node *create() {

   node *temp;

   printf("Enter data: "); temp = (node

   *)malloc(sizeof(node));   scanf("%d",

   &temp->data);  temp->left  =  temp-

   >right = NULL; return temp;

}

void insert(node *root, node *temp) {
    if (temp->data < root->data) { if
        (root->left != NULL)
            insert(root->left, temp);
        else
            root->left = temp;
    } else if (temp->data > root->data) { if
        (root->right != NULL)
            insert(root->right, temp);
        else
            root->right = temp;
    }
}

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

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

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

int main() { char ch; int n = 1;
    node *root = NULL, *temp;
    do {
        temp = create(); if
        (root == NULL)
            root = temp;
        else
            insert(root, temp);
            printf("\nEnter 0 to exit
            "); scanf("%d",&n);
    } while (n!=0);

    printf("\nPreorder  Traversal:  ");
    preorder(root);
    printf("\nInorder   Traversal:   ");
    inorder(root);
    printf("\nPostorder Traversal: ");
    postorder(root);

    return 0;
}
```

Output:



```
Enter data: 8

Enter 0 to exit 1
Enter data: 6

Enter 0 to exit 1
Enter data: 10

Enter 0 to exit 1
Enter data: 5

Enter 0 to exit 1
Enter data: 7

Enter 0 to exit 1
Enter data: 9

Enter 0 to exit 1
Enter data: 11

Enter 0 to exit 0

Preorder Traversal: 8 6 5 7 10 9 11
Inorder Traversal: 5 6 7 8 9 10 11
Postorder Traversal: 5 7 6 9 11 10 8
Process returned 0 (0x0)   execution time : 82.579 s
Press any key to continue.
```

Lab-9

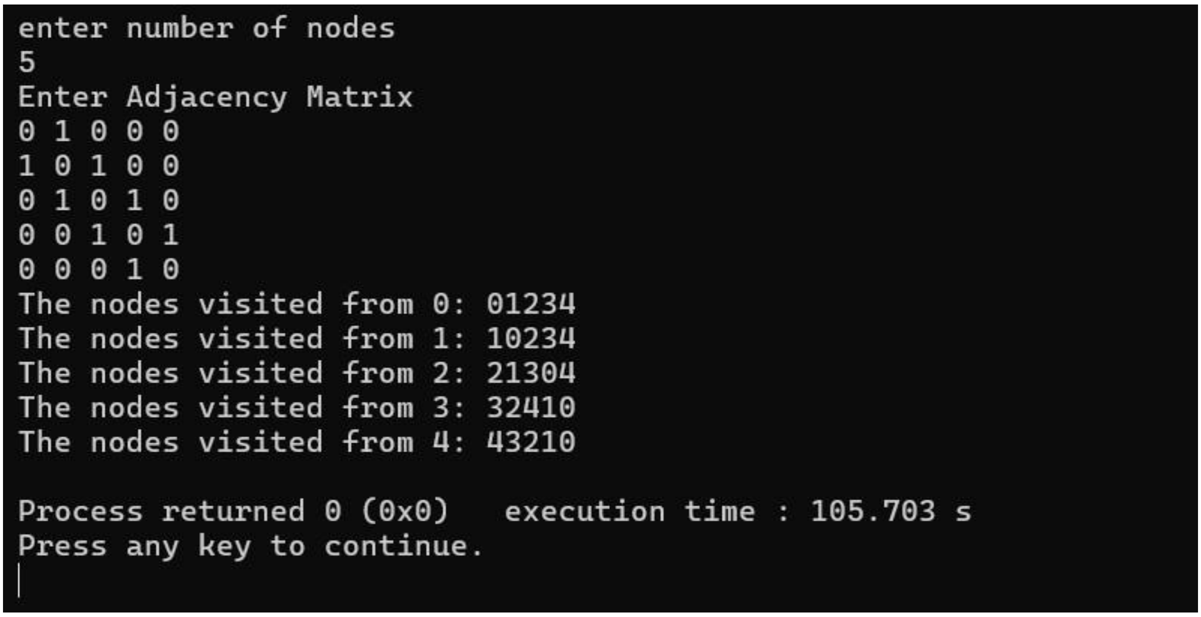
9a) Write a program to traverse a graph using BFS method 9b) Write a program to check whether given graph is connected or not using DFS method.

BFS-Method

