VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Kavana M A (1BM23CS145)

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 **Kavana M A (1BM23CS145)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

Prof. Namratha M
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 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>
#define STACK_SIZE 5
void push(int st[],int *top)
{
        int item;
        if(*top==STACK_SIZE-1)
                printf("Stack overflow\n");
        else
        {
                printf("\nEnter an item :");
                scanf("%d",&item);
                (*top)++;
                st[*top]=item;
        }
}
void pop(int st[],int *top)
{
        if(*top==-1)
                printf("Stack underflow\n");
        else
        {
                printf("\n%d item was deleted",st[(*top)--]);
        }
}
void display(int st[],int *top)
{
        int i;
        if(*top==-1)
                printf("Stack is empty\n");
        for(i=0;i<=*top;i++)
                printf("%d\t",st[i]);
}
void main()
{
        int st[10],top=-1, c,val_del;
        while(1)
        {
                printf("\n1. Push\n2. Pop\n3. Display\n");
                printf("\nEnter your choice :");
                scanf("%d",&c);
                switch(c)
                {
```

```
case 1: push(st,&top);
                        break;
                case 2: pop(st,&top);
                        break;
                case 3: display(st,&top);
                        break;
                default: printf("\nInvalid choice!!!");
                        exit(0);
            }
        }
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\jyothika\DST> cd "d:\jyothika\DST\" ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :12

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :65

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :45

1. Push
2. Pop
3. Display

Enter your choice :1
Stack overflow
```

```
1. Push
2. Pop
3. Display

Enter your choice :2

45 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2

65 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :3
12
1. Push
2. Pop
3. Display

Enter your choice :2

12 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2
Stack underflow

1. Push
2. Pop
3. Display

Enter your choice :4

Invalid choice!!!
```

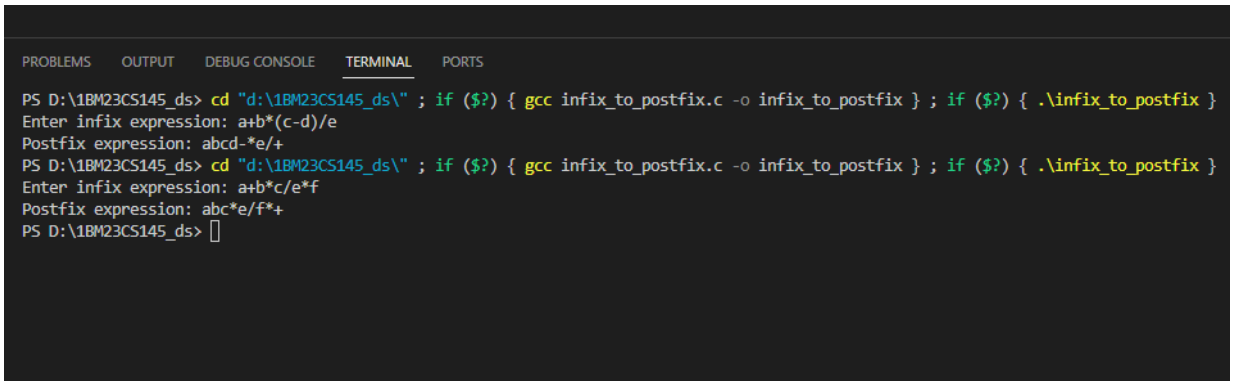
Lab program 2:

Program to convert a given valid parenthesized infix arithmetic expression to postfix expression.

```
#include <stdio.h>
#define STACK_SIZE 15

void push(char s[], int *top, char item) {
   (*top)++;
   s[*top] = item;
}

char pop(char s[], int *top) {
   return s[(*top)--];
}

int pr(char op) {
   switch(op) {
      case '#': return 0;
      case '(': return 1;
      case '+': return 2;
      case '-': return 2;
      case '*': return 3;
      case '/': return 3;
      default: return 0;
   }
}

int main() {
   char s[STACK_SIZE];
   int top = -1;
   char str[30], postfix[30];
   int i = 0, j = 0;

   push(s, &top, '#');
   printf("Enter infix expression: ");
   scanf("%s", str);

   while (str[i] != '\0') {
      if (str[i] != '+' && str[i] != '-' && str[i] != '*' && str[i] != '/' && str[i] != '(' && str[i] !=
')') {
         postfix[j++] = str[i];
      } else if (str[i] == '(') {
         push(s, &top, str[i]);
      } else if (str[i] == ')') {
         while (s[top] != '(') {
```

```
            postfix[j++] = pop(s, &top);
        }
        pop(s, &top);
    } else {
        while (pr(str[i]) <= pr(s[top])) {
            postfix[j++] = pop(s, &top);
        }
        push(s, &top, str[i]);
    }
    i++;
  }

  while (top > 0) {
    postfix[j++] = pop(s, &top);
  }

  postfix[j] = '\0';
  printf("Postfix expression: %s\n", postfix);
  return 0;
}
```

Output:



Lab program 3:

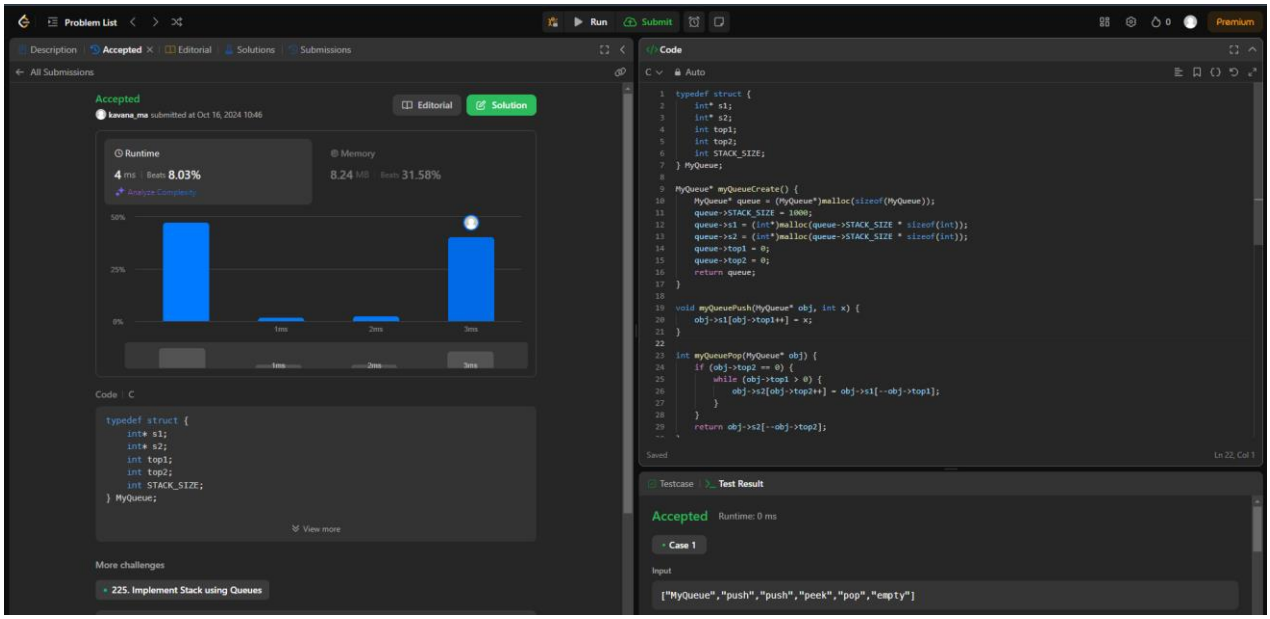
Program to simulate the working of a queue
LeetCode 1: Queue using Stack
Program to simulate the working of a circular queue
LeetCode 2: Valid Parentheses

LeetCode 1: Queue using Stack

```
typedef struct {
    int* s1;
    int* s2;
    int top1;
    int top2;
    int STACK_SIZE;
} MyQueue;

MyQueue* myQueueCreate() {
    MyQueue* queue = (MyQueue*)malloc(sizeof(MyQueue));
    queue->STACK_SIZE = 1000;
    queue->s1 = (int*)malloc(queue->STACK_SIZE * sizeof(int));
    queue->s2 = (int*)malloc(queue->STACK_SIZE * sizeof(int));
    queue->top1 = 0;
    queue->top2 = 0;
    return queue;
}

void myQueuePush(MyQueue* obj, int x) {
    obj->s1[obj->top1++] = x;
}

int myQueuePop(MyQueue* obj) {
    if (obj->top2 == 0) {
        while (obj->top1 > 0) {
            obj->s2[obj->top2++] = obj->s1[--obj->top1];
        }
    }
    return obj->s2[--obj->top2];
}

int myQueuePeek(MyQueue* obj) {
    if (obj->top2 == 0) {
        while (obj->top1 > 0) {
            obj->s2[obj->top2++] = obj->s1[--obj->top1];
        }
```

```
    }
    return obj->s2[obj->top2 - 1];
}

bool myQueueEmpty(MyQueue* obj) {
    return obj->top1 == 0 && obj->top2 == 0;
}

void myQueueFree(MyQueue* obj) {
    free(obj->s1);
    free(obj->s2);
    free(obj);
}

