VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ROSHNI P(1BM22CS223)

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 ROSHNI P**(1BM22CS223)**, 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. Sneha S Bagalkot Dr. Jyothi S Nayak Assistant Professor Professor and Head Department of CSE Department of CSE BMSCE, Bengaluru BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.

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.

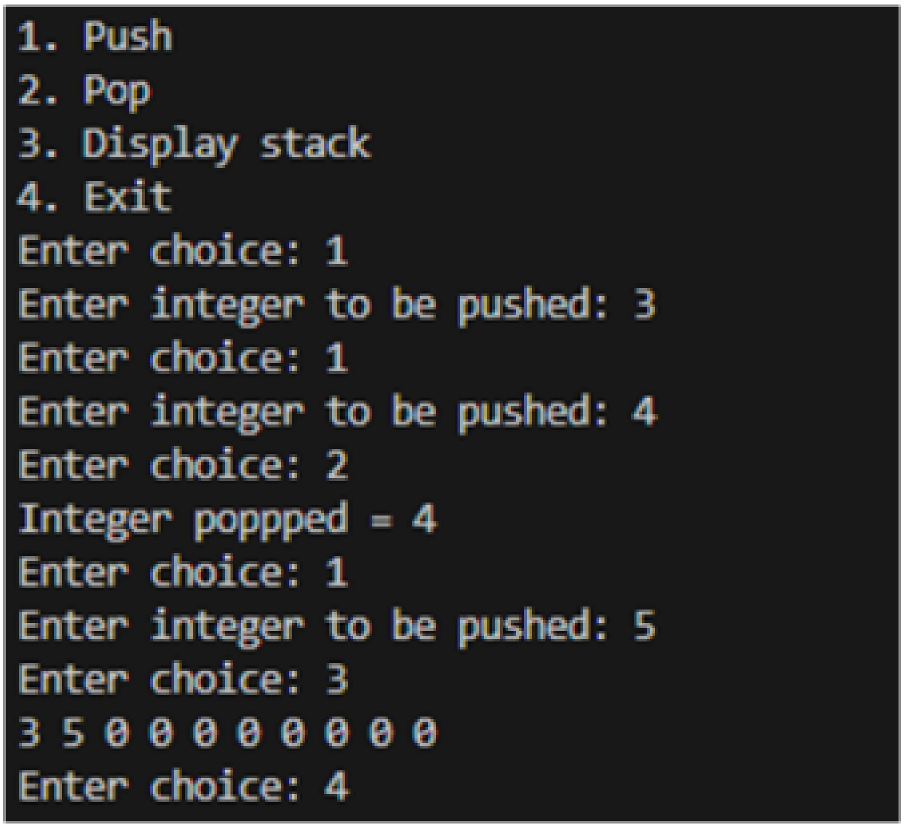
```
#include <stdio.h>
#include<stdlib.h> #include <string.h>

#definesize10 int pos = -1;
```

```
int stack[size];

voidpush(inta); int pop();
void display();

int main(){
printf("1.Push\n2.Pop\n3.Displaystack\n4.Exit\nEnter choice: ");
intchoice; int a;
scanf("%d",&choice); while(choice != 4){
switch(choice){ case 1:
printf("Enterintegertobepushed:"); scanf("%d", &a);
push(a); break;
case 2:
a = pop();
printf("Integerpoppped=%d\n",a); break;
case 3:
display(); break;
default:
printf("Invalid input"); break;

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

}

void push(int a){ if (pos == 9){
printf("Stack Overflow condition"); return;
}
stack[++pos] = a;
}

int pop(){
if (pos == -1){
printf("Stack Underflow condition"); return (int) NULL;
}
return stack[pos--];
}

void display(){
```

```
for(inti=0;i<size;i++){ printf("%d ", stack[i]);
}
```

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

Output:



```
1. Push
2. Pop
3. Display stack
4. Exit
Enter choice: 1
Enter integer to be pushed: 3
Enter choice: 1
Enter integer to be pushed: 4
Enter choice: 2
Integer poppped = 4
Enter choice: 1
Enter integer to be pushed: 5
Enter choice: 3
3 5 0 0 0 0 0 0 0 0
Enter choice: 4
```

LAB PROGRAM 2:

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

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

#define size 20

void push(chara);
char pop();

void display();
```

```
char*postfix(char*exp);
```

```
char* prefix(char* exp);
bool character(char c);
boollower_precedence(charop1,charop2);
 bool isEmpty();
int pos = -1;
char stack[size]; int n;

int main(){


        printf("Entersizeofexpressionintermsofcharacters:"); scanf("%d", &n);

        fflush(stdin);

        char*expr=(char*)malloc(size*sizeof(char)); printf("Enter infix expression: ");

        scanf("%[^\n]s", expr);

        char* postfixexp = postfix(expr);
        printf("Postfixexpression:%s\n",postfixexp); char* prefixexp =
        prefix(expr);

        printf("Prefix expression: %s", prefixexp);



        return 0;

}


bool isEmpty(){

        return pos == -1;
}


void push(char a){

        if (pos == size-1){
                printf("StackOverflowcondition");
                return;

        }

        stack[++pos] = a;
}
```

```
char pop(){

        if (pos == -1){
                printf("StackUnderflowcondition");
                 return (char) NULL;

        }



        charreturn_value=stack[pos]; stack[pos] =
        (char) NULL;

        pos--;

        return return_value;
}


void display(){

        printf("Stack: ");

                for(inti=0;i<size;i++){ printf("%c ",
                        stack[i]);

        }

        printf("\n");
}


bool character(char c){

        return (c >= 'a'&& c <= 'z') || (c >= 'A'&& c <= 'Z') || (c
>= '0'&& c <= '9');
}


bool lower_precedence(char op1, char op2){
        if (op1 == op2 && op2 == '^') return false;
        charop_order[]={'^','/','*','+','-'}; int o1, o2;

        for(int i = 0; i <5; i++){

                if(op_order[i]==op1)o1=i; if
                (op_order[i] == op2) o2 = i;
        }


        return o1 <= o2;


  }


  char* postfix(char* exp){

        char*return_exp=(char*)malloc((size+3)*sizeof(char)); int current =

        0; for(int i = 0; i < n; i++){


              if (character(exp[i])){

                    return_exp[current++] = exp[i];
              }
              else if(exp[i] == '+' || exp[i] == '-' || exp[i] == '*'||
  exp[i] == '/' || exp[i] == '^'){
                    if (isEmpty()) push(exp[i]);


                    else if(lower_precedence(stack[pos], exp[i])){

                                while(lower_precedence(stack[pos], exp[i]) &&

  !isEmpty()){ pop();



  if(stack[pos]!='(')return_exp[current++]= else{




                                 pos--; break;

                              }

                         }
                          push(exp[i]);
```

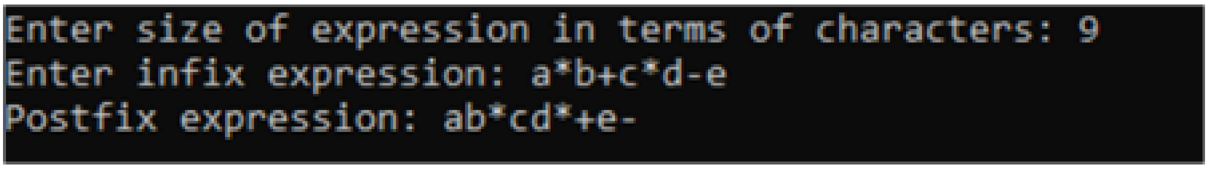
```
            }
```

```
            else push(exp[i]);
        }
        else if(exp[i] == '('){
            push('(');
        }
        else if(exp[i] == ')'){
                while(stack[pos]!='('&&!isEmpty()){ return_exp[current++] = pop();

            }

        }




    }


    while(!isEmpty()){
        if(stack[pos]!='(')return_exp[current++]=pop(); else pos--;

    }


    return return_exp;


}


char* prefix(char* exp){
    char*buffer=malloc(size*sizeof(char)); for(int i =
    n-1; i >= 0; i--){

        buffer[n-i-1] = exp[i];

        if (buffer[n-i-1] == '(') buffer[n-i-1] = ')';
        else if (buffer[n-i-1] == ')') buffer[n-i-1] = '(';
    }


    printf("Reversed String: %s\n", buffer);
    char*return_exp=malloc(size*sizeof(char)); int current =
    0;
```

```
    for(int i = 0; i < n; i++){


        if (character(buffer[i])){
            return_exp[current++] = buffer[i];
        }
        else if(buffer[i] == '+' || buffer[i] == '-' || buffer[i]
== '*' || buffer[i] == '/' || buffer[i] == '^'){
            if (isEmpty()) push(buffer[i]);


            else if(lower_precedence(stack[pos], buffer[i])){

                while(lower_precedence(stack[pos], buffer[i]) &&

!isEmpty()){ pop();


if(stack[pos]!='(')return_exp[current++]= else{


    pos--; break;


                    }

                }
                push(buffer[i]);
            }
            else push(buffer[i]);
        }
        else if(buffer[i] == '('){
            push('(');
        }
        else if(buffer[i] == ')'){
                while(stack[pos]!='('&&!isEmpty()){ return_exp[current++] = pop();

            }

        }
        }
```

```
        while(!isEmpty()){

                if(stack[pos]!='(')return_exp[current++]=pop(); else pos--;

        }



        char*final=(char*)malloc(size*sizeof(char)); for(int i = 0; i <
        strlen(return_exp); i++)

                final[i] = return_exp[strlen(return_exp)-i-1];


        return final;

  }
```

OUTPUT:



```
Enter size of expression in terms of characters: 9
Enter infix expression: a*b+c*d-e
Postfix expression: ab*cd*+e-
```

LAB 3 PROGRAMS:

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

#define size 5

void push(int a);
int pop();
void display();
```

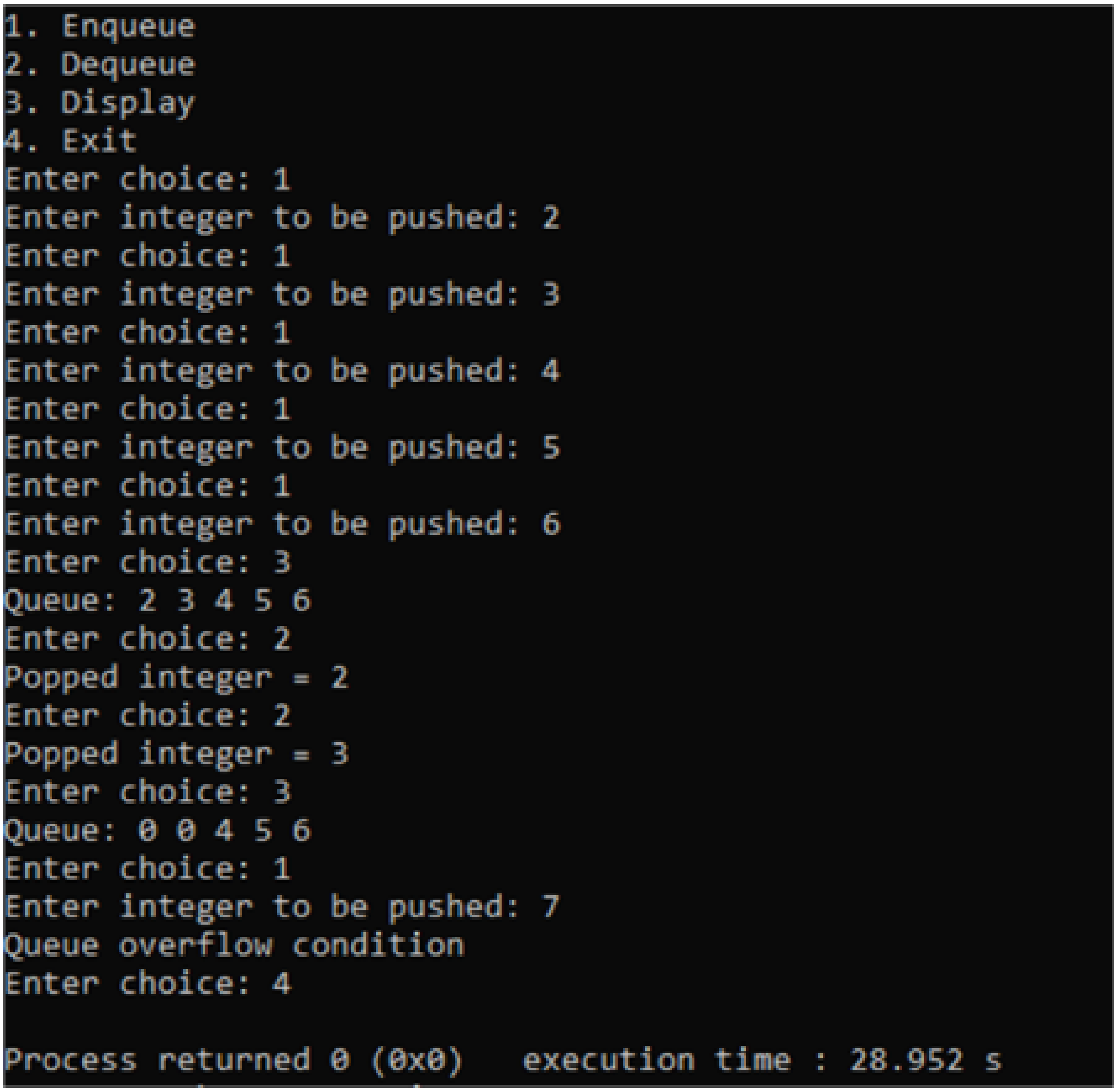
```
int fpos = -1, rpos = -1;
int queue[size];

int main(){
    int choice;
```

```
    printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter choice: ");
    scanf("%d", &choice);
    int a;
    while(choice != 4){
        switch(choice){
            case 1:
                printf("Enter integer to be pushed: ");
                scanf("%d", &a);
                push(a);
                break;
            case 2:
                a = pop();
                printf("Popped integer = %d\n", a);
                break;
            case 3:
                display();
                break;
            default:
                printf("Idk");
                break;
        }
        printf("Enter choice: ");
        scanf("%d", &choice);
    }

}

void push(int a){
    if (fpos == -1 && rpos == -1){
        queue[++rpos] = a;
        fpos++;
        return;
    }
    else if (rpos == size-1){
        printf("Queue overflow condition\n");
        return;
    }
    else{
        queue[++rpos] = a;
        return;
    }
}
```

```
int pop(){
    if (fpos == -1){
        printf("Queue Underflow condition\n");
    }
    int n = queue[fpos];
    queue[fpos] = (int) NULL;
    fpos++;
```

```
    return n;
}

void display(){
    printf("Queue: ");
    for(int i = 0; i < size; i++)
        printf("%d ", queue[i]);
    printf("\n");
}
```

OUTPUT:



```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter integer to be pushed: 2
Enter choice: 1
Enter integer to be pushed: 3
Enter choice: 1
Enter integer to be pushed: 4
Enter choice: 1
Enter integer to be pushed: 5
Enter choice: 1
Enter integer to be pushed: 6
Enter choice: 3
Queue: 2 3 4 5 6
Enter choice: 2
Popped integer = 2
Enter choice: 2
Popped integer = 3
Enter choice: 3
Queue: 0 0 4 5 6
Enter choice: 1
Enter integer to be pushed: 7
Queue overflow condition
Enter choice: 4

Process returned 0 (0x0)   execution time : 28.952 s
```

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

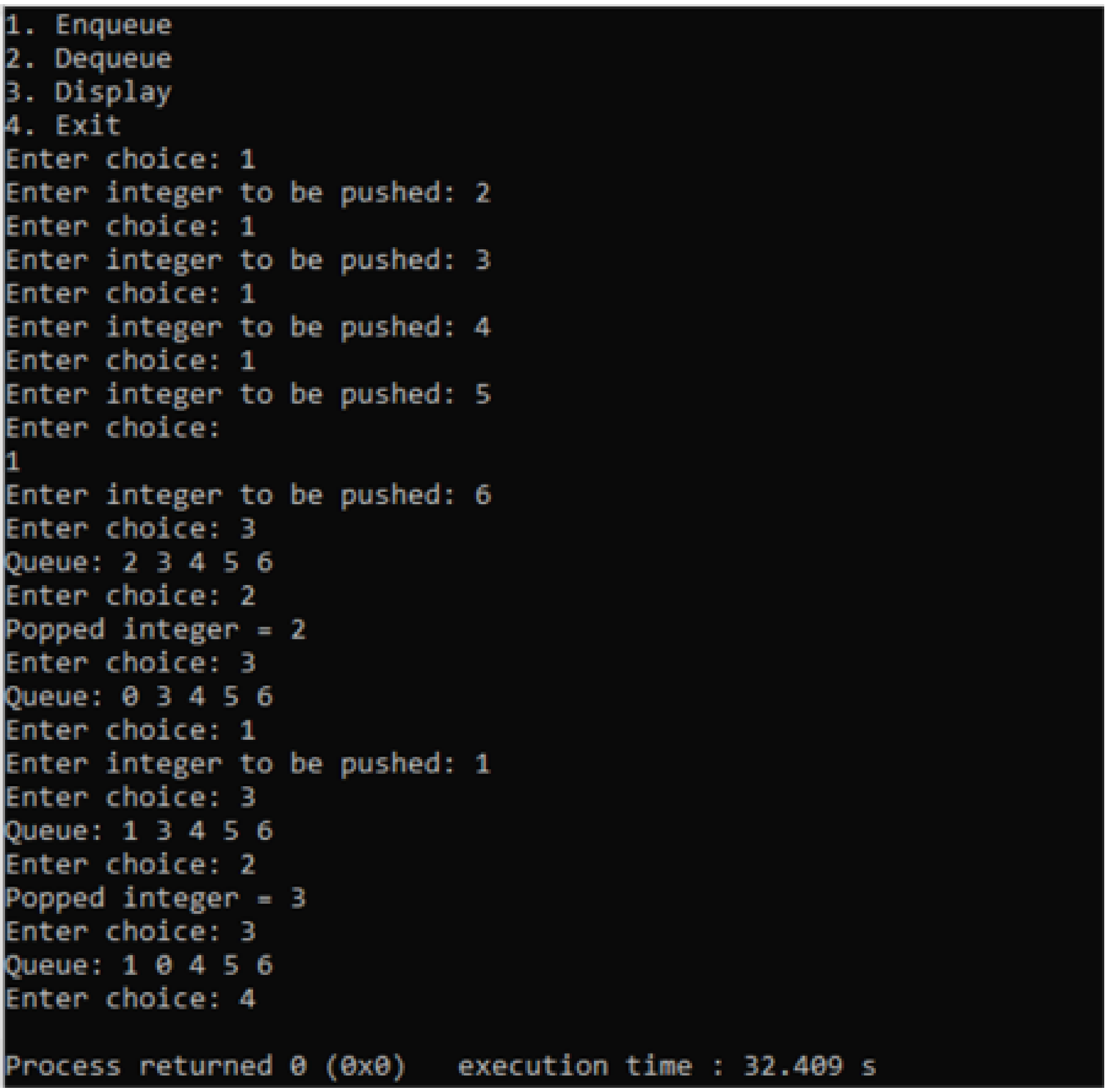
```
#include <stdio.h>
```

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

#define size 5

void push(int a);
int pop();
void display();

int fpos = -1, rpos = -1;
int queue[size];

int main(){
    int choice;
    printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter choice: ");
    scanf("%d", &choice);
    int a;
    while(choice != 4){
        switch(choice){
            case 1:
                printf("Enter integer to be pushed: ");
                scanf("%d", &a);
                push(a);
                // printf("fpos = %d; rpos = %d\n", fpos%size, rpos%size);
                break;
            case 2:
                a = pop();
                printf("Popped integer = %d\n", a);
                // printf("fpos = %d; rpos = %d\n", fpos%size, rpos%size);
                break;
            case 3:
                display();
                break;
            default:
                printf("Idk");
                break;
        }
        printf("Enter choice: ");
        scanf("%d", &choice);
    }

}

void push(int a){
    if (fpos == -1 && rpos == -1){
        queue[++rpos] = a;
        fpos++;
        return;
    }
```

```
    else if ((rpos+1)%size == (fpos%size)){
        printf("Queue overflow condition\n");
        return;
    }
    else{
        rpos++;
        queue[(rpos%size)] = a;
        return;
    }
```

```
}

int pop(){
   if (fpos == -1){
      printf("Queue Underflow condition\n");
   }
   int n = queue[fpos%size];
   queue[fpos%size] = (int) NULL;
   fpos++;
   return n;
}

void display(){
   printf("Queue: ");
   for(int i = 0; i < size; i++)
      printf("%d ", queue[i]);
   printf("\n");
}
```

OUTPUT:



```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter integer to be pushed: 2
Enter choice: 1
Enter integer to be pushed: 3
Enter choice: 1
Enter integer to be pushed: 4
Enter choice: 1
Enter integer to be pushed: 5
Enter choice:
1
Enter integer to be pushed: 6
Enter choice: 3
Queue: 2 3 4 5 6
Enter choice: 2
Popped integer = 2
Enter choice: 3
Queue: 0 3 4 5 6
Enter choice: 1
Enter integer to be pushed: 1
Enter choice: 3
Queue: 1 3 4 5 6
Enter choice: 2
Popped integer = 3
Enter choice: 3
Queue: 1 0 4 5 6
Enter choice: 4

Process returned 0 (0x0)   execution time : 32.409 s
```

LAB 4 PROGRAMS:
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.