```
#include <stdio.h>

void bfs(int adj[10][10], int n, int source){
   int  que[10];  int  front=0,rear=-1;  int
   visited[10]={0}; int node;
   printf("The nodes visited from %d: ", source);
   que[++rear]=source;
   visited[source]=1;
   printf("%d",source);
   while(front<=rear){
      int  u=  que[front++];
      for(int v=0; v<n; v++){
      if(adj[u][v]==1){
      if(visited[v]==0){
      printf("%d",v);
      visited[v]=1;
               que[++rear]=v;
            }
         }
      }
   }
   printf("\n");
}

int main() { int n;
   int adj[10][10];
   int source;
```

```
    printf("enter number of nodes \n");
    scanf("%d",&n);          printf("Enter
    Adjacency Matrix \n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d",&adj[i][j]);
        }
    }
    for(source=0; source<n; source++){
        bfs(adj,n,source);
    }

    return 0;
}
```

Output:



```
enter number of nodes
5
Enter Adjacency Matrix
0 1 0 0 0
1 0 1 0 0
0 1 0 1 0
0 0 1 0 1
0 0 0 1 0
The nodes visited from 0: 01234
The nodes visited from 1: 10234
The nodes visited from 2: 21304
The nodes visited from 3: 32410
The nodes visited from 4: 43210

Process returned 0 (0x0)   execution time : 105.703 s
Press any key to continue.
```

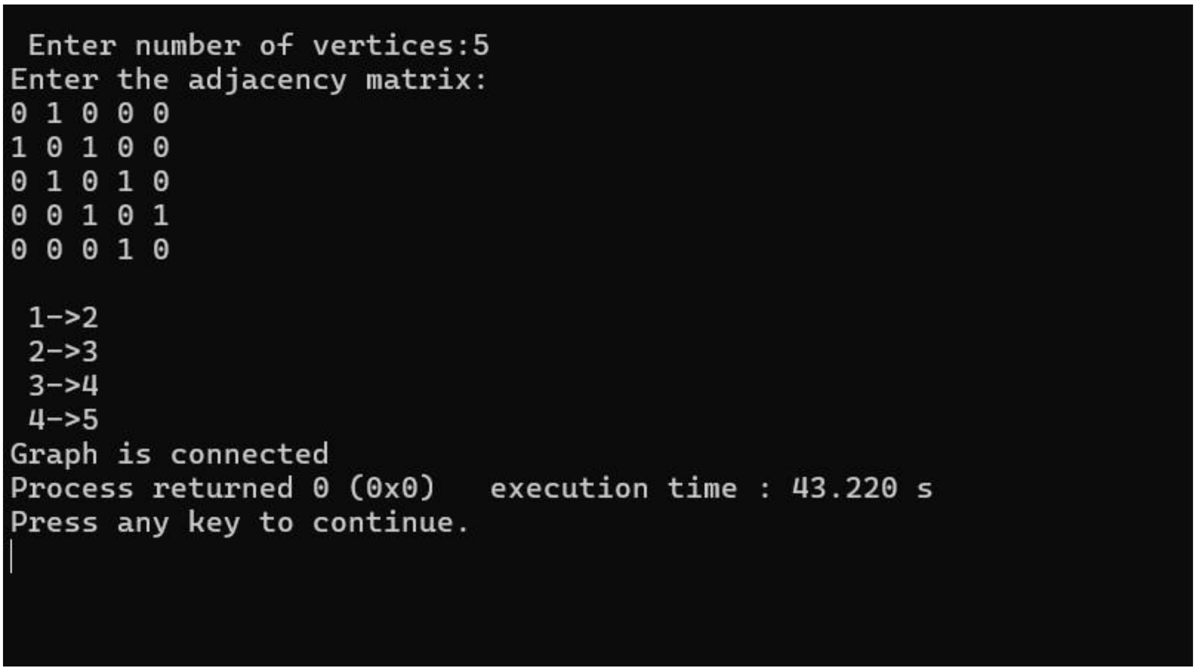
DFS – Method

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

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

int a[20][20], s[20], n;

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

```
s[i]=0; for(j=1;
j<=n; j++)
a[i][j]=0;
}
printf("Enter the adjacency matrix:\n");
for(i=1; i<=n; i++) for(j=1; j<=n; j++)
scanf("%d", &a[i][j]); dfs(1);
printf("\n"); for(i=1; i<=n; i++)
{ if(s[i])
count++;
}
if(count==n) printf("Graph is
connected"); else
printf("Graph is not connected"); return
0;
}
```

Output:



```
 Enter number of vertices:5
Enter the adjacency matrix:
0 1 0 0 0
1 0 1 0 0
0 1 0 1 0
0 0 1 0 1
0 0 0 1 0

 1->2
 2->3
 3->4
 4->5
Graph is connected
Process returned 0 (0x0)   execution time : 43.220 s
Press any key to continue.
```

Lab-10

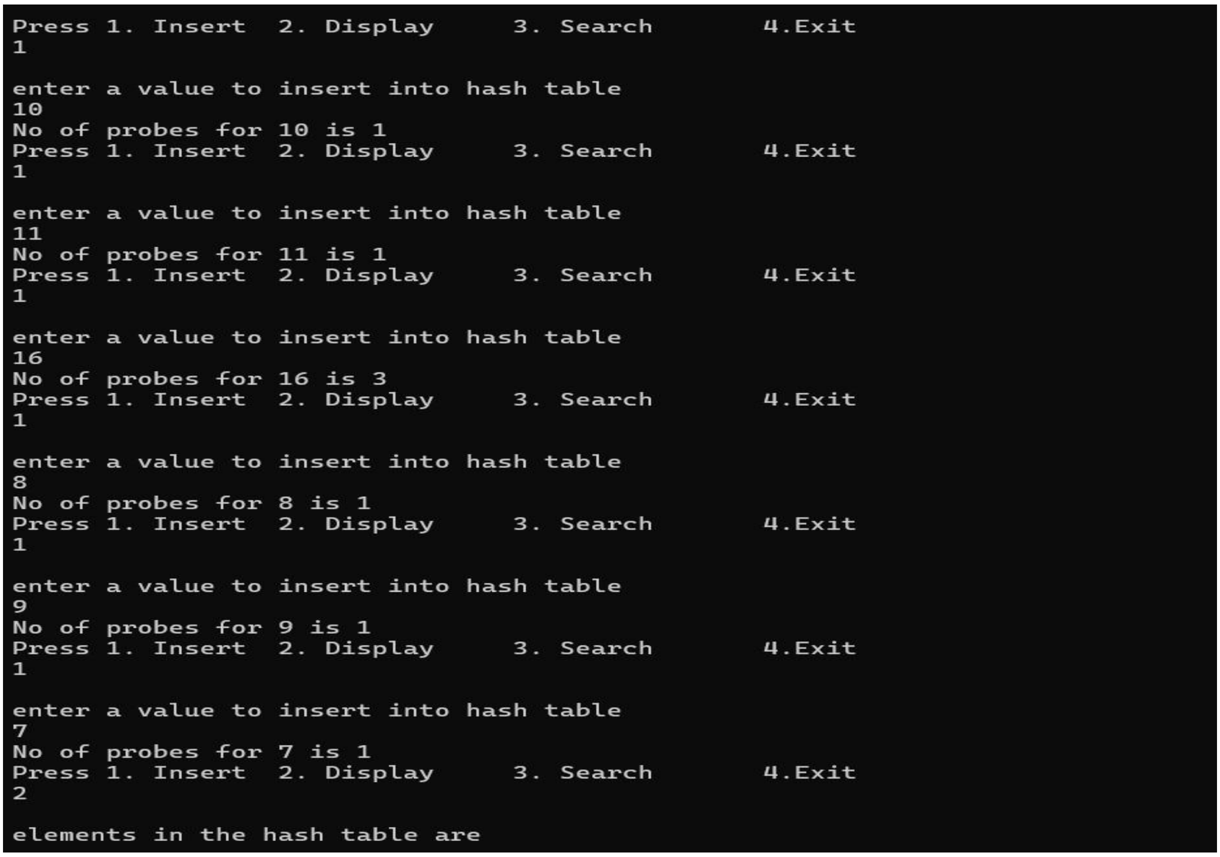
->Write a program to demonstrate linear Probing

```
#include <stdio.h>
#include<stdlib.h> #define
TABLE_SIZE        6        int
h[TABLE_SIZE]={NULL};
void insert()
{
    int key,index,i,flag=0,hkey;
    printf("\nenter a value to insert into hash
    table\n"); scanf("%d",&key);
    hkey=key%TABLE_SIZE; for(i=0;i<TABLE_SIZE;i++)
    {
     index=(hkey+i)%TABLE_SIZE;
        if(h[index] == NULL)
        {
        h[index]=key;
         break;
        }
    }
    printf("No of probes for %d is %d", key,i+1); if(i
    == TABLE_SIZE)
    printf("\nelement cannot be inserted\n");
}
```

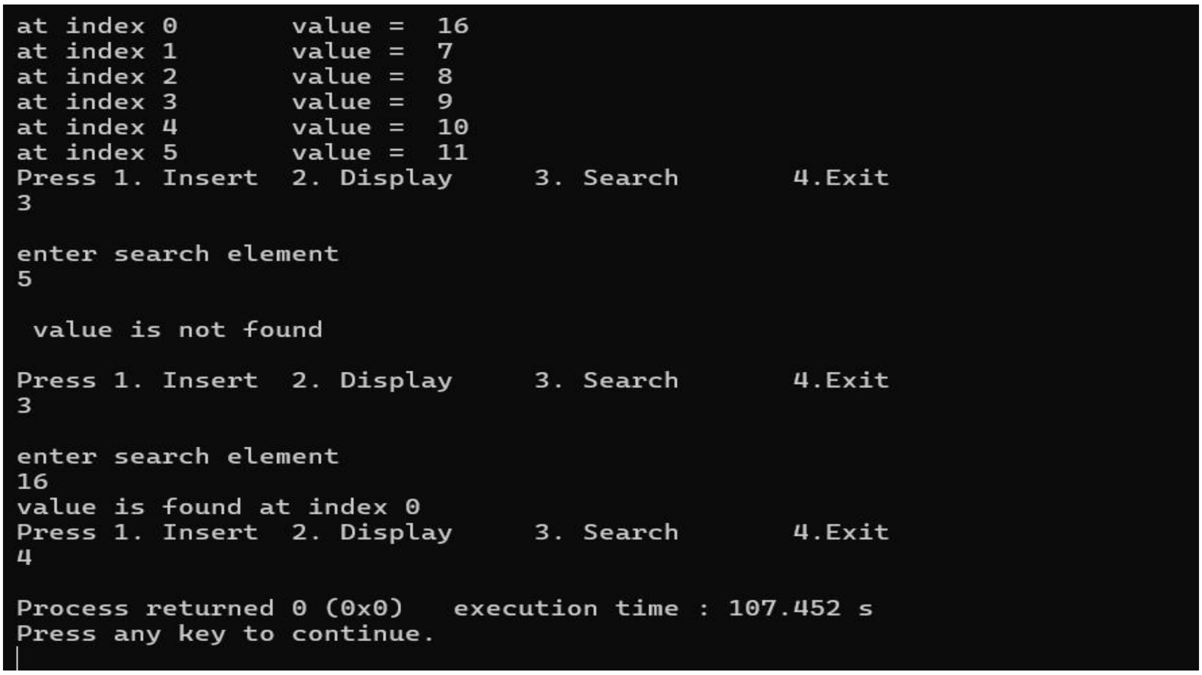
```
void search()
{
int              key,index,i,flag=0,hkey;
printf("\nenter search element\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
   for(i=0;i<TABLE_SIZE; i++)
   {
    index=(hkey+i)%TABLE_SIZE;
      if(h[index]==key)
      {
         printf("value is found at index %d",index); break;
      }
   }
   if(i   ==   TABLE_SIZE)   printf("\n
   value is not found\n");
}
void display()
{ int i;
printf("\nelements in the hash table are \n"); for(i=0;i< TABLE_SIZE;
i++)
printf("\nat index %d \t value = %d",i,h[i]);
}
main()
```

```
{  int opt,i; while(1)
{   printf("\nPress  1.  Insert\t  2.  Display  \t3.  Search  \t4.Exit  \n");
    scanf("%d",&opt);
   switch(opt)
   {
    case 1:insert(); break;
    case 2:display(); break;
   case 3:search(); break;
   case 4:exit(0);
   }
}
}
```

Output:



```
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
10
No of probes for 10 is 1
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
11
No of probes for 11 is 1
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
16
No of probes for 16 is 3
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
8
No of probes for 8 is 1
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
9
No of probes for 9 is 1
Press 1. Insert  2. Display     3. Search       4.Exit
1

enter a value to insert into hash table
7
No of probes for 7 is 1
Press 1. Insert  2. Display     3. Search       4.Exit
2

elements in the hash table are
```



```
at index 0       value =  16
at index 1       value =  7
at index 2       value =  8
at index 3       value =  9
at index 4       value =  10
at index 5       value =  11
Press 1. Insert  2. Display     3. Search       4.Exit
3

enter search element
5

 value is not found

Press 1. Insert  2. Display     3. Search       4.Exit
3

enter search element
16
value is found at index 0
Press 1. Insert  2. Display     3. Search       4.Exit
4

Process returned 0 (0x0)   execution time : 107.452 s
Press any key to continue.
```

All Leetcode Problems

->[Remove Outermost Parentheses]

A valid parentheses string is either empty "", "(" + A + ")", or A + B, where A and B are valid parentheses strings, and + represents string concatenation.

Solution:

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

char* removeOuterParentheses(char* s) {
    int balance = 0;
    int length = strlen(s);
    char* result = (char*)malloc(length + 1);
    int index = 0;

    for (int i = 0; i < length; i++) {
        if (s[i] == '(') {
            if (balance > 0) {
                result[index++] = s[i];
            }
            balance++;
        } else {
            balance--;
            if (balance > 0) {
                result[index++] = s[i];
            }
        }
    }
    result[index] = '\0';
    return result;
}

int main() {
    char s[] = "(()())(())";
    char* result = removeOuterParentheses(s);
    printf("%s\n", result);
    free(result);
    return 0;
}
```

->Number of Students Unable to Eat Lunch

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches. The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a ***stack****. At each step:*

- *If the student at the front of the queue* ***prefers*** *the sandwich on the top of the stack, they will* ***take it*** *and leave the queue.*
- *Otherwise, they will* ***leave it*** *and go to the queue's end.*

This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

Solution: #include <stdio.h>

```
int countStudents(int* students, int studentsSize, int* sandwiches, int sandwichesSize) {
    int count[2] = {0}; // Count of students preferring circular (0) and square (1) sandwiches
```

```
  for (int i = 0; i < studentsSize; i++) {
    count[students[i]]++;
  }

  for (int i = 0; i < sandwichesSize; i++) {
    if (count[sandwiches[i]] > 0) {
      count[sandwiches[i]]--;
    } else {
      return sandwichesSize - i;
    }
  }
  return 0;
}

int main() {
  int students[] = {1, 1, 0, 0};
  int sandwiches[] = {0, 1, 0, 1};
  int studentsSize = sizeof(students) / sizeof(students[0]);
  int sandwichesSize = sizeof(sandwiches) / sizeof(sandwiches[0]);

  int result = countStudents(students, studentsSize, sandwiches,
sandwichesSize);
  printf("%d\n", result);
  return 0;
}
```

->Reveal Cards In Increasing Order

You are given an integer array deck. There is a deck of cards where every card has a unique integer. The integer on the i^{th} card is deck[i]. You can order the deck in any order you want. Initially, all the cards start face down (unrevealed) in one deck.

You will do the following steps repeatedly until all cards are revealed:

1. *Take the top card of the deck, reveal it, and take it out of the deck.*
2. *If there are still cards in the deck then put the next top card of the deck at the bottom of the deck.*
3. *If there are still unrevealed cards, go back to step 1. Otherwise, stop.*

Return an ordering of the deck that would reveal the cards in increasing order.

Solution:

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

int* deckRevealedIncreasing(int* deck, int deckSize, int* returnSize) {
    int* result = (int*)malloc(deckSize * sizeof(int));
    int* indexQueue = (int*)malloc(deckSize * sizeof(int));
    int front = 0, rear = 0;

    for (int i = 0; i < deckSize; i++) {
        indexQueue[rear++] = i;
    }

    for (int i = 0; i < deckSize; i++) {
        int smallestIndex = indexQueue[front++];
        result[smallestIndex] = deck[i];
        if (front < rear) {
            indexQueue[rear++] = indexQueue[front++];
        }
    }

    *returnSize = deckSize;
    free(indexQueue);
    return result;
}

int cmpfunc(const void* a, const void* b) {
   return (*(int*)a - *(int*)b);
}

int main() {
   int deck[] = {17, 13, 11, 2, 3, 5, 7};
   int deckSize = sizeof(deck) / sizeof(deck[0]);
   int returnSize;

   qsort(deck, deckSize, sizeof(int), cmpfunc);

   int* result = deckRevealedIncreasing(deck, deckSize, &returnSize);

   printf("[");
   for (int i = 0; i < returnSize; i++) {
      printf("%d", result[i]);
      if (i < returnSize - 1) printf(", ");
   }
   printf("]\n");

   free(result);
   return 0;
}
```