VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On
DATA STRUCTURES (23CS3PCDST)

Submitted by

Aisha Ali Nyaz (1BM23CS017)

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 Aisha Ali Nyaz **(1BM23CS017)**, 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.

Dr. Rajeshwari B S
Associate 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: Stack Operations

Write a program to simulate the working of stack using an array with the following:

i. **Push**
ii. **Pop**
iii. **Display**

The program should print appropriate messages for stack overflow, stack underflow

```
#include<stdio.h>
#include<stdlib.h>
int stack[10],top=-1,i,item;
#define max 9

void push(){
   if(top==max-1){
      printf("Stack Overflow\n");
   }
   else{
      top++;
      printf("Enter Element to Push: ");
      scanf("%d",&item);
      stack[top]=item;
   }
}

int pop(){
   if (top==-1){
      printf("Stack Underflow\n");
      return -1;
   }
   item=stack[top];
   top=top-1;
   return (item);
}

void display(){
   if (top==-1){
      printf("Stack Empty\n");
   }
   else{
      printf("The Stack is: \n");
      for(i=top;i>-1;i--){
         printf("%d\n",stack[i]);
      }
   }
```

```
}

void main(){
   while(1){
      int userInput;
      printf("Enter \n1 to Push,\n2 to Pop,\n3 to Display, and \n4 to Exit\n");
      scanf("%d",&userInput);
      switch(userInput){
         case 1: push();
            break;
         case 2: item=pop();
            if(item!=-1){
               printf("The Popped Element is: %d \n",item);
            }
            break;
         case 3: display();
            break;
         case 4: exit(0);
            break;
      }
   }
}
```

Output:

```
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 1
Enter Element to Push: 18
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 1
Enter Element to Push: 45
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 1
Enter Element to Push: 7
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 1
Stack Overflow
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 3
The Stack is:
7
45
18
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 2
The Popped Element is: 7
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 2
The Popped Element is: 45
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 2
The Popped Element is: 18
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 2
Stack Underflow
Enter (1) to Push,(2) to Pop, (3) to Display, and (4) to Exit: 4
```

Lab 2: Infix to Postfix

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 i = 0, pos = 0, top = -1, length;
char symbol, temp, infix[20], postfix[20], stack[20];

void infixtopostfix();
void push(char symbol);
char pop();
int pred(char symb);

int main()
{
  printf("Enter infix expression:\n");
  scanf("%s", infix);

  infixtopostfix();
  printf("\nInfix expression:\n%s", infix);
  printf("\nPostfix expression:\n%s", postfix);
  return 0;
}

void infixtopostfix() {
  length = strlen(infix);
  push('#');
  while (i < length) {
```

```
    symbol = infix[i];
    switch (symbol) {
        case '(':
            push(symbol);
            break;

        case ')':
            temp = pop();
            while (temp != '(') {
                postfix[pos++] = temp;
                temp = pop();
            }
            break;

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

        default:
            postfix[pos++] = symbol;
    }
    i++;
}
```

```
    while (top > 0) {
        temp = pop();
        postfix[pos++] = temp;
    }
    postfix[pos] = '\0';
}

void push(char symbol) {
    top = top + 1;
    stack[top] = symbol;
}

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

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;

    default:
        p = -1;
        break;
  }
  return p;
}
```

Output:

```
Enter infix expression:
A^B*C-D+E/F/(G+H)

Infix expression:
A^B*C-D+E/F/(G+H)
Postfix expression:
AB^C*D-EF/GH+/+
```

Leetcode

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

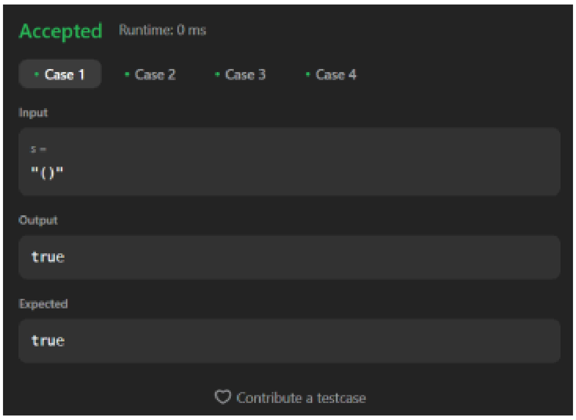
An input string is valid if:

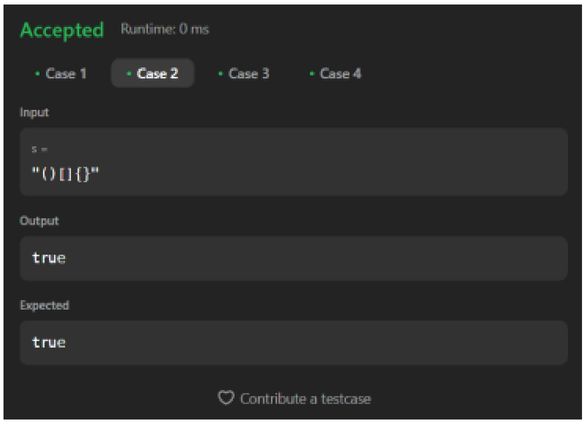
i. **Open brackets must be closed by the same type of brackets.**
ii. **Open brackets must be closed in the correct order.**
iii. **Every close bracket has a corresponding open bracket of the same type.**

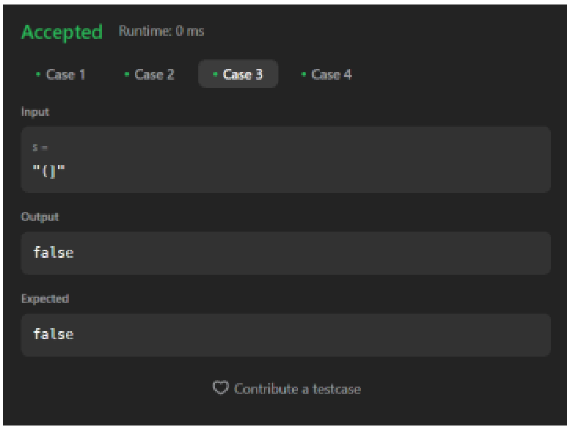
```
#include<string.h>

bool isValid(char* s) {
   int len = strlen(s);
   char stack[len];
   int top = -1;

   for(int i = 0; i < len; i++) {
      if(s[i] == '(' || s[i] == '{' || s[i] == '[') {
         top++;
         stack[top] = s[i];
      } else {
         if (top == -1) {
            return false;
         }
         if((s[i] == ')' && stack[top] == '(') ||
           (s[i] == '}' && stack[top] == '{') ||
           (s[i] == ']' && stack[top] == '[')) {
            top--;
         } else {
            return false;
         }
      }
```

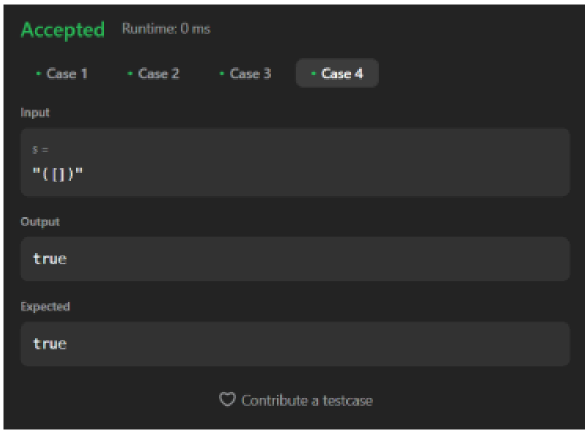
```
    }
    return top == -1;
}
```

Output:









Lab 3: Linear Queue

WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete and Display.

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

```
#include <stdio.h>

#include <stdlib.h>

#define max 3


int front=-1, rear=-1, i, queue[10], ch, item;


void insert();

int del();

void display();


void main()

{

  while (1)

  {

        printf("\n1. INSERT \n2. DELETE \n3. DISPLAY \n4. EXIT \nEnter Your Choice: ");

      scanf("%d",&ch);

      switch (ch)

      {

          case 1: insert();

                break;

          case 2: item=del();

                if (item!=-1)

                {

                    printf("The Deleted Item is:%d\n", item);

                } break;
```

```
            case 3: display();
                  break;
            case 4: exit(0);
        }
    }
}

void insert()
{
    if (rear==max-1)
    {
        printf("Queue is Full \n");
        return;
    }
    printf("Enter Element: \n");
    scanf("%d",&item);
    if (rear ==-1 && front ==-1)
    {
        rear=0;
        front=0;
    }
    else
    {
        rear=rear+1;
    }
    queue[rear]=item;
    return;
}

int del()
{
```

```
    if (front==-1 && rear==-1)
    {
        printf("Queue is Empty\n");
        return -1;
    }
    item=queue[front];

    if (front==rear)
    {
        front=-1;
        rear=-1;
    }
    else
    {
        front=front+1;
    }
    return item;
}

void display()
{
    if (front==-1 && rear==-1)
    {
        printf("Queue is Empty \n");
        return;
    }
    printf("The Elements of the Queue are: \n");
    for (i=front;i<=max-1;i++)
    {
        printf("%d \n", queue[i]);
    }
```

```
    return;
}
```

Output:

```
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 3
Queue is Empty

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 1
Enter Element:
4

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 1
Enter Element:
7

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 1
Enter Element:
9
```

```
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 1
Queue is Full

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 3
The Elements of the Queue are:
4
7
9

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 2
The Deleted Item is:4

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 2
The Deleted Item is:7
```

```
1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 2
The Deleted Item is:9

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 2
Queue is Empty

1. INSERT
2. DELETE
3. DISPLAY
4. EXIT
Enter Your Choice: 4


=== Code Execution Successful ===
```

Circular Queue

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>
#include <stdlib.h>
#include <string.h>
#define MAX 4

void Insert();
int Delete();
void Display();

int cq[20];
int front=-1, rear=-1, item, ch, i;

void main()
{
  while(1)
  {
    printf(" \n1. Insert \n2. Delete \n3. Display \n4. Exit");
    printf("\nEnter Your Choice: ");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1: Insert();
        break;
      case 2: item=Delete();
          if (item!=-1)
          {
```

```
                printf("The Dequeued Element is: %d",item);
              }
              break;
          case 3: Display();
            break;
          case 4: exit(0);
        }
    }
}

void Insert()
{
    if (front == (rear+1) % MAX)
        {
          printf("Circular Queue is Full. \n");
          return;
        }
    if (rear==-1 && front==-1)
    {
        rear=0;
        front=0;
    }
    else
        rear=(rear+1)%MAX;
    printf("Enter the Element to be Inserted: ");
    scanf("%d",&item);
    cq[rear]=item;
    return;
}

int Delete()
```

```
{
   if(front==-1 && rear==-1)
   {
      printf("Circular Queue is Empty. \n");
      return (-1);
   }
   item=cq[front];
   if(front==rear)
   {
      front=-1;
      rear=-1;
   }
   else
      front=(front+1)%MAX;
   return item;
}

void Display()
{
   if(front==-1 && rear==-1)
   {
      printf("Circular Queue is Empty. \n");
      return;
   }

   printf("Circular Queue Contents: \n");
   if (front<=rear)
   {
      for (int i=front;i<=rear;i++)
      {
         printf("%d\n",cq[i]);
```

```
        }
    }

  else
  {
      for(int i=front;i<=MAX-1;i++)
      {
         printf("%d\n",cq[i]);
      }
      for (int i=0;i<=rear;i++)
      {
         printf("%d\n",cq[i]);
      }
  }
  return;
}
```

Output:

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Enter the Element to be Inserted: 10

1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Enter the Element to be Inserted: 20

1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Enter the Element to be Inserted: 30

1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Enter the Element to be Inserted: 40

1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Circular Queue is Full.

1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 3
Circular Queue Contents:
10
20
30
40
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
The Dequeued Element is: 10
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
The Dequeued Element is: 20
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
The Dequeued Element is: 30
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
The Dequeued Element is: 40
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
Circular Queue is Empty.

1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 4

Process returned 0 (0x0)   execution time : 43.123 s
Press any key to continue.
```

Lab 4: Insertion in Singly Linked List

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.

c) Display the contents of the linked list.

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

struct Node {
   int data;
   struct Node *link;
};

typedef struct Node node;
node *start = NULL;
node *new1, *curr, *ptr;


void create();
void display();
void InsertStart();
void InsertPosition();
void InsertEnd();

void main() {
   int ch;
   while (1) {
         printf("\n1. Create \n2. Display \n3. Insert at Beginning \n4. Insert at Position \n5. Insert at End \n6. Exit");
      printf("\nEnter Your Choice: ");
```

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

        switch (ch) {
            case 1: create();
                break;
            case 2: display();
                break;
            case 3: InsertStart();
                break;
            case 4: InsertPosition();
                break;
            case 5: InsertEnd();
                break;
            case 6: exit(0);
        }
    }
}

void create() {
    char ch;

    do {
        new1 = (node*)malloc(sizeof(node));
       printf("\nEnter Value: ");
       scanf("%d",&new1->data);
       if (start==NULL)
       {
           start=new1;
           curr=new1;
       }
       else {
```

```
        curr->link = new1;
        curr=new1;
      }

      printf("Do You Want to Add an Element (Y/N)? ");
      scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.");
        return;
    }

    ptr = start;
    printf("\nElements in Linked List: \n");

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

void InsertStart() {
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter Value: ");
    scanf("%d",&new1->data);
    if(start==NULL)
```

```
    {
        start=new1;
        new1->link=NULL;
        return;
    }
    else {
        new1->link=start;
        start=new1;
        return;
    }
}

void InsertEnd() {
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter Value: ");
    scanf("%d",&new1->data);
    if(start==NULL)
    {
        start=new1;
        new1->link=NULL;
        return;
    }

    ptr=start;
    while(ptr->link !=NULL)
    {
        ptr=ptr->link;
    }
    ptr->link=new1;
    new1->link=NULL;
    return;
```

```
}

void InsertPosition() {
   new1 = (node*)malloc(sizeof(node));
   printf("\nEnter Value: ");
   scanf("%d",&new1->data);
   if(start==NULL)
   {
      start=new1;
      new1->link=NULL;
      return;
   }

   int i=1, pos;
   ptr=start;
   printf("\nEnter Position: ");
   scanf("%d",&pos);
   while (ptr!=NULL && i<pos-1)
   {
      ptr=ptr->link;
      i++;
   }
   if(ptr==NULL)
   {
      return;
   }

   new1->link=ptr->link;
   ptr->link=new1;
}
```

Output:

```
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 1

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 20
Do You Want to Add an Element (Y/N)? n

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
10 20

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 3

Enter Value: 30
```

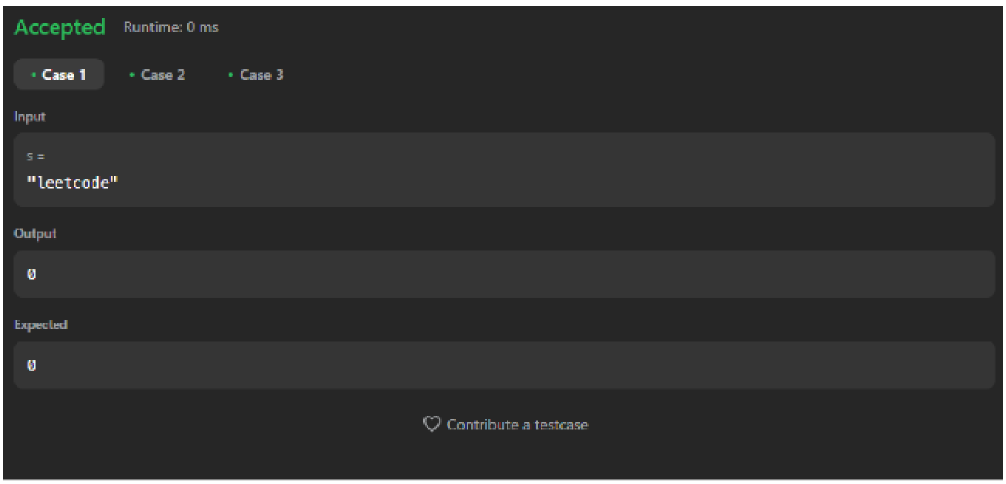
```
Enter Value: 30

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 4

Enter Value: 40

Enter Position: 2

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 5

Enter Value: 50

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
30 40 10 20 50

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 6
```

Leetcode

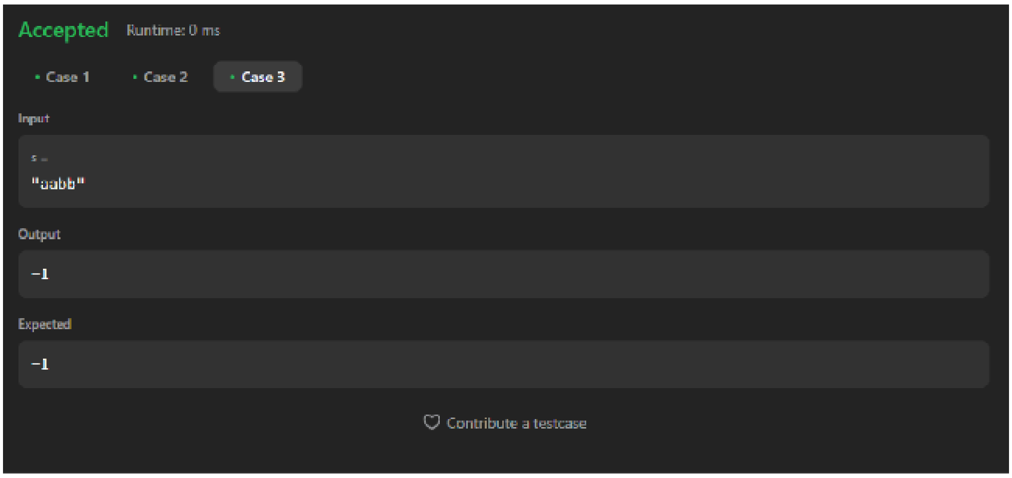
First Unique Character in a String: Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.

```
int firstUniqChar(char* s) {
    int l = strlen(s);
    int a[26] = {0};
    int i=0;
    for (i=0;i<l;i++)
    {
        a[s[i] - 'a']++;
    }
    for (i=0;i<l;i++)
    {
        if (a[s[i]-'a']==1) {
            return i;
        }
    }
    return -1;
}
```

Output:




Accepted Runtime: 0 ms
Case 1
Case 2
Case 3
Input
s =
"loveleetcode"
Output
2
Expected
2
Contribute a testcase


Accepted Runtime: 0 ms
Case 1
Case 2
Case 3
Input
s =
"aabb"
Output
-1
Expected
-1
Contribute a testcase

Lab 5: Deletion in Linked List

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 *link;
};

typedef struct Node node;
node *start = NULL;
node *new1, *curr, *ptr;


void create();
void display();
void DeleteStart();
void DeletePosition();
void DeleteEnd();


void main() {
   int ch;
   while (1) {
        printf("\n1. Create \n2. Display \n3. Delete from Beginning \n4. Delete at Position \n5. Delete at End \n6. Exit");
```

```
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);

        switch (ch) {
          case 1: create();
            break;
          case 2: display();
            break;
          case 3: DeleteStart();
            break;
          case 4: DeletePosition();
            break;
          case 5: DeleteEnd();
            break;
          case 6: exit(0);
        }
    }
}

void create() {
    char ch;

    do {
        new1 = (node*)malloc(sizeof(node));
       printf("\nEnter Value: ");
       scanf("%d",&new1->data);
       if (start==NULL)
       {
          start=new1;
          curr=new1;
       }
```

```
        else {
            curr->link = new1;
            curr=new1;
        }

        printf("Do You Want to Add an Element (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.");
        return;
    }

    ptr = start;
    printf("\nElements in Linked List: \n");

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

void DeleteStart() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
```

```
    }

    node *temp = start;
    start = start->link;
    free(temp);
    printf("\nFirst Element Deleted.\n");
}

void DeletePosition() {
    int i=1,pos;
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    printf("\nEnter Position: ");
    scanf("%d", &pos);

    node *temp = start;
    node *prev = NULL;

    if (pos == 1) {
        start = temp->link;
        free(temp);
        printf("\nElement at Position %d Deleted.\n", pos);
        return;
    }

    while (temp != NULL && i < pos) {
        prev = temp;
        temp = temp->link;
```

```
        i++;
    }

    if (temp == NULL) {
        printf("\nPosition Not Found.\n");
        return;
    }

    prev->link = temp->link;
    free(temp);
    printf("\nElement at Position %d Deleted\n", pos);
}

void DeleteEnd() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    node *temp = start;
    node *prev = NULL;

    if (start->link == NULL) {
        start = NULL;
        free(temp);
        printf("\nLast Element Deleted.\n");
        return;
    }

    while (temp->link != NULL) {
        prev = temp;
```

```
        temp = temp->link;
    }

    prev->link = NULL;
    free(temp);
    printf("\nLast element Deleted.\n");
}
```

Output:

```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 1

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 20
Do You Want to Add an Element (Y/N)? y

Enter Value: 30
Do You Want to Add an Element (Y/N)? y

Enter Value: 40
Do You Want to Add an Element (Y/N)? y

Enter Value: 50
Do You Want to Add an Element (Y/N)? y

Enter Value: 60
Do You Want to Add an Element (Y/N)? n
```

```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
10 20 30 40 50 60
```

```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 3

First Element Deleted.

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
20 30 40 50 60

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 4

Enter Position: 3

Element at Position 3 Deleted
```

```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
20 30 50 60

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 5

Last element Deleted.

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
20 30 50

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 6
```

Lab 6: Sort, Reverse and Concatenate Linked Lists

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 *link;
};
typedef struct Node node;

node *start = NULL, *temp, *new1, *curr;
int ch;
char c;

void createList();
void sort();
void reverse();
void display();
void concatenate();

void createList() {
    do {
        new1 = (node*)malloc(sizeof(node));
        printf("Enter Value: ");
        scanf("%d", &new1->data);
        new1->link = NULL;

        if (start == NULL) {
```

```
            start = new1;
            curr = new1;
        } else {
            curr->link = new1;
            curr = new1;
        }
        printf("Do you want to add another element (Y/N): ");
        scanf(" %c", &c);
    } while (c == 'y' || c == 'Y');
}

void sort() {
    if (start == NULL) {
        printf("The Linked List is Empty.\n");
        return;
    }

    node *i, *j;
    int tempData;
    for (i = start; i != NULL; i = i->link) {
        for (j = i->link; j != NULL; j = j->link) {
            if (i->data > j->data) {
                tempData = i->data;
                i->data = j->data;
                j->data = tempData;
            }
        }
    }
    printf("Linked List is Sorted.\n");
}
```

```
void reverse() {
    node *a = start, *b = NULL;
    while (a != NULL) {
        temp = a->link;
        a->link = b;
        b = a;
        a = temp;
    }
    start = b;
    printf("Linked List is Reversed.\n");
}

void display() {
    if (start == NULL) {
        printf("Linked list is Empty\n");
        return;
    }

    temp = start;
    printf("Elements in Linked List:\n");
    while (temp != NULL) {
        printf("%d\t", temp->data);
        temp = temp->link;
    }
    printf("\n");
}

void concatenate() {
    node *start2 = NULL, *curr2 = NULL;

    printf("Enter the second linked list:\n");
```

```
    createList();

    do {
        new1 = (node*)malloc(sizeof(node));
        printf("Enter value for second list: ");
        scanf("%d", &new1->data);
        new1->link = NULL;

        if (start2 == NULL) {
            start2 = new1;
            curr2 = new1;
        } else {
            curr2->link = new1;
            curr2 = new1;
        }
        printf("Do you want to add another element (Y/N): ");
        scanf(" %c", &c);
    } while (c == 'y' || c == 'Y');

    if (start == NULL) {
        start = start2;
    } else {
        temp = start;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        temp->link = start2;
    }
    start2 = NULL;
    printf("Lists concatenated successfully.\n");
}
```

```
int main() {
  while (1) {
     printf("\n1. Create 1st Linked List\n2. Sort Linked List\n3. Reverse Linked List\n4. Concatenate Linked Lists\n5. Display Linked List\n6. Exit\n");
    printf("Enter Your Choice: ");
    scanf("%d", &ch);
    switch (ch) {
      case 1:
        createList();
        break;
      case 2:
        sort();
        break;
      case 3:
        reverse();
        break;
      case 4:
        concatenate();
        break;
      case 5:
        display();
        break;
      case 6:
        exit(0);
        break;
      default:
        printf("Invalid choice. Please try again.\n");
        break;
    }
  }
```

}

Output:

```
1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 1
Enter Value: 10
Do you want to add another element (Y/N): y
Enter Value: 80
Do you want to add another element (Y/N): y
Enter Value: 60
Do you want to add another element (Y/N): y
Enter Value: 20
Do you want to add another element (Y/N): y
Enter Value: 70
Do you want to add another element (Y/N): y
Enter Value: 30
Do you want to add another element (Y/N): n

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
10      80      60      20      70      30

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 3
Linked List is Reversed.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
30      70      20      60      80      10
```

```
1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 2
Linked List is Sorted.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
10      20      30      60      70      80

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 4
Enter the second linked list:
Enter Value: 10
Do you want to add another element (Y/N): y
Enter Value: 70
Do you want to add another element (Y/N): y
Enter Value: 80
Do you want to add another element (Y/N): y
Enter Value: 60
Do you want to add another element (Y/N): y
Enter Value: 30
Do you want to add another element (Y/N): n
Enter value for second list: 10
Do you want to add another element (Y/N): y
Enter value for second list: 50
Do you want to add another element (Y/N): y
Enter value for second list: 60
Do you want to add another element (Y/N): y
Enter value for second list: 40
Do you want to add another element (Y/N): n
Lists concatenated successfully.
```

```
1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
10      10      70      80      60      30      10      50      60      40

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 2
Linked List is Sorted.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
10      10      10      30      40      50      60      60      70      80

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 3
Linked List is Reversed.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
80      70      60      60      50      40      30      10      10      10
```

Stack and Queue Operations

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

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

struct Node{
   int data;
   struct Node *link;
};

typedef struct Node node;

//Stack
node *top=NULL;
void push();
void pop();
void displayStack();

void push(){
   node *new1=(node*)malloc(sizeof(node));
   if(new1==NULL){
      printf("\nStack Overflow.\n");
      return;
   }

   printf("\nEnter Value to Push: ");
   scanf("%d", &new1->data);
   new1->link=top;
   top=new1;
}
```

```
void pop(){
   if(top==NULL){
      printf("\nStack Underflow.\n");
      return;
   }

   node *temp=top;
   printf("\nPopped Element: %d\n", temp->data);
   top=top->link;
   free(temp);
}

void displayStack(){
   if(top==NULL){
      printf("\nThe Stack is Empty.\n");
      return;
   }

   printf("\nElements in the Stack: ");
   node *temp=top;
   while(temp!=NULL){
      printf("%d ", temp->data);
      temp=temp->link;
   }
   printf("\n");
}


//Queue
node *front=NULL, *rear=NULL;
```

```
void insert();
void del();
void displayQueue();

void insert(){
   node *new1=(node*)malloc(sizeof(node));
   if(new1==NULL){
      printf("\nQueue Full.\n");
      return;
   }

   printf("\nEnter Value to Insert: ");
   scanf("%d", &new1->data);
   new1->link=NULL;

   if(rear==NULL){
      front=rear=new1;
      return;
   }
   rear->link=new1;
   rear=new1;
}

void del(){
   if(front==NULL){
      printf("\nQueue Empty.\n");
      return;
   }

   node *temp=front;
```

```
    printf("\nDeleted Element: %d\n", temp->data);
    front=front->link;

    if(front==NULL){
        rear=NULL;
    }
    free(temp);
}

void displayQueue(){
    if(front==NULL){
        printf("\nThe Queue is Empty.\n");
        return;
    }

    printf("\nElements in the Queue: ");
    node *temp=front;
    while(temp!=NULL){
        printf("%d ", temp->data);
        temp=temp->link;
    }
    printf("\n");
}


// Main
void main(){
    int ch;

    while(1){
        printf("\n1. Push (Stack) \n2. Pop (Stack) \n3. Display (Stack)");
```

```
        printf("\n4. Insert (Queue) \n5. Delete (Queue) \n6. Display (Queue) \n7. Exit");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);

        switch(ch){
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                insert();
                break;
            case 5:
                del();
                break;
            case 6:
                displayQueue();
                break;
            case 7:
                exit(0);
            default:
                printf("\nEnter Your Choice: \n");
        }
    }
}
```

Output:

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 10

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 20

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 30

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1

Enter Value to Push: 40
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 3

Elements in the Stack: 40 30 20 10

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 2

Popped Element: 40

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 2

Popped Element: 30

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 3

Elements in the Stack: 20 10
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 10

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 20

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 30

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4

Enter Value to Insert: 40
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 6

Elements in the Queue: 10 20 30 40

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 5

Deleted Element: 10

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 5

Deleted Element: 20

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 5

Deleted Element: 30
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 6

Elements in the Queue: 40

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 7
```

Lab 7: Doubly Linked Lists

WAP to Implement doubly link list with primitive operations

i. **Create a doubly linked list.**
ii. **Insert a new node to the left of the node.**
iii. **Delete the node based on a specific value.**
iv. **Display the contents of the list.**

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node *left;

   struct Node *right;

};


typedef struct Node node;

node *start = NULL;

node *new1, *curr, *ptr;


void create();

void display();

void InsertLeft();

void DeleteSpecificElement();


void main() {

   int ch;

   while (1) {

         printf("\n1. Create \n2. Display \n3. Insert Left \n4. Delete Specific Element
\n5. Exit");

       printf("\nEnter Your Choice: ");

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

```
        switch (ch) {
            case 1: create();
                break;
            case 2: display();
                break;
            case 3: InsertLeft();
                break;
            case 4: DeleteSpecificElement();
                break;
            case 5: exit(0);
        }
    }
}

void create() {
    char ch;

    do {
        new1 = (node*)malloc(sizeof(node));
        printf("\nEnter Value: ");
        scanf("%d", &new1->data);
        new1->left = NULL;
        new1->right = NULL;

        if (start == NULL) {
            start = new1;
            curr = new1;
        } else {
            curr->right = new1;
            new1->left = curr;
```

```
        curr = new1;
    }

    printf("Do You Want to Add an Element (Y/N)? ");
    scanf(" %c", &ch);
  } while (ch == 'y' || ch == 'Y');
}

void display() {
  if (start == NULL) {
    printf("\nLinked List is Empty.");
    return;
  }

  ptr = start;
  printf("\nElements in Linked List: \n");

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

void InsertLeft() {
  int val;
  printf("\nEnter Value: ");
  scanf("%d", &val);

  new1 = (node*)malloc(sizeof(node));
  new1->data = val;
```

```
    new1->left = NULL;
    new1->right = NULL;

    printf("\nEnter the Value to Insert Left of: ");
    scanf("%d", &val);

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

    if (ptr != NULL) {
        new1->right = ptr;
        new1->left = ptr->left;
        if (ptr->left != NULL) {
            ptr->left->right = new1;
        }
        ptr->left = new1;
        if (ptr == start) {
            start = new1;
        }
    } else {
        printf("\nValue not found.\n");
    }
}

void DeleteSpecificElement() {
    int value;
    printf("\nEnter Value to Delete: ");
    scanf("%d", &value);
```

```
    ptr = start;
    while (ptr != NULL && ptr->data != value) {
        ptr = ptr->right;
    }

    if (ptr == NULL) {
        printf("\nValue not found.\n");
        return;
    }

    if (ptr->left != NULL) {
        ptr->left->right = ptr->right;
    }
    if (ptr->right != NULL) {
        ptr->right->left = ptr->left;
    }
    if (ptr == start) {
        start = ptr->right;
    }

    free(ptr);
    printf("\nElement with value %d deleted.\n", value);
}
```

Output:

```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 1

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 20
Do You Want to Add an Element (Y/N)? y

Enter Value: 30
Do You Want to Add an Element (Y/N)? n

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 3

Enter Value: 40

Enter the Value to Insert Left of: 20

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
10 40 20 30
```

```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 4

Enter Value to Delete: 20

Element with value 20 deleted.

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
10 40 30

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 5

Process returned 0 (0x0)   execution time : 397.571 s
Press any key to continue.
```

Lab 8: Binary Search Tree

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>

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

node* createNode(int data) {
   node* new1 = (node*)malloc(sizeof(node));
   new1->data = data;
   new1->left = new1->right = NULL;
   return new1;
}

node* insertNode(node* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }
   if (data < root->data) {
      root->left = insertNode(root->left, data);
   } else {
      root->right = insertNode(root->right, data);
   }
   return root;
}
```

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

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

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

void displayTree(node* root, int space) {
   if (root == NULL) {
      return;
   }

   space += 10;
```

```
    displayTree(root->right, space);

    printf("\n");
    for (int i = 10; i < space; i++) {
        printf(" ");
    }
    printf("%d\n", root->data);

    displayTree(root->left, space);
}

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

    printf("Binary Search Tree Operations:\n");
    while (1) {
                printf("\n1. Insert\n2. In-order Traversal\n3. Pre-order Traversal\n4.
Post-order Traversal\n5. Display Tree\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorderTraversal(root);
```

```
            printf("\n");
            break;
        case 3:
            printf("Pre-order Traversal: ");
            preorderTraversal(root);
            printf("\n");
            break;
        case 4:
            printf("Post-order Traversal: ");
            postorderTraversal(root);
            printf("\n");
            break;
        case 5:
            printf("Tree Representation:\n");
            displayTree(root, 0);
            printf("\n");
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
      }
    }

    return 0;
}
```

Output:

```
Binary Search Tree Operations:

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 50

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 40

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 75

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 10

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 25

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 80
```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 20

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 2
In-order Traversal: 10 20 25 40 50 75 80

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 3
Pre-order Traversal: 50 40 10 25 20 75 80

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 4
Post-order Traversal: 20 25 10 40 80 75 50

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 5
```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display Tree
6. Exit
Enter your choice: 5
Tree Representation:

                    80

          75

50

          40

                              25

                                        20

                    10
```

Lab 9: Traverse a Graph using BFS Method

Write a program to traverse a graph using BFS method.

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

#define MAX 10

int queue[MAX], front = -1, rear = -1;

void enqueue(int item) {
    if (rear == MAX - 1) {
        printf("Queue is Full\n");
        return;
    }
    if (front == -1){
            front = 0;
    }
    queue[++rear] = item;
}

int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is Empty\n");
        return -1;
    }
    return queue[front++];
}

void bfs(int graph[MAX][MAX], int visited[MAX], int start, int n) {
    int i;
```

```
    enqueue(start);
    visited[start] = 1;

    printf("BFS Traversal: ");
    while (front <= rear) {
        int current = dequeue();
        printf("%d ", current);

        for (i = 0; i < n; i++) {
            if (graph[current][i] == 1 && visited[i] == 0){
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}

void main() {
    int n, i, j, start;
    int graph[MAX][MAX], visited[MAX] = {0};

    printf("Enter the Number of Vertices: ");
    scanf("%d", &n);

    printf("Enter the Adjacency Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
```

```
    printf("Enter the Starting Vertex: ");
    scanf("%d", &start);

    bfs(graph, visited, start, n);
}
```

Output:

```
Enter the Number of Vertices: 5
Enter the Adjacency Matrix:
0 0 1 1 1
0 0 0 1 1
1 0 0 1 0
1 1 1 0 0
1 1 0 0 0
Enter the Starting Vertex: 1
BFS Traversal: 1 3 4 0 2

Process returned 10 (0xA)   execution time : 30.652 s
Press any key to continue.
```

DFS Connected

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

```
#include <stdio.h>

#define MAX 10

int a[MAX][MAX], vis[MAX], n;

void dfs(int v);
int isConnected();

void main() {
  int i, j;

  printf("Enter Number of Vertices: ");
  scanf("%d", &n);

  printf("Enter Adjacency Matrix:\n");
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      scanf("%d", &a[i][j]);
    }
  }

  printf("\nDFS Traversal: ");

  if (isConnected()) {
    printf("\nThe graph is connected.\n");
  } else {
    printf("\nThe graph is disconnected.\n");
```

```
    }

    for (i = 0; i < n; i++) {
        vis[i] = 0;
    }

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

    printf("\n");
}

void dfs(int v) {
    printf("%d ", v);
    vis[v] = 1;

    for (int i = 0; i < n; i++) {
        if (a[v][i] == 1 && vis[i] == 0) {
            dfs(i);
        }
    }
}

int isConnected() {
    int i;

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

```
        vis[i] = 0;
    }

    dfs(0);

    for (i = 0; i < n; i++) {
        if (vis[i] == 0) {
            return 0;
        }
    }
    return 1;
}
```

Output:

```
Enter Number of Vertices: 5
Enter Adjacency Matrix:
0 0 1 1 1
0 0 0 1 1
1 0 0 1 0
1 1 1 0 0
1 1 0 0 0

DFS Traversal: 0 2 3 1 4
The graph is connected.
```

Lab 10: Hashing

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 -> 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_EMPLOYEES 100
#define m 100
typedef struct {
    int key;
    int address;
} EmployeeRecord;
int hashTable[m];
int hashFunction(int key) {
    return key % m;
}
int insert(int key) {
    int index = hashFunction(key);
    while (hashTable[index] != -1) {
        index = (index + 1) % m;
    }
    hashTable[index] = key;
    return index;
}
void displayHashTable() {
    printf("\nHash Table:\n");
    printf("Index  Key\n");
    for (int i = 0; i < m; i++) {
```

```
        if (hashTable[i] != -1) {
            printf("%d     %d\n", i, hashTable[i]);
        }
    }
}
int main() {
    for (int i = 0; i < m; i++) {
        hashTable[i] = -1;
    }
    int employeeKeys[MAX_EMPLOYEES];
    int numEmployees;

    printf("Enter number of employees: ");
    scanf("%d", &numEmployees);

    printf("Enter the employee keys (4-digit integers):\n");
    for (int i = 0; i < numEmployees; i++) {
        scanf("%d", &employeeKeys[i]);
    }
    for (int i = 0; i < numEmployees; i++) {
        int address = insert(employeeKeys[i]);
        printf("Employee key %d inserted at address %d\n", employeeKeys[i], address);
    }
    displayHashTable();
    return 0;
}
```

Output:

```
Enter number of employees: 9
Enter the employee keys (4-digit integers):
1234
1111
1444
1342
1567
1777
1980
1665
1343
Employee key 1234 inserted at address 34
Employee key 1111 inserted at address 11
Employee key 1444 inserted at address 44
Employee key 1342 inserted at address 42
Employee key 1567 inserted at address 67
Employee key 1777 inserted at address 77
Employee key 1980 inserted at address 80
Employee key 1665 inserted at address 65
Employee key 1343 inserted at address 43

Hash Table:
Index  Key
11       1111
34       1234
42       1342
43       1343
44       1444
65       1665
67       1567
77       1777
80       1980
```