```
#include <stdio.h>

#include <stdlib.h>



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


Node* head = NULL;


void push();
void append();
void insert();
void display();


void main() {
        int choice;
   while (1) {
      printf("1. Insert at beginning\n");
      printf("2. Insert at end\n");
      printf("3. Insert at position\n");
      printf("4. Display\n");
      printf("5. Exit\n");
      printf("Enter choice: ");
      scanf("%d", &choice);
         switch (choice) {
        case 1:
           push();
           break;
        case 2:
           append();
```

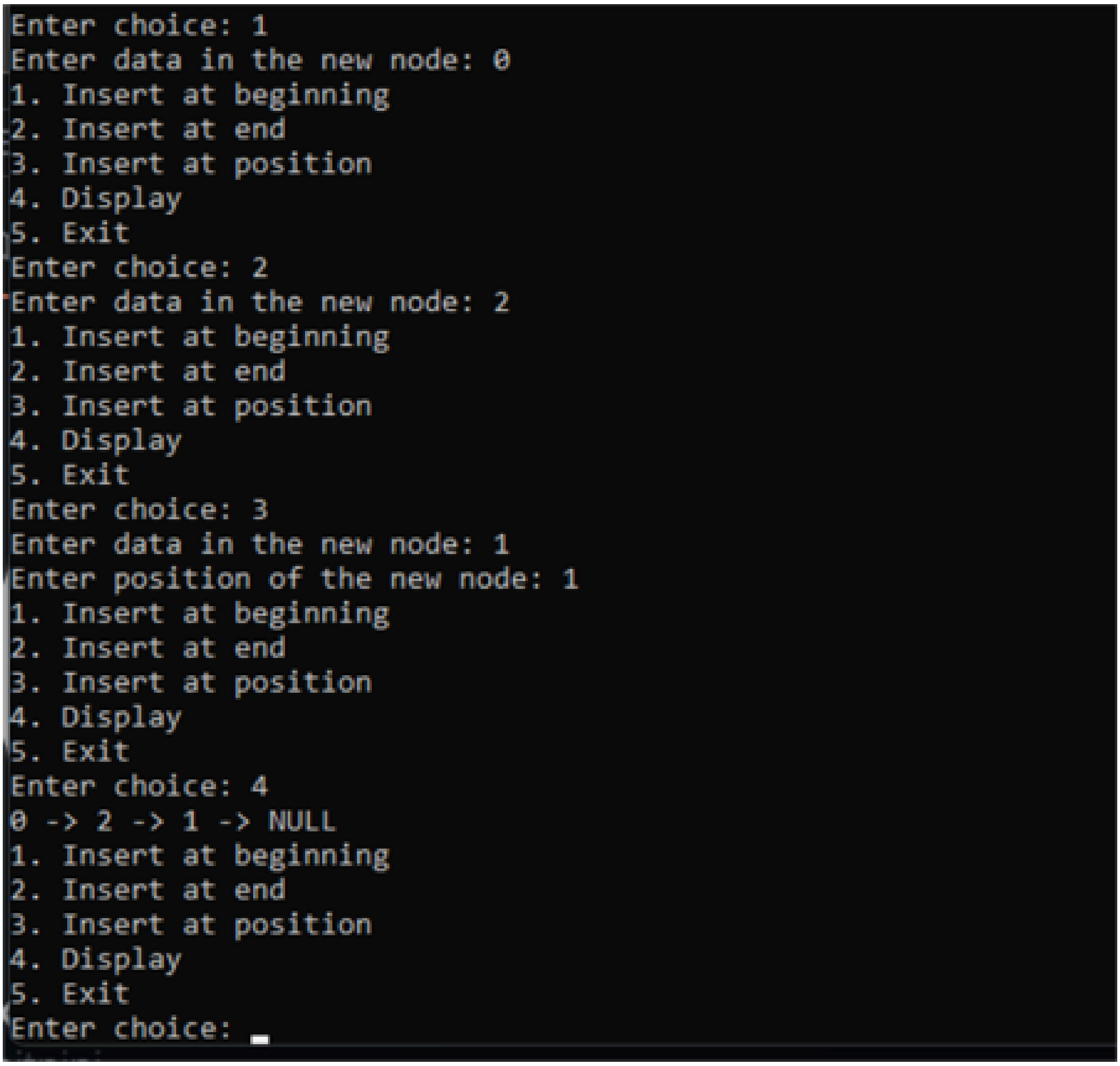
```
                break;

            case 3:

                insert();

                break;

            case 4:

                display();

                break;

            default:

                printf("Exiting the program");

        }

            }

}


void push() {

    Node* temp = (Node*)malloc(sizeof(Node));

            int new_data;

    printf("Enter data in the new node: ");

    scanf("%d", &new_data);

    temp->data = new_data;

    temp->next = head;

            head = temp;

}


void append() {

    Node* temp = (Node*)malloc(sizeof(Node));

            int new_data;
```

```
  printf("Enter data in the new node: ");
```

```
  scanf("%d", &new_data);
  temp->data = new_data;
  temp->next = NULL;
      if (head == NULL) {
    head = temp;
    return;
      }
  Node* temp1 = head;
  while (temp1->next != NULL) {
    temp1 = temp1->next;
      }
  temp1->next = temp;
}

void insert() {
  Node* temp = (Node*)malloc(sizeof(Node));
      int new_data, pos;
  printf("Enter data in the new node: ");
  scanf("%d", &new_data);
  printf("Enter position of the new node: ");
  scanf("%d", &pos);
  temp->data = new_data;
  temp->next = NULL;
      if (pos == 0) {
    temp->next = head;

    head = temp;

    return;
```

```
            }

    Node* temp1 = head;

    while (pos--) {

        temp1 = temp1->next;

            }

    Node* temp2 = temp1->next;

    temp->next = temp2;

    temp1->next = temp;

}


void display() {

    Node* temp1 = head;

    while (temp1 != NULL) {

        printf("%d -> ", temp1->data);

        temp1 = temp1->next;

            }

    printf("NULL\n");

}
```

OUTPUT:



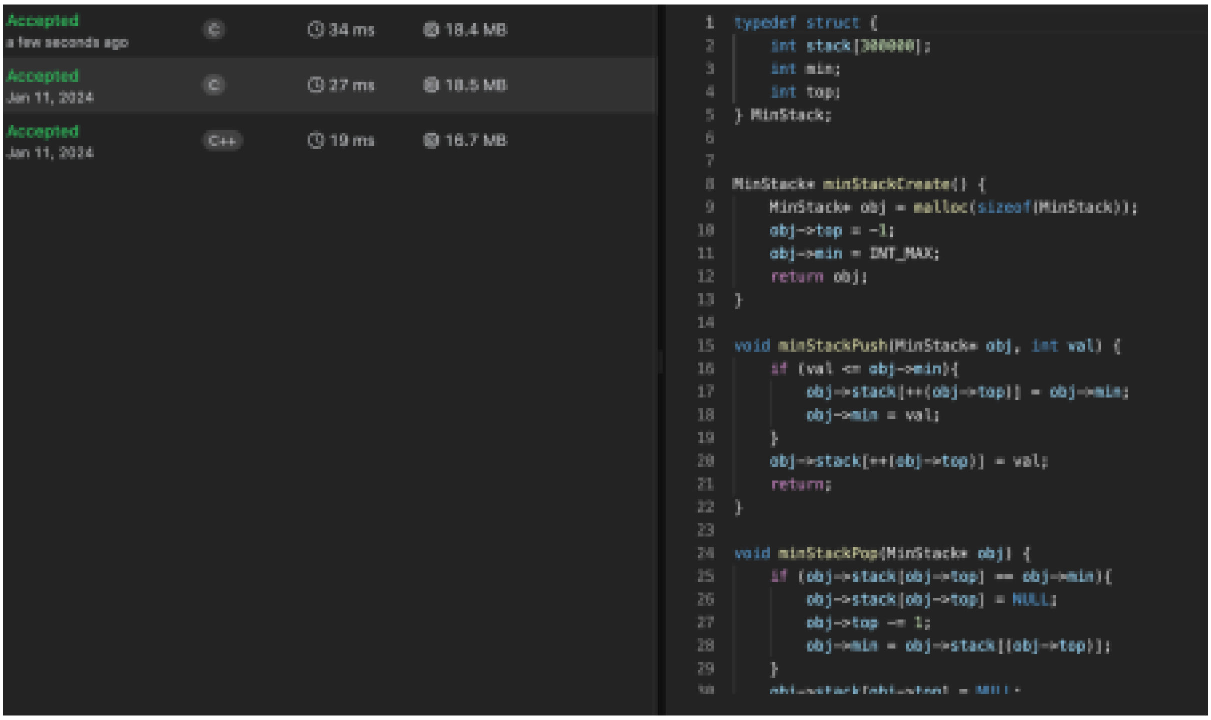
```
Enter choice: 1
Enter data in the new node: 0
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 2
Enter data in the new node: 2
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 3
Enter data in the new node: 1
Enter position of the new node: 1
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 4
0 -> 2 -> 1 -> NULL
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice:
```

Program - Leetcode platform

```
typedef struct {
    int stack[300000];
    int min;
    int top;
} MinStack;


MinStack* minStackCreate() {
    MinStack* obj = malloc(sizeof(MinStack));
    obj->top = -1;
    obj->min = INT_MAX;
    return obj;
}

void minStackPush(MinStack* obj, int val) {
    if (val <= obj->min){
        obj->stack[++(obj->top)] = obj->min;
```

```
        obj->min = val;
    }
    obj->stack[++(obj->top)] = val;
    return;
}

void minStackPop(MinStack* obj) {
    if (obj->stack[obj->top] == obj->min){
        obj->stack[obj->top] = NULL;
        obj->top -= 1;
        obj->min = obj->stack[(obj->top)];
    }
    obj->stack[obj->top] = NULL;
    obj->top -= 1;
}

int minStackTop(MinStack* obj) {
    return obj->stack[obj->top];
}

int minStackGetMin(MinStack* obj) {
    return obj->min;
}

void minStackFree(MinStack* obj) {
    free(obj);
}
```

OUTPUT:



Accepted a few seconds ago	C	34 ms	18.4 MB
Accepted Jan 11, 2024	C	27 ms	18.5 MB
Accepted Jan 11, 2024	C++	19 ms	18.7 MB