/**
 * Your MyQueue struct will be instantiated and called as such:
 * MyQueue* obj = myQueueCreate();
 * myQueuePush(obj, x);
 * int param_2 = myQueuePop(obj);
 * int param_3 = myQueuePeek(obj);
 * bool param_4 = myQueueEmpty(obj);
 * myQueueFree(obj);
*/
```

Output:



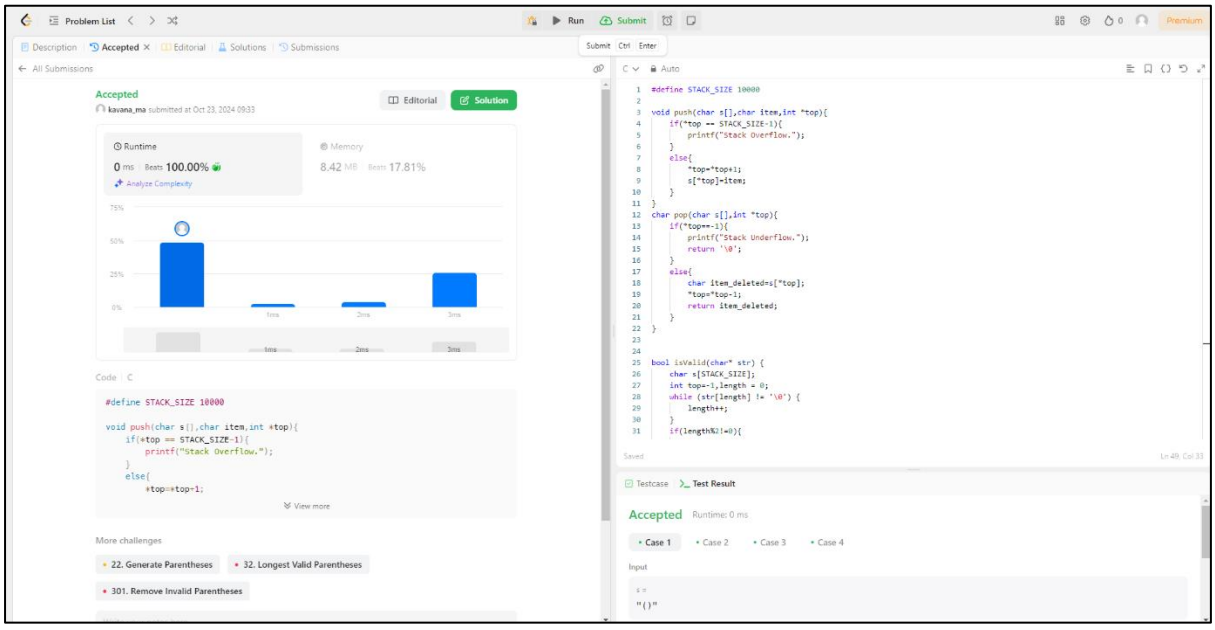
LeetCode 2: Valid Parentheses

```
#define STACK_SIZE 10000
```

```
void push(char s[],char item,int *top){
   if(*top == STACK_SIZE-1){
      printf("Stack Overflow.");
   }
   else{
      *top=*top+1;
      s[*top]=item;
   }
}
char pop(char s[],int *top){
   if(*top==-1){
      printf("Stack Underflow.");
      return '\0';
   }
   else{
      char item_deleted=s[*top];
      *top=*top-1;
      return item_deleted;
   }
}

bool isValid(char* str) {
   char s[STACK_SIZE];
   int top=-1,length = 0;
   while (str[length] != '\0') {
      length++;
   }
   if(length%2!=0){
      return false;
   }

   for(int i=0;i<length;i++){
      if(str[i]=='(' || str[i]=='[' || str[i]=='{'){
         push(s,str[i],&top);
      }
      else{
         if (top == -1) {
            return false;
         }
         if(str[i]==')'){
            if(s[top]=='('){
            pop(s,&top);}
            else{return false;}
         }
```

```
            else if(str[i]==']'){
                if(s[top]=='['){
                pop(s,&top);}
                else{return false;}
            }
            else{
                if(s[top]=='{'){
                pop(s,&top);}
                else{return false;}
            }
        }
    }
    return top == -1;
}
```

Output:



Lab program 4:

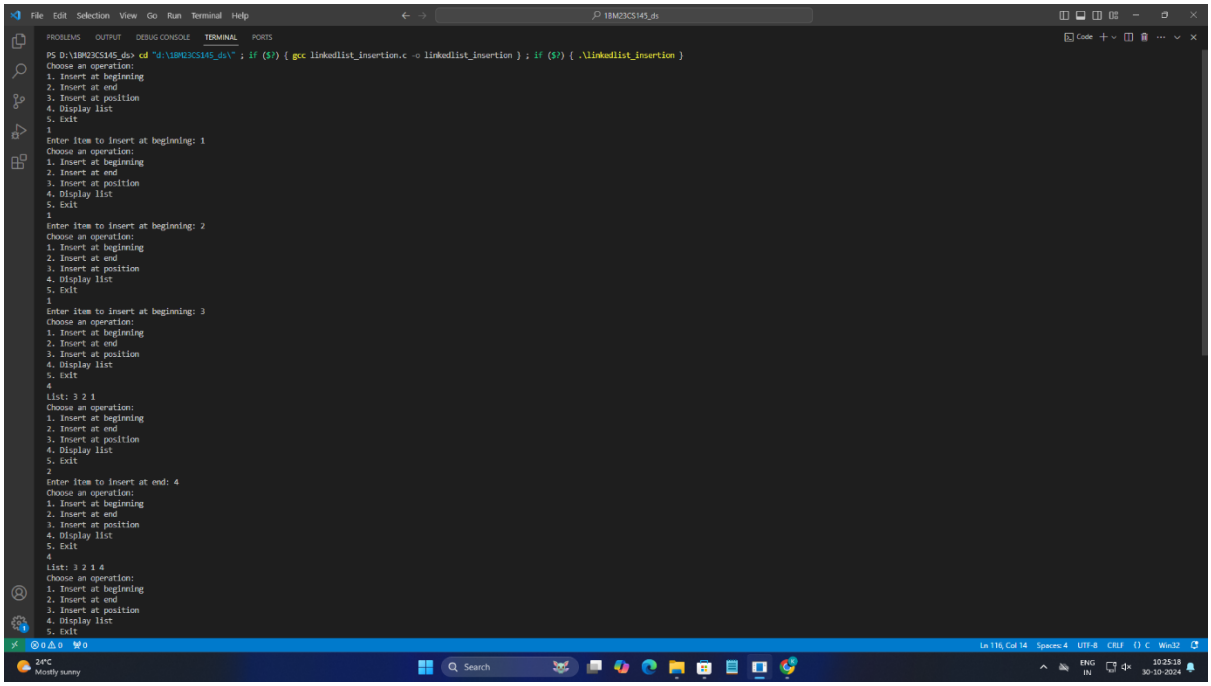
Program to Implement Singly Linked List (Insertion at first position, at any position and at end of list). Display the contents of the linked list.

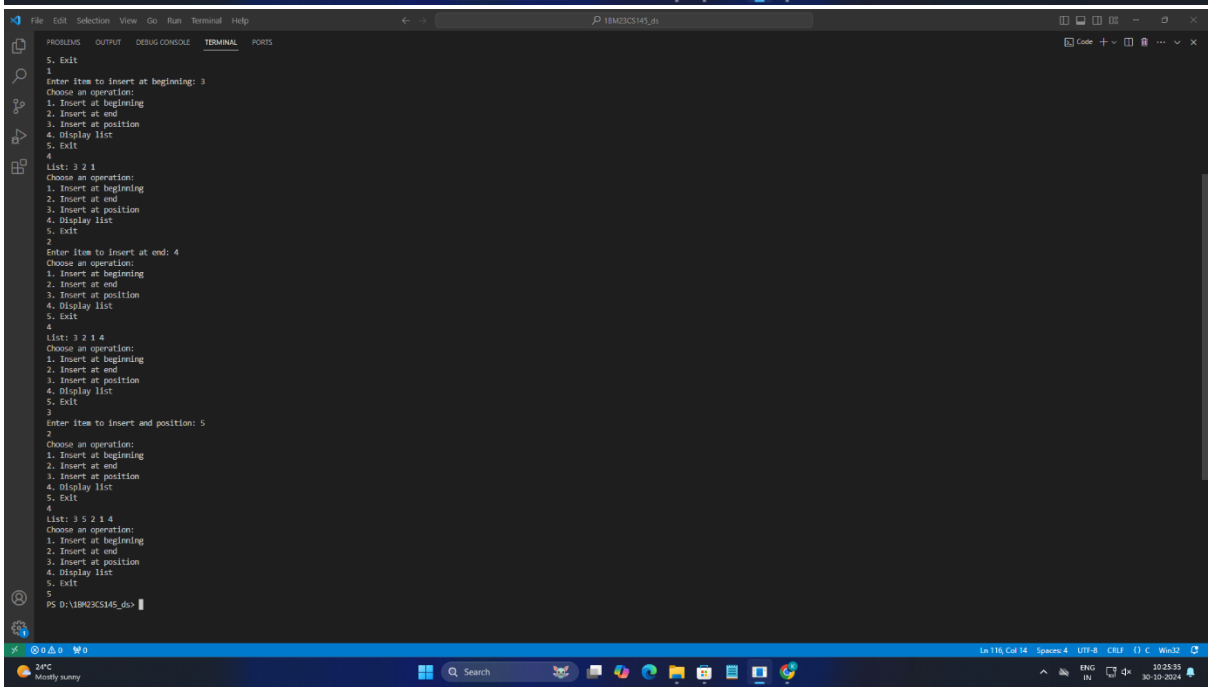
```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int value;
   struct node *next;
};
typedef struct node *NODE;

NODE get_node() {
   NODE ptr = (NODE)malloc(sizeof(struct node));
   if (ptr == NULL) {
      printf("Memory not allocated\n");
   }
   return ptr;
}

NODE insert_beginning(NODE first, int item) {
   NODE new_node = get_node();
   new_node->value = item;
   new_node->next = first;
   return new_node;
}

NODE insert_end(NODE first, int item) {
   NODE new_node = get_node();
   new_node->value = item;
   new_node->next = NULL;
   if (first == NULL) {
      return new_node;
   }
   NODE temp = first;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = new_node;
   return first;
}
```

```
NODE insert_pos(NODE first, int item, int pos) {
   NODE new_node = get_node();
   new_node->value = item;
   if (pos == 1) {
      new_node->next = first;
      return new_node;
   }
   int count = 1;
   NODE prev = NULL, current = first;
   while (count < pos && current != NULL) {
      prev = current;
      current = current->next;
      count++;
   }
   if (prev != NULL) {
      prev->next = new_node;
      new_node->next = current;
   } else {
      printf("Invalid position\n");
   }
   return first;
}

void display(NODE first) {
   NODE temp = first;
   if (first == NULL) {
      printf("Empty\n");
      return;
   }
   while (temp != NULL) {
      printf("%d ", temp->value);
      temp = temp->next;
   }
   printf("\n");
}

int main() {
   int item, pos, choice;
   NODE first = NULL;

   while (1) {
      printf("Choose an operation:\n");
      printf("1. Insert at beginning\n");
      printf("2. Insert at end\n");
```

```
        printf("3. Insert at position\n");
        printf("4. Display list\n");
        printf("5. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter item to insert at beginning: ");
                scanf("%d", &item);
                first = insert_beginning(first, item);
                break;
            case 2:
                printf("Enter item to insert at end: ");
                scanf("%d", &item);
                first = insert_end(first, item);
                break;
            case 3:
                printf("Enter item to insert and position: ");
                scanf("%d %d", &item, &pos);
                first = insert_pos(first, item, pos);
                break;
            case 4:
                printf("List: ");
                display(first);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output:


PS D:\18M23CS145_ds> cd "d:\18M23CS145_ds\" ; if ($?) { gcc linkedlist_insertion.c -o linkedlist_insertion } ; if ($?) { .\linkedlist_insertion }
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
1
Enter item to insert at beginning: 1
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
1
Enter item to insert at beginning: 2
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
1
Enter item to insert at beginning: 3
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
4
List: 3 2 1
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
2
Enter item to insert at end: 4
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
4
List: 3 2 1 4
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit


5. Exit
1
Enter item to insert at beginning: 3
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
4
List: 3 2 1
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
2
Enter item to insert at end: 4
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
4
List: 3 2 1 4
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
3
Enter item to insert and position: 5
2
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
4
List: 3 5 2 1 4
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display list
5. Exit
5
PS D:\18M23CS145_ds>

Lab Program 5:
Program to Implement Singly Linked List (Delete first element,Delete last element, Delete,Display)

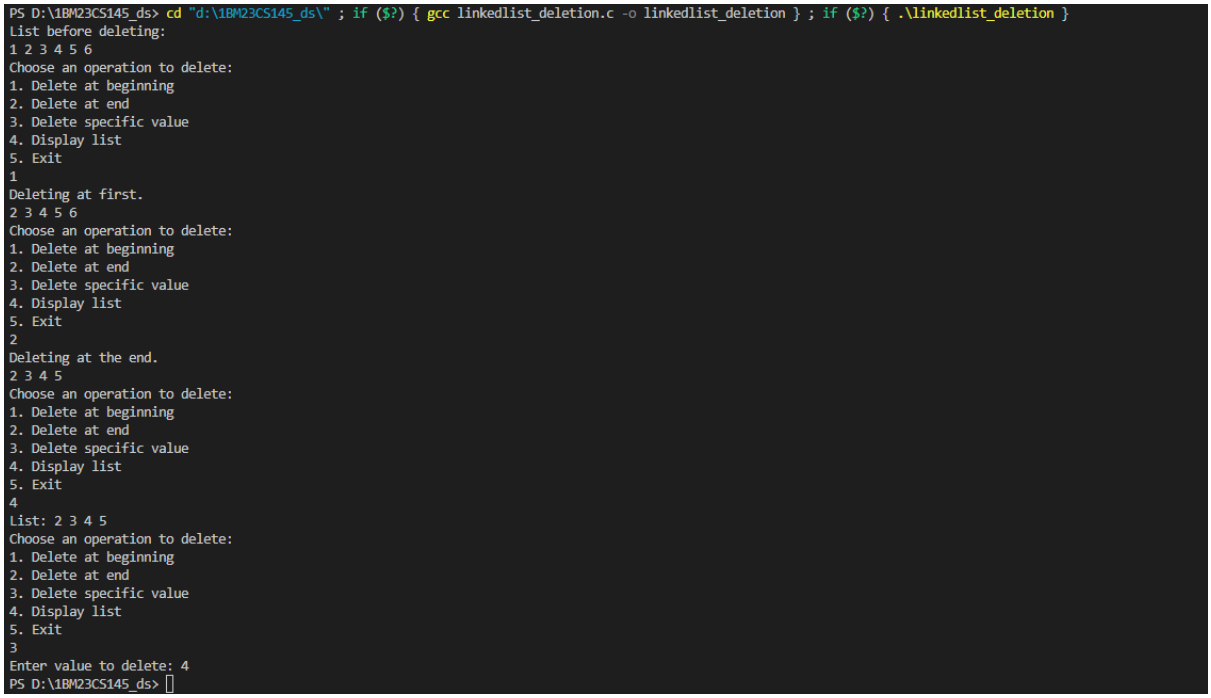
```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int value;
   struct node *next;
};
typedef struct node *NODE;

NODE get_node() {
   NODE ptr = (NODE)malloc(sizeof(struct node));
   if (ptr == NULL) {
      printf("Memory not allocated\n");
   }
   return ptr;
}

NODE delete_first(NODE first){
   NODE temp=first;
   if (first == NULL) {
      printf("Empty\n");
      return NULL;
   }
   first=first->next;
   free(temp);
   return first;
}

NODE delete_end(NODE first){
   if (first == NULL) {
      printf("Empty\n");
      return NULL;
   }
   NODE prev,last;
   prev=NULL;
   last=first;
   while(last->next!=NULL){
      prev=last;
      last=last->next;
   }
   prev->next=NULL;
```

```
    free(last);
    return first;
}

NODE delete_value(NODE first,int value){
    if (first == NULL) {
        printf("Empty\n");
        return NULL;
    }
    NODE prev,current;
    prev=NULL;
    current=first;
    while(value!=current->value || current->next!=NULL){
        prev=current;
        current=current->next;
    }
    if(current==NULL){
        printf("Value notfound");
        return first;
    }
    prev->next=current->next;
    free(current);
    return first;
}

NODE insert_beginning(NODE first, int item) {
    NODE new_node = get_node();
    new_node->value = item;
    new_node->next = first;
    return new_node;
}

void display(NODE first) {
    NODE temp = first;
    if (first == NULL) {
        printf("Empty\n");
        return;
    }
    while (temp != NULL) {
        printf("%d ", temp->value);
        temp = temp->next;
    }
    printf("\n");
}
```

```
int main() {
    int item, choice;
    NODE first = NULL;

    first = insert_beginning(first, 6);
    first = insert_beginning(first, 5);
    first = insert_beginning(first, 4);
    first = insert_beginning(first, 3);
    first = insert_beginning(first, 2);
    first = insert_beginning(first, 1);

    printf("List before deleting:\n");
    display(first);

    while (1) {
        printf("Choose an operation to delete:\n");
        printf("1. Delete at beginning\n");
        printf("2. Delete at end\n");
        printf("3. Delete specific value\n");
        printf("4. Display list\n");
        printf("5. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Deleting at first.\n");
                first = delete_first(first);
                display(first);
                break;
            case 2:
                printf("Deleting at the end.\n");
                first = delete_end(first);
                display(first);
                break;
            case 3:
                printf("Enter value to delete: ");
                scanf("%d", &item);
                first = delete_value(first, item);
                display(first);
                break;
            case 4:
                printf("List: ");
                display(first);
```

```
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}
```

Output:



Lab Program 6:

a) **Linked list operations (Sort, Reverse, Concatenate)**
b) **Implementation of Queue and Stack using Linked List**

a)

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int value;
   struct node *next;
};
typedef struct node *NODE;
NODE get_node() {
   NODE ptr = (NODE)malloc(sizeof(struct node));
   if (ptr == NULL) {
      printf("Memory not allocated\n");
   }
   return ptr;
}
NODE insert_beginning(NODE first, int item) {
   NODE new_node = get_node();
   new_node->value = item;
   new_node->next = first;
   return new_node;
}
NODE insert_end(NODE first, int item) {
   NODE new_node = get_node();
   new_node->value = item;
   new_node->next = NULL;
   if (first == NULL) {
      return new_node;
   }
   NODE temp = first;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = new_node;
   return first;
}
NODE delete_first(NODE first){
   NODE temp=first;
   if (first == NULL) {
      printf("Empty\n");
      return NULL;
```

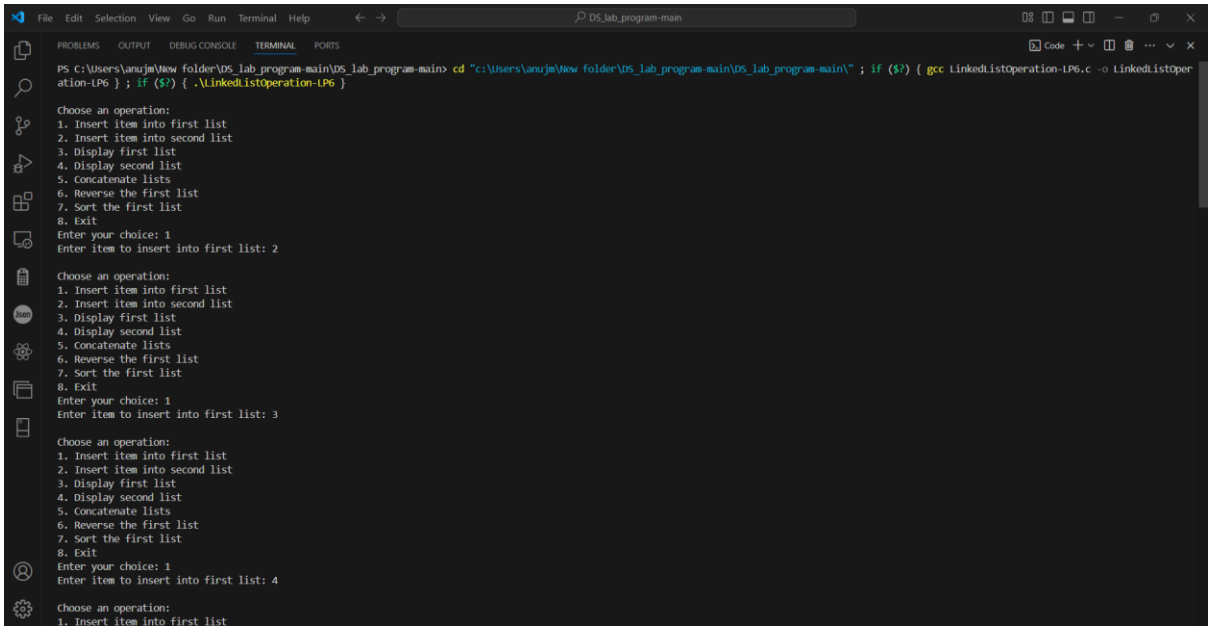
```
    }
    first=first->next;
    free(temp);
    return first;
}
NODE delete_end(NODE first){
    if (first == NULL) {
        printf("Empty\n");
        return NULL;
    }
    NODE prev,last;
    prev=NULL;
    last=first;
    while(last->next!=NULL){
        prev=last;
        last=last->next;
    }
    prev->next=NULL;
    free(last);
    return first;
}
void display(NODE first) {
    NODE temp = first;
    if (first == NULL) {
        printf("Empty\n");
        return;
    }
    while (temp != NULL) {
        printf("%d ", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

NODE concatenate(NODE first1, NODE first2){
    if(first1 == NULL && first2==NULL){ return NULL;}
    if(first1==NULL){ return first2; }
    if(first2==NULL){ return first1; }
    NODE temp=first1;
    while(temp->next!= NULL){
        temp=temp->next;
    }
    temp->next=first2;
    return first1;
```

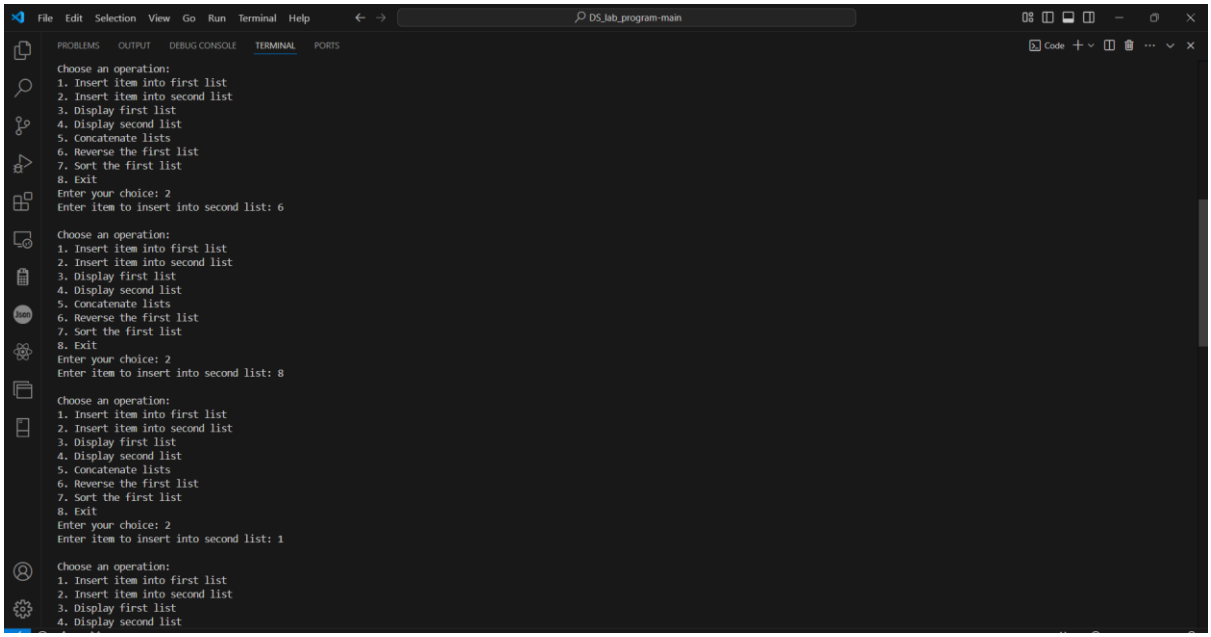
```
}

NODE reverse(NODE first){
   if(first==NULL){
      return NULL;
   }
   NODE curr=NULL,temp;
   while(first!=NULL){
      temp=first;
      first=first->next;
      temp->next=curr;
      curr=temp;
   }
   return curr;
}

void sort(NODE first){
   NODE temp1=first,temp2=first->next;
   while((temp1->next)!=NULL){
      while(temp2!=NULL){
         if(temp1->value >= temp2->value){
            int x=temp1->value;
            temp1->value=temp2->value;
            temp2->value=x;
         }
      temp2=temp2->next;
      }
   temp1=temp1->next;
   }
}

int main() {
   int choice, item;
   NODE first1 = NULL, first2 = NULL, mergedList = NULL;
   NODE ConcatenatedList;

   // Menu to choose operations
   while (1) {
      printf("\nChoose an operation:\n");
      printf("1. Insert item into first list\n");
      printf("2. Insert item into second list\n");
      printf("3. Display first list\n");
      printf("4. Display second list\n");
      printf("5. Concatenate lists\n");
```

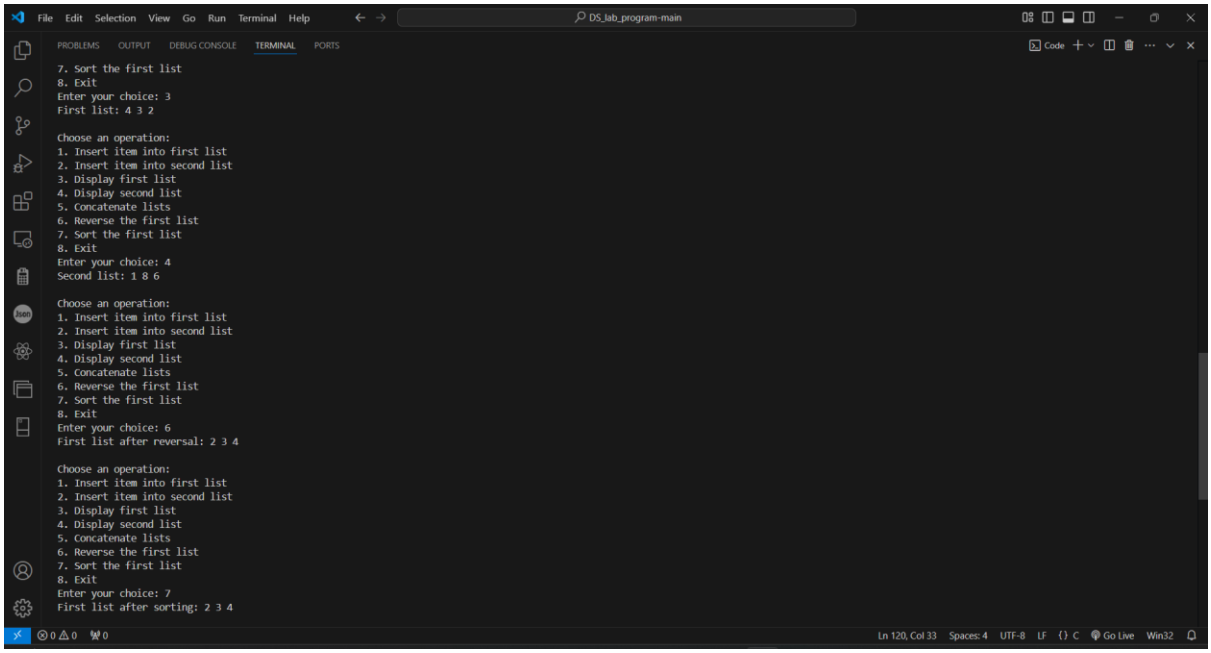
```
printf("6. Reverse the first list\n");
printf("7. Sort the first list\n");
printf("8. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
   case 1:
      printf("Enter item to insert into first list: ");
      scanf("%d", &item);
      first1 = insert_beginning(first1, item);
      break;
   case 2:
      printf("Enter item to insert into second list: ");
      scanf("%d", &item);
      first2 = insert_beginning(first2, item);
      break;
   case 3:
      printf("First list: ");
      display(first1);
      break;
   case 4:
      printf("Second list: ");
      display(first2);
      break;
   case 5:
      ConcatenatedList = concatenate(first1, first2);
      printf(" Concatenated List: ");
      display( ConcatenatedList);
      break;
   case 6:
      first1 = reverse(first1);
      printf("First list after reversal: ");
      display(first1);
      break;
   case 7:
      sort(first1);
      printf("First list after sorting: ");
      display(first1);
      break;
   case 8:
      exit(0);
   default:
      printf("Invalid choice..\n");
```

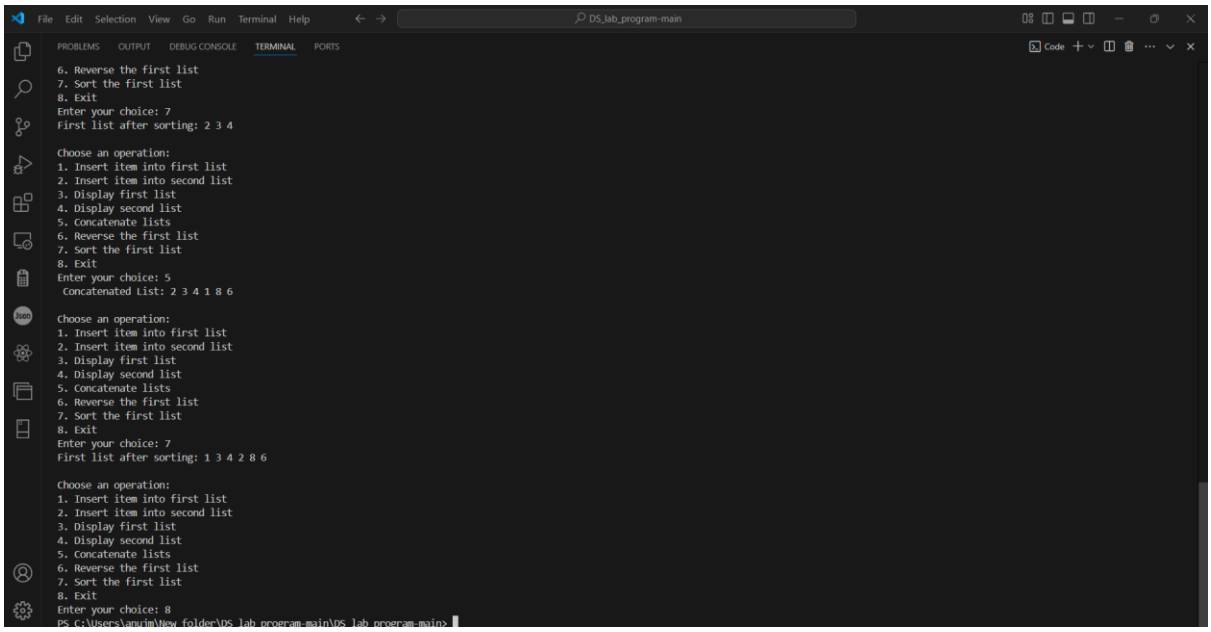
```
        }
    }

    return 0;
}
```

Output:









b)

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

struct node {
    int value;
    struct node *next;
};
typedef struct node *NODE;

NODE get_node() {
    NODE ptr = (NODE)malloc(sizeof(struct node));
```

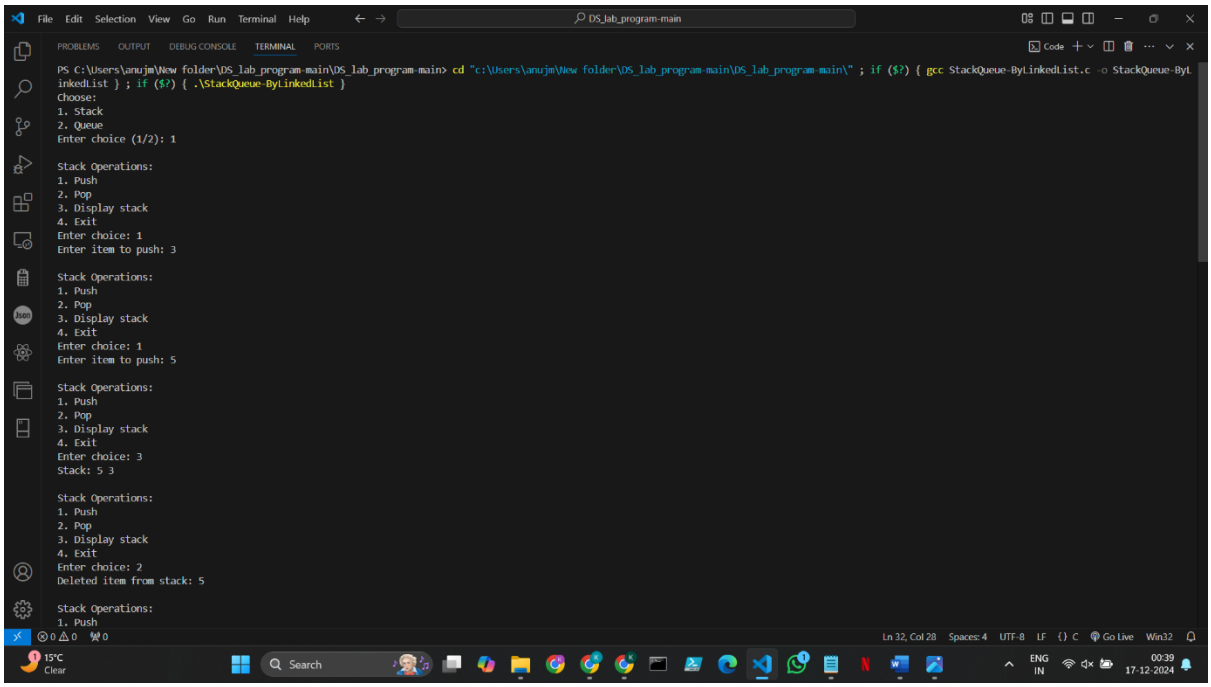
```
    if (ptr == NULL) {
        printf("Memory not allocated\n");
    }
    return ptr;
}

NODE delete_first(NODE first){
    NODE temp=first;
    if (first == NULL) {
        printf("Empty\n");
        return NULL;
    }
    first=first->next;
    free(temp);
    return first;
}

NODE insert_beginning(NODE first, int item) {
    NODE new_node = get_node();
    new_node->value = item;
    new_node->next = first;
    return new_node;
}

NODE insert_end(NODE first, int item) {
    NODE new_node = get_node();
    new_node->value = item;
    new_node->next = NULL;
    if (first == NULL) {
        return new_node;
    }
    NODE temp = first;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = new_node;
    return first;
}

void display(NODE first) {
    NODE temp = first;
    if (first == NULL) {
        printf("Empty\n");
        return;
```

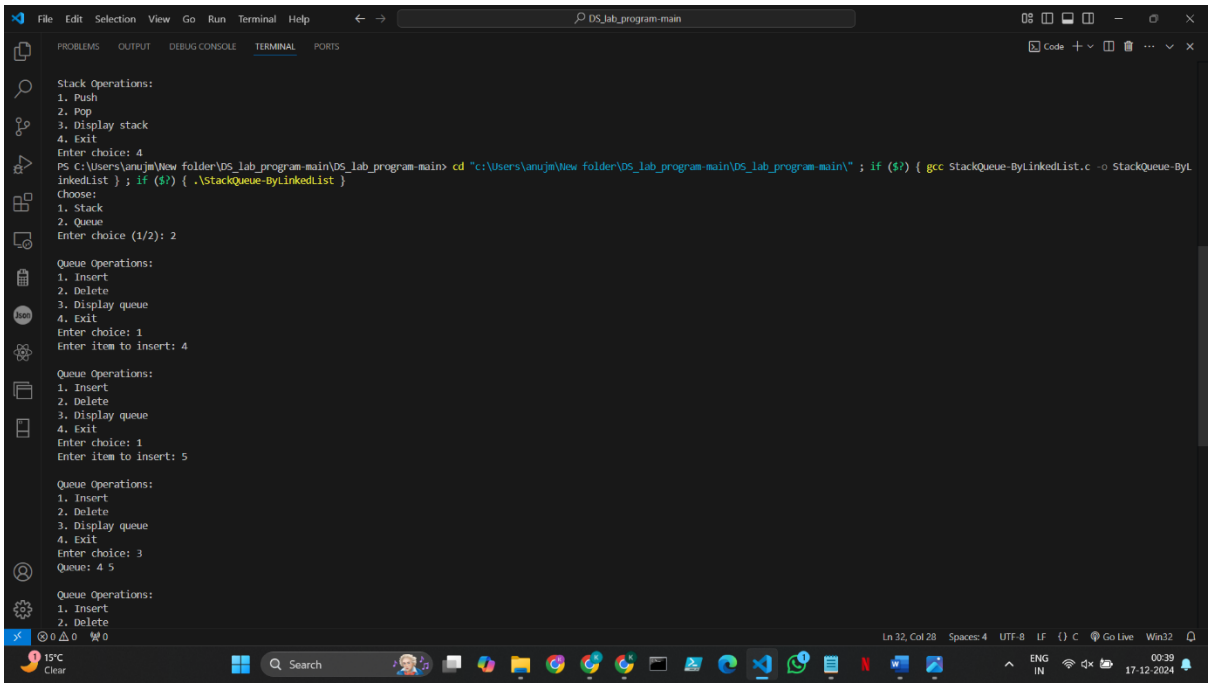
```
    }
    while (temp != NULL) {
        printf("%d ", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int item, choice, deleted_item;
    NODE first = NULL;

    printf("Choose:\n");
    printf("1. Stack\n");
    printf("2. Queue\n");
    printf("Enter choice (1/2): ");
    scanf("%d", &choice);

    if (choice == 1) {
        while (1) {
            printf("\nStack Operations:\n");
            printf("1. Push\n");
            printf("2. Pop\n");
            printf("3. Display stack\n");
            printf("4. Exit\n");
            printf("Enter choice: ");
            scanf("%d", &choice);

            switch (choice) {
                case 1:
                    printf("Enter item to push: ");
                    scanf("%d", &item);
                    first = insert_beginning(first, item);
                    break;
                case 2:
                    if (first != NULL) {
                        deleted_item = first->value;
                        first = delete_first(first);
                        printf("Deleted item from stack: %d\n", deleted_item);
                    } else {
                        printf("Stack is empty\n");
                    }
                    break;
                case 3:
```

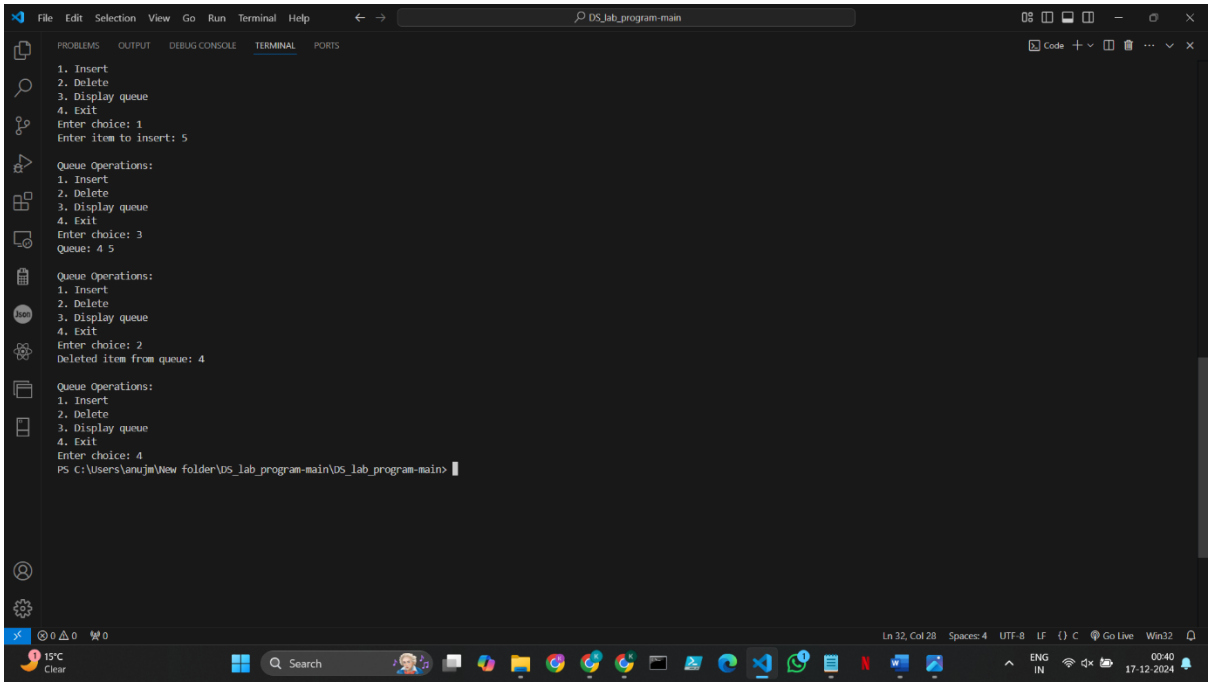
```
                printf("Stack: ");
                display(first);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }
}
else if (choice == 2) {
    while (1) {
        printf("\nQueue Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display queue\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter item to insert: ");
                scanf("%d", &item);
                first = insert_end(first, item);
                break;
            case 2:
                if (first != NULL) {
                    deleted_item = first->value;
                    first = delete_first(first);
                    printf("Deleted item from queue: %d\n", deleted_item);
                } else {
                    printf("Queue is empty!\n");
                }
                break;
            case 3:
                printf("Queue: ");
                display(first);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
```

```
    }
}
else {
    printf("Invalid operation.\n");
}

return 0;
}
```

Output:






File Edit Selection View Go Run Terminal Help
DS_lab_program-main
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Code
1. Insert
2. Delete
3. Display queue
4. Exit
Enter choice: 1
Enter item to insert: 5
Queue Operations:
1. Insert
2. Delete
3. Display queue
4. Exit
Enter choice: 3
Queue: 4 5
Queue Operations:
1. Insert
2. Delete
3. Display queue
4. Exit
Enter choice: 2
Deleted item from queue: 4
Queue Operations:
1. Insert
2. Delete
3. Display queue
4. Exit
Enter choice: 4
PS C:\Users\anujm\New folder\DS_lab_program-main\DS_lab_program-main>
Ln 32, Col 28 Spaces: 4 UTF-8 LF C Go Live Win32
15°C Clear
Search
ENG IN
00:40 17-12-2024

Lab Program 7:

a) **Program to Implement doubly link list**
b) **LeetCode 3: Finding middle element of a linked list**

a)

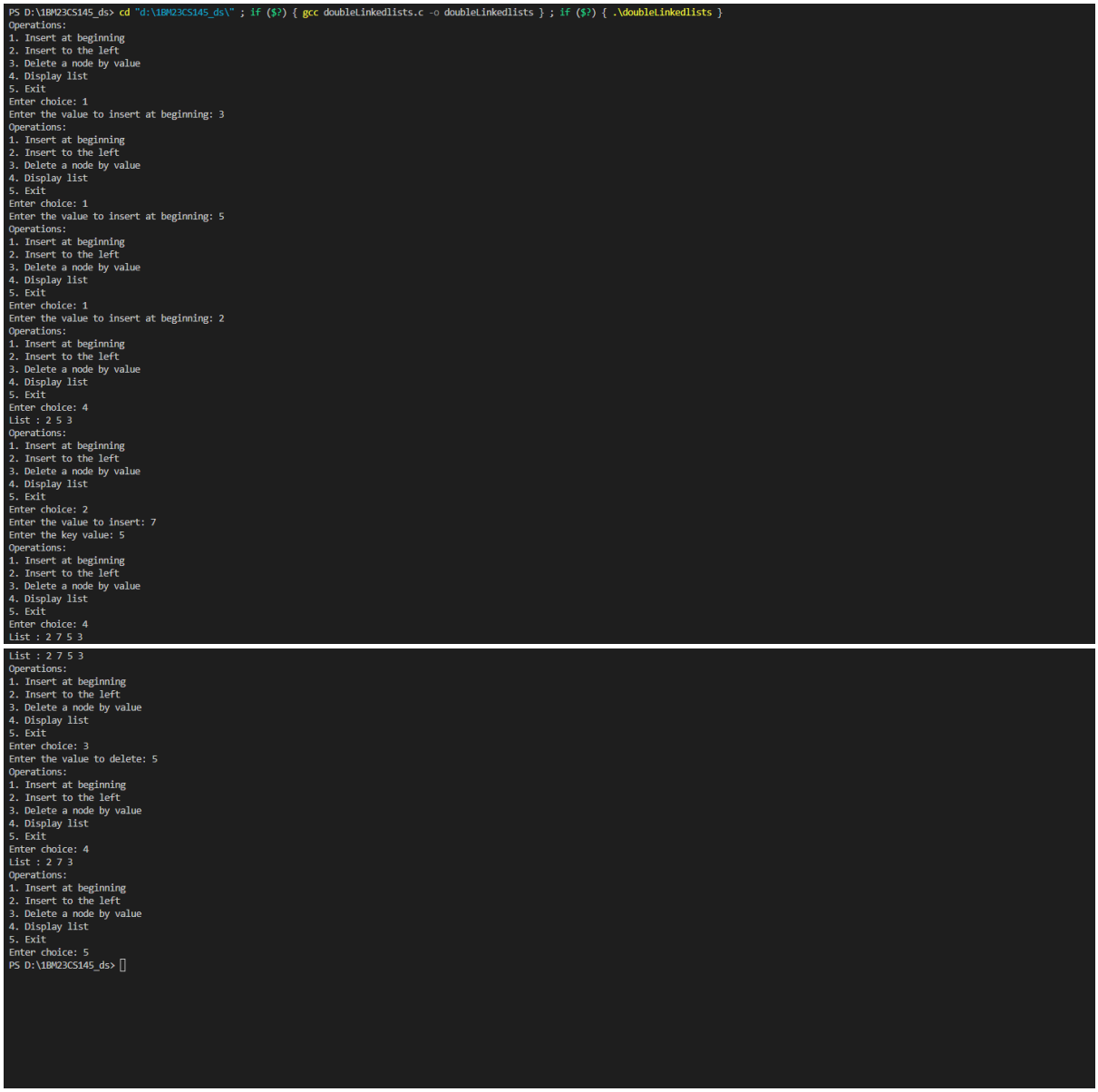
```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int value;
   struct Node* prev;
   struct Node* next;
};
typedef struct Node* NODE;

NODE get_node() {
   NODE ptr = (NODE)malloc(sizeof(struct Node));
   if (ptr == NULL) {
      printf("Memory not allocated\n");
   }
   return ptr;
}

NODE insert_beginning(NODE first, int item) {
   NODE new_node = get_node();
   new_node->value = item;
   new_node->next = first;
   new_node->prev = NULL;
   if (first != NULL) {
      first->prev = new_node;
   }
   return new_node;
}

NODE insert_left_value(NODE first, int item, int key) {
   NODE curr = first;
   NODE new_node = get_node();
   new_node->value = item;

    if (curr == NULL) {
      printf("value not found.\n");
      return first;
   }
```

```
    while (curr != NULL && curr->value != key) {
        curr = curr->next;
    }

    new_node->next = curr;
    new_node->prev = curr->prev;
    (curr->prev)->next = new_node;
    curr->prev = new_node;
}

NODE deleteNode(NODE first, int value) {
    NODE curr = first;
    if (curr == NULL) {
        printf("Value %d not found.\n", value);
        return first;
    }

    while (curr != NULL && curr->value != value) {
        curr = curr->next;
    }
    (curr->prev)->next = curr->next;
    (curr->next)->prev = curr->prev;
    free(curr);
    return first;
}

void displayList(NODE first) {
    if (first == NULL) {
        printf("List is empty.\n");
        return;
    }

    NODE curr = first;
    while (curr != NULL) {
        printf("%d ", curr->value);
        curr = curr->next;
    }
    printf("\n");
}

int main() {
    NODE first = NULL;
    int choice, value, key;
```

```
do {
    printf("Operations:\n");
    printf("1. Insert at beginning\n");
    printf("2. Insert to the left \n");
    printf("3. Delete a node by value\n");
    printf("4. Display list\n");
    printf("5. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the value to insert at beginning: ");
            scanf("%d", &value);
            first = insert_beginning(first, value);
            break;

        case 2:
            printf("Enter the value to insert: ");
            scanf("%d", &value);
            printf("Enter the key value: ");
            scanf("%d", &key);
            insert_left_value(first, value, key);
            break;

        case 3:
            printf("Enter the value to delete: ");
            scanf("%d", &value);
            first = deleteNode(first, value);
            break;

        case 4:
            printf("List : ");
            displayList(first);
            break;

        case 5:
            break;

        default:
            printf("Invalid choice\n");
    }
} while (choice != 5);
```

```
    return 0;
}
```

Output:



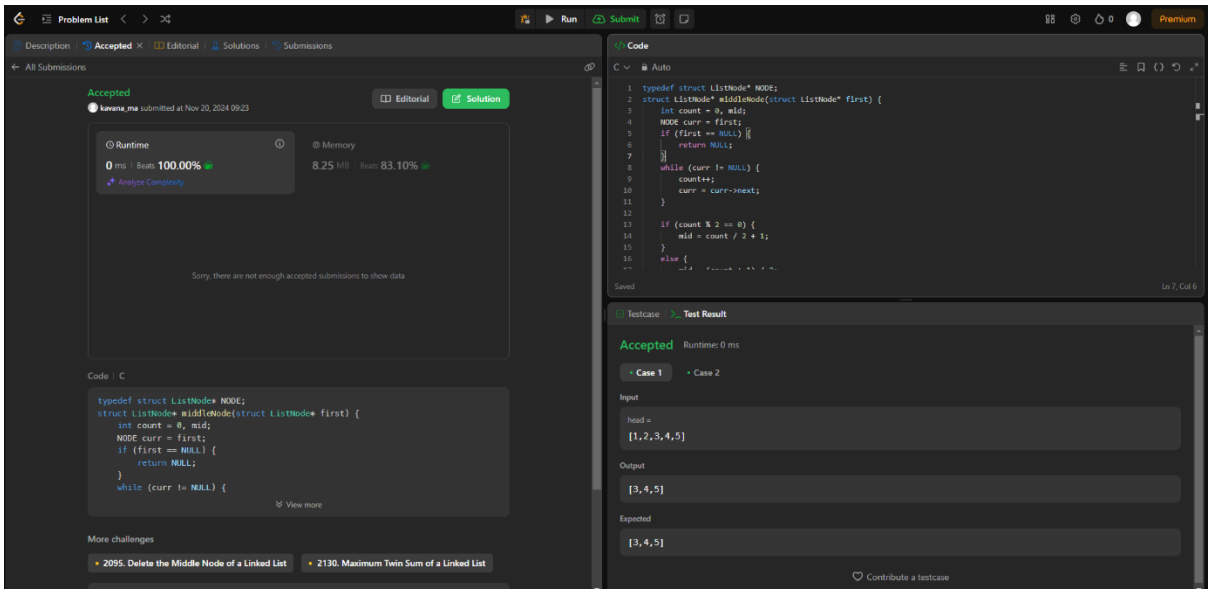
```
PS D:\1BM23CS145_ds> cd "d:\1BM23CS145_ds\" ; if ($?) { gcc doubleLinkedlists.c -o doubleLinkedlists } ; if ($?) { .\doubleLinkedlists }
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 1
Enter the value to insert at beginning: 3
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 1
Enter the value to insert at beginning: 5
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 1
Enter the value to insert at beginning: 2
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 4
List : 2 5 3
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 2
Enter the value to insert: 7
Enter the key value: 5
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 4
List : 2 7 5 3
List : 2 7 5 3
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 3
Enter the value to delete: 5
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 4
List : 2 7 3
Operations:
1. Insert at beginning
2. Insert to the left
3. Delete a node by value
4. Display list
5. Exit
Enter choice: 5
PS D:\1BM23CS145_ds>
```

b)

```
typedef struct ListNode* NODE;
struct ListNode* middleNode(struct ListNode* first) {
    int count = 0, mid;
    NODE curr = first;
    if (first == NULL) {
        return NULL;
    }
    while (curr != NULL) {
        count++;
```

```
        curr = curr->next;
    }

    if (count % 2 == 0) {
        mid = count / 2 + 1;
    }
    else {
        mid = (count + 1) / 2;
    }

    curr = first;
    count = 1;
    while (curr != NULL && count < mid) {
        curr = curr->next;
        count++;
    }
    return curr;
}
```

Output:



Lab Program 8:

a) **Program to Implement Binary Search Tree**
b) **LeetCode 4: Intersection of Two Linked List**

a)

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

struct Node {
   int value;
   struct Node *left, *right;
};
typedef struct Node *NODE;

NODE newNode(int key) {
   NODE node = (NODE)malloc(sizeof(struct Node));
   node->value = key;
   node->left = node->right = NULL;
   return node;
}

NODE insert(NODE root, int key) {
   if (root == NULL) {
      return newNode(key);
   }

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

   return root;
}

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

void preorder(NODE root) {
```

```
    if (root != NULL) {
        printf("%d ", root->value);
        preorder(root->left);
        preorder(root->right);
    }
}

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

int main() {
    NODE root = NULL;
    int choice, value;

    do {
        printf("\nBST Operations\n");
        printf("1. Insert a node\n");
        printf("2. Display: In-order\n");
        printf("3. Display: Pre-order\n");
        printf("4. Display: Post-order\n");
        printf("5. Exit\n");

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

        switch(choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                printf("Value inserted:%d\n",value);
                break;
            case 2:
                if (root == NULL) {
                    printf("The tree is empty!\n");
                } else {
                    inorder(root);
                }
                break;
```

```
            case 3:
                if (root == NULL) {
                    printf("The tree is empty!\n");
                } else {
                    preorder(root);
                }
                break;
            case 4:
                if (root == NULL) {
                    printf("The tree is empty!\n");
                } else {
                    postorder(root);
                }
                break;
            case 5:
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 5);

    return 0;
}
```

Output:

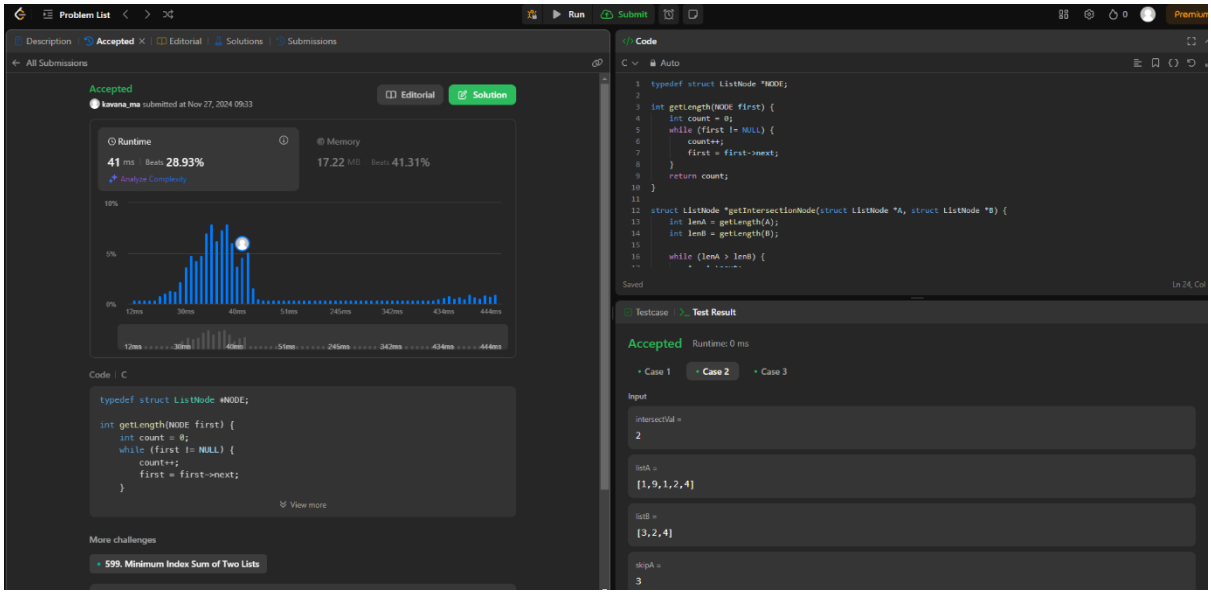
```
PS D:\1BM23CS145_ds> cd "d:\1BM23CS145_ds\" ; if ($?) { gcc BinarySearchTree.c -o BinarySearchTree } ; if ($?) { .\BinarySearchTree }

BST Operations
1. Insert a node
2. Display: In-order
3. Display: Pre-order
4. Display: Post-order
5. Exit
Enter your choice: 1
Enter the value to insert: 6
Value inserted:6

BST Operations
1. Insert a node
2. Display: In-order
3. Display: Pre-order
4. Display: Post-order
5. Exit
Enter your choice: 1
Enter the value to insert: 8
Value inserted:8

BST Operations
1. Insert a node
2. Display: In-order
3. Display: Pre-order
4. Display: Post-order
5. Exit
Enter your choice: 1
Enter the value to insert: 5
Value inserted:5

BST Operations
1. Insert a node
2. Display: In-order
3. Display: Pre-order
4. Display: Post-order
5. Exit
Enter your choice: 1
Enter the value to insert: 7
Value inserted:7

BST Operations
1. Insert a node
2. Display: In-order
3. Display: Pre-order
4. Display: Post-order
5. Exit
Enter your choice: 1
Enter the value to insert: 9
Value inserted:9

BST Operations
1. Insert a node
```

b)

```
typedef struct ListNode *NODE;

int getLength(NODE first) {
    int count = 0;
    while (first != NULL) {
        count++;
        first = first->next;
    }
    return count;
}

struct ListNode *getIntersectionNode(struct ListNode *A, struct ListNode *B) {
    int lenA = getLength(A);
    int lenB = getLength(B);

    while (lenA > lenB) {
        A = A->next;
        lenA--;
    }

    while (lenB > lenA) {
        B = B->next;
        lenB--;
    }
```

```
    while (A != NULL && B != NULL) {
        if (A == B) {
            return A;
        }
        A = A->next;
        B = B->next;
    }

    return NULL;
}
```

Output:



Lab Program 9:

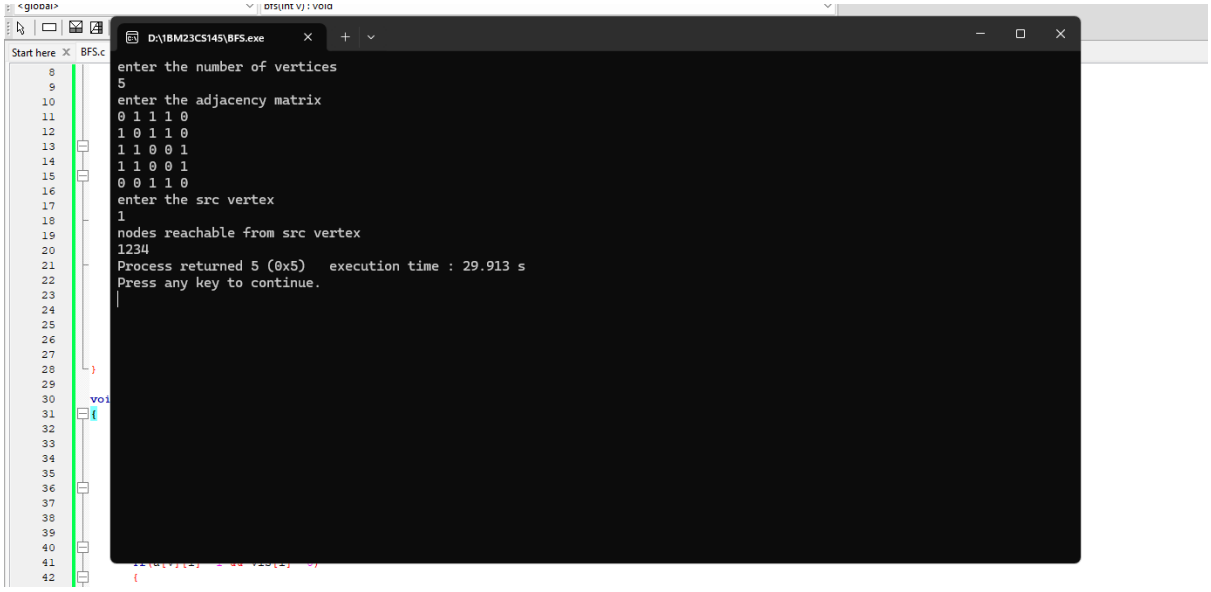
a) **BFS traversal**
b) **DFS traversal and other functions**

a)

```
#include<stdio.h>
void bfs(int);
int a[10][10],vis[10],n;

void main()
{
  int i,j,src;

  printf("enter the number of vertices\n");
  scanf("%d",&n);
  printf("enter the adjacency matrix\n");
  for(i=1;i<=n;i++)
  {
    for(j=1;j<=n;j++)
    {
   scanf("%d",&a[i][j]);

    }

    vis[i]=0;
  }

  printf("enter the src vertex\n");
  scanf("%d",&src);
  printf("nodes reachable from src vertex\n");
  bfs(src);

}

void bfs(int v)
{
   int q[10],f=1,r=1,u,i;
   q[r]=v;
   vis[v]=1;
   while(f<=r)
   {
     u=q[f];
     printf("%d",u);
     for(i=1;i<=n;i++)
```

```
    {
  if(a[v][i]==1 && vis[i]==0)
  {
    vis[i]=1;
    r=r+1;
    q[r]=i;
  }
    }
   f=f+1;
  }
}
```

Output:

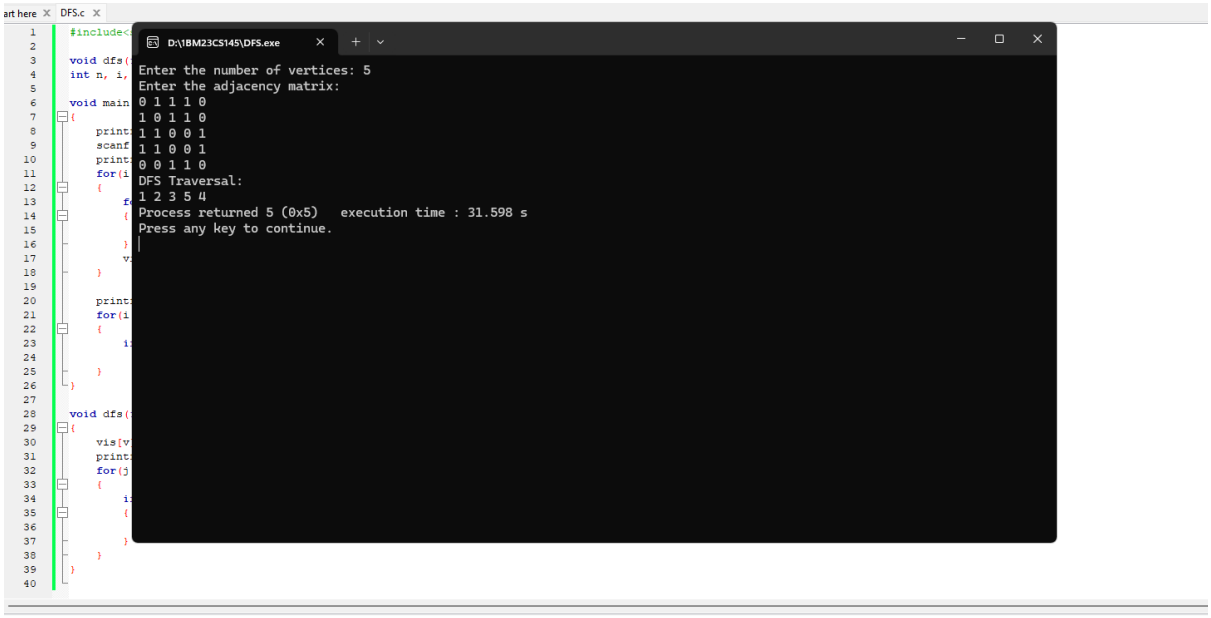


b)

```
#include<stdio.h>

void dfs(int);
int n, i, j, a[10][10], vis[10];

void main()
{
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   printf("Enter the adjacency matrix:\n");
   for(i = 0; i < n; i++)  // 0-based indexing
   {
      for(j = 0; j < n; j++)
      {
         scanf("%d", &a[i][j]);
```

```
        }
        vis[i] = 0;  // Initialize visited array
    }

    printf("DFS Traversal:\n");
    for(i = 0; i < n; i++)  // 0-based indexing
    {
        if(vis[i] == 0)
            dfs(i);
    }
}

void dfs(int v)
{
    vis[v] = 1;
    printf("%d ", v + 1);  // Print vertices as 1-based indexing
    for(j = 0; j < n; j++)  // 0-based indexing
    {
        if(a[v][j] == 1 && vis[j] == 0)
        {
            dfs(j);
        }
    }
}
```

Output:



Lab Program 10:
Design and develop a Program in C that uses Hash function H: K-> 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>
#define MAX_RECORDS 100
void displayHashTable(int HT[], int m) {
printf("\nHash Table:\n");
for (int i = 0; i < m; i++) {
if (HT[i] != -1) {
printf("Index %d: %d\n", i, HT[i]);
} else {
printf("Index %d: Empty\n", i);
}
}
}
int hashFunction(int key, int m) {
return key % m;
}
void insert(int HT[], int key, int m) {
int hash = hashFunction(key, m);
int i = 0;
while (HT[(hash + i) % m] != -1) {
i++;
if (i == m) {
printf("Hash table is full. Unable to insert key %d\n", key);
return;
}
}

HT[(hash + i) % m] = key;
printf("Inserted key %d at index %d\n", key, (hash + i) % m);
}
int main() {
int m, n;
int HT[MAX_RECORDS];
for (int i = 0; i < MAX_RECORDS; i++) {
HT[i] = -1;
}

printf("Enter the number of memory locations in the hash table (m): ");
scanf("%d", &m);
```

```
printf("Enter the number of employee records: ");
scanf("%d", &n);
for (int i = 0; i < n; i++) {
int key;
printf("Enter the 4-digit key for employee record %d: ", i + 1);
scanf("%d", &key);
if (key < 1000 || key > 9999) {
printf("Invalid key! Please enter a 4-digit number.\n");
i--;
} else {
insert(HT, key, m);
}
}
displayHashTable(HT, m);
return 0;
}
```

Output: