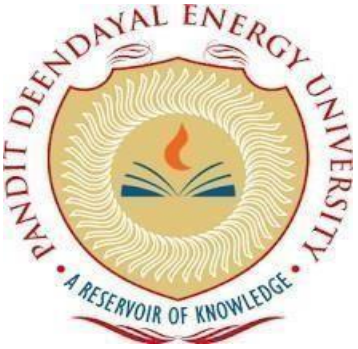
# Lab Experiments
# Data Structures and Algorithms
# (24IC203P)

SCHOOL OF TECHNOLOGY
PANDIT DEENDAYAL ENERGY UNIVERSITY
GANDHINAGAR, GUJARAT, INDIA

# Information and Communication Technology LAB File (2025-26)

# Data Structures and Algorithms LAB (24IC203P)




**Student Name: Ronit Kundnani**

**Enrollment No.: 24BIT100**

**Course with Semester: 3rd**

**Division: 2**

**Group: H5**

**Lab Instructors: Dr. Rutvij H. Jhaveri**

**Dr. Rajendra Choudhary**

**Dr. Davinder Singh**

# Table of Contents

**Course Outcomes (COs):**

On completion of the course, student will be able to

**CO1** - Differentiate linear and non-linear data structures.
**CO2** - Enhance logical reasoning and programming skills.
**CO3** - Implement linear and non-linear data structures.
**CO4** - Identify suitable data structures to solve complex computing problems.
**CO5** - Apply the algorithms on the small and large data sets.
**CO6** - Design and implement an appropriate hashing function for an application.

**List of Experiments:**



**Pre-requisites:**

1. Basic knowledge of Computer Programming Fundamentals: Understanding of Arrays, Strings, Structure, Pointers and other fundamentals.

# Instructions

i. Make all the programs using C language only.

ii. You are allowed to use gcc compiler and command prompt for running the programming. Also, you are allowed to use any IDE (like, CODEBLOCKS) for implementation.

iii. Each program must print your name and enrollment number in the starting of your each C program.

iv. In each lab after making the programs, paste the code (in text format) with the output (snapshot of the output) in this file.

v. Evaluation will be carried out based on Correctness of Code, Code Efficiency (Optimization), Use of Appropriate Data Structures, Code Readability and Structure, Documentation and Comments, Modularity and Reusability, Test Cases and Output, Handling of Errors, Memory Management, Innovative Approach and Viva Performance,.

vi. You must submit only a soft copy of the lab manual.

| Exp. No. | Experiment Title | Da |
|---|---|---|
| 1 | **[Revision of Arrays and Structures]**<br>Programs covering basics of Arrays and Structure.<br>1. Write a program in C to perform addition of two 3x3 matrices. | |

**CODE**

```
#include <stdio.h>

void add(int arr1[3][3],int arr2[3][3],int res[3][3]){
   for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
         res[i][j]=arr1[i][j]+arr2[i][j];
      }
   }
}

int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   int arr1[3][3];
   int arr2[3][3];
   int res[3][3];

   printf("Enter the Elements of the matrix 1 \n");
   for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
         printf("Enter %d %d th element:",i,j);
         scanf("%d",&arr1[i][j]);
      }
      printf("\n");
   }
   printf("Enter the Elements of the matrix 2: \n");
   for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
         printf("Enter %d %d th element:",i,j);
         scanf("%d",&arr2[i][j]);
      }
      printf("\n");
   }

   add(arr1,arr2,res);

   for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
         printf("%d \t",res[i][j]);
      }
      printf("\n");
   }
   return 0;
}
```

**OUTPUT**

```
D:\college\DS-Sem3\Exp1>.\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter the Elements of the matrix 1
Enter 0 0 th element:1
Enter 0 1 th element:1
Enter 0 2 th element:1

Enter 1 0 th element:1
Enter 1 1 th element:1
Enter 1 2 th element:1

Enter 2 0 th element:1
Enter 2 1 th element:1
Enter 2 2 th element:1

Enter the Elements of the matrix 2:
Enter 0 0 th element:2
Enter 0 1 th element:2
Enter 0 2 th element:2

Enter 1 0 th element:2
Enter 1 1 th element:2
Enter 1 2 th element:2

Enter 2 0 th element:2
Enter 2 1 th element:22
Enter 2 2 th element:2

3       3       3
3       3       3
3       23      3
```

2. Create a structure Student in C with student name, student roll number and student address as its data members. Create the array of type student and print their values.

**CODE**

```
#include <stdio.h>
struct Student{
      char name[10];
      int rollno;
      char address[20];
};
int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   int n;
   printf("Enter the number of students:");
   scanf("%d",&n);

   struct Student s1[n];
   for(int i=0;i<n;i++){
      printf("Enter The details of the %d th student \n",(i+1));
      printf("Enter student name:");
      scanf("%s",&s1[i].name);
      printf("Enter student RollNo:");
      scanf("%d",&s1[i].rollno);
      printf("Enter student address:");
      scanf("%s",&s1[i].address);
      printf("\n");
   }
   printf("\n \n");
   for(int i=0;i<n;i++){
      printf("Student %d :\n",i+1);
      printf("student name: %s\n",s1[i].name);
      printf("student RollNo:%d\n",s1[i].rollno);
      printf("student address:%s\n",s1[i].address);
   }

   return 0;
}
```

**OUTPUT**

```
D:\college\DS-Sem3\Exp1>.\Program2.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter the number of students:2
Enter The details of the 1 th student
Enter student name:krish
Enter student RollNo:120
Enter student address:hjACdsv

Enter The details of the 2 th student
Enter student name:Yug
Enter student RollNo:099
Enter student address:hjsjafv



Student 1 :
student name: krish
student RollNo:120
student address:hjACdsv
Student 2 :
student name: Yug
student RollNo:99
student address:hjsjafv
```

**Practice Problem:**

3. Write a program in C that obtains the minimum and maximum element from the array. Modify this program to give the second largest and second smallest element of the array.

**CODE**

```
#include <stdio.h>
int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   int n;
   printf("Enter the size of array: ");
   scanf("%d",&n);
   int arr[n];
   printf("Enter the elements of the array: ");
   for(int i=0;i<n;i++){
      scanf("%d",&arr[i]);
   }

   int largest=arr[0] , secondLarge=arr[0];
   int smallest=arr[0] , secondSmall=arr[0];

   for(int i=0;i<n;i++){
      if(arr[i]>largest){
         secondLarge=largest;
         largest=arr[i];
      }
      else if(arr[i]>secondLarge && arr[i]<largest){
         secondLarge=arr[i];
      }

      if(arr[i]<smallest){
         secondSmall=smallest;
         smallest=arr[i];
      }
      else if(arr[i]<secondSmall && arr[i]>smallest){
         secondSmall=arr[i];
      }
   }

   printf("The smallest and Second smallest value of the array are %d and %d \n",smallest,secondSmall);
   printf("The largest and Second largest value of the array are %d and %d",largest,secondLarge);

   return 0;
}
```

**OUTPUT**

```
D:\college\DS-Sem3\Exp1>.\Program3.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter the size of array: 10
Enter the elements of the array: 2
9
54
21
40
87
4
1
66
99
The smallest and Second smallest value of the array are 1 and 2
The largest and Second largest value of the array are 99 and 87
```

2

**[Revision of Pointers and Structures]**

Programs covering structure definition, array of structure, nested structures etc.

1. Write a program in C to implement arrays of pointers and pointers to arrays.

**CODE**

```
#include <stdio.h>
int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   int a=10,b=20,c=30;
   int *ptr[]={&a,&b,&c};
   printf("Array Of Pointers\n");
   for(int i=0;i<3;i++){
      printf("%d\n",*ptr[i]);
   }
   printf("Pointers Of Array \n");
   int arr[]={1,2,3,4,5,6,7,8,9,10};
   int *p=arr;
   for(int i=0;i<10;i++){
      printf("%d\n",*(p+i));
   }
   return 0;
}
```

**OUTPUT**

```
D:\college\DS-Sem3\Exp2>.\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Array Of Pointers
10
20
30
Pointers Of Array
1
2
3
4
5
6
7
8
9
10
```

**CODE**

2. Write a program in C to implement pointers to structures.

```
#include <stdio.h>
#include <string.h>
struct Student{
        char name[10];
        int rollno;
        char address[20];
};
int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   struct Student s1;
   struct Student *ptr;
   ptr=&s1;

   strcpy(ptr->name,"ronit");
   ptr->rollno=10;
   strcpy(ptr->address,"BlahBlah");

   printf("name:%s\n",ptr->name);
   printf("rollno:%d\n",ptr->rollno);
   printf("address:%s\n",ptr->address);
   return 0;
}
```

OUTPUT

```
D:\college\DS-Sem3\Exp2>.\Program2.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
name:ronit
rollno:10
address:BlahBlah
```

**Practice Problem:**

3. Write a program in C to perform swapping of two numbers by passing addresses of the variables to the functions.

**CODE**

```
#include <stdio.h>
void swap(int *a,int *b){
   int temp=*a;
   *a=*b;
   *b=temp;

}
int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   int a=10,b=20;
   printf("Values Before swapping a:%d and b:%d\n",a,b);
   swap(&a,&b);
   printf("swapped values: a:%d b:%d",a,b);
   return 0;
}
```

**OUTPUT**

```
D:\college\DS-Sem3\Exp2>.\Program3.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Values Before swapping a:10 and b:20
swapped values: a:20 b:10
```

| 3 | **[Stack]** | | |
|---|---|---|---|

**CODE**

Programs covering stack and applications.

1. Implement a stack using an array. Implement the following functions:

   a. isEmpty(): Tests whether stack is empty or not.
   b. push(): Adds new element to the stack.
   c. pop(): Removes top element from the stack.
   d. top(): Returns value of the top element.

```
 #include<stdio.h>
#define MAX 20

struct stack{
   int s[MAX];
   int top;
};

struct stack s1;

int isEmpty(){
   if(s1.top == -1){
      printf("Stack is Empty or Underflow\n");
      return 1;
   }
   return 0;
}

int isFull(){
   if(s1.top == MAX-1){
      printf("Stack is Full or Overflow\n");
      return 1;
   }
   return 0;
}

void push(int ele){
   if(isFull()){
     printf("You can't push an element\n");
   }else{
      s1.top += 1;
         s1.s[s1.top]=ele;
         printf("%d is pushed into stack\n", ele);
   }
}

void pop(){
   if(isEmpty()){
     printf("You can't pop an element\n");
   }else{
      printf("%d is popped from the stack\n" ,s1.s[s1.top]);
      s1.top -= 1;
   }
```

```
}

void Top(){
   printf("Top element of stack is %d\n" ,s1.s[s1.top]);

}

int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   s1.top = -1;
push(1);
push(3);
push(5);
pop();
push(2);
push(4);
pop();
pop();
Top();
pop();
pop();
pop();
   return 0;
}
```

**OUTPUT**

```
PS D:\college\DS-Sem3\Exp3> .\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
1 is pushed into stack
3 is pushed into stack
5 is pushed into stack
5 is popped from the stack
2 is pushed into stack
4 is pushed into stack
4 is popped from the stack
2 is popped from the stack
Top element of stack is 3
3 is popped from the stack
1 is popped from the stack
Stack is Empty or Underflow
You can't pop an element
```

**CODE**

**Practice Problems:**

2. Write a program to reverse a given string using a stack.

```
#include<stdio.h>
#include<string.h>
#define MAX 20

struct stack{
   char s[MAX];
   int top;
};

struct stack s1;

int isEmpty(){
   if(s1.top == -1){
      printf("Stack is Empty or Underflow\n");
      return 1;
   }
   return 0;
}

int isFull(){
   if(s1.top == MAX-1){
      printf("Stack is Full or Overflow\n");
      return 1;
   }
   return 0;
}

void push(char ele){
   if(isFull()){
     printf("You can't push an element\n");
   }else{
      s1.top += 1;
         s1.s[s1.top]=ele;
         //printf("%c is pushed into stack\n", ele);
   }
}

char pop(){
   if(isEmpty()){
     printf("You can't pop an element\n");
   }else{
      char popped = s1.s[s1.top];
      //printf("%c is popped from the stack\n" ,s1.s[s1.top]);
      s1.top -= 1;
      return popped;
   }
}

void Top(){
   printf("Top element of stack is %c\n" ,s1.s[s1.top]);

}
```

```
int main(){
   printf("Student Name: Rudresh Monpara\n");
   printf("Student RollNo: 24BIT136\n");
   s1.top = -1;
   int i;
   char A[MAX];
   gets(A);
   for(i=0;i<=strlen(A);i++){
      push(A[i]);
   }
   printf("Reverse of String :\n");
   for(i=0;i<=strlen(A);i++){
      printf("%c",pop());
   }
   return 0;
}
```

**OUTPUT**

```
PS D:\college\DS-Sem3\Exp3> .\Program2.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Ronitt
Reverse of String :
ttinoR
```

3. Given an expression, write a program to examine whether the pairs and the orders of "{", "}", "(", ")", "[", "]" are correct in the expression or not.
Example:
Input: exp = "[( )]{}{[( )( )]( )}" Output: Balanced
Input: exp = "" Output: Not Balanced
**Code**:

```
#include <stdio.h>
#include <stdlib.h>
struct stack
{
   int size;
   int top;
   char *arr;
};

int isEmpty(struct stack *s)
{
   return s->top == -1;
}

int isFull(struct stack *s)
{
   return s->top == s->size - 1;
}

char push(struct stack *s, char value)
{
   if (isFull(s))
   {
      printf("Stack overflow\n");
      return 0;
   }
   else
   {
      s->top++;
      s->arr[s->top] = value;
      return value;
   }
}

char pop(struct stack *s)
{
   if (isEmpty(s))
   {
      printf("Stack underflown\n");
      return 0;
   }
   else
   {
      return s->arr[s->top--];
   }
```

```
}

int match(char f, char b)
{
  if (f == '{' && b == '}')
  {
    return 1;
  }
  if (f == '[' && b == ']')
  {
    return 1;
  }
  if (f == '(' && b == ')')
  {
    return 1;
  }
  return 0;
}

int parenthesisMatch(char *exp)
{
  struct stack *sp;
  sp->size = 100;
  sp->top = -1;
  sp->arr = (char *)malloc(sp->size * sizeof(char));
  char popped_ch;
  for (int i = 0; exp[i] != '\0'; i++)
  {
    if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[')
    {
      push(sp, exp[i]);
    }
    else if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']')
    {
      if (isEmpty(sp))
      {
        return 0;
      }

      popped_ch = pop(sp);
      if (!match(popped_ch, exp[i]))
      {
        return 0;
      }
    }
  }
  if (isEmpty(sp))
  {
    return 1;
  }
  else
  {
    return 0;
  }
```

```
}

int main()
{
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");

   char *exp = "[( )]{}{[( )(  )]( )}";

   if (parenthesisMatch(exp))
   {
      printf("Parenthesis Matching\n");
   }
   else
   {
      printf("Parenthesis not Matching \n");
   }

   return 0;
}
```

```
D:\college\DS-Sem3\Exp3>.\Program3.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Parenthesis Matching
```

4 **[Stack]**

1. Convert the given infix expression into postfix expression using stack.

Code

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

struct stack {
   int size;
   int top;
   char *arr;
};

int isEmpty(struct stack *s) {
   return s->top == -1;
}

int isFull(struct stack *s) {
   return s->top == s->size - 1;
}

char pop(struct stack *s) {
   if (isEmpty(s)) {
      printf("stack underflow\n");
      return 0;
   }
   return s->arr[s->top--];
}

char push(struct stack *s, char value) {
   if (isFull(s)) {
      printf("stack overflow\n");
      return -1;
   }
   s->arr[++s->top] = value;
   return value;
}

char top(struct stack *s) {
   if (isEmpty(s)) return '\0';
   return s->arr[s->top];
}

int isOperator(char a) {
   return (a == '+' || a == '-' || a == '*' || a == '/');
}

int precedence(char a) {
   if (a == '*' || a == '/')
      return 3;
```

```
    else if (a == '+' || a == '-')
        return 2;
    else if (a == '(')
        return 1; // lowest precedence
    else
        return 0;
}

char *inftopost(char *infix) {
    struct stack *sp = (struct stack *)malloc(sizeof(struct stack));
    sp->size = 100;
    sp->top = -1;
    sp->arr = (char *)malloc(sp->size * sizeof(char));

    char *postfix = (char *)malloc((strlen(infix) + 1) * sizeof(char));
    int i = 0, j = 0;

    while (infix[i] != '\0') {
        if (infix[i] == '(') {
            push(sp, infix[i]);
            i++;
        }
        else if (infix[i] == ')') {
            while (!isEmpty(sp) && top(sp) != '(') {
                postfix[j++] = pop(sp);
            }
            pop(sp); // remove '('
            i++;
        }
        else if (!isOperator(infix[i])) {
            postfix[j++] = infix[i++];
        }
        else {
            while (!isEmpty(sp) && precedence(top(sp)) >= precedence(infix[i])) {
                postfix[j++] = pop(sp);
            }
            push(sp, infix[i]);
            i++;
        }
    }

    while (!isEmpty(sp)) {
        postfix[j++] = pop(sp);
    }
    postfix[j] = '\0';
    return postfix;
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");
    char *infix = "a+b+c";
    printf("%s\n", inftopost(infix));
    return 0;
```

```
}
```

```
D:\college\DS-Sem3\Exp4>.\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
ab+c+
```

**Practice Problems:**

2. Write a program to evaluate the following given postfix expressions:

e.g. Input: 231 * +9–  Output: -4

```
#include <stdio.h>
#include <stdlib.h>
struct stack
{
   int top;
   int size;
   char *arr;
};

int isEmpty(struct stack *s)
{
   return s->top == -1;
}

int isFull(struct stack *s)
{
   return s->top == s->size - 1;
}

char pop(struct stack *s)
{
   if (isEmpty(s))
   {
      printf("stack underflow\n");
      return 0;
   }
   else
   {
      return s->arr[s->top--];
   }
}

char push(struct stack *s, char value)
{
   if (isFull(s))
   {
      printf("stack overflow\n");
      return -1;
   }
   else
   {
      s->top++;
      s->arr[s->top] = value;
      return value;
   }
}
```

```
char top(struct stack *s)
{
   return s->arr[s->top];
}

int isOperant(char a)
{
   return (a >= '0' && a <= '9');
}

int inftopost(char *exp)
{
   struct stack *s = (struct stack *)malloc(sizeof(struct stack));
   s->top = -1;
   s->size = 100;
   s->arr = (char *)malloc(s->size * sizeof(char));
   int i = 0;
   while (exp[i] != '\0')
   {
      if (isOperant(exp[i]))
      {
         push(s, exp[i]);
      }
      else
      {
         if (exp[i] == '+')
         {
            int second = pop(s) - '0';
            int first = pop(s) - '0';
            char c = first + second + '0';
            push(s, c);
         }
         else if (exp[i] == '-')
         {
            int second = pop(s) - '0';
            int first = pop(s) - '0';
            char c = first - second + '0';
            push(s, c);
         }
         else if (exp[i] == '*')
         {
            int second = pop(s) - '0';
            int first = pop(s) - '0';
            char c = first * second + '0';
            push(s, c);
         }
         else if (exp[i] == '/')
         {
            int second = pop(s) - '0';
            int first = pop(s) - '0';
            char c = first / second + '0';
            push(s, c);
         }
      }
```

```
        i++;
    }

    return pop(s) - '0';
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");
    char * exp="23+54*+";
    // char *exp = "22/";
    printf("Result:%d", inftopost(exp));
    return 0;
}
```

```
D:\college\DS-Sem3\Exp4>.\Program2.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Result:25
```

3. Convert the given infix expression into prefix expression using stack.

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

struct stack {
   int size;
   int top;
   char *arr;
};

int isEmpty(struct stack *s) {
   return s->top == -1;
}

int isFull(struct stack *s) {
   return s->top == s->size - 1;
}

char pop(struct stack *s) {
   if (isEmpty(s)) {
      return 0;
   }
   return s->arr[s->top--];
}

char push(struct stack *s, char value) {
   if (isFull(s)) {
      return -1;
   }
   s->arr[++s->top] = value;
   return value;
}

char top(struct stack *s) {
   if (isEmpty(s)) return '\0';
   return s->arr[s->top];
}

int isOperator(char a) {
   return (a == '+' || a == '-' || a == '*' || a == '/');
}

int precedence(char a) {
   if (a == '*' || a == '/')
      return 2;
   else if (a == '+' || a == '-')
      return 1;
   else
      return 0;
}
```

```
void reverse(char *exp) {
    int n = strlen(exp);
    for (int i = 0; i < n / 2; i++) {
        char temp = exp[i];
        exp[i] = exp[n - i - 1];
        exp[n - i - 1] = temp;
    }
}

char *inftopost(char *infix) {
    struct stack *sp = (struct stack *)malloc(sizeof(struct stack));
    sp->size = 100;
    sp->top = -1;
    sp->arr = (char *)malloc(sp->size * sizeof(char));

    char *postfix = (char *)malloc((strlen(infix) + 1) * sizeof(char));
    int i = 0, j = 0;

    while (infix[i] != '\0') {
        if (infix[i] == '(') {
            push(sp, infix[i]);
        }
        else if (infix[i] == ')') {
            while (!isEmpty(sp) && top(sp) != '(') {
                postfix[j++] = pop(sp);
            }
            pop(sp); // remove '('
        }
        else if (!isOperator(infix[i])) {
            postfix[j++] = infix[i];
        }
        else {
            while (!isEmpty(sp) && precedence(top(sp)) >= precedence(infix[i])) {
                postfix[j++] = pop(sp);
            }
            push(sp, infix[i]);
        }
        i++;
    }

    while (!isEmpty(sp)) {
        postfix[j++] = pop(sp);
    }
    postfix[j] = '\0';
    free(sp->arr);
    free(sp);
    return postfix;
}

char *inftopre(char *infix) {
    int n = strlen(infix);

    char *rev = (char *)malloc((n + 1) * sizeof(char));
    strcpy(rev, infix);
    reverse(rev);
```

```
    // Swap brackets
    for (int i = 0; i < n; i++) {
        if (rev[i] == '(') rev[i] = ')';
        else if (rev[i] == ')') rev[i] = '(';
    }

    char *post = inftopost(rev);

    reverse(post); // reverse postfix to get prefix

    free(rev);
    return post;
}

int main() {
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    char exp[] = "a+b+c";

    printf("Infix: %s\n", exp);
    printf("Prefix: %s\n", inftopre(exp));

    return 0;
}
```

```
D:\college\DS-Sem3\Exp4>.\Program3.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Result: +a+bc
```

5 **[Queue]**

1. Implement various functionalities of Queue using Arrays. For example: insertion, deletion, front element, rear element etc.

Code:-

```
#include <stdio.h>
#include<stdlib.h>
struct Queue
{
   int size;
   int front;
   int rear;
   int *arr;
};

int isFull(struct Queue *queue){
   if (queue->rear==queue->size)
   {
      return 1;
   }
   else{
      return 0;
   }

}

int isEmpty(struct Queue *queue){
   if (queue->front==-1)
   {
      return 1;
   }
   else{
      return 0;
   }

}

int insertion(struct Queue *queue,int value){
   if (isFull(queue))
   {
      printf("Queue is Full\n");
      return 0;
   }
   else{
      queue->rear++;
      if (queue->front==-1)
      {
         queue->front=0;
      }

      queue->arr[queue->rear]=value;
      return value;
   }

}

int deletion(struct Queue *queue){
   if (isEmpty(queue))
```

```
    {
        printf("Queue is Empty\n");
        return 0;
    }
    else{
        int value=queue->arr[queue->rear];
        queue->rear--;
        return value;
    }
}

int front(struct Queue *queue){
    if (isEmpty(queue))
    {
        printf("Queue is empty\n");
        return 0;
    }
    else{
        return queue->arr[queue->front];
    }
}

int rear(struct Queue *queue){
    if (!isEmpty(queue))
    {
        return queue->arr[queue->rear];
    }
    else{
        printf("Queue is empty\n");
        return 0;
    }
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");
    struct Queue *queue=(struct Queue*)malloc(sizeof(struct Queue));
    queue->size=100;
    queue->front=-1;
    queue->rear=-1;
    queue->arr=(int *)malloc(sizeof(int)*queue->size);
    insertion(queue,10);
    printf("Front: %d, Rear: %d\n", front(queue), rear(queue));
    printf("Value inserted %d\n",insertion(queue,20));
    printf("Value inserted %d\n",insertion(queue,40));
    printf("Value inserted %d\n",insertion(queue,39));
    printf("Value inserted %d\n",insertion(queue,30));
    printf("Front: %d, Rear: %d\n", front(queue), rear(queue));
    printf("Value deleted %d\n",deletion(queue));
    printf("Front: %d, Rear: %d\n", front(queue), rear(queue));
    return 0;
}
```

```
D:\college\DS-Sem3\Exp5>.\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
10
Value inserted 20
Value inserted 40
Value inserted 39
Value inserted 30
Value deleted 30
10
39
```

**Practice Problems:**

2. Implement Priority Queue, where every element has a priority associated with it. Perform operations like Insertion and Deletion in a priority queue.

**Code:**

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

struct Node{
   int data;
   int priority;
   struct Node *next;
};
struct Node *front = NULL;
int isEmpty(){
   return (front == NULL);
}
void insert(int data, int priority)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   new->data = data;
   new->priority = priority;
   new->next = NULL;

   if (isEmpty() || priority < front->priority)
   {
      new->next = front;
      front = new;
   }
   else
   {
      struct Node *temp = front;
      while (temp->next != NULL && temp->next->priority <= priority)
      {
         temp = temp->next;
      }
      new->next = temp->next;
      temp->next = new;
   }
   printf("Inserted: %d with priority %d\n", data, priority);
}

void delete(){
   if (isEmpty()){
      printf("Priority Queue Underflow!\n");
      return;
   }

   struct Node *temp = front;
   printf("Deleted element: %d with priority %d\n", temp->data, temp->priority);
   front = front->next;
   free(temp);
}
void display(){
```

```
    if (isEmpty()){
        printf("Priority Queue is empty.\n");
        return;
    }
    struct Node *temp = front;
    printf("Priority Queue elements:\n");
    while (temp != NULL)
    {
        printf("Data: %d  |  Priority: %d\n", temp->data, temp->priority);
        temp = temp->next;
    }
}
int main(){
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");
    insert(10, 3);
    insert(20, 2);
    insert(30, 4);
    insert(40, 1);
    insert(50, 2);
    display();
    delete();
    delete();
    display();
    return 0;
}
```

```
D:\college\DS-Sem3\Exp5>.\Program2.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Inserted: 10 with priority 3
Inserted: 20 with priority 2
Inserted: 30 with priority 4
Inserted: 40 with priority 1
Inserted: 50 with priority 2
Priority Queue elements:
Data: 40  |  Priority: 1
Data: 20  |  Priority: 2
Data: 50  |  Priority: 2
Data: 10  |  Priority: 3
Data: 30  |  Priority: 4
Deleted element: 40 with priority 1
Deleted element: 20 with priority 2
Priority Queue elements:
Data: 50  |  Priority: 2
Data: 10  |  Priority: 3
Data: 30  |  Priority: 4
```

3. Implement Double Ended Queue that supports following operation:
    a. insertFront(): Adds an item at the front of Deque.
    b. insertLast(): Adds an item at the rear of Deque.
    c. deleteFront(): Deletes an item from the front of Deque.
    d. deleteLast(): Deletes an item from the rear of Deque.

**Code**:

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

struct Deque {
   int size;
   int front;
   int rear;
   int *arr;
};

int isFull(struct Deque *dq) {
   if (dq->rear==dq->size)
   {
      return 1;
   }
   else{
      return 0;
   }
}

int isEmpty(struct Deque *dq) {
   return dq->front == -1;
}

void insertFront(struct Deque *dq, int value) {
   if (isFull(dq)) {
      printf("Deque is Full\n");
      return;
   }

   if (dq->front == -1) {
      dq->front = dq->rear = 0;
   }
   else if (dq->front == 0) {
      dq->front = dq->size - 1;
   }
   else {
      dq->front--;
   }
   dq->arr[dq->front] = value;
   printf("Inserted %d at front\n", value);
}
```

```
void insertLast(struct Deque *dq, int value) {
    if (isFull(dq)) {
        printf("Deque is Full\n");
        return;
    }

    if (dq->front == -1) {
        dq->front = dq->rear = 0;
    }
    else if (dq->rear == dq->size - 1) {
        dq->rear = 0;
    }
    else {
        dq->rear++;
    }
    dq->arr[dq->rear] = value;
    printf("Inserted %d at rear\n", value);
}

void deleteFront(struct Deque *dq) {
    if (isEmpty(dq)) {
        printf("Deque is Empty\n");
        return;
    }

    int value = dq->arr[dq->front];
    printf("Deleted %d from front\n", value);

    if (dq->front == dq->rear) {
        dq->front = dq->rear = -1;
    }
    else if (dq->front == dq->size - 1) {
        dq->front = 0;
    }
    else {
        dq->front++;
    }
}

void deleteLast(struct Deque *dq) {
    if (isEmpty(dq)) {
        printf("Deque is Empty\n");
        return;
    }

    int value = dq->arr[dq->rear];
    printf("Deleted %d from rear\n", value);
```

```
    if (dq->front == dq->rear) {
        dq->front = dq->rear = -1;
    }
    else if (dq->rear == 0) {
        dq->rear = dq->size - 1;
    }
    else {
        dq->rear--;
    }
}


int front(struct Deque *dq) {
    if (isEmpty(dq)) {
        printf("Deque is Empty\n");
        return -1;
    }
    return dq->arr[dq->front];
}

int rear(struct Deque *dq) {
    if (isEmpty(dq)) {
        printf("Deque is Empty\n");
        return -1;
    }
    return dq->arr[dq->rear];
}

int main() {
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    struct Deque *dq = (struct Deque*)malloc(sizeof(struct Deque));
    dq->size = 100;
    dq->front = dq->rear = -1;
    dq->arr = (int*)malloc(dq->size * sizeof(int));

    insertLast(dq, 10);
    insertLast(dq, 20);
    insertFront(dq, 5);
    insertLast(dq, 30);
    printf("Front: %d, Rear: %d\n", front(dq), rear(dq));
    deleteFront(dq);
    deleteLast(dq);
    printf("Front: %d, Rear: %d\n", front(dq), rear(dq));
    return 0;
}
```

```
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Inserted 10 at rear
Inserted 20 at rear
Inserted 5 at front
Inserted 30 at rear
Front: 5, Rear: 30
Deleted 5 from front
Deleted 30 from rear
Front: 10, Rear: 20
```

6 **[Linked List]**

1. Implement doubly linked list with following functions:
   a. insertAtBeginning(): Inserts a node at the end of the list
   b. insertAtEnd(): Inserts a node at the end of the list.
   c. insertAtPosition() – Inserts a node at a specific index.
   d. deleteFromBeginning(): Deletes the first node.
   e. deleteFromEnd(): Deletes the last node.
   f. deleteFromPosition(position) : Deletes a node from a specific position.
   g. forwardTraverse(): Traverses and displays the list from head to tail.
   h. backwardTraverse(): Traverses and displays the list from tail to head
   i. search: search(): Searches for a node with given data and returns its position (or index).

**Code**:

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

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

int len(struct Node *ptr)
{
   int count = 0;
   while (ptr != NULL)
   {
      count++;
      ptr = ptr->next;
   }

   return count;
}

void forwardTraversal(struct Node *ptr)
{
   while (ptr!= NULL)
   {
      printf("Data:%d\n", ptr->data);
      ptr = ptr->next;
   }

}

void backwardTraversal(struct Node *ptr){
   while (ptr->next!= NULL)
   {
      ptr = ptr->next;
   }
   while (ptr!=NULL)
   {
      printf("Data:%d\n", ptr->data);
      ptr=ptr->prev;
   }
```

```
}

struct Node *insertAtBeginning(struct Node *ptr, int data)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   new->next = ptr;
   new->data = data;
   new->prev = NULL;
   return new;
}

struct Node* insertAtEnd(struct Node *ptr,int data){
   struct Node *new=(struct Node*)malloc(sizeof(struct Node));
   new->next=NULL;
   new->data=data;
   struct Node *head=ptr;
   while (head->next!= NULL)
   {
      head = head->next;
   }
   head->next=new;
   new->prev=head;
   return ptr;
}

struct Node *insertAtPosition(struct Node *ptr, int data, int index)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   struct Node *p = ptr;
   int i = 0;
   while (i < index - 1)
   {
      p = p->next;
      i++;
   }
   new->next = p->next;
   new->prev = p;
   p->next = new;
   new->data = data;
   if (new->next != NULL)
   {
      p = new->next;
      p->prev = new;
   }
   return ptr;
}

int search(struct Node *ptr, int value)
{
   int i=0;
   while (ptr!=NULL)
   {
      i++;
      if (ptr->data==value){
         return i;
      }
      else{
         ptr = ptr->next;
      }
```

```
    }
    return -1;
}

struct Node *deleteFromBeginning(struct Node *ptr)
{
    struct Node *p = ptr;
    ptr = ptr->next;
    ptr->prev = NULL;
    free(p);

    return ptr;
}

struct Node* deleteAtEnd(struct Node *ptr){
    struct Node *head = ptr;

    while ((ptr->next)->next!= NULL)
    {
        ptr = ptr->next;
    }
    struct Node *last=ptr->next;
    free(last);
    ptr->next=NULL;

    return head;
}

struct Node *deleteFromPosition(struct Node *ptr, int index)
{
    if(len(ptr)<index){
        printf("Invalid index\n");
        return ptr;
    }
    else{
        struct Node *p = ptr;
        struct Node *q = ptr->next;
        for (int i = 1; i < index-1 ; i++)
        {
            printf("yes\n");
            p = p->next;
            q = q->next;
        }
        p->next = q->next;
        if (q->next!=NULL)
        {
            (q->next)->prev=p;
        }


        free(q);
        return ptr;
    }
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");
```

```
    struct Node *head;
    struct Node *second;
    struct Node *third;

    head = (struct Node *)malloc(sizeof(struct Node));
    second = (struct Node *)malloc(sizeof(struct Node));
    third = (struct Node *)malloc(sizeof(struct Node));

    head->data = 1;
    head->next = second;
    head->prev = NULL;

    second->data = 2;
    second->next = third;
    second->prev = head;

    third->data = 99;
    third->prev = second;
    third->next = NULL;
    head=insertAtBeginning(head,0);
    head=insertAtEnd(head,10);
    head=insertAtPosition(head,100,3);
    forwardTraversal(head);
    printf("The index of 99 is %d",search(head,99));
    head=deleteFromBeginning(head);
    head=deleteFromPosition(head,3);
    head=deleteAtEnd(head);
    backwardTraversal(head);
    return 0;
}
```

```
D:\college\DS-Sem3\Exp6>.\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Data:0
Data:1
Data:2
Data:100
Data:99
Data:10
The index of 99 is 5yes
Data:99
Data:2
Data:1
```

**Practice Problems:**

2. Implement Stack and Queue using linked list with suitable functions (e.g. insert, delete, checking stack/queue is empty or full).

**Code(**Stack)

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

struct Node
{
   int data;
   struct Node *next;
};
void push(struct Node **top, int val)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   if (new == NULL)
   {
      printf("stack overflow\n");
   }
   else
   {
      new->data = val;
      new->next = *top;
      *top = new;
      printf("Element %d pushed successfully\n", new->data);
   }
}

void pop(struct Node **top)
{
   if (*top == NULL)
   {
      printf("Stack underflown\n");
   }
   else
   {
      struct Node *temp = *top;
      *top = (*top)->next;
      printf("Element %d popped succesfuly\n", temp->data);
      free(temp);
   }
}

void peek(struct Node *top)
{
   if (top == NULL)
   {
      printf("Stack underflown\n");
   }
   else
   {
      printf("%d is the top element\n",top->data);
   }
```

```
}

void display(struct Node *top)
{
   if (top == NULL)
   {
      printf("Stack underflown\n");
   }
   else
   {
      struct Node *temp = top;
      printf("Stack elements:");
      while (temp != NULL)
      {
         printf("%d\t", temp->data);
         temp = temp->next;
      }
      printf("\n");
   }
}

int main()
{
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   struct Node *top = NULL;
   push(&top, 10);
   push(&top, 20);
   push(&top, 30);
   push(&top, 40);
   push(&top, 50);
   push(&top, 60);
   display(top);
   pop(&top);
   pop(&top);
   display(top);
   peek(top);
   return 0;
}
```

```
D:\college\DS-Sem3\Exp6>.\Program2a.exe
Element 10 pushed successfully
Element 20 pushed successfully
Element 30 pushed successfully
Element 40 pushed successfully
Element 50 pushed successfully
Element 60 pushed successfully
Stack elements:60       50      40      30      20      10
Element 60 popped succesfuly
Element 50 popped succesfuly
Stack elements:40       30      20      10
40 is the top element
```

**Code:(Queue)**

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
void enqueue(struct Node** front,struct Node** rear,int val){
   struct Node* new=(struct Node*)malloc(sizeof(struct Node));
   if(new==NULL)
   {
      printf("Queue overflow\n");
   }
   else{
      new->data=val;
      new->next=NULL;
      if (*rear==NULL)
      {
         *front=new;
         *rear=new;
      }
      else
      {
         (*rear)->next=new;
         *rear=new;
      }
      printf("value:%d enqued\n",new->data);
   }
}
void dequeue(struct Node** front,struct Node** rear){
   if (*front==NULL)
   {
      printf("Queue underflown\n");
   }
   else{
      struct Node* temp=*front;
      *front=(*front)->next;
      if (*front==NULL)
      {
         *rear=NULL;
      }
      printf("Value:%d dequed\n",temp->data);
      free(temp);
   }
}
void peek(struct Node *front)
{
   if (front == NULL)
   {
      printf("Queue underflown\n");
   }
   else
```

```
    {
        printf("%d is the front element\n",front->data);
    }
}
void display(struct Node *front)
{
    if (front == NULL)
    {
        printf("Queue underflown\n");
    }
    else
    {
        struct Node *temp = front;
        printf("Queue elements:");
        while (temp != NULL)
        {
            printf("%d\t", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main(){
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");
    struct Node* front=NULL;
    struct Node* rear=NULL;
    enqueue(&front,&rear,10);
    enqueue(&front,&rear,20);
    enqueue(&front,&rear,30);
    enqueue(&front,&rear,40);
    enqueue(&front,&rear,50);
    display(front);
    dequeue(&front,&rear);
    dequeue(&front,&rear);
    display(front);
    peek(front);
    return 0;
}
```

```
D:\college\DS-Sem3\Exp6>.\Program2b.exe
value:10 enqued
value:20 enqued
value:30 enqued
value:40 enqued
value:50 enqued
Queue elements:10       20      30      40      50
Value:10 dequed
Value:20 dequed
Queue elements:30       40      50
30 is the front element
```

3. Implement Circular and Double ended queues using linked list with suitable functions.

**Code**: (Circular)

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

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

struct Node *rear = NULL;
struct Node *front = NULL;

int isEmpty()
{
   return (front == NULL);
}

void enqueue(int data)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   new->data = data;
   if (front == NULL)
   {
      front = new;
      rear = new;
      new->next = front;
   }
   else
   {
      rear->next = new;
      rear = new;
      rear->next = front;
   }
   printf("Enqueued: %d\n", data);
}

void dequeue()
{
   if (isEmpty())
   {
      printf("Queue Underflow!\n");
      return;
   }
   struct Node *temp = front;
   int val = temp->data;

   if (front == rear)
   {
      front = NULL;
```

```
        rear = NULL;
    }
    else
    {
        front = front->next;
        rear->next = front;
    }
    free(temp);
    printf("Dequeued: %d\n", val);
}

void display()
{
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return;
    }
    struct Node *temp = front;
    printf("Queue elements:\n");
    do
    {
        printf("Data: %d\n", temp->data);
        temp = temp->next;
    } while (temp != front);
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);

    display();

    dequeue();
    dequeue();

    display();

    enqueue(50);
    display();

    return 0;
}
```

```
D:\college\DS-Sem3\Exp6>.\Program3a.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enqueued: 10
Enqueued: 20
Enqueued: 30
Enqueued: 40
Queue elements:
Data: 10
Data: 20
Data: 30
Data: 40
Dequeued: 10
Dequeued: 20
Queue elements:
Data: 30
Data: 40
Enqueued: 50
Queue elements:
Data: 30
Data: 40
Data: 50
```

Code:(Dequeue)

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

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

struct Node *front = NULL;
struct Node *rear = NULL;

int isEmpty()
{
   return (front == NULL);
}

void insertFront(int data)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   new->data = data;
   new->prev = NULL;
   new->next = front;
```

```
    if (isEmpty())
    {
        rear = new;
    }
    else
    {
        front->prev = new;
    }
    front = new;
    printf("Inserted at Front: %d\n", data);
}

void insertRear(int data)
{
    struct Node *new = (struct Node *)malloc(sizeof(struct Node));
    new->data = data;
    new->next = NULL;
    new->prev = rear;

    if (isEmpty())
    {
        front = new;
    }
    else
    {
        rear->next = new;
    }
    rear = new;
    printf("Inserted at Rear: %d\n", data);
}

void deleteFront()
{
    if (isEmpty())
    {
        printf("Deque Underflow at Front!\n");
        return;
    }
    struct Node *temp = front;
    int val = temp->data;

    front = front->next;
    if (front == NULL)
    {
        rear = NULL;
    }
    else
    {
        front->prev = NULL;
    }
    free(temp);
    printf("Deleted from Front: %d\n", val);
}
```

```
void deleteRear()
{
   if (isEmpty())
   {
      printf("Deque Underflow at Rear!\n");
      return;
   }
   struct Node *temp = rear;
   int val = temp->data;

   rear = rear->prev;
   if (rear == NULL)
   {
      front = NULL;
   }
   else
   {
      rear->next = NULL;
   }
   free(temp);
   printf("Deleted from Rear: %d\n", val);
}

void displayForward()
{
   if (isEmpty())
   {
      printf("Deque is empty\n");
      return;
   }
   struct Node *temp = front;
   printf("Deque (Front -> Rear):\n");
   while (temp != NULL)
   {
      printf("Data: %d\n", temp->data);
      temp = temp->next;
   }
}

void displayBackward()
{
   if (isEmpty())
   {
      printf("Deque is empty\n");
      return;
   }
   struct Node *temp = rear;
   printf("Deque (Rear -> Front):\n");
   while (temp != NULL)
   {
      printf("Data: %d\n", temp->data);
      temp = temp->prev;
   }
```

```
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    insertRear(10);
    insertRear(20);
    insertFront(5);
    insertFront(1);

    displayForward();

    deleteFront();
    deleteRear();

    displayBackward();

    insertRear(50);
    displayForward();

    return 0;
}
```

```
D:\college\DS-Sem3\Exp6>.\Program3b.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Inserted at Rear: 10
Inserted at Rear: 20
Inserted at Front: 5
Inserted at Front: 1
Deque (Front -> Rear):
Data: 1
Data: 5
Data: 10
Data: 20
Deleted from Front: 1
Deleted from Rear: 20
Deque (Rear -> Front):
Data: 10
Data: 5
Inserted at Rear: 50
Deque (Front -> Rear):
Data: 5
Data: 10
Data: 50
```

4. Write a program that takes ***two sorted lists*** as inputs and merge them into one sorted list.
   For example, if the first linked list *A* is 5 =>10 =>15, and the other linked list *B* is 2 => 3 => 20, then output should be 2 => 3 => 5 => 10 => 15 => 20.

Code:

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

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

void traversal(struct Node *ptr)
{
   while (ptr != NULL)
   {
      printf("Data: %d\n", ptr->data);
      ptr = ptr->next;
   }
}

struct Node *insertAtEnd(struct Node *ptr, int data)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   new->data = data;
   new->next = NULL;

   if (ptr == NULL)
   {
      return new;
   }

   struct Node *p = ptr;
   while (p->next != NULL)
   {
      p = p->next;
   }
   p->next = new;
   return ptr;
}

struct Node *mergeSortedLists(struct Node *a, struct Node *b)
{
   if (a == NULL)
      return b;
   if (b == NULL)
      return a;

   struct Node *result = NULL;
```

```
    if (a->data <= b->data)
    {
        result = a;
        result->next = mergeSortedLists(a->next, b);
    }
    else
    {
        result = b;
        result->next = mergeSortedLists(a, b->next);
    }

    return result;
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    struct Node *A = NULL;
    struct Node *B = NULL;
    struct Node *merged = NULL;

    //5 -> 10 -> 15
    A = insertAtEnd(A, 5);
    A = insertAtEnd(A, 10);
    A = insertAtEnd(A, 15);

    //2 -> 3 -> 20
    B = insertAtEnd(B, 2);
    B = insertAtEnd(B, 3);
    B = insertAtEnd(B, 20);

    printf("List A:\n");
    traversal(A);

    printf("\nList B:\n");
    traversal(B);

    merged = mergeSortedLists(A, B);

    printf("\nMerged Sorted List:\n");
    traversal(merged);

    return 0;
}
```

```
D:\college\DS-Sem3\Exp6>.\Program4.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
List A:
Data: 5
Data: 10
Data: 15

List B:
Data: 2
Data: 3
Data: 20

Merged Sorted List:
Data: 2
Data: 3
Data: 5
Data: 10
Data: 15
Data: 20
```

7 **[Trees]**

1. Implement binary search tree with recursive functions: insert(),inorderTraverse(), preorderTraverse(), postorderTraverse(), search() and findHeight().

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

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

struct Node* createNode(int data){
   struct Node* new=(struct Node*)malloc(sizeof(struct Node));
   new->data=data;
   new->left=NULL;
   new->right=NULL;
   return new;
}

struct Node * insert(struct Node* root,int val){
   if (root==NULL)
   {
      return createNode(val);
   }
   if (root->data>val)
   {
      root->left=insert(root->left,val);
   }
   else if (root->data<val)
   {
      root->right=insert(root->right,val);
   }
   return root;
}

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

}
```

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

}

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

}

struct Node* search(struct Node* root, int val) {
   if (root == NULL || root->data == val) {
      return root;
   }
   if (val < root->data)
      return search(root->left, val);
   else
      return search(root->right, val);
}

int findHeight(struct Node* root){
   if (root==NULL)
      return -1;
   int leftHeight=findHeight(root->left);
   int rightHeight=findHeight(root->right);
   if(leftHeight>rightHeight) return leftHeight+1;
   else return rightHeight+1;

}

int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
   struct  Node * root=createNode(5);
   struct  Node * p1=createNode(3);
   struct  Node * p2=createNode(6);
   struct  Node * p3=createNode(1);
```

```
    struct  Node * p4=createNode(4);
    root->left=p1;
    root->right=p2;
    p1->left=p3;
    p1->right=p4;
    printf("InOrder Traversal:  ");
    inorder(root);
    printf("\n");
    printf("PreOrder Traversal: ");
    preorder(root);
    printf("\n");
    printf("PostOrder Traversal:");
    postorder(root);
    printf("\n");

    root=insert(root,10);
    root=insert(root,9);
    root=insert(root,7);


    printf("InOrder Traversal after insertion:");
    inorder(root);
    printf("\n");
    printf("%d searching in tree.\n",70);
    if (search(root,70)!=NULL)
    {
       printf("value found\n");
    }
    else{
       printf("value not found.\n");
    }
    printf("Height of tree is %d\n",findHeight(root));
    return 0;
}
```

```
D:\college\DS-Sem3\Exp7>.\Program1.exe
InOrder Traversal:  1   3       4       5       6
PreOrder Traversal: 5   3       1       4       6
PostOrder Traversal:1   4       3       6       5
InOrder Traversal after insertion:1     3       4       5       6       7       9       10
70 searching in tree.
value not found.
Height of tree is 4
```

**Practice Problems:**

2. Implement binary search tree with non-recursive functions: insert(), inorderTraverse(), preorderTraverse(), postorderTraverse(), countNodes(), and countLeafNodes().

CODE:

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

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

struct Node *createNode(int data)
{
   struct Node *new = (struct Node *)malloc(sizeof(struct Node));
   new->data = data;
   new->left = NULL;
   new->right = NULL;
   return new;
}

struct Node *insert(struct Node *root, int val)
{
   if (root == NULL)
   {
      return createNode(val);
   }
   struct Node *current = root;
   while (current != NULL)
   {
      if (val < current->data)
      {
         if (current->left == NULL)
         {
            current->left = createNode(val);
            return root;
         }
         current = current->left;
      }
      else if (val > current->data)
      {
         if (current->right == NULL)
         {
            current->right = createNode(val);
            return root;
         }
         current = current->right;
      }
      else
```

```
        {
            return root;
        }
    }
}

void inorder(struct Node *root)
{
    struct Node *stack[100];
    int top = -1;
    struct Node *current = root;
    while (current != NULL || top != -1)
    {
        while (current != NULL)
        {
            stack[++top] = current;
            current = current->left;
        }
        current = stack[top--];
        printf("%d\t", current->data);
        current = current->right;
    }
}

void preorder(struct Node *root)
{
    if (root == NULL)
        return;
    struct Node *stack[100];
    int top = -1;
    stack[++top] = root;
    while (top != -1)
    {
        struct Node *current = stack[top--];
        printf("%d\t", current->data);

        if (current->right != NULL)
            stack[++top] = current->right;
        if (current->left != NULL)
            stack[++top] = current->left;
    }
}

void postorder(struct Node* root) {
    if (root == NULL) return;

    struct Node* stack1[100], *stack2[100];
    int top1 = -1, top2 = -1;

    stack1[++top1] = root;

    while (top1 != -1) {
        struct Node* current = stack1[top1--];
        stack2[++top2] = current;
```

```
        if (current->left != NULL)
            stack1[++top1] = current->left;
        if (current->right != NULL)
            stack1[++top1] = current->right;
    }

    while (top2 != -1) {
        printf("%d\t", stack2[top2--]->data);
    }
}

int countNodes(struct Node* root) {
    if (root == NULL) return 0;

    int count = 0;
    struct Node* stack[100];
    int top = -1;
    stack[++top] = root;

    while (top != -1) {
        struct Node* current = stack[top--];
        count++;
        if (current->right != NULL)
            stack[++top] = current->right;
        if (current->left != NULL)
            stack[++top] = current->left;
    }
    return count;
}

int countLeafNodes(struct Node* root) {
    if (root == NULL) return 0;

    int count = 0;
    struct Node* stack[100];
    int top = -1;
    stack[++top] = root;

    while (top != -1) {
        struct Node* current = stack[top--];
        if (current->left == NULL && current->right == NULL)
            count++;
        if (current->right != NULL)
            stack[++top] = current->right;
        if (current->left != NULL)
            stack[++top] = current->left;
    }
    return count;
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");
    struct Node *root = createNode(5);
```

```
    struct Node *p1 = createNode(3);
    struct Node *p2 = createNode(6);
    struct Node *p3 = createNode(1);
    struct Node *p4 = createNode(4);
    root->left = p1;
    root->right = p2;
    p1->left = p3;
    p1->right = p4;
    insert(root, 2);
    printf("InOrder Traversal:  ");
    inorder(root);
    printf("\n");
    printf("preOrder Traversal:  ");
    preorder(root);
    printf("\n");

    printf("postOrder Traversal:  ");
    postorder(root);
    printf("\n");
}
```

```
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
InOrder Traversal:  1   2       3       4       5       6
preOrder Traversal:  5  3       1       2       4       6
postOrder Traversal:  2 1       4       3       6       5
```

8 **[Graphs]**

1. Implement Minimum Spanning Tree (MST) using the greedy Kruskal's algorithm

**Code**:

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

struct Edge
{
   int src;
   int dest;
   int weight;
};

int parent[20];

int findParent(int i)
{
   if (parent[i] == i)
      return i;
   return parent[i] = findParent(parent[i]);
}

void unionSet(int a, int b)
{
   int rootA = findParent(a);
   int rootB = findParent(b);
   if (rootA != rootB)
      parent[rootB] = rootA;
}

int compare(const void *a, const void *b)
{
   struct Edge *edgeA = (struct Edge *)a;
   struct Edge *edgeB = (struct Edge *)b;
   return edgeA->weight - edgeB->weight;
}

void makeSet(int V)
{
   for (int i = 0; i < V; i++)
      parent[i] = i;
}

void kruskalMST(struct Edge edges[], int V, int E)
{
   qsort(edges, E, sizeof(struct Edge), compare);
   makeSet(V);
```

```
    struct Edge result[V - 1];
    int edgeCount = 0;
    int totalCost = 0;

    for (int i = 0; i < E && edgeCount < V - 1; i++){
        int srcRoot = findParent(edges[i].src);
        int destRoot = findParent(edges[i].dest);

        if (srcRoot != destRoot)
        {
            result[edgeCount++] = edges[i];
            unionSet(srcRoot, destRoot);
        }
    }

    printf("Edges in the Minimum Spanning Tree:\n");
    for (int i = 0; i < edgeCount; i++){
        printf("%d -- %d (Weight: %d)\n", result[i].src, result[i].dest, result[i].weight);
        totalCost += result[i].weight;
    }

    printf("Total Minimum Cost: %d\n", totalCost);
}

int main(){
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    int V = 7;
    int E = 9;

    struct Edge edges[9] = {
        {0, 1, 2},
        {0, 3, 4},
        {1, 2, 3},
        {1, 4, 7},
        {2, 3, 5},
        {2, 5, 8},
        {3, 6, 1},
        {4, 5, 6},
        {5, 6, 9}
    };

    printf("Minimum Spanning Tree using Kruskal's Algorithm\n");
    printf("Number of Vertices: %d\n", V);
    printf("Number of Edges: %d\n\n", E);

    kruskalMST(edges, V, E);
    return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp8> .\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Minimum Spanning Tree using Kruskal's Algorithm
Number of Vertices: 7
Number of Edges: 9

Edges in the Minimum Spanning Tree:
3 -- 6 (Weight: 1)
0 -- 1 (Weight: 2)
1 -- 2 (Weight: 3)
0 -- 3 (Weight: 4)
4 -- 5 (Weight: 6)
1 -- 4 (Weight: 7)
Total Minimum Cost: 23
```

**Practice Problems:**

2. For a given graph $G = (V, E)$, study and implement the Breadth First Search and Depth First Search.

**Code**:

```
#include <stdio.h>

#define MAX 20

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

// Function to add element to queue
void enqueue(int v)
{
   if (rear == MAX - 1)
      return;
   if (front == -1)
      front = 0;
   rear++;
   queue[rear] = v;
}

int dequeue()
{
   if (front == -1 || front > rear)
      return -1;
   int v = queue[front];
   front++;
   return v;
}
void BFS(int adj[MAX][MAX], int n, int start)
```

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

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

    printf("BFS Traversal starting from vertex %d: ", start);
    while (front <= rear)
    {
        int node = dequeue();
        printf("%d ", node);

        for (int j = 0; j < n; j++)
        {
            if (adj[node][j] == 1 && !visited[j])
            {
                enqueue(j);
                visited[j] = 1;
            }
        }
    }
    printf("\n");
}

void DFSUtil(int adj[MAX][MAX], int n, int v)
{
    printf("%d ", v);
    visited[v] = 1;

    for (int i = 0; i < n; i++)
    {
        if (adj[v][i] == 1 && !visited[i])
            DFSUtil(adj, n, i);
    }
}

void DFS(int adj[MAX][MAX], int n, int start)
{
    for (int i = 0; i < n; i++)
        visited[i] = 0;

    printf("DFS Traversal starting from vertex %d: ", start);
    DFSUtil(adj, n, start);
    printf("\n");
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
```

```
    printf("Student RollNo: 24BIT100\n");

    int n, adj[MAX][MAX], start;

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            scanf("%d", &adj[i][j]);
        }
    }

    printf("Enter starting vertex (0 - %d): ", n - 1);
    scanf("%d", &start);

    printf("\n");
    BFS(adj, n, start);
    DFS(adj, n, start);

    return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp8> .\Program2.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter number of vertices: 5
Enter adjacency matrix (5 x 5):
0 1 1 0 0
1 0 0 1 0
1 0 0 1 1
0 1 1 0 1
0 0 1 1 0
Enter starting vertex (0 - 4): 0

BFS Traversal starting from vertex 0: 0 1 2 3 4
DFS Traversal starting from vertex 0: 0 1 3 2 4
```

3. Given a directed graph, check whether the graph contains a cycle or not. Your function should return true if t
   contains at least one cycle, else return false. Perform same task for undirected graph as well.

**Code**:

```
#include <stdio.h>

#define MAX 20

int adj[MAX][MAX];
int visited[MAX];
int recStack[MAX];  // Used for directed graph cycle detection

int isCyclicDirectedUtil(int v, int n)
{
   visited[v] = 1;
   recStack[v] = 1;

   for (int i = 0; i < n; i++)
   {
      if (adj[v][i])
      {
         if (!visited[i] && isCyclicDirectedUtil(i, n))
            return 1;
         else if (recStack[i])
            return 1;
      }
   }

   recStack[v] = 0;
   return 0;
}

int isCyclicDirected(int n)
{
   for (int i = 0; i < n; i++)
   {
      visited[i] = 0;
      recStack[i] = 0;
   }

   for (int i = 0; i < n; i++)
   {
      if (!visited[i])
      {
         if (isCyclicDirectedUtil(i, n))
            return 1;
      }
   }
   return 0;
}
int isCyclicUndirectedUtil(int v, int visited[], int parent, int n)
{
   visited[v] = 1;
```

```
    for (int i = 0; i < n; i++)
    {
        if (adj[v][i])
        {
            if (!visited[i])
            {
                if (isCyclicUndirectedUtil(i, visited, v, n))
                    return 1;
            }
            else if (i != parent)
            {
                return 1;
            }
        }
    }
    return 0;
}

int isCyclicUndirected(int n)
{
    for (int i = 0; i < n; i++)
        visited[i] = 0;

    for (int i = 0; i < n; i++)
    {
        if (!visited[i])
        {
            if (isCyclicUndirectedUtil(i, visited, -1, n))
                return 1;
        }
    }
    return 0;
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    int n, choice;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            scanf("%d", &adj[i][j]);
        }
    }
```

```
    printf("\nChoose Graph Type:\n1. Directed Graph\n2. Undirected Graph\nEnter your
choice: ");
    scanf("%d", &choice);

    int result = 0;
    if (choice == 1){
        result = isCyclicDirected(n);
        if (result)
            printf("\nCycle detected in the Directed Graph.\n");
        else
            printf("\nNo cycle found in the Directed Graph.\n");
    }
    else if (choice == 2){
        result = isCyclicUndirected(n);
        if (result)
            printf("\nCycle detected in the Undirected Graph.\n");
        else
            printf("\nNo cycle found in the Undirected Graph.\n");
    }
    else{
        printf("Invalid choice!\n");
    }
    return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp8> .\Program3.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter number of vertices: 4
Enter adjacency matrix (4 x 4):
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0

Choose Graph Type:
1. Directed Graph
2. Undirected Graph
Enter your choice: 2

No cycle found in the Undirected Graph.
```

4. Implement Minimum Spanning Tree (MST) using the Prim's algorithm.

**Code**:

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

#define MAX 20
```

```
int findMinKey(int key[], int mstSet[], int n){
   int min = INT_MAX, minIndex;
   for (int v = 0; v < n; v++){
      if (mstSet[v] == 0 && key[v] < min){
         min = key[v];
         minIndex = v;
      }
   }
   return minIndex;
}
void printMST(int parent[], int graph[MAX][MAX], int n){
   int totalCost = 0;
   printf("Edges in the Minimum Spanning Tree:\n");
   for (int i = 1; i < n; i++){
      printf("%d -- %d (Weight: %d)\n", parent[i], i, graph[i][parent[i]]);
      totalCost += graph[i][parent[i]];
   }
   printf("Total Minimum Cost: %d\n", totalCost);
}
void primMST(int graph[MAX][MAX], int n){
   int parent[MAX];
   int key[MAX];
   int mstSet[MAX];

   for (int i = 0; i < n; i++){
      key[i] = INT_MAX;
      mstSet[i] = 0;
   }
   key[0] = 0;
   parent[0] = -1;
   for (int count = 0; count < n - 1; count++)
   {
      int u = findMinKey(key, mstSet, n);
      mstSet[u] = 1;

      for (int v = 0; v < n; v++)
      {
         if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
         {
            parent[v] = u;
            key[v] = graph[u][v];
         }
      }
   }
   printMST(parent, graph, n);
}
int main(){
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");
```

```
    int n;
    int graph[MAX][MAX];
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix (%d x %d):\n", n, n);
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            scanf("%d", &graph[i][j]);
        }
    }
    printf("\nMinimum Spanning Tree using Prim's Algorithm\n");
    primMST(graph, n);
    return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp8> .\Program4.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter number of vertices: 5
Enter adjacency matrix (5 x 5):
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0

Minimum Spanning Tree using Prim's Algorithm
Edges in the Minimum Spanning Tree:
0 -- 1 (Weight: 2)
1 -- 2 (Weight: 3)
0 -- 3 (Weight: 6)
1 -- 4 (Weight: 5)
Total Minimum Cost: 16
```

9 **[Sorting]**

1. Implement Insertion sort to arrange the numbers in ascending order.

**Code**:

```
#include <stdio.h>

void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++)
   {
      key = arr[i];
      j = i - 1;

      // Move elements greater than key to one position ahead
      while (j >= 0 && arr[j] > key)
      {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}

void printArray(int arr[], int n)
{
   for (int i = 0; i < n; i++)
   {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main()
{
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");

   int arr[10];
   int n;

   printf("Enter number of elements: ");
   scanf("%d", &n);

   printf("Enter %d elements:\n", n);
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }

   printf("\nOriginal Array:\n");
   printArray(arr, n);

   insertionSort(arr, n);
```

```
    printf("\nArray after Insertion Sort (Ascending Order):\n");
    printArray(arr, n);

    return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp9> .\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter number of elements: 10
Enter 10 elements:
9 22 1 6 99 4 5 3 100 100000

Original Array:
9 22 1 6 99 4 5 3 100 100000

Array after Insertion Sort (Ascending Order):
1 3 4 5 6 9 22 99 100 100000
```

2. Implement Quick sort to arrange the numbers in ascending order.

**Code:**

```
#include <stdio.h>

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j < high; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
```

```
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int n){
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main(){
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    int n, arr[50];

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }

    printf("\nOriginal Array:\n");
    printArray(arr, n);

    quickSort(arr, 0, n - 1);

    printf("\nArray after Quick Sort (Ascending Order):\n");
    printArray(arr, n);

    return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp9> .\Program2.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter number of elements: 7
Enter 7 elements:
9 1 5 3 8 2 0

Original Array:
9 1 5 3 8 2 0

Array after Quick Sort (Ascending Order):
0 1 2 3 5 8 9
```

**Practice Problems:**

3. Implement Bubble sort to arrange the numbers in ascending order.

**Code**:

```
#include <stdio.h>

void bubbleSort(int arr[], int n)
{
   for (int i = 0; i < n - 1; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (arr[j] > arr[j + 1])
         {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

void printArray(int arr[], int n)
{
   for (int i = 0; i < n; i++)
   {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main()
{
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");

   int n, arr[50];

   printf("Enter number of elements: ");
   scanf("%d", &n);

   printf("Enter %d elements:\n", n);
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }

   printf("\nOriginal Array:\n");
   printArray(arr, n);

   bubbleSort(arr, n);

   printf("\nArray after Bubble Sort (Ascending Order):\n");
   printArray(arr, n);
```

```
    return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp9> .\Program3.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter number of elements: 7
Enter 7 elements:
9 1 5 3 8 2 0

Original Array:
9 1 5 3 8 2 0

Array after Bubble Sort (Ascending Order):
0 1 2 3 5 8 9
```

4. Implement Selection sort to arrange the numbers in descending order.

**Code**:

```
#include <stdio.h>

void selectionSort(int arr[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        int maxIndex = i;
        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] > arr[maxIndex])
            {
                maxIndex = j;
            }
        }

        int temp = arr[i];
        arr[i] = arr[maxIndex];
        arr[maxIndex] = temp;
    }
}

void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
```

```
    printf("Student RollNo: 24BIT100\n");

    int n, arr[50];

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    printf("\nOriginal Array:\n");
    printArray(arr, n);

    selectionSort(arr, n);

    printf("\nArray after Selection Sort (Descending Order):\n");
    printArray(arr, n);

    return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp9> .\Program4.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter number of elements: 7
Enter 7 elements:
9 1 5 3 8 2 0

Original Array:
9 1 5 3 8 2 0

Array after Selection Sort (Descending Order):
9 8 5 3 2 1 0
```

5. Implement Merge sort to arrange the numbers in ascending order

**Code**:

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right)
{
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
```

```
    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right)
{
    if (left < right)
    {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

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

```c
int main()
{
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");

   int n, arr[50];

   printf("Enter number of elements: ");
   scanf("%d", &n);

   printf("Enter %d elements:\n", n);
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }

   printf("\nOriginal Array:\n");
   printArray(arr, n);

   mergeSort(arr, 0, n - 1);

   printf("\nArray after Merge Sort (Ascending Order):\n");
   printArray(arr, n);

   return 0;
}
```

```
(base) PS D:\college\DS-Sem3\Exp9> .\Program5.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter number of elements: 7
Enter 7 elements:
9 1 5 3 8 2 0

Original Array:
9 1 5 3 8 2 0

Array after Merge Sort (Ascending Order):
0 1 2 3 5 8 9
```

10 **[Searching]**

1. Implement Linear search to search a number from a sorted array.

```
#include <stdio.h>

int linearSearch(int arr[10], int x)
{
   for (int i = 0; i < 10; i++)
   {
      if (arr[i] == x)
      {
         printf("%d found in the array.\n", x);
         return i;
      }
   }
   return -1;
}

int main()
{
   printf("Student Name: Ronit Kundnani\n");
   printf("Student RollNo: 24BIT100\n");

   int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
   int x;

   printf("Enter any number to search: ");
   scanf("%d", &x);

   int index = linearSearch(arr, x);

   if (index == -1)
   {
      printf("Value not found in the array.\n");
   }
   else
   {
      printf("Value found at index %d.\n", index);
   }

   return 0;
}
```

```
D:\college\DS-Sem3\Exp10>.\Program1.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter any number to search: 8
8 found in the array.
Value found at index 8.
```

2. Implement Binary search to search a number from a sorted array.

**CODE:**

```
#include <stdio.h>
int binarySearch(int arr[10], int size, int x){
    int st = 0;
    int end = size - 1;
    int mid;
    while (st <= end){
        mid = (st + end) / 2;
        if (arr[mid] == x){
            printf("Value found at index %d\n", mid);
            return mid;
        }
        else if (x < arr[mid]){
            end = mid - 1;
        }
        else{
            st = mid + 1;
        }
    }
    return -1;
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    int arr[10] = {10, 20, 30, 40, 50, 60, 69, 70, 80, 96};
    int x;

    printf("Enter the value you want to find: ");
    scanf("%d", &x);

    int index = binarySearch(arr, 10, x);

    if (index == -1)
    {
        printf("Value not found in the array.\n");
    }

    return 0;
}
```

```
D:\college\DS-Sem3\Exp10>.\Program2.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter the value you want to find: 60
Value found at index 5
```

**Practice Problem:**

3. Given a sorted array of integers (in descending order), writea C program to find the first and last occurrence of a given number x. If the element is not found, print suitable message.

CODE:

```
#include <stdio.h>

int firstOccurrence(int arr[], int size, int x)
{
   int st = 0;
   int end = size - 1;
   int result = -1;

   while (st <= end)
   {
      int mid = (st + end) / 2;

      if (arr[mid] == x)
      {
         result = mid;
         st = mid + 1;
      }
      else if (arr[mid] < x)
      {
         end = mid - 1;
      }
      else
      {
         st = mid + 1;
      }
   }
   return result;
}

int lastOccurrence(int arr[], int size, int x)
{
   int st = 0;
   int end = size - 1;
   int result = -1;

   while (st <= end)
   {
      int mid = (st + end) / 2;

      if (arr[mid] == x)
      {
         result = mid;
         // Move left to find later occurrence in descending order
         end = mid - 1;
      }
      else if (arr[mid] < x)
      {
```

```
            end = mid - 1;
        }
        else
        {
            st = mid + 1;
        }
    }
    return result;
}

int main()
{
    printf("Student Name: Ronit Kundnani\n");
    printf("Student RollNo: 24BIT100\n");

    int arr[] = {99, 90, 90, 80, 75, 60, 60, 45, 30, 10};
    int size = sizeof(arr) / sizeof(arr[0]);
    int x;

    printf("Enter the value you want to find: ");
    scanf("%d", &x);

    int first = firstOccurrence(arr, size, x);
    int last = lastOccurrence(arr, size, x);

    if (first == -1 && last == -1)
    {
        printf("Value %d not found in the array.\n", x);
    }
    else
    {
        printf("First occurrence of %d is at index %d\n", x, first);
        printf("Last occurrence of %d is at index %d\n", x, last);
    }

    return 0;
}
```

```
D:\college\DS-Sem3\Exp10>.\Program3.exe
Student Name: Ronit Kundnani
Student RollNo: 24BIT100
Enter the value you want to find: 60
First occurrence of 60 is at index 6
Last occurrence of 60 is at index 5
```

11

**Experiment No. 11, 12, 13 [Mini Project and Problem Solving]**

These laboratory sessions are dedicated for doing a mini project and doubt solving sessions. Students are expected to do a mini project by forming a group of 5 on the concepts they have learned in data structures. They are free to select any mini project as per their choice. If they are not able to decide, they can select any one from the following:

1. **Hospital Patient Record System:** This system manages patient information in a hospital. It stores patient records, including ID, name, age, disease, date of admission, and status (admitted/discharged). You can implement: Add patient details, Search patient by ID, Discharge patient, View all current admitted patients

   Data Structures Used: Linked Lists (for dynamic record handling), Queues (for patient appointments)
2. **Bank Management System:** This project simulates basic banking operations: Account creation, Deposit/withdrawal, Balance check, Displaying account details. Each account can be stored in a hash table or binary search tree to allow fast searching and updates.
3. **College Admission Management System:** Simulate the admission process of students into different courses based on merit and availability. Features: Student registration, Merit list sorting, Course seat allocation, Waiting list generation Data Structures Used: Arrays and Structures (for sorting using marks), Queues (for waitlist), Priority Queue (optional for merit)
4. **Online Food Ordering System:** Simulate an online food delivery system. Functions include: Display menu, Place order, Cancel order, Show bill, Maintain order history Data Structures Used: Stacks (for order undo), Queues or Linked Lists (for order processing), Arrays (for menu)

**Online Food Ordering System**

**Code:**

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

struct MenuItem {
   int id;
   char name[50];
   float price;
};

struct MenuItem menu[] = {
   {1, "Burger", 120.0},
   {2, "Pizza", 250.0},
```

```
    {3, "Pasta", 180.0},
    {4, "French Fries", 90.0},
    {5, "Cold Coffee", 110.0}
};
int menuSize = 5;

struct Order {
    int id;
    char name[50];
    int quantity;
    float price;
    struct Order *next;
};

struct Stack {
    int id;
    int quantity;
    struct Stack *next;
};

void displayMenu() {
    printf("\n--------- MENU ---------\n");
    for (int i = 0; i < menuSize; i++) {
        printf("%d. %s - Rs. %.2f\n", menu[i].id, menu[i].name, menu[i].price);
    }
    printf("------------------------\n");
}

void placeOrder(struct Order **head, struct Stack **top) {
    int id, qty;
    displayMenu();
    printf("Enter item id to order: ");
    scanf("%d", &id);
    printf("Enter quantity: ");
    scanf("%d", &qty);

    if (id < 1 || id > menuSize) {
        printf("Invalid item ID!\n");
        return;
    }

    struct Order *newOrder = (struct Order *)malloc(sizeof(struct Order));
    newOrder->id = id;
    strcpy(newOrder->name, menu[id - 1].name);
    newOrder->quantity = qty;
    newOrder->price = menu[id - 1].price * qty;
    newOrder->next = NULL;

    if (*head == NULL)
        *head = newOrder;
    else {
        struct Order *temp = *head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newOrder;
    }
```

```
    struct Stack *newNode = (struct Stack *)malloc(sizeof(struct Stack));
    newNode->id = id;
    newNode->quantity = qty;
    newNode->next = *top;
    *top = newNode;

    FILE *fp = fopen("order_history.txt", "a");
    if (fp != NULL) {
        fprintf(fp, "%s (x%d) - Rs. %.2f\n", newOrder->name, qty, newOrder->price);
        fclose(fp);
    }

    printf("%s (x%d) added to order!\n", newOrder->name, qty);
}

void cancelLastOrder(struct Order **head, struct Stack **top) {
    if (*top == NULL) {
        printf("No orders to cancel!\n");
        return;
    }

    int id = (*top)->id;
    int qty = (*top)->quantity;
    struct Stack *tempStack = *top;
    *top = (*top)->next;
    free(tempStack);

    struct Order *temp = *head, *prev = NULL;
    while (temp != NULL) {
        if (temp->id == id && temp->quantity == qty) {
            if (prev == NULL)
                *head = temp->next;
            else
                prev->next = temp->next;
            printf("Order for %s (x%d) cancelled!\n", temp->name, qty);
            free(temp);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
}


void showBill(struct Order *head) {
    if (head == NULL) {
        printf("No items in order!\n");
        return;
    }

    float total = 0;
    printf("\n------ Your Bill ------\n");
    while (head != NULL) {
        printf("%s (x%d) - Rs. %.2f\n", head->name, head->quantity, head->price);
        total += head->price;
```

```
        head = head->next;
    }
    printf("----------------------\nTotal Amount: Rs. %.2f\n", total);
}

void showOrderHistory() {
    FILE *fp = fopen("order_history.txt", "r");
    if (fp == NULL) {
        printf("No order history available.\n");
        return;
    }

    printf("\n------ ORDER HISTORY ------\n");
    char ch;
    while ((ch = fgetc(fp)) != EOF)
        putchar(ch);
    printf("----------------------------\n");
    fclose(fp);
}

void clearOrderHistory() {
    FILE *fp = fopen("order_history.txt", "w");
    if (fp != NULL) {
        fclose(fp);
        printf("Order history cleared successfully!\n");
    } else {
        printf("Error clearing history.\n");
    }
}

int main() {
    struct Order *orderList = NULL;
    struct Stack *undoStack = NULL;
    int choice;

    printf("\n=============================================\n");
    printf("     ONLINE FOOD ORDERING SYSTEM \n");
    printf("=============================================\n");

    do {
        printf("\n1. Display Menu\n");
        printf("2. Place Order\n");
        printf("3. Cancel Last Order\n");
        printf("4. Show Bill\n");
        printf("5. Show Order History\n");
        printf("6. Clear Order History\n");
        printf("7. Exit\n");
        printf("-----------------------------------------\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                displayMenu();
                break;
```

```
            case 2:
                placeOrder(&orderList, &undoStack);
                break;
            case 3:
                cancelLastOrder(&orderList, &undoStack);
                break;
            case 4:
                showBill(orderList);
                break;
            case 5:
                showOrderHistory();
                break;
            case 6:
                clearOrderHistory();
                break;
            case 7:
                printf("Thank you for using Online Food Ordering System!\n");
                break;
            default:
                printf("Invalid choice. Try again.\n");
        }

    } while (choice != 7);

    return 0;
}
```