```
typedef struct {
    int stack[300000];
    int min;
    int top;
} MinStack;


MinStack* minStackCreate() {
    MinStack* obj = malloc(sizeof(MinStack));
    obj->top = -1;
    obj->min = INT_MAX;
    return obj;
}

void minStackPush(MinStack* obj, int val) {
    if (val <= obj->min){
        obj->stack[++(obj->top)] = obj->min;
        obj->min = val;
    }
    obj->stack[++(obj->top)] = val;
    return;
}

void minStackPop(MinStack* obj) {
    if (obj->stack[obj->top] == obj->min){
        obj->stack[obj->top] = NULL;
        obj->top -= 1;
        obj->min = obj->stack[(obj->top)];
    }
    obj->stack[obj->top] = NULL;
```

LAB 5 PROGRAMS:

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 create_ll(struct node **start);

void display(struct node *start);
void pop(struct node **start);

void end_delete(struct node **start);

void delete_at_pos(struct node **start);

void free_list(struct node *start);


int main(void)

{

   struct node *start = NULL;

   int option;


   do
```

```
          {

printf("\n\n *****MAIN MENU *****");

printf("\n 1: Create a list");

printf("\n 2: Display the list");

printf("\n 3: Delete a node from the beginning");

printf("\n 4: Delete a node from the end");

printf("\n 5: Delete a from a specific position");

printf("\n 6: EXIT");

printf("\n Enter your option : ");

scanf("%d", &option);


      switch (option)

      {

      case 1:

create_ll(&start);
```

```
printf("\n LINKED LIST CREATED");

            break;

        case 2:

            display(start);

            break;

        case 3:

            pop(&start);

            break;

        case 4:

end_delete(&start);

            break;

        case 5:
```

```
delete_at_pos(&start);

            break;

        case 6:

free_list(start);

printf("\nExiting....\n");

            break;

        }

            } while (option != 6);


    return 0;

}


void create_ll(struct node **start)

 {
   struct node *new_node, *ptr;

   int num;

printf("Enter -1 to end\n");

printf("Enter the data : \n");

scanf("%d", &num);


   while (num != -1)

         {

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


      if (new_node == NULL)

       {

printf("Memory allocation failed\n");
```

```
exit(EXIT_FAILURE);

       }


new_node->data = num;

new_node->next = NULL;


      if (*start == NULL)

       {

         *start = new_node;

       }

      else

       {

ptr = *start;
```

```
            while (ptr->next != NULL)
ptr = ptr->next;

ptr->next = new_node;
        }

printf("\nEnter the data : ");
scanf("%d", &num);
            }
}

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

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

void pop(struct node **start)
 {
    if (*start == NULL)
            {
printf("List is empty\n");
```

```
      return;
         }

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

void end_delete(struct node **start)
 {
   if (*start == NULL)
         {
printf("List is empty\n");
      return;
```

```
         }

   struct node *ptr = *start;
   struct node *ptr1 = NULL;

   while (ptr->next != NULL)
         {
      ptr1 = ptr;
ptr = ptr->next;
         }

   if (ptr1 != NULL)
```

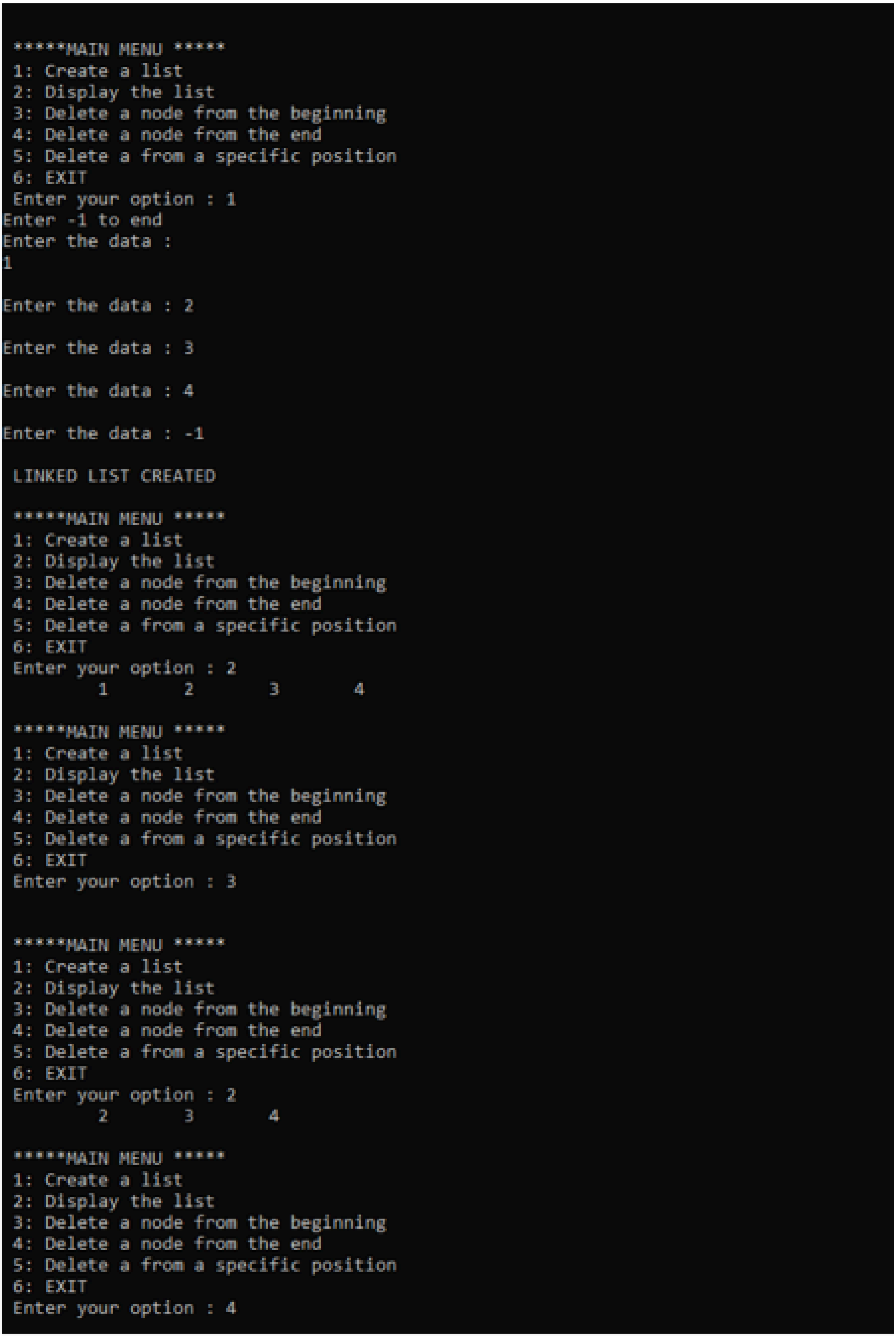
```
            {
        ptr1->next = NULL;
        free(ptr);
            }
    else
            {
        // Only one node in the list
        free(ptr);
        *start = NULL;
            }
}

void delete_at_pos(struct node **start)
 {
    if (*start == NULL)
```

```
            {
printf("List is empty\n");
        return;
            }

    int loc;
printf("\nEnter the location of the node which has to be deleted : ");
scanf("%d", &loc);

    struct node *ptr = *start;
    struct node *ptr1 = NULL;
    for (int i = 0; i<loc; i++)
            {
        ptr1 = ptr;
ptr = ptr->next;


        if (ptr == NULL)
         {
printf("There are less than %d elements in the list\n", loc);
          return;
         }
            }


    if (ptr1 != NULL)
            {
        ptr1->next = ptr->next;
```

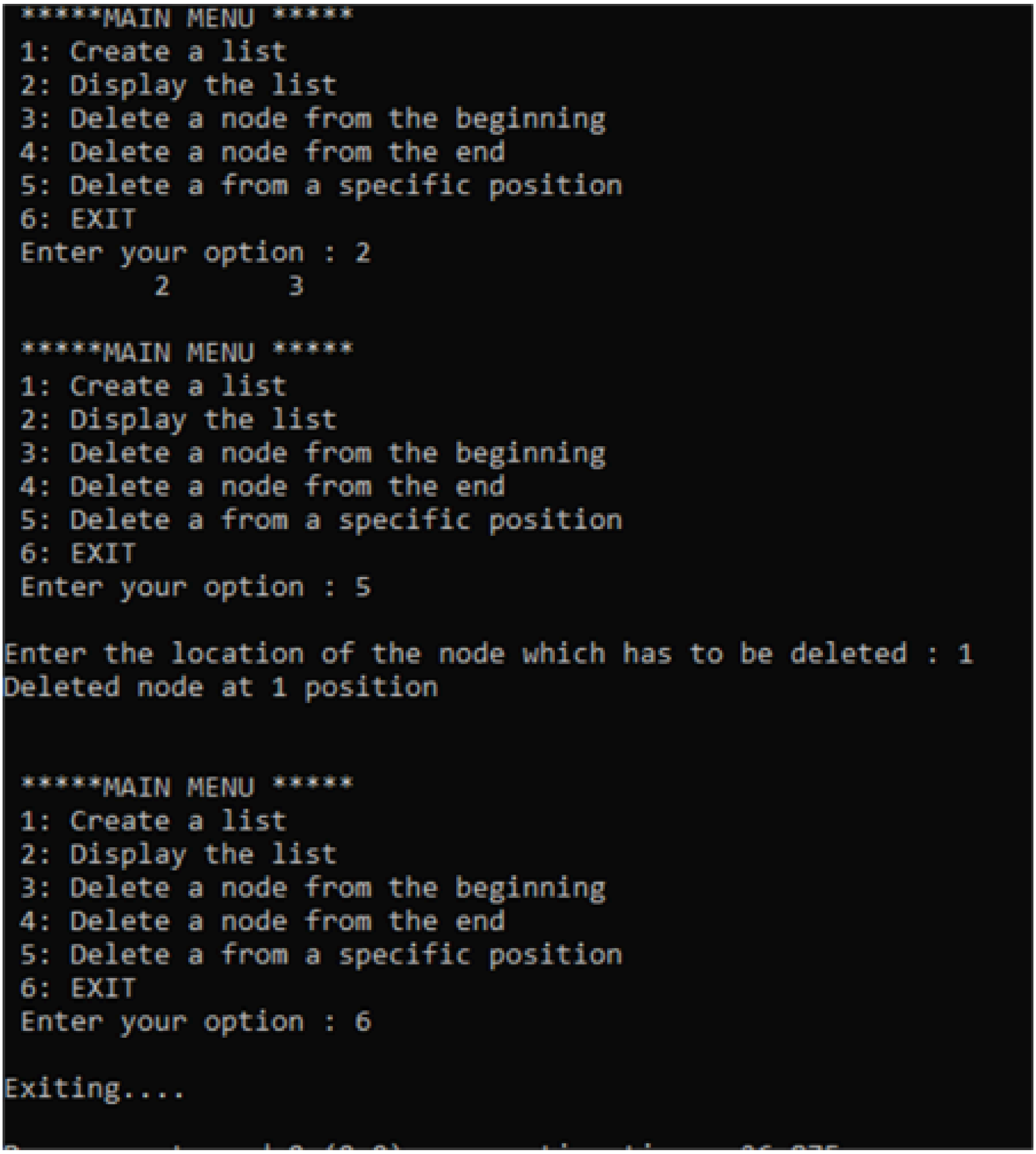
```
       free(ptr);
printf("Deleted node at %d position\n", loc);
           }
    else
           {
       // Deleting the first node
       *start = ptr->next;
       free(ptr);
printf("Deleted node at %d position\n", loc);
           }
```

```
}


void free_list(struct node *start)
 {
    struct node *ptr = start;
    struct node *next_node;


    while (ptr != NULL)
            {
next_node = ptr->next;
        free(ptr);
ptr = next_node;
            }
}
```

OUTPUT:



```

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node from the beginning
 4: Delete a node from the end
 5: Delete a from a specific position
 6: EXIT
 Enter your option : 1
Enter -1 to end
Enter the data :
1

Enter the data : 2

Enter the data : 3

Enter the data : 4

Enter the data : -1

 LINKED LIST CREATED

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node from the beginning
 4: Delete a node from the end
 5: Delete a from a specific position
 6: EXIT
 Enter your option : 2
         1       2       3       4

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node from the beginning
 4: Delete a node from the end
 5: Delete a from a specific position
 6: EXIT
 Enter your option : 3


 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node from the beginning
 4: Delete a node from the end
 5: Delete a from a specific position
 6: EXIT
 Enter your option : 2
         2       3       4

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node from the beginning
 4: Delete a node from the end
 5: Delete a from a specific position
 6: EXIT
 Enter your option : 4
```

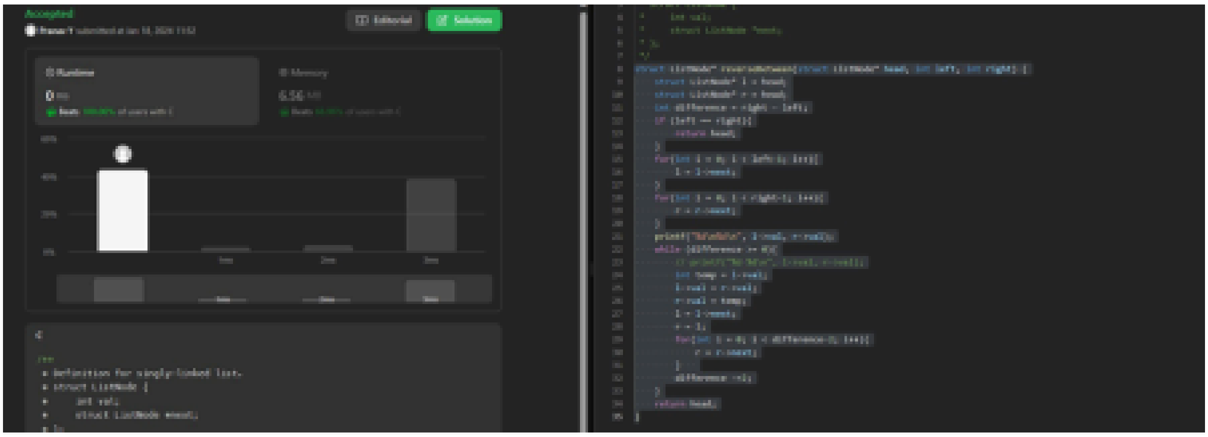


```
 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node from the beginning
 4: Delete a node from the end
 5: Delete a from a specific position
 6: EXIT
 Enter your option : 2
         2       3

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node from the beginning
 4: Delete a node from the end
 5: Delete a from a specific position
 6: EXIT
 Enter your option : 5

Enter the location of the node which has to be deleted : 1
Deleted node at 1 position


 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node from the beginning
 4: Delete a node from the end
 5: Delete a from a specific position
 6: EXIT
 Enter your option : 6

Exiting....
```

LEETCODE PROGRAM:

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
   struct ListNode* l = head;
   struct ListNode* r = head;
   int difference = right - left;
   if (left == right){
      return head;
   }
   for(int i = 0; i < left-1; i++){
      l = l->next;
   }
   for(int i = 0; i < right-1; i++){
        r = r->next;
    }
    printf("%d\n%d\n", l->val, r->val);
    while (difference >= 0){
        // printf("%d %d\n", l->val, r->val);
        int temp = l->val;
        l->val = r->val;
        r->val = temp;
        l = l->next;
        r = l;
        for(int i = 0; i < difference-2; i++){
            r = r->next;
        }
        difference -=2;
    }
    return head;
}
```

OUTPUT:



LAB 6 PROGRAMS:

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>
#include <stdbool.h>

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

node *head = NULL;
node *head1 = NULL;
int count = 0;

void insert(int data, int position);
void delete(int position);
```

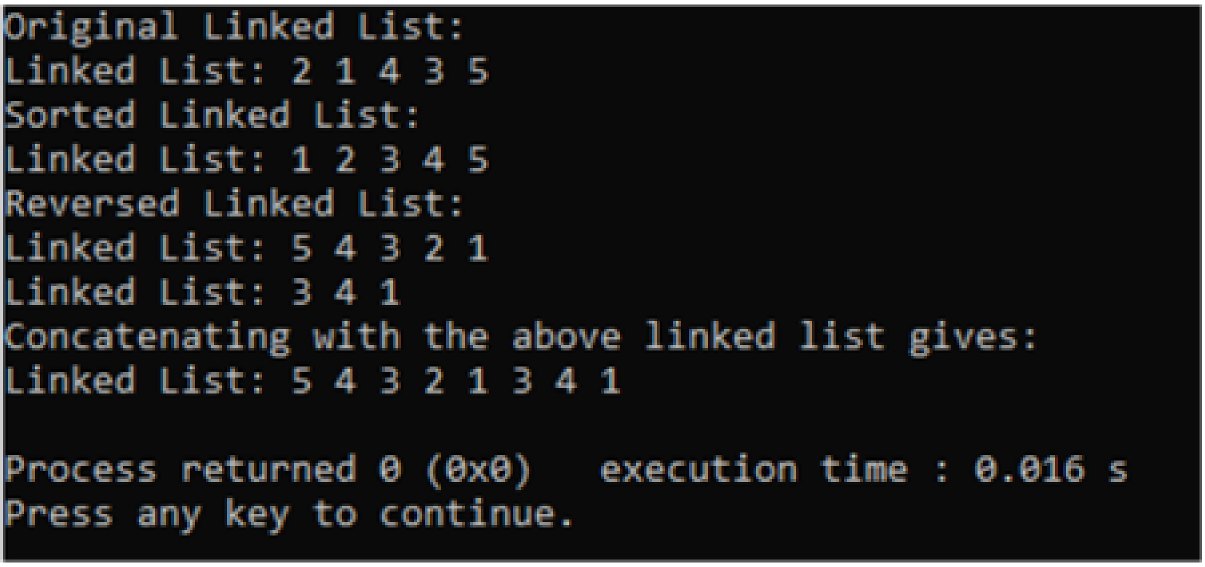
```
void display();
void sort();
void reverse();
void concat(node** head1, node** head2);

int main(){
    insert(2, 0);
    insert(1, 1);
    insert(4, 2);
    insert(3, 3);
    insert(5, 4);
    printf("Original Linked List: \n");
    display();
    sort();
    printf("Sorted Linked List: \n");
    display();
    reverse();
    printf("Reversed Linked List: \n");
    display();
    head1 = head;
    head = NULL;
    insert(3, 0);
    insert(4, 1);
    insert(1, 2);
    display();
    concat(&head1, &head);
    head = head1;
    printf("Concatenating with the above linked list gives: \n");
    display();

    return 0;
}

void insert(int data, int position){
    if (position == 0){
        node* new_node = (node*)malloc(sizeof(node));
        new_node->data = data;
        new_node->next = head;
        head = new_node;
        count++;
        return;

    } else if (position == count){
        node* new_node = malloc(sizeof(node));
        new_node->data = data;
        new_node->next = NULL;
        node* temp = head;
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = new_node;
```

```
        count++;
        return;

    } else if (position > count || position < 0){
        printf("Unable to insert at given position\n");
        return;
    } else {
        node* temp = head;
        for(int i = 0; i < position-1; i++)
            temp = temp->next;
        node* new_node = malloc(sizeof(node));
        new_node->data = data;
        new_node->next = temp->next;
        temp->next = new_node;
        count++;
        return;
    }
}

void delete(int position){
    if (position == 0){
        node* temp = head;
        head = head->next;
        free(temp);
        count--;
        return;
    } else if (position == count-1){
        node* temp = head;
        for(int i = 1; i < count-1; i++)
            temp = temp->next;
        node* temp1 = temp->next;
        temp->next = NULL;
        free(temp1);
        count--;
        return;
    } else if (position > count || position < 0){
        printf("Unable to delete at given position\n");
        return;
    } else {
        node* temp = head;
        for(int i = 0; i < position-1; i++)
            temp = temp->next;
        node* temp1 = temp->next;
        temp->next = temp1->next;
        free(temp1);
        count--;
        return;
    }
}
```

```
void sort(){
    int i, j, min_index;
    node *i_node=head, *j_node=head, *min_node=NULL;
    for(int i = 0; i < count-1; i++, i_node=i_node->next){ //
    printf("Got here\n");
        min_index = i;
        min_node = i_node;
        j_node = i_node->next;
        for(int j = i+1; j < count; j++, j_node=j_node->next){
            // printf("Got here too\n");
            if (j_node->data < i_node->data){
                min_index = j;
                min_node = j_node;
            }
        }
        // printf("%d\n", min_index);
        if (min_index != i){
            // printf("Found a min element\n");
            int temp = i_node->data;
            i_node->data = min_node->data;
            min_node->data = temp;
            // display();
        }
    }
}

void reverse(){
    node *prev = NULL, *next=NULL;
    while(head != NULL){
        next = head->next;
        head->next = prev;
        prev = head;
        head = next;
    }
    head = prev;
}

void concat(node **head1, node **head2){
    node *temp1 = *head1;
    while (temp1->next != NULL){
        temp1 = temp1->next;
    }
    // printf("Got here atleast?\n");
    temp1->next = *head2;
    // printf("Got here?\n");
}

void display(){
    node* temp = head;
    printf("Linked List: ");
```

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

OUTPUT:



```
Original Linked List:
Linked List: 2 1 4 3 5
Sorted Linked List:
Linked List: 1 2 3 4 5
Reversed Linked List:
Linked List: 5 4 3 2 1
Linked List: 3 4 1
Concatenating with the above linked list gives:
Linked List: 5 4 3 2 1 3 4 1

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

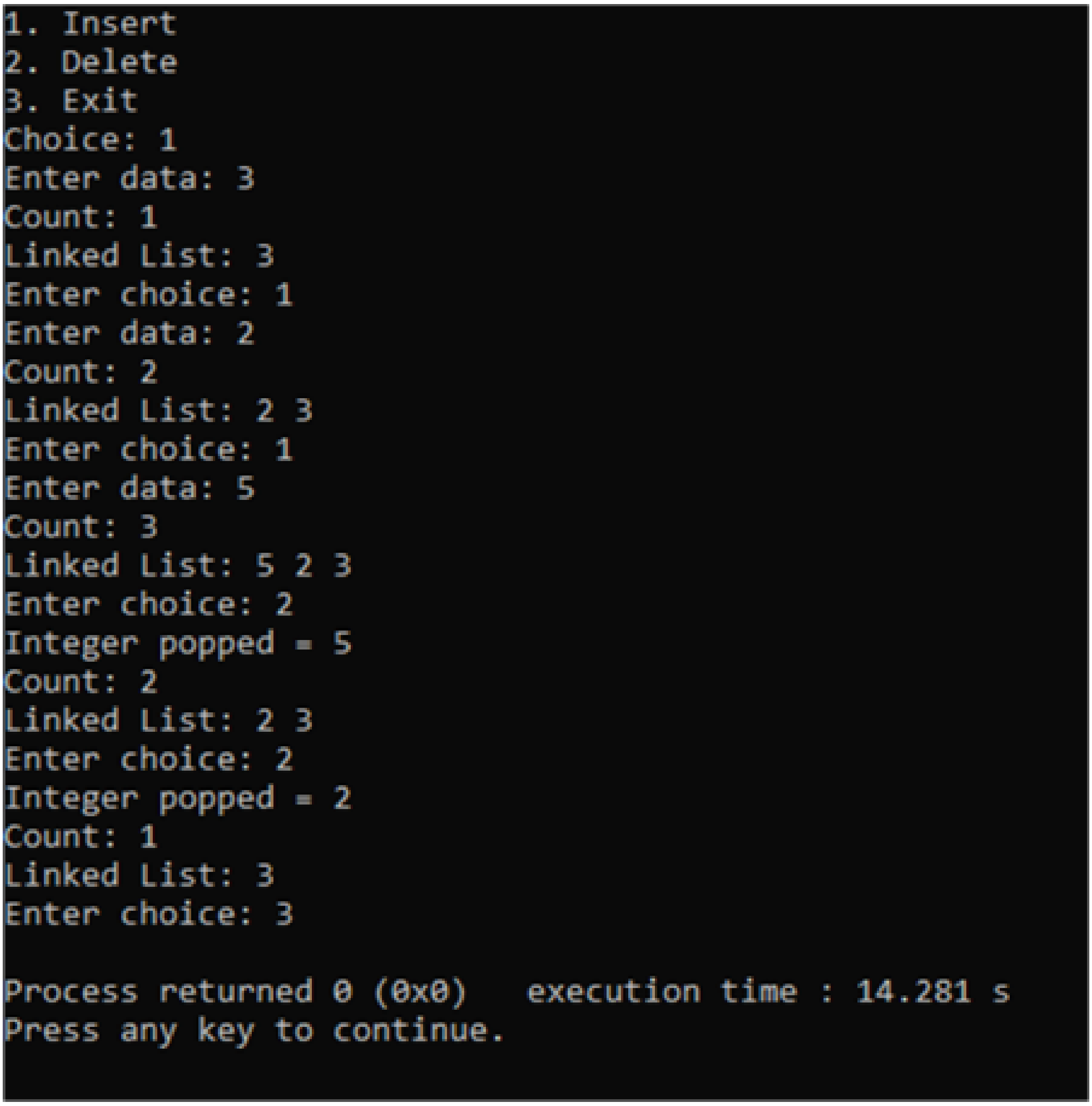
6b) WAP to Implement Single Link List to simulate Stack &amp; Queue Operations.

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

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

node* head = NULL;
int count = 0;

void insert(int data);
int delete();
void display();
```

```
int main(){
    int data, choice, pos;
    printf("1. Insert\n2. Delete\n3. Exit\nChoice: ");
    scanf("%d", &choice);
    while(choice != 3){
        if (choice == 1){
            printf("Enter data: ");
            scanf("%d", &data);
```

```
            insert(data);
            printf("Count: %d\n", count);
        } else if (choice == 2){
            printf("Integer popped = %d\n", delete());
            printf("Count: %d\n", count);
        }
        display();
        printf("Enter choice: ");
        scanf("%d", &choice);
    }


    return 0;
}

void insert(int data){
    node* new_node = (node*)malloc(sizeof(node));
    new_node->data = data;
    new_node->next = head;
    head = new_node;
    count++;
    return;

}

int delete(){
    node* temp = head;
    head = head->next;
    int t = temp->data;
    free(temp);
    count--;
    return t;

}

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

OUTPUT:



```
1. Insert
2. Delete
3. Exit
Choice: 1
Enter data: 3
Count: 1
Linked List: 3
Enter choice: 1
Enter data: 2
Count: 2
Linked List: 2 3
Enter choice: 1
Enter data: 5
Count: 3
Linked List: 5 2 3
Enter choice: 2
Integer popped = 5
Count: 2
Linked List: 2 3
Enter choice: 2
Integer popped = 2
Count: 1
Linked List: 3
Enter choice: 3

Process returned 0 (0x0)   execution time : 14.281 s
Press any key to continue.
```

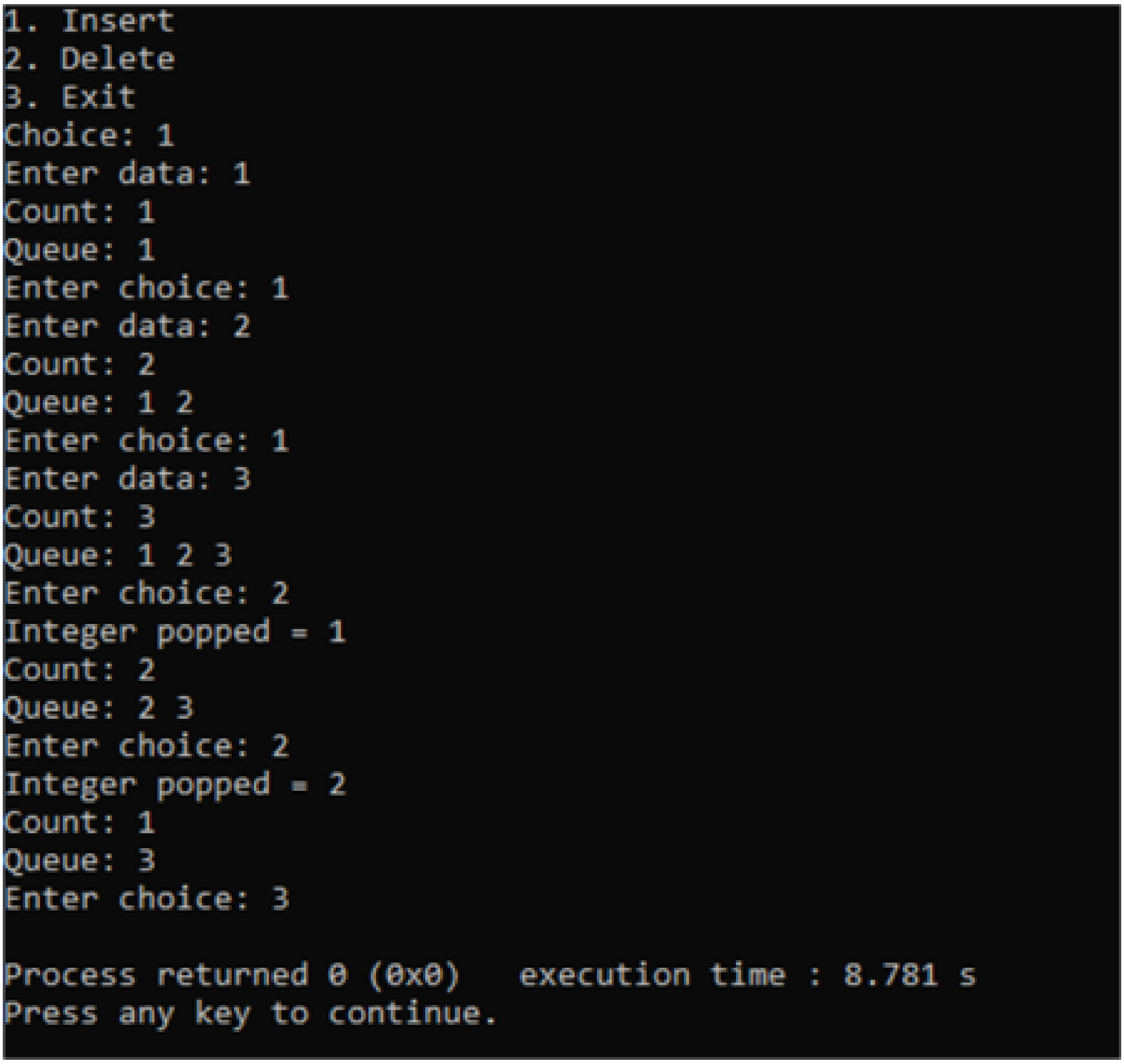
II)QUEUE:

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

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

node* head = NULL;
int count = 0;

void insert(int data);
int delete();
void display();
```

```
int main(){
    int data, choice, pos;
    printf("1. Insert\n2. Delete\n3. Exit\nChoice: ");
    scanf("%d", &choice);
    while(choice != 3){
        if (choice == 1){
            printf("Enter data: ");
            scanf("%d", &data);
            insert(data);
            printf("Count: %d\n", count);
        } else if (choice == 2){
            printf("Integer popped = %d\n", delete());
            printf("Count: %d\n", count);
        }
        display();
        printf("Enter choice: ");
        scanf("%d", &choice);
    }


    return 0;
}

void insert(int data){
    node* new_node = malloc(sizeof(node));
    new_node->data = data;
    new_node->next = NULL;
    if (head == NULL){
        head = new_node;
        count++;
        return;
    }
    node* temp = head;
    while(temp->next != NULL)
        temp = temp->next;
    temp->next = new_node;
    count++;
    return;

}

int delete(){
    node* temp = head;
    head = head->next;
    int t = temp->data;
    free(temp);
    count--;
    return t;

}
```

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

OUTPUT:



```
1. Insert
2. Delete
3. Exit
Choice: 1
Enter data: 1
Count: 1
Queue: 1
Enter choice: 1
Enter data: 2
Count: 2
Queue: 1 2
Enter choice: 1
Enter data: 3
Count: 3
Queue: 1 2 3
Enter choice: 2
Integer popped = 1
Count: 2
Queue: 2 3
Enter choice: 2
Integer popped = 2
Count: 1
Queue: 3
Enter choice: 3

Process returned 0 (0x0)   execution time : 8.781 s
Press any key to continue.
```

LAB 7 PROGRAMS:
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>
#include <stdbool.h>

typedef struct Node{
   int data;
   struct Node *next;
   struct Node *prev;
} node;
```

```
node* head = NULL;
int count = 0;

void insert(int data, int position);
void delete(int element);
void display();

int main(){
  int data, choice, pos;
  printf("1. Insert\n2. Delete\n3. Exit\nChoice: ");
  scanf("%d", &choice);
  while(choice != 3){
     if (choice == 1){
        printf("Enter data and position: ");
        scanf("%d%d", &data, &pos);
        insert(data, pos);
        printf("Count: %d\n", count);
     } else if (choice == 2){
        printf("Enter element: ");
        scanf("%d", &pos);
        delete(pos);
        printf("Count: %d\n", count);
     }
     display();
     printf("Enter choice: ");
     scanf("%d", &choice);
  }


  return 0;
}

void insert(int data, int position){
  if (position == 0){
     node* new_node = malloc(sizeof(node));
     new_node->data = data;
     new_node->next = head;
```

```
     new_node->prev = NULL;
     if (head != NULL) head->prev =
     new_node; head = new_node;
     count++;
     return;
  } else if (position == count){
     node* new_node = malloc(sizeof(node));
     new_node->data = data;
     new_node->next = NULL;
     node* temp = head;
```

```
        while(temp->next != NULL)
            temp = temp->next;
        temp->next = new_node;
        new_node->prev = temp;
        count++;
        return;

    } else if (position > count || position < 0){
        printf("Unable to insert at given position\n");
        return;
    } else {
        node* temp = head;
        for(int i = 0; i < position-1; i++)
            temp = temp->next;
        node* new_node = malloc(sizeof(node));
        new_node->data = data;
        new_node->next = temp->next;
        new_node->prev = temp;
        temp->next->prev = new_node;
        temp->next = new_node;
        count++;
        return;
    }
}

void delete(int element){
    int position = 0; node *temp = head;
    if (head == NULL){
        printf("List is empty, cannot delete"); return;
    }
    for(;position < count; temp=temp->next,
        position++) if (temp->data == element) break;
    if (temp == NULL){
        printf("Element does not exist in list"); return;
    }
    if (position == 0){
        node* temp = head;
        temp = temp->next;
        temp->prev = NULL;
        free(head);
```

```
        head = temp;
        count--;
        return;
    } else if (position == count-1){
        node* temp = head;
        for(int i = 1; i < count-1; i++)
            temp = temp->next;
        node* temp1 = temp->next;
        temp->next = NULL;
        free(temp1);
        count--;
        return;
    } else if (position > count || position <
        0){ printf("Unable to delete at
        position\n"); return;
    } else {
        node* temp = head;
        for(int i = 0; i < position; i++)
            temp = temp->next;
        temp->next->prev = temp->prev;
        temp->prev->next = temp->next;
        free(temp);
        count--;
        return;
    }
}

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

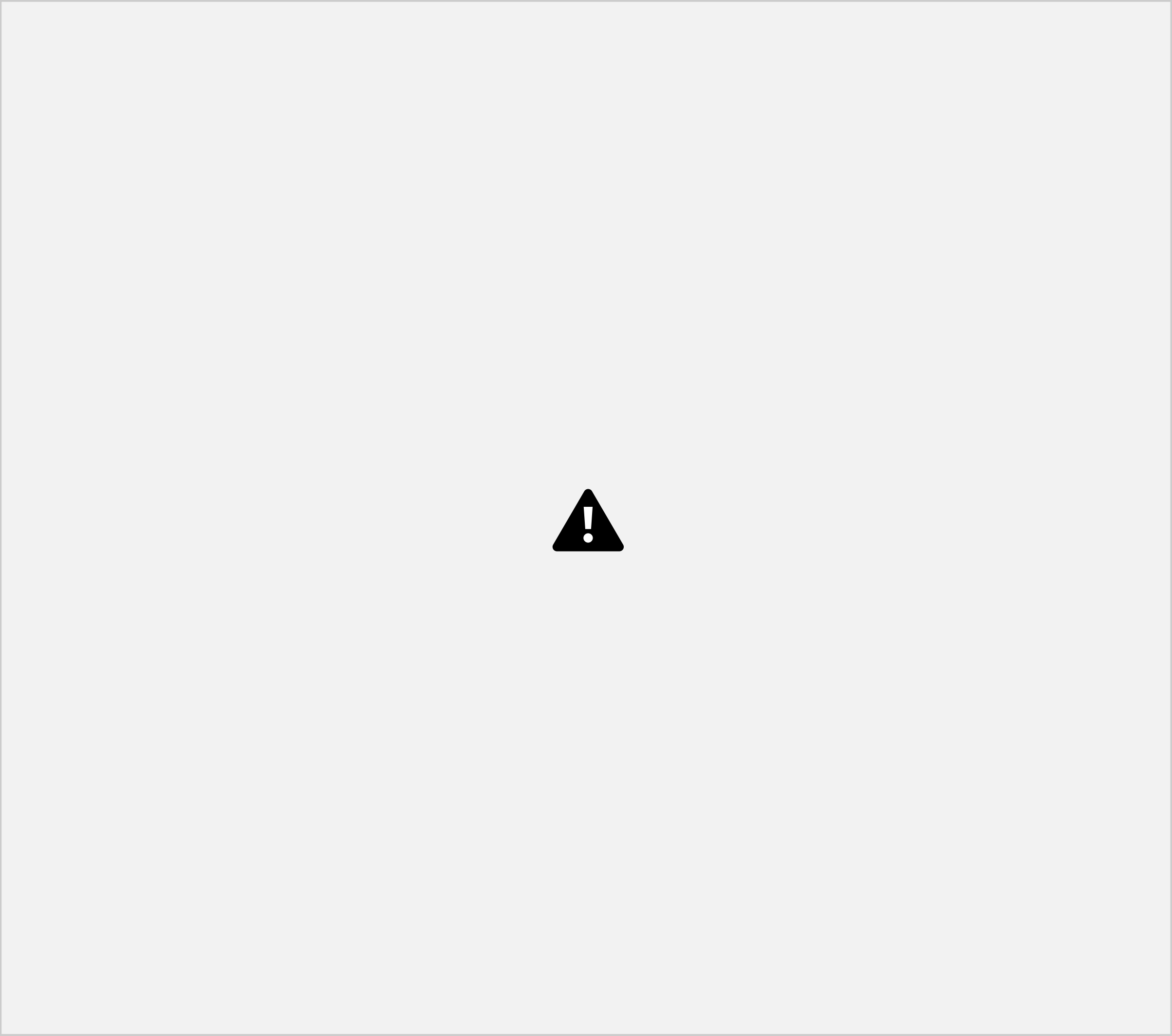
OUTPUT:

Program - Leetcode platform

```
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
    struct ListNode* temp = head; int n = 0;
    for(; temp != NULL; temp=temp->next, n++);
    struct ListNode** lists = (struct ListNode**)malloc(k*sizeof(struct
    ListNode*)); for(int i = 0; i < k; i++) lists[i] = NULL;
    int earlier_lists = n%k, size=n/k;
    int current = 0; bool list_over = false;
    temp = head;
    *returnSize = k;
```

```
  for(int i = earlier_lists; i > 0; i--){
        // printf("Entering here\n");
        struct ListNode* temp1 = temp;
        lists[current++] = temp;
        for(int j = 0; j < size; j++) temp1 = temp1->next;
        temp = temp1->next;
        temp1->next = NULL;
  }
  // printf("%d %d %d", lists[0]->val, lists[1]->val, lists[2]->val);
  if (temp == NULL) return lists;
  for(int i = 0; i < k-earlier_lists; i++){
        struct ListNode* temp1 = temp;
        if (temp1 == NULL) break;
        for(int j = 0; j < size-1; j++) temp1 = temp1->next;
        lists[current++] = temp;
        temp = temp1->next;
        temp1->next = NULL;
        // for(int l = 0; l < k; l++){
        // for(struct ListNode* temp2 = lists[l]; temp2 != NULL; temp2 = temp2->next){ //
        printf("%d ", temp2->val);
        // }
        // printf("\n");
        // }
  }
  return lists;
}
```

OUTPUT:



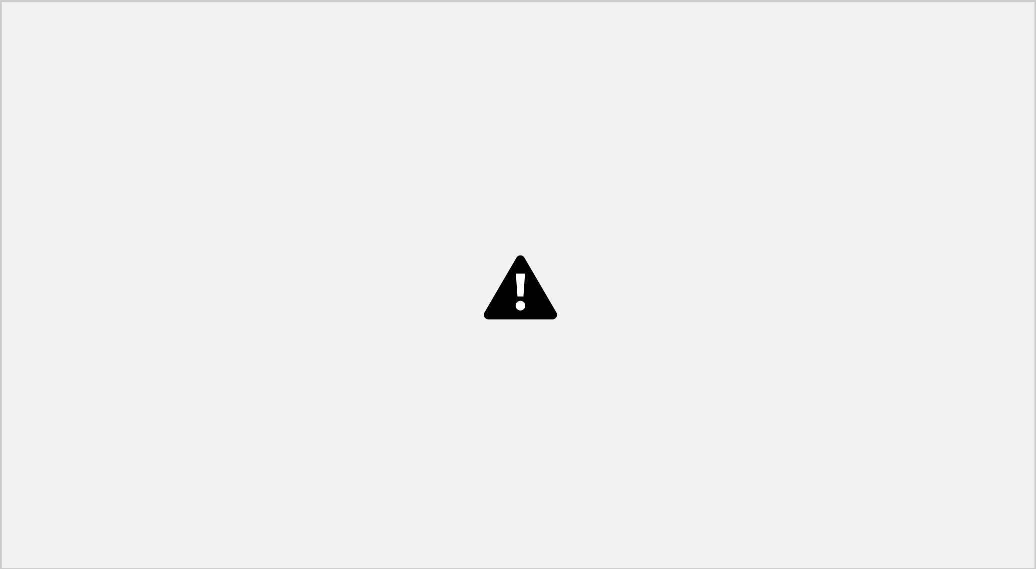
LAB 8 PROGRAMS:
Write a program

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>
#include <stdbool.h>

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

node *root = NULL;

void insert(node **root, int data);
void preorder(node **root);
void postorder(node **root);
void inorder(node **root);

int main(){
    int choice, data;
    insert(&root, 8);
    insert(&root, 3);
    insert(&root, 1);
    insert(&root, 6);
    insert(&root, 4);
    insert(&root, 7);
    insert(&root, 10);
    insert(&root, 14);
    insert(&root, 13);
    printf("1. Preorder\n2. Inorder\n3. Postorder\n4. Exit\nChoice: ");
    scanf("%d", &choice);
    while (choice != 4){
        if (choice == 1){
            preorder(&root);
            printf("\n");
        } else if (choice == 2){
            inorder(&root);
            printf("\n");
        } else if (choice == 3){
            postorder(&root);
            printf("\n");
        }
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
}

void insert(node **root, int data){
    if (*root == NULL) {
        node *new_node = malloc(sizeof(node));
        new_node->data = data;
        new_node->right = NULL;
        new_node->left = NULL;
        *root = new_node;
        return;
    }
    if (data < (*root)->data){
        insert(&((*root)->left), data);
    } else if (data > (*root)->data){
        insert(&((*root)->right), data);
    }
    return;
}
```

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

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

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

OUTPUT:



LEETCODE PROGRAM:

```
struct ListNode* rotateRight(struct ListNode* head, int k) {
   struct ListNode *temp = head;
   if (head == NULL) return NULL;
   if (head->next == NULL) return head;
   if (k == 0) return head;
   int size = 1;
   for(; temp->next != NULL; temp=temp->next, size++);
   k %= size;
```

```
   if (k == 0) return head;
   temp->next = head;
   struct ListNode *temp1 = head;
   for(int i = 0; i < (size-k-1); temp1 = temp1->next, i++);
   head = temp1->next;
   temp1->next = NULL;
   return head;
}
```

OUTPUT:

LAB 9 PROGRAMS:

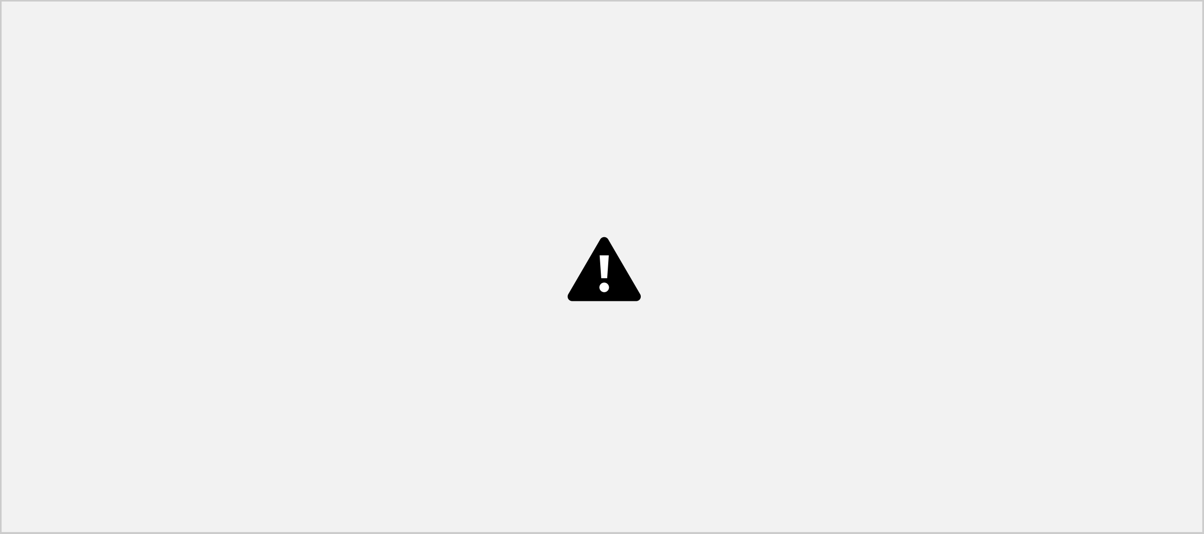
9a) Write a program to traverse a graph using BFS method.

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

#define size 7

void push(int a);
int pop();
void display();
void bfs(int graph[][7]);

int fpos = -1, rpos = -1;
int queue[size];

int main(){
  int adj_matrix[7][7] = {
       {0, 1, 0, 1, 0, 0, 0},
       {1, 0, 1, 1, 0, 1, 1},
       {0, 1, 0, 1, 1, 1, 0},
       {1, 1, 1, 0, 0, 0, 0},
       {0, 0, 1, 0, 0, 0, 1},
```

```
        {0, 1, 1, 0, 0, 0, 0},
        {0, 1, 0, 0, 1, 0, 0},
    };
    for(int i = 0; i < 7; i++) queue[i] = NULL;
    // display();
    bfs(adj_matrix);
    return 0;

}

void bfs(int graph[][7]){
    int visited[7];
    for(int i = 0; i < 7; i++) visited[i] = 0;
    push(0); visited[0]= 1;
    while (fpos != size){
        for(int i = 0; i < 7; i++){
            if(graph[queue[fpos]][i] == 1 && visited[i] != 1){
                push(i);
                visited[i] = 1;
                // break;
            }
        }
        printf("%d ", pop());
        // printf("%d\n", new_node);
    }
}

void push(int a){
    if (fpos == -1 && rpos == -1){
        queue[++rpos] = a;
        fpos++;
        return;
    }
    else if (rpos == size-1){
        printf("Queue overflow condition\n");
        return;
    }
    else{
        queue[++rpos] = a;
        return;
    }
}

int pop(){
    if (fpos == -1){
       printf("Queue Underflow condition\n");
    }
    int n = queue[fpos];
    queue[fpos] = (int) NULL;
    fpos++;
```

```
   return n;
}

void display(){
   printf("Queue: ");
    for(int i = 0; i < size; i++)
       printf("%d ", queue[i]);
   printf("\n");
}
```

OUTPUT:



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

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

#define size 7
int pos = -1;
int stack[size];

void push(int a);
int pop();
void display();
void dfs(int graph[][7]);

int main(){
    int adj_matrix[7][7] = {
        {0, 1, 0, 1, 0, 0, 0},
        {1, 0, 1, 1, 1, 0, 1, 1},
        {0, 1, 0, 1, 1, 1, 0},
        {1, 1, 1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 1},
        {0, 1, 1, 0, 0, 0, 0},
```

```
        {0, 1, 0, 0, 1, 0, 0},
    };
    for(int i = 0; i < 7; i++) stack[i] = NULL;
    // display();
    dfs(adj_matrix);
    return 0;


}

void dfs(int graph[][7]){
    int visited[7];
    for (int i = 0; i < 7; i++) visited[i] = 0;
    push(0); visited[0] = 1; printf("0 ");
    // printf("%d ", pos);
    // return;
    // display();
    while(pos != -1){
        bool new_node = false;
        for(int i = 0; i < 7; i++){
            // printf("%d ", graph[stack[pos]][i]);
            if(graph[stack[pos]][i] == 1 && visited[i] != 1){
                new_node = true;
                // printf("Current top: %d\n", i);
                push(i);
                // display();
                visited[i] = 1; printf("%d ", i);
                break;
            }
        }
        // printf("%d\n", new_node);
        if (!new_node) pop();
    }

}

void push(int a){
    if (pos == size-1){
        printf("Stack Overflow condition");
        return;
    }
```

```
   stack[++pos] = a;
}

int pop(){
   if (pos == -1){
      printf("Stack Underflow condition");
      return (int) NULL;
   }
   return stack[pos--];
```

```
}

void display(){
   for(int i = 0; i < size; i++){
      printf("%d ", stack[i]);
   }
   printf("\n");
}
```

OUTPUT:

LAB 10 PROGRAMS:

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.

Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.

Let the keys in K and addresses in L are integers.

Design and develop a Program in C that uses Hash function H: K -&gt; L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L.

Resolve the collision (if any) using linear probing.

```
#include <stdio.h>

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

```
#include <string.h>

#define size 10

int table[size];

void push(int data);
int pop(int data);
void search(int data);
void display();
```

```
int main(){
    for(int i = 0; i < size; i++) table[i] = -1;
    int choice;
    printf("1. Insert\n2. Delete \n3. Display\n4. Exit\nChoice: ");
    scanf("%d", &choice);
    int a;
    while(choice != 4){
        switch(choice){
            case 1:
                printf("Enter integer to be pushed: ");
                scanf("%d", &a);
                push(a);
                break;
            case 2:
                printf("Enter integer to be popped: ");
                scanf("%d", &a);
                int res = pop(a);
                if (res == 0) printf("Integer popped\n");
                else printf("Integer not found\n");
                break;
            case 3:
                display();
                break;
            default:
                printf("Idk");
                break;
        }
        printf("Enter choice: ");
        scanf("%d", &choice);
    }
}

void push(int data){
    int hash = data%size;
    while (table[hash] != -1 && hash <= (hash+size-1)) hash = (hash+1)%size;
    if (table[hash] == -1) table[hash] = data;
    else printf("Table is full");
}
```

```
int pop(int data){
    int hash = data%size;
     for(int i = 0; (table[hash] != data) || (i < size); i++, hash =
     (hash+1)%size); if (table[hash] == data) {
        table[hash] = -1; return 0;
     }
    return -1;
}

void display(){
```

```
printf("Table: ");
for(int i = 0; i < size; i++)
printf("%d ", table[i]);
printf("\n");
}
```

OUTPUT: