

DATA STRUCTURE LAB

(20MCA135)

LAB RECORD

Submitted in partial fulfilment of the requirements for the award of the degree
of Master of Computer Applications of A P J Abdul Kalam Technological
University, Kerala.

Submitted by:

ABHIRAMI VINOD (SJC23MCA-2001)



**MASTER OF COMPUTER APPLICATIONS
ST. JOSEPH'S COLLEGE OF ENGINEERING AND
TECHNOLOGY, PALAI**

CHOONDACHERRY P.O, KOTTAYAM

KERALA

December 2023

ST. JOSEPH' S COLLEGE OF ENGINEERING AND TECHNOLOGY, PALAI

(An ISO 9001: 2015 Certified College)

CHOONDACHERRY P.O, KOTTAYAM, KERALA



CERTIFICATE

This is to certify that the Data Structure Lab Record (20MCA135) submitted by Abhirami Vinod, student of First semester MCA at ST. JOSEPH'S COLLEGE OF ENGINEERING AND TECHNOLOGY, PALAI in partial fulfilment for the award of Master of Computer Applications is a bonafide record of the lab work carried out by her under our guidance and supervision. This record in any form has not been submitted to any other University or Institute for any purpose.

Prof. Anish Augustine K

Faculty In- Charge

Dr. Rahul Shajan

(Head of the Department)

Submitted for the End Semester Examination held on _____

Examiner 1:

Examiner 2:

DECLARATION

I Abhirami Vinod, do hereby declare that the Data Structure Lab Record (20 MCA 135) is a record of work carried out under the guidance of Mr. Anish Augustine, Asst. Professor, Department of Computer Applications, SJCET, Palai as per the requirement of the curriculum of Master of Computer Applications Programme of A P J Abdul Kalam Technology University, Thiruvananthapuram. Further, I also declare that this record has not been submitted, full or part thereof, in any University / Institution for the award of any Degree / Diploma.

Place: Choondacherry

ABHIRAMI VINOD

Date :

(SJC23MCA-2001)

CONTENT

Sl. No.	Program List	Page No.
1	Array Operation (Insertion, Deletion, Sorting, Merging).	1
2	Searching an array element (Linear Search, Binary Search).	6
3	Matrix Operations (Addition, Multiplication, Transpose).	9
4	Using Structure, add two distances in the inch-feet system.	14
5	Implement Stack Operations.	16
6	String Operations (Searching, Concatenation, Substring).	19
7	Sorting an Array (Bubble Sort, Selection Sort, Insertion Sort).	21
8	Implement Queue operations (Insert, delete, display front & rear values).	25
9	Implement Circular Queue operations (Insert, delete, display front & rear values).	28
10	Implement singly linked list (Insert at the head, insert at tail, insert at a position, delete at the head, delete at tail, delete form a position, search an element).	31
11	Implement doubly linked list (Insert at the head, insert at tail, insert at a position, delete at the head, delete at tail, delete form a position, search an element).	38
12	Implement circular linked list (Insert at the head, insert at tail, insert at a position, delete at the head, delete at tail, delete form a position, search an element).	45
13	Implement binary search tree.	51
14	Implement balanced-binary-search tree.	56

15	Implement set operations (union, intersection, difference).	61
16	Implement disjoint set operations.	66
17	Implement tree traversal methods DFS (In-order, Pre-Order, Post-Order), and BFS.	69
18	Implement Binomial Heaps and operations (Create, Insert, Delete).	74
19	Implement B Trees and its operations.	86
20	Implement Red Black Trees and its operations.	97
21	Graph Traversal techniques (DFS and BFS) and Topological Sorting.	103
22	Finding the Strongly connected Components in a directed graph.	108
23	Prim's Algorithm for finding the minimum cost spanning tree.	112
24	Kruskal's algorithm using the Disjoint set data structure.	115
25	Single Source shortest path algorithm using any heap structure that supports mergeable heap operations.	118

1. Array Operation (Insertion, Deletion, Sorting, Merging).**Program:**

```
#include <stdio.h>

int a[50], n, choice, i, x, j;

void insert(){
    int pos;
    printf("Enter the position :");
    scanf("%d", &pos);
    if (pos < 0 || pos > n)
    {
        printf("Invalid position");
        return;
    }
    else
    {
        printf("Enter the element to insert: ");
        scanf("%d", &x);

        for (i = n; i > pos; i--)
        {
            a[i] = a[i - 1];
        }
        a[pos] = x;
        n++;
    }
}

void delete(){
    int pos;
    printf("Enter the position of element to delete: ");
    scanf("%d", &pos);
```

```
    if (pos < 0 || pos > n)
    {
        printf("Invalid position");
        return;
    }
    else
    {
        if (pos >= n + 1)
        {
            printf("Deletion is not possible");
        }
        else
        {
            for (i = pos; i < n - 1; i++)
            {
                a[i] = a[i + 1];
            }
            n--;
        }
    }
}

void display(){
    printf("Array Elements:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", a[i]);
    }
}

void sort(){
    int temp;
```

```
for (i = 0; i < n - 1; i++)
{
    for (j = 0; j < n - i - 1; j++)
    {
        if (a[j] > a[j + 1])
        {
            temp = a[j];
            a[j] = a[j + 1];
            a[j + 1] = temp;
        }
    }
}

void merge(){
    int n2, n3, b[50], c[50];
    printf("Enter the size of second array: ");
    scanf("%d", &n2);
    printf("Enter the array of second elements: ");
    for (i = 0; i < n2; i++)
    {
        scanf("%d", &b[i]);
    }
    n3 = n + n2;
    for (i = 0; i < n; i++)
    {
        c[i] = a[i];
    }
    for (i = 0; i < n2; i++)
    {
        c[i + n] = b[i];
    }
}
```



```
    }  
    n=n3;  
    for(i = 0; i< n3; i++){  
        a[i] = c[i];  
    }  
    printf("The merged array: ");  
    for (i = 0; i < n3; i++)  
    {  
        printf("%d ", a[i]);  
    }  
}  
int main(){  
    printf("Enter the size of array: ");  
    scanf("%d", &n);  
    printf("Enter the elements: ");  
    for (i = 0; i < n; i++)  
    {  
        scanf("%d", &a[i]);  
    }  
    while (choice != 6)  
    {  
        printf("\nEnter the choice (1.Insert 2.Delete 3.Sort 4.Merge 5.Display 6.Exit): ");  
        scanf("%d", &choice);  
        switch (choice){  
            case 1:  
                insert();  
                break;  
            case 2:  
                delete();  
                break;
```

```

        case 3:

            sort();

            break;

        case 4:

            merge();

            break;

        case 5:

            display();

            break;

        case 6:

            printf("\nExit\n");

            break;

        default:

            printf("Enter the invalid option");

    }

}

return 0;

}

```

Output:

```

mca@HP-Z238:~/abhirami$ gcc array.c
mca@HP-Z238:~/abhirami$ ./a.out
Enter the size of array: 4
Enter the elements: 7
2
3
9

Enter the choice (1.Insert 2.Delete 3.Sort 4.Merge 5.Display 6.Exit): 1
Enter the position :3
Enter the element to insert: 0

Enter the choice (1.Insert 2.Delete 3.Sort 4.Merge 5.Display 6.Exit): 5
Array Elements:
7
2
3
0
9

Enter the choice (1.Insert 2.Delete 3.Sort 4.Merge 5.Display 6.Exit): 2
Enter the position of element to delete: 2

Enter the choice (1.Insert 2.Delete 3.Sort 4.Merge 5.Display 6.Exit): 5
Array Elements:
7
2
0
9

Enter the choice (1.Insert 2.Delete 3.Sort 4.Merge 5.Display 6.Exit): 4
Enter the size of second array: 3
Enter the elements of second array: 1
6
4
The merged array: 7 2 0 9 1 6 4
Enter the choice (1.Insert 2.Delete 3.Sort 4.Merge 5.Display 6.Exit): 5
Array Elements:
7
2
0
9
1
6
4

```

2. Searching an array element (Linear Search, Binary Search).**Program:**

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}

int binarySearch(int arr[], int n, int key) {
    int left = 0;
    int right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;
}

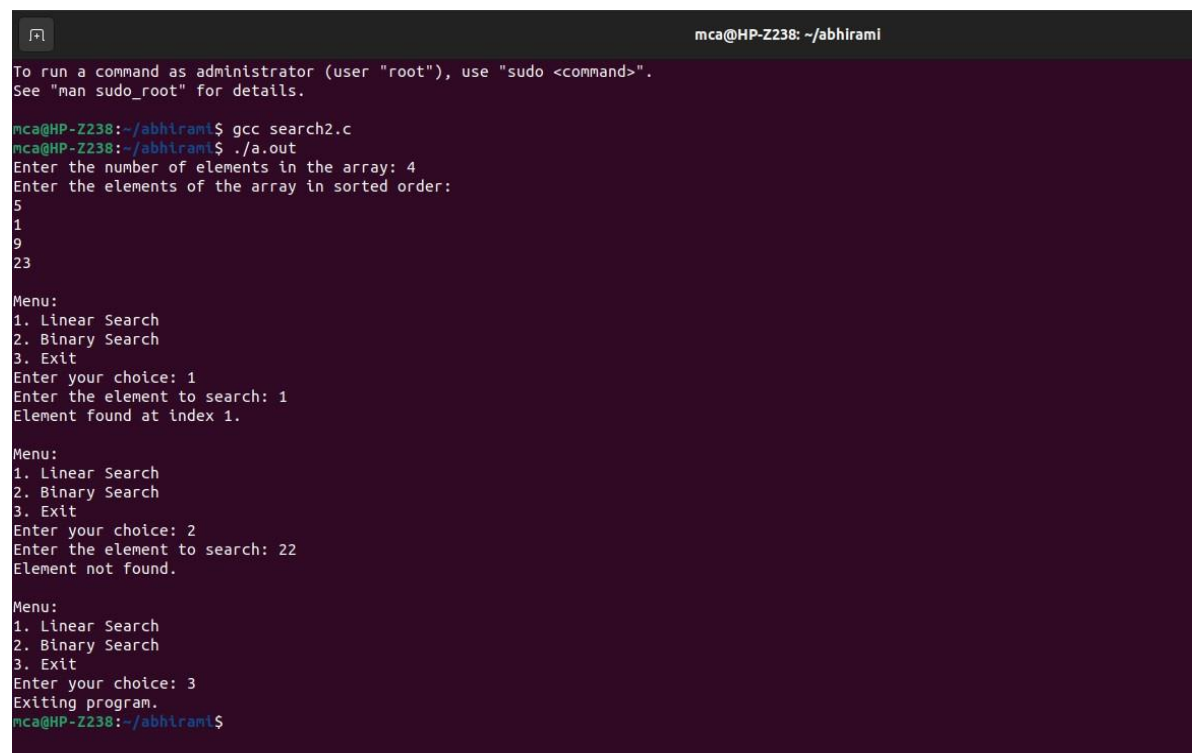
int main() {
    int n, choice, key, result;
```

```
printf("Enter the number of elements in the array: ");
scanf("%d", &n);
int arr[n];
printf("Enter the elements of the array in sorted order:\n");
for (int i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
}

while (1) {
    printf("\nMenu:\n");
    printf("1. Linear Search\n");
    printf("2. Binary Search\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            printf("Enter the element to search: ");
            scanf("%d", &key);
            result = linearSearch(arr, n, key);
            if (result != -1) {
                printf("Element found at index %d.\n", result);
            } else {
                printf("Element not found.\n");
            }
            break;
        case 2:
            printf("Enter the element to search: ");
            scanf("%d", &key);
            result = binarySearch(arr, n, key);
```

```
        if (result != -1) {  
            printf("Element found at index %d.\n", result);  
        } else {  
            printf("Element not found.\n");  
        }  
        break;  
    case 3:  
        printf("Exiting program.\n");  
        return 0;  
    default:  
        printf("Invalid choice. Please select a valid option.\n");  
    }  
}  
}  
return 0;  
}
```

Output:



```
mca@HP-Z238: ~/abhirami  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
mca@HP-Z238:~/abhirami$ gcc search2.c  
mca@HP-Z238:~/abhirami$ ./a.out  
Enter the number of elements in the array: 4  
Enter the elements of the array in sorted order:  
5  
1  
9  
23  
  
Menu:  
1. Linear Search  
2. Binary Search  
3. Exit  
Enter your choice: 1  
Enter the element to search: 1  
Element found at index 1.  
  
Menu:  
1. Linear Search  
2. Binary Search  
3. Exit  
Enter your choice: 2  
Enter the element to search: 22  
Element not found.  
  
Menu:  
1. Linear Search  
2. Binary Search  
3. Exit  
Enter your choice: 3  
Exiting program.  
mca@HP-Z238:~/abhirami$
```

3. Matrix Operations (Addition, Multiplication, Transpose)

Program:

```
#include <stdio.h>

int a[10][10], b[10][10], c[10][10], d[10][10], i, j, row, col, choice;

void add()
{
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    printf("\nMatrix: \n");
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            printf("%d\t", c[i][j]);
        }
        printf("\n");
    }
}

void multiply()
{
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            d[i][j] = a[i][j] * b[i][j];
        }
    }
}
```

```
    }  
}  
printf("\nMatrix: \n");  
for (i = 0; i < row; i++)  
{  
    for (j = 0; j < col; j++)  
    {  
        printf("%d\t", d[i][j]);  
    }  
    printf("\n");  
}  
}  
void transpose()  
{  
    printf("Transpose of First Matix: \n");  
    for (i = 0; i < row; i++)  
    {  
        for (j = 0; j < col; j++)  
        {  
            printf("%d\t", a[j][i]);  
        }  
        printf("\n");  
    }  
    printf("Transpose of Second Matix: \n");  
    for (i = 0; i < row; i++)  
    {  
        for (j = 0; j < col; j++)  
        {  
            printf("%d\t", a[j][i]);  
        }  
    }
```

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

int main()
{
    printf("Enter the no of rows: ");
    scanf("%d", &row);

    printf("Enter the no of columns: ");
    scanf("%d", &col);

    printf("Enter the elements of first matrix:\n ");
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            printf("a[%d][%d]: ", i, j);
            scanf("%d", &a[i][j]);
        }
    }

    printf("Enter the elements of second matrix:\n ");
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            printf("b[%d][%d]: ", i, j);
            scanf("%d", &b[i][j]);
        }
    }

    printf("\nFirst Matrix: \n");
    for (i = 0; i < row; i++)
    {
```



```
        for (j = 0; j < col; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    printf("\nSecond Matrix: \n");
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            printf("%d\t", b[i][j]);
        }
        printf("\n");
    }
    while (choice != 4)
    {
        printf("\nEnter the choice (1.Add 2.Multiply 3.Transpose 4.Exit): ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            {
                add();
                break;
            }
            case 2:
            {
                multiply();
                break;
            }
        }
    }
}
```

```

    }

    case 3:

    {

        transpose();

        break;

    }

    case 4:

    {

        printf("\nExit\n");

        break;

    }

    default:

    {

        printf("Enter the invalid option");

    }

    }

    return 0;

}

```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_1$ gcc matrix.c
mca@HP-Z238:~/abhirami/c_labcycle_1$ ./a.out
Enter the no of rows: 2
Enter the no of columns: 2
Enter the elements of first matrix:
a[0][0]: 1
a[0][1]: 3
a[1][0]: 2
a[1][1]: 4
Enter the elements of second matrix:
b[0][0]: 1
b[0][1]: 3
b[1][0]: 4
b[1][1]: 5

First Matrix:
1      3
2      4

Second Matrix:
1      3
4      5

Enter the choice (1.Add 2.Multiply 3.Transpose 4.Exit): 1

Matrix:
2      6
6      9

Enter the choice (1.Add 2.Multiply 3.Transpose 4.Exit): 2

Matrix:
1      9
8      20

Enter the choice (1.Add 2.Multiply 3.Transpose 4.Exit): 3
Transpose of First Matix A:
1      2
3      4
Transpose of Second Matix B:
1      4
3      5

Enter the choice (1.Add 2.Multiply 3.Transpose 4.Exit): 4
Exit

```

4. Using Structure, add two distances in the inch-feet system.**Program:**

```
#include <stdio.h>

struct Distance {
    int feet;
    float inches;
};

struct Distance addDistances(struct Distance d1, struct Distance d2) {
    struct Distance result;
    result.inches = d1.inches + d2.inches;
    result.feet = d1.feet + d2.feet;
    if (result.inches >= 12.0) {
        result.feet++;
        result.inches -= 12.0;
    }
    return result;
}

int main() {
    struct Distance distance1, distance2, sum;

    printf("Enter distance 1:\n");

    printf("Feet: ");
    scanf("%d", &distance1.feet);

    printf("Inches: ");
    scanf("%f", &distance1.inches);
```

```
printf("Enter distance 2:\n");

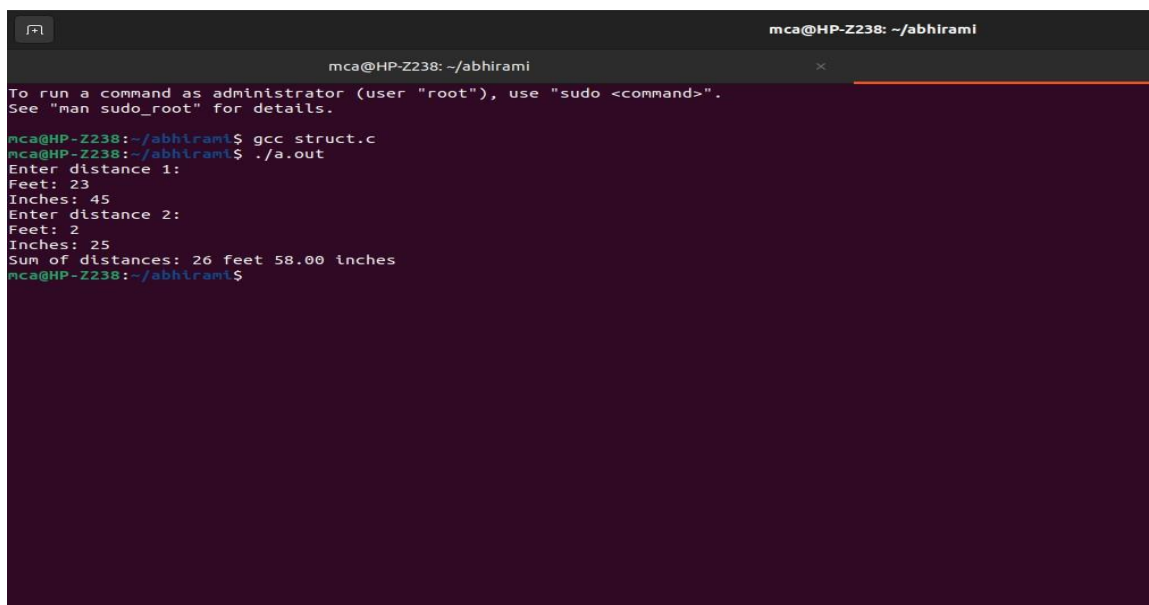
printf("Feet: ");
scanf("%d", &distance2.feet);
printf("Inches: ");

scanf("%f", &distance2.inches);

sum = addDistances(distance1, distance2);

printf("Sum of distances: %d feet %.2f inches\n", sum.feet, sum.inches);
return 0;
}
```

Output:



```
mca@HP-Z238: ~/abhirami
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
mca@HP-Z238:~/abhirami$ gcc struct.c
mca@HP-Z238:~/abhirami$ ./a.out
Enter distance 1:
Feet: 23
Inches: 45
Enter distance 2:
Feet: 2
Inches: 25
Sum of distances: 26 feet 58.00 inches
mca@HP-Z238:~/abhirami$
```

5. Implement Stack Operations.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#define max 20
int top=-1,s[max],n;
void push(int n)
{
    if(top==max-1)
    {
        printf("Stack is overflow");
        return;
    }
    else
    {
        top=top+1;
        s[top]=n;
    }
}
void pop()
{
    int del;
    if(top== -1)
    {
        printf("stack is underflow");
        return;
    }
    else
    {
        del=s[top];
        top=top-1;
    }
}
```

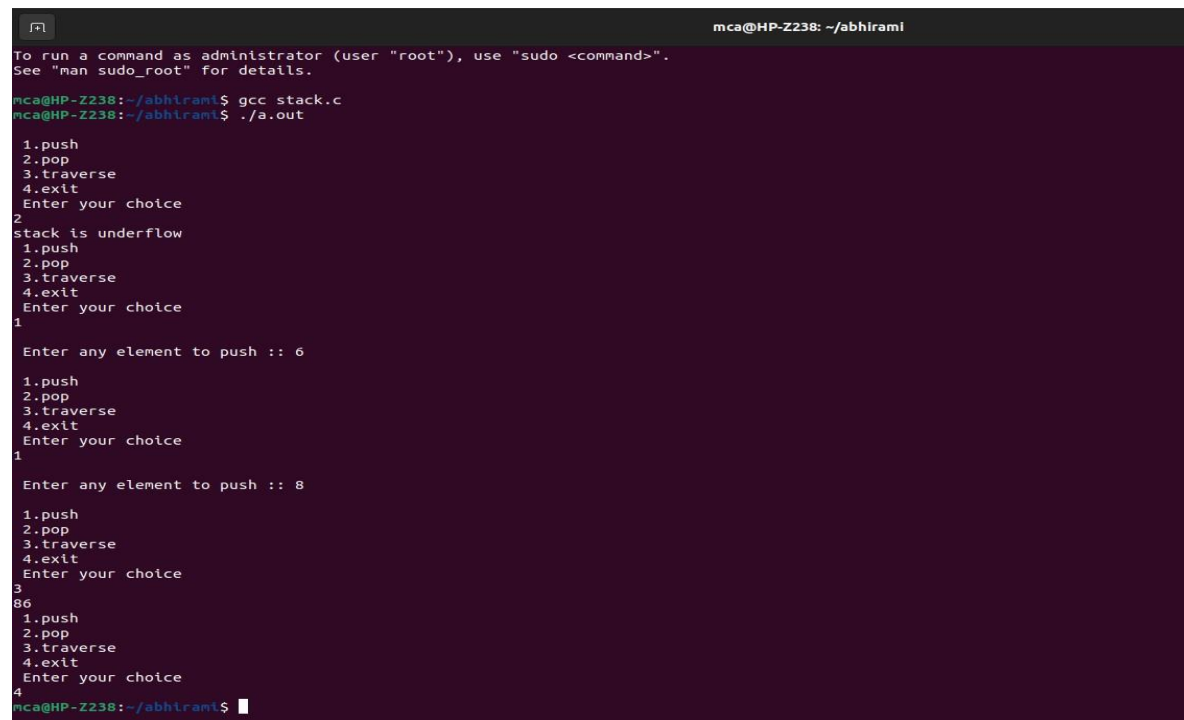
```
void traverse()
{
    int i;
    if(top== -1)
        printf("Stack is underflow");
    else
    {
        for(i=top; i>=0; i--)
            printf("%d", s[i]);
    }
}

int main()
{
    int op, n;
    do
    {
        printf("\n 1.push\n 2.pop \n 3.traverse \n 4.exit\n Enter your choice\n");
        scanf("%d", &op);
        switch(op)
        {
            case 1:
                printf("\n Enter any element to push :: ");
                scanf("%d", &n);
                push(n);
                break;
            case 2:
                pop();
                break;
            case 3:
                traverse();
                break;
            case 4:

                exit(0);
        }
    } while (op != 4);
}
```

```
        break;
    }
}
while(1);
return 0;
}
```

Output:



```
mca@HP-Z238: ~/abhirami
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
mca@HP-Z238:~/abhirami$ gcc stack.c
mca@HP-Z238:~/abhirami$ ./a.out
1.push
2.pop
3.traverse
4.exit
Enter your choice
2
stack is underflow
1.push
2.pop
3.traverse
4.exit
Enter your choice
1
Enter any element to push :: 6
1.push
2.pop
3.traverse
4.exit
Enter your choice
1
Enter any element to push :: 8
1.push
2.pop
3.traverse
4.exit
Enter your choice
3
86
1.push
2.pop
3.traverse
4.exit
Enter your choice
4
mca@HP-Z238:~/abhirami$
```

6. String Operations (Searching, Concatenation, Substring).

Program:

```
#include <stdio.h>

#include <string.h>

int main()
{
    char str1[100], str2[100];
    int choice=0;
    printf("Enter the first string: ");
    scanf("%s", str1);
    printf("Enter the second string: ");
    scanf("%s", str2);
    while (choice != 4){
        printf("\nSelect a String Operation\n");
        printf("1. Search\n2. Concatenate\n3. Substring\n4. Exit\n ");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                if (strstr(str1, str2) != NULL){
                    printf("'s' found in 's'\n", str2, str1);
                }
                else{
                    printf("'s' not found in 's'\n", str2, str1);
                }
                break;
            case 2:
                strcat(str1, str2);
                printf("Concatenated string: %s\n", str1);
```



```
        break;

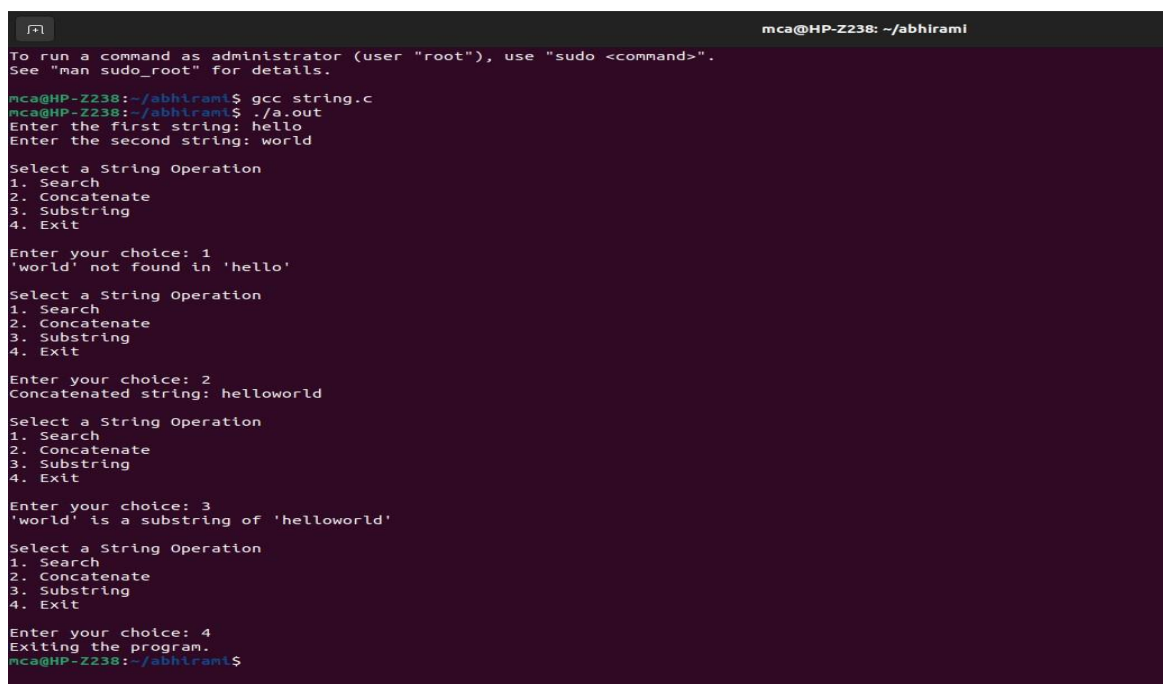
    case 3:
        if (strstr(str1, str2) != NULL){
            printf("%s' is a substring of '%s'\n", str2, str1);
        }
        else{
            printf("%s' is not a substring of '%s'\n", str2, str1);
        }
        break;

    case 4:
        printf("Exiting the program.\n");
        break;

    default:
        printf("Invalid choice.\n");
    }}

    return 0;
}
```

Output:



```
mca@HP-Z238: ~/abhirami
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mca@HP-Z238:~/abhirami$ gcc string.c
mca@HP-Z238:~/abhirami$ ./a.out
Enter the first string: hello
Enter the second string: world

Select a String Operation
1. Search
2. Concatenate
3. Substring
4. Exit

Enter your choice: 1
'world' not found in 'hello'

Select a String Operation
1. Search
2. Concatenate
3. Substring
4. Exit

Enter your choice: 2
Concatenated string: helloworld

Select a String Operation
1. Search
2. Concatenate
3. Substring
4. Exit

Enter your choice: 3
'world' is a substring of 'helloworld'

Select a String Operation
1. Search
2. Concatenate
3. Substring
4. Exit

Enter your choice: 4
Exiting the program.
mca@HP-Z238:~/abhirami$
```

7. Sorting an Array (Bubble Sort, Selection Sort, Insertion Sort).**Program:**

```
#include<stdio.h>
#include<stdlib.h>
void display(int a[],int n);
void bubble_sort(int a[],int n);
void selection_sort(int a[],int n);
void insertion_sort(int a[],int n);
int main()
{
    int n,choice,i;
    char ch[20];
    printf("Enter no. of elements u want to sort : ");
    scanf("%d",&n);
    int arr[n];
    for(i=0;i<n;i++)
    {
        printf("Enter %d Element : ",i+1);
        scanf("%d",&arr[i]);
    }
    printf("Please select any option Given Below for Sorting : \n");

    while(1)
    {

        printf("\n1. Bubble Sort\n2. Selection Sort\n3. Insertion Sort\n4. Display Array.\n5.
Exit the Program.\n");
        printf("\nEnter your Choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
```

```
        bubble_sort(arr,n);
        break;
    case 2:
        selection_sort(arr,n);
        break;
    case 3:
        insertion_sort(arr,n);
        break;
    case 4:

        display(arr,n);
        break;

    case 5:
        return 0;
    default:
        printf("\nPlease Select only 1-5 option-----\n");
    }
}
return 0;
}

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

}

void bubble_sort(int arr[],int n)
{
    int i,j,temp;
    for(i=0;i<n;i++)
```

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

printf("After Bubble sort Elements are : ");
display(arr,n);
}

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

    printf("After Selection sort Elements are : ");
    display(arr,n);
}
```

```

void insertion_sort(int arr[],int n)
{
    int i,j,min;
    for(i=1;i<n;i++)
    {
        min=arr[i];
        j=i-1;
        while(min<arr[j] && j>=0)
        {
            arr[j+1]=arr[j];
            j=j-1;
        }
        arr[j+1]=min;
    }

    printf("After Insertion sort Elements are : ");
    display(arr,n);
}

```

Output:

```

mca@HP-Z238: ~/abhirami
mca@HP-Z238: ~/abhirami$ gcc sort.c
mca@HP-Z238: ~/abhirami$ ./a.out
Enter no. of elements u want to sort : 4
Enter 1 Element : 3
Enter 2 Element : 7
Enter 3 Element : 9
Enter 4 Element : 2
Please select any option Given Below for Sorting :
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array.
5. Exit the Program.
Enter your Choice : 1
After Bubble sort Elements are : 2 3 7 9
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array.
5. Exit the Program.
Enter your Choice : 2
After Selection sort Elements are : 2 3 7 9
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array.
5. Exit the Program.
Enter your Choice : 3
After Insertion sort Elements are : 2 3 7 9
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array.
5. Exit the Program.
Enter your Choice : 4
2 3 7 9
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Display Array.
5. Exit the Program.

```

8. Implement Queue operations (Insert, delete, display front & rear values).**Program:**

```
#include<stdio.h>

#define MAX 5

int q[20],choice,n,rear=-1,front=-1,x,i;

void insert(){

int item;

if (rear==MAX-1){

printf("Queue Overflow \n");

}else{

if (front== -1)

front=0;

printf("Inset the element in queue: ");

scanf("%d",&item);

rear=rear+1;

q[rear]=item;

}

}

void delete(){

if(front== -1||front>rear){

printf("Queue Underflow \n");

return ;

}else{

printf("Element deleted from queue is : %d\n", q[front]);

front=front+1;

}

}

void display(){

int i;

if (front== -1)
```

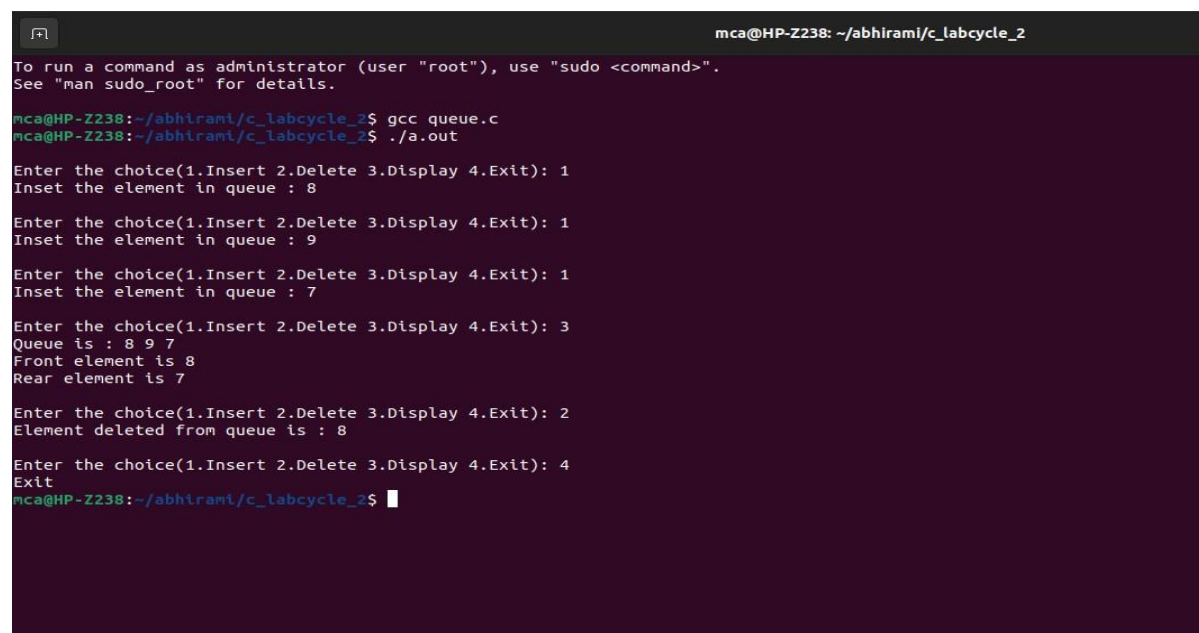
```
printf("Queue is empty \n");
else
{
printf("Queue is : ");
for (i=front;i<=rear;i++)
printf("%d ", q[i]);
printf("\nFront element is %d ", q[front]);
printf("\nRear element is %d ", q[rear]);
printf("\n");
}
}
int main()
{
while(choice != 4)
{
printf("\nEnter the choice(1.Insert 2.Delete 3.Display 4.Exit): ");

scanf("%d",&choice);
switch(choice)
{
case 1:
{
insert();
break;
}
case 2:
{
delete();
break;
}
}
```

```
        case 3:{
            display();
            break;
        }
        case 4:
        {
            printf("Exit\n");
            break;
        }
        default:{
            printf("Enter valid option");
        }
    }
}

return 0;
}
```

Output:



```
mca@HP-Z238: ~/abhirami/c_labcycle_2
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc queue.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 1
Inset the element in queue : 8

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 1
Inset the element in queue : 9

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 1
Inset the element in queue : 7

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 3
Queue is : 8 9 7
Front element is 8
Rear element is 7

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 2
Element deleted from queue is : 8

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 4
Exit
mca@HP-Z238:~/abhirami/c_labcycle_2$
```


9. Implement Circular Queue operations (Insert, delete, display front & rear values).**Program:**

```
#include<stdio.h>

#define MAX 10

int q[10],n,choice,front=-1,rear=-1;

void insert()
{
    int x;
    if((front==0 &&rear==MAX-1)||((front==rear+1))){
        printf("queue is full");
        return;
    }else {
        printf("Enter the element: ");
        scanf("%d", &x);
        if (front == -1) {
            front = 0;
        }
        rear=(rear+1)%MAX;
        q[rear] = x;
    }
}

void delete()
{
    if (front== -1){
        printf("Queue is empty ");
    }else {
        int removed =q[front];
        if (front == rear){
            front = rear = -1;
        }else{
            front = (front + 1) % MAX;
        }
    }
}
```

```
    }
    printf("Element deleted is %d.\n", removed);
    }}
void display()
{
    int i;
    if(rear== -1 && front== -1){
        printf("Queue is empty");
    }else if (front > rear){
        for (i = front; i < n; i++){
            printf("%d ", q[i]);
        }
        for(i=0; i <= rear; i++)
            printf("%d ", q[i]);
    }else{
        printf("Elements: ");
        for (i = front; i <= rear; i++)
            printf("%d ", q[i]);
        printf("\nFront element is %d ", q[front]);
        printf("\nRear element is %d ", q[rear]);
    }
}
int main()
{
    while(choice != 4)
    {
        printf("\nEnter the choice(1.Insert 2.Delete 3.Display 4.Exit): ");
        scanf("%d",&choice);
        switch(choice) {
            case 1:
                insert();
```

```
        break;

    case 2:

        delete();

        break;

    case 3:

        display();

        break;

    case 4:

        printf("Exit\n");

        break;

    default:

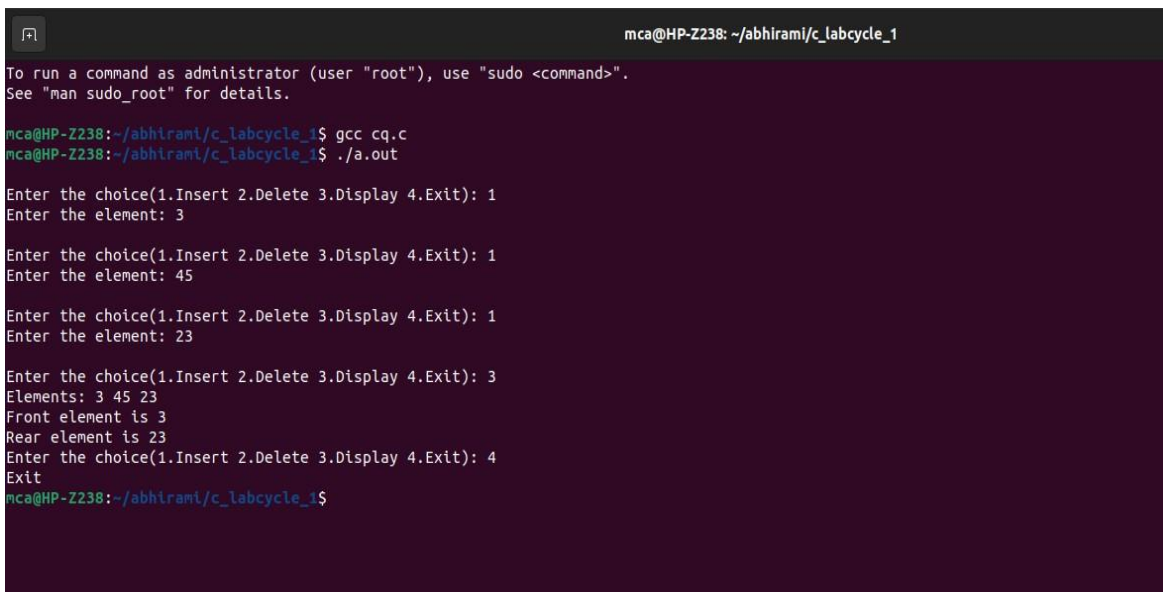
        printf("Enter valid option");

    }}

return 0;

}
```

Output:



```
mca@HP-Z238: ~/abhirami/c_labcycle_1
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mca@HP-Z238:~/abhirami/c_labcycle_1$ gcc cq.c
mca@HP-Z238:~/abhirami/c_labcycle_1$ ./a.out

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 1
Enter the element: 3

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 1
Enter the element: 45

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 1
Enter the element: 23

Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 3
Elements: 3 45 23
Front element is 3
Rear element is 23
Enter the choice(1.Insert 2.Delete 3.Display 4.Exit): 4
Exit
mca@HP-Z238:~/abhirami/c_labcycle_1$
```

10. Implement singly linked list (Insert at the head, insert at tail, insert at a position, delete at the head, delete at tail, delete form a position, search an element).**Program:**

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAtHead(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

void insertAtTail(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    struct Node* current = *head;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

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

    current->next = newNode;
}

void insertAtPosition(struct Node** head, int data, int position) {
    if (position < 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 0 || *head == NULL) {
```

```
        insertAtHead(head, data);
        return;
    }

    struct Node* newNode = createNode(data);
    struct Node* current = *head;
    int currentPosition = 0;

    while (currentPosition < position - 1 && current->next != NULL) {
        current = current->next;
        currentPosition++;
    }

    newNode->next = current->next;
    current->next = newNode;
}

void deleteAtHead(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

void deleteAtTail(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }

    struct Node* current = *head;
    while (current->next->next != NULL) {
        current = current->next;
    }

    free(current->next);
    current->next = NULL;
}
```

```
void deleteAtPosition(struct Node** head, int position) {

    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    if (position < 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 0) {
        deleteAtHead(head);
        return;
    }

    struct Node* current = *head;
    int currentPosition = 0;

    while (currentPosition < position - 1 && current->next != NULL) {
        current = current->next;
        currentPosition++;
    }

    if (current->next == NULL || current->next->next == NULL) {
        printf("Invalid position\n");
        return;
    }

    struct Node* temp = current->next;
    current->next = current->next->next;
    free(temp);
}

int searchElement(struct Node* head, int key) {
    struct Node* current = head;
    int position = 0;

    while (current != NULL) {
        if (current->data == key) {
            return position;
        }
        current = current->next;
        position++;
    }

    return -1;
}
```

```
}
void printList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node* head = NULL;
    int choice, data, position, key;
    while (1) {
        printf("\nMenu:\n1. Insert at head\n2. Insert at tail\n3. Insert at a position\n");
        printf("4. Delete at head\n5. Delete at tail\n6. Delete at a position\n");
        printf("7. Search \n8. Display\n9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data to insert at head: ");
                scanf("%d", &data);
                insertAtHead(&head, data);
                break;
            case 2:
                printf("Enter data to insert at tail: ");
                scanf("%d", &data);
                insertAtTail(&head, data);
                break;
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter position to insert at: ");
                scanf("%d", &position);
                insertAtPosition(&head, data, position);
                break;
            case 4:
                deleteAtHead(&head);
                break;
            case 5:
                deleteAtTail(&head);
                break;
            case 6:
                printf("Enter position to delete from: ");
```

```
        scanf("%d", &position);
        deleteAtPosition(&head, position);
        break;
    case 7:
        printf("Enter element to search for: ");
        scanf("%d", &key);
        position = searchElement(head, key);
        if (position != -1) {
            printf("%d found at position %d\n", key, position);
        } else {
            printf("%d not found in the linked list\n", key);
        }
        break;
    case 8:
        printf("Linked List: ");
        printList(head);
        break;
    case 9:
        printf("\nExit\n");
        exit(0);
    default:
        printf("Invalid choice\n");
    }
}

return 0;
}
```


Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_2
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc singly.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 1
Enter data to insert at head: 10

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 2
Enter data to insert at tail: 20

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 3
Enter data to insert: 30
Enter position to insert at: 1

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit

```

```

mca@HP-Z238: ~/abhirami/c_labcycle_2

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 8
Linked List: 10 -> 30 -> 20 -> NULL

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 5

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 8
Linked List: 10 -> 30 -> NULL

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit

```

```
mca@HP-Z238: ~/abhirami/c_labcycle_2
8. Print list
9. Exit
Enter your choice: 8
Linked List: 10 -> 30 -> NULL

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 7
Enter element to search for: 30
30 found at position 1

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 7
Enter element to search for: 60
60 not found in the linked list

Menu:
1. Insert at head
2. Insert at tail
3. Insert at a position
4. Delete at head
5. Delete at tail
6. Delete at a position
7. Search for an element
8. Print list
9. Exit
Enter your choice: 9
mca@HP-Z238:~/abhirami/c_labcycle_2$
```

11. Implement doubly linked list (Insert at the head, insert at tail, insert at a position, delete at the head, delete at tail, delete from a position, search an element).

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtHead(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    newNode->next = *head;
    *head = newNode;
}

void insertAtTail(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
}

void insertAtPosition(struct Node** head, int data, int position) {
    if (position < 0) {
        printf("Invalid position. Position must be non-negative.\n");
    }
}
```

```
        return;
    }
    if (position == 0) {
        insertAtHead(head, data);
        return;
    }
    struct Node* newNode = createNode(data);
    struct Node* current = *head;
    int currentPosition = 0;
    while (current != NULL && currentPosition < position - 1) {
        current = current->next;
        currentPosition++;
    }
    if (current == NULL) {
        printf("Position exceeds the length of the list.\n");
        return;
    }
    newNode->prev = current;
    newNode->next = current->next;
    if (current->next != NULL) {
        current->next->prev = newNode;
    }
    current->next = newNode;
}

void deleteAtHead(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    if (*head != NULL) {
        (*head)->prev = NULL;
    }
    free(temp);
}

void deleteAtTail(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }
}
```

```
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->prev->next = NULL;
    free(current);
}
void deleteAtPosition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }
    if (position < 0) {
        printf("Invalid position. Position must be non-negative.\n");
        return;
    }
    if (position == 0) {
        deleteAtHead(head);
        return;
    }
    struct Node* current = *head;
    int currentPosition = 0;
    while (current != NULL && currentPosition < position) {
        current = current->next;
        currentPosition++;
    }
    if (current == NULL) {
        printf("Position exceeds the length of the list.\n");
        return;
    }
    current->prev->next = current->next;
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    free(current);
}
int search(struct Node* head, int key) {
    struct Node* current = head;
    int position = 0;
    while (current != NULL) {
        if (current->data == key) {
            return position;
        }
        current = current->next;
        position++;
    }
}
```

```
    return -1;
}
void displayForward(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
void displayBackward(struct Node* head) {
    struct Node* current = head;
    while (current != NULL && current->next != NULL) {
        current = current->next;
    }
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->prev;
    }
    printf("NULL\n");
}
int main()
{
    struct Node* head = NULL;
    int choice, data, position;

    while (1) {
        printf("\nDoubly Linked List Operations:\n")

        printf("1. Insert at the Head\n2. Insert at the Tail\n3. Insert at a Position\n");
        printf("4. Delete at the Head\n5. Delete at the Tail\n6. Delete at a Position\n");
        printf("7. Search for an Element\n8. Display Forward\n9. Display Backward\n");
        printf("10. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at the head: ");
                scanf("%d", &data);
                insertAtHead(&head, data);
                break;

            case 2:
```

```
printf("Enter data to insert at the tail: ");
scanf("%d", &data);
insertAtTail(&head, data);
break;

case 3:
    printf("Enter data to insert: ");
    scanf("%d", &data);
    printf("Enter position to insert at: ");
    scanf("%d", &position);
    insertAtPosition(&head, data, position);
    break;

case 4:
    deleteAtHead(&head);
    break;

case 5:
    deleteAtTail(&head);
    break;

case 6:
    printf("Enter position to delete: ");
    scanf("%d", &position);
    deleteAtPosition(&head, position);
    break;

case 7:
    printf("Enter element to search: ");
    scanf("%d", &data);
    position = search(head, data);
    if (position != -1) {
        printf("Element %d found at position %d\n", data, position);
    } else {
        printf("Element %d not found in the linked list\n", data);
    }
    break;

case 8:
    printf("Doubly Linked List (Forward): ");
    displayForward(head);
    break;

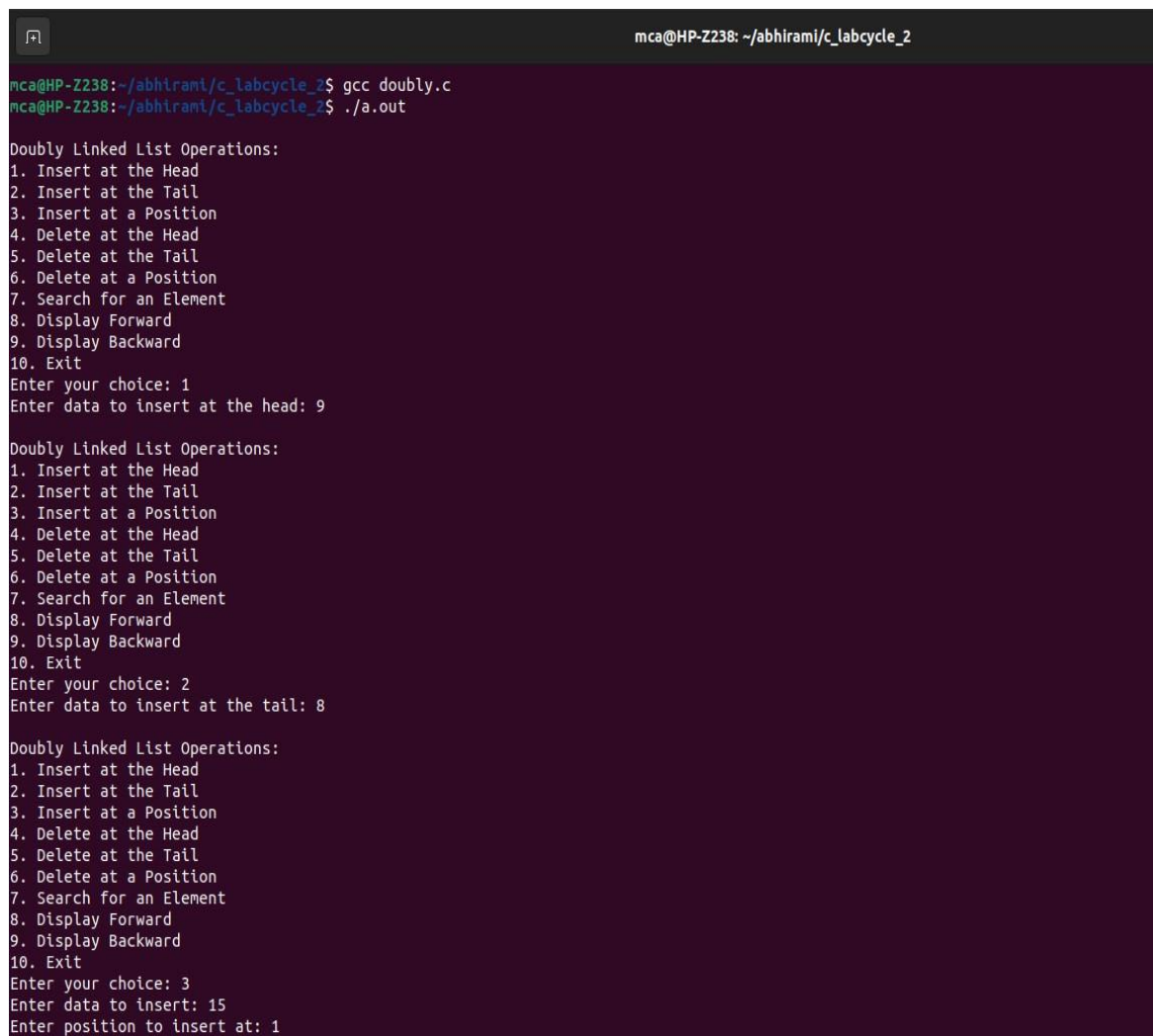
case 9:
    printf("Doubly Linked List (Backward): ");
    displayBackward(head);
    break;
```

```
        case 10:
            printf("\nExit\n");
            exit(0);
            break;

        default:
            printf("Invalid choice. Please enter a valid option.\n");
    }
}

return 0;
}
```

Output:



```
mca@HP-Z238: ~/abhirami/c_labcycle_2
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc doubly.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 1
Enter data to insert at the head: 9

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 2
Enter data to insert at the tail: 8

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 3
Enter data to insert: 15
Enter position to insert at: 1
```



```
mca@HP-Z238: ~/abhirami/c_labcycle_2

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 8
Doubly Linked List (Forward): 9 -> 15 -> 8 -> NULL

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 5

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 9
Doubly Linked List (Backward): 15 -> 9 -> NULL

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
```

```
mca@HP-Z238: ~/abhirami/c_labcycle_2

10. Exit
Enter your choice: 9
Doubly Linked List (Backward): 15 -> 9 -> NULL

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 6
Enter position to delete: 1

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 8
Doubly Linked List (Forward): 9 -> NULL

Doubly Linked List Operations:
1. Insert at the Head
2. Insert at the Tail
3. Insert at a Position
4. Delete at the Head
5. Delete at the Tail
6. Delete at a Position
7. Search for an Element
8. Display Forward
9. Display Backward
10. Exit
Enter your choice: 10
mca@HP-Z238:~/abhirami/c_labcycle_2$
```

12. Implement circular linked list (Insert at the head, insert at tail, insert at a position, delete at the head, delete at tail, delete from a position, search an element).**Program:**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

struct Node* head = NULL;
void insertAtHead(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (head == NULL) {
        newNode->next = newNode;
        head = newNode;
        return;
    }

    struct Node* temp = head;
    while (temp->next != head)
        temp = temp->next;

    temp->next = newNode;
    newNode->next = head;
    head = newNode;
}

void insertAtTail(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (head == NULL) {
        newNode->next = newNode;
        head = newNode;
        return;
    }

    struct Node* temp = head;
    while (temp->next != head)
        temp = temp->next;

    temp->next = newNode;
    newNode->next = head;
}

void insertAtPosition(int value, int position) {
```

```
if (position == 1) {
    insertAtHead(value);
    return;
}

struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;

struct Node* temp = head;
for (int i = 1; i < position - 1 && temp->next != head; i++)
    temp = temp->next;
newNode->next = temp->next;
temp->next = newNode;
}

void deleteAtHead() {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }

    if (head->next == head) {
        free(head);
        head = NULL;
        return;
    }

    struct Node* temp = head;
    while (temp->next != head)
        temp = temp->next;
    temp->next = head->next;
    struct Node* toDelete = head;
    head = head->next;
    free(toDelete);
}

void deleteAtTail() {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }

    if (head->next == head) {
        free(head);
        head = NULL;
        return;
    }
}
```

```
    struct Node* temp = head;
    while (temp->next->next != head)
        temp = temp->next;

    struct Node* toDelete = temp->next;
    temp->next = head;
    free(toDelete);
}

void deleteAtPosition(int position) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }

    if (position == 1) {
        deleteAtHead();
        return;
    }

    struct Node* temp = head;
    for (int i = 1; i < position - 1 && temp->next != head; i++)
        temp = temp->next;

    if (temp->next == head) {
        printf("Invalid position.\n");
        return;
    }

    struct Node* toDelete = temp->next;
    temp->next = temp->next->next;
    free(toDelete);
}

void search(int value) {
    struct Node* temp = head;
    int position = 1;
    do {
        if (temp->data == value) {
            printf("%d found at position %d.\n", value, position);
            return;
        }
        temp = temp->next;
        position++;
    } while (temp != head);
    printf("%d not found in the list.\n", value);
}

void display() {
```

```
if (head == NULL) {
    printf("List is empty.\n");
    return;
}

struct Node* temp = head;
do {
    printf("%d ", temp->data);
    temp = temp->next;
} while (temp != head);
printf("\n");
}

int main() {
    int choice, value, position;
    do {
        printf("\nCircular Linked List Operations:\n");
        printf("1. Insert at Head\n2. Insert at Tail\n3. Insert at Position\n4. Delete at Head\n5. Delete at Tail\n6. Delete at Position\n7. Search\n8. Display\n9. Quit\n");

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

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtHead(value);
                break;

            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtTail(value);
                break;

            case 3:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                printf("Enter position: ");
                scanf("%d", &position);
                insertAtPosition(value, position);
                break;

            case 4:
                deleteAtHead();
                break;
```

```
        case 5:
            deleteAtTail();
            break;

        case 6:
            printf("Enter position: ");
            scanf("%d", &position);
            deleteAtPosition(position);
            break;

        case 7:
            printf("Enter value to search: ");
            scanf("%d", &value);
            search(value);
            break;
        case 8:
            display();
            break;
        case 9:
            printf("Exiting...\n");
            break;

        default:
            printf("Invalid choice. Please try again.\n");
            break;
    } } while (choice != 9);
return 0;
}
```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_2
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc circular.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out

Circular Linked List Operations:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete at Head
5. Delete at Tail
6. Delete at Position
7. Search
8. Display
9. Quit
Enter your choice: 1
Enter value to insert: 10

Circular Linked List Operations:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete at Head
5. Delete at Tail
6. Delete at Position
7. Search
8. Display
9. Quit
Enter your choice: 2
Enter value to insert: 20

Circular Linked List Operations:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete at Head
5. Delete at Tail
6. Delete at Position
7. Search
8. Display
9. Quit
Enter your choice: 3
Enter value to insert: 30
Enter position: 2

Circular Linked List Operations:
1. Insert at Head
2. Insert at Tail

```

```

mca@HP-Z238: ~/abhirami/c_labcycle_2
7. Search
8. Display
9. Quit
Enter your choice: 8
10 30 20

Circular Linked List Operations:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete at Head
5. Delete at Tail
6. Delete at Position
7. Search
8. Display
9. Quit
Enter your choice: 6
Enter position: 3

Circular Linked List Operations:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete at Head
5. Delete at Tail
6. Delete at Position
7. Search
8. Display
9. Quit
Enter your choice: 7
Enter value to search: 20
20 not found in the list.

Circular Linked List Operations:
1. Insert at Head
2. Insert at Tail
3. Insert at Position
4. Delete at Head
5. Delete at Tail
6. Delete at Position
7. Search
8. Display
9. Quit
Enter your choice: 9
Exiting...
mca@HP-Z238:~/abhirami/c_labcycle_2$

```

13. Implement binary search tree.**Program:**

```
#include <stdio.h>

#include <stdlib.h>

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

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

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

struct Node* findMin(struct Node* node) {
    struct Node* current = node;
```



```
while (current && current->left != NULL)
    current = current->left;
return current;
}

struct Node* deleteNode(struct Node* root, int value) {
    if (root == NULL)
        return root;
    if (value < root->data)
        root->left = deleteNode(root->left, value);
    else if (value > root->data)
        root->right = deleteNode(root->right, value);
    else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

struct Node* search(struct Node* root, int value) {
    if (root == NULL || root->data == value)
```

```
        return root;
    if (value < root->data)
        return search(root->left, value);
    return search(root->right, value);
}

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

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

    do {
        printf("\nBinary Search Tree Operations:\n");
        printf("1. Insert\n2. Delete\n3. Search\n4. In-order Traversal\n5. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("Enter value to delete: ");
```

```
        scanf("%d", &value);
        root = deleteNode(root, value);
        break;
    case 3:
        printf("Enter value to search: ");
        scanf("%d", &value);
        struct Node* result = search(root, value);
        if (result != NULL)
            printf("%d found in the tree.\n", value);
        else
            printf("%d not found in the tree.\n", value);
        break;
    case 4:
        printf("In-order Traversal: ");
        inOrderTraversal(root);
        printf("\n");
        break;
    case 5:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
        break;
}
} while (choice != 5);

return 0;
}
```

Output:

```
mca@HP-Z238: ~/abhirami/c_labcycle_2
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc binarysearchtree.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out
Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 1
Enter value to insert: 5

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 1
Enter value to insert: 6

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 1
Enter value to insert: 4

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 4
In-order Traversal: 4 5 6

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 3
```

```
mca@HP-Z238: ~/abhirami/c_labcycle_2
Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 4
In-order Traversal: 4 5 6

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 3
Enter value to search: 5
5 found in the tree.

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 2
Enter value to delete: 5

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 4
In-order Traversal: 4 6

Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 5
Exiting...
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc circular.c
```

14. Implement balanced-binary-search tree.**Program:**

```

#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Node {
    int data;
    int height;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->height = 1;
    newNode->left = newNode->right = NULL;
    return newNode;
}

int calculateHeight(struct Node* node) {
    int leftHeight = (node->left) ? node->left->height : 0;
    int rightHeight = (node->right) ? node->right->height : 0;
    return (leftHeight > rightHeight) ? leftHeight + 1 : rightHeight + 1;
}

void updateHeight(struct Node* node) {
    node->height = calculateHeight(node);
}

struct Node* rightRotate(struct Node* y) {
    struct Node* x = y->left;
    struct Node* T = x->right;
    x->right = y;
    y->left = T;
    updateHeight(y);
    updateHeight(x);
    return x;
}

struct Node* leftRotate(struct Node* x) {
    struct Node* y = x->right;
    struct Node* T = y->left;
    y->left = x;
    x->right = T;
    updateHeight(x);
    updateHeight(y);
    return y;
}

struct Node* balanceNode(struct Node* node) {

```

```

    int balance = (node->left ? node->left->height : 0) - (node->right ? node->right->height
: 0);
    if (balance > 1) {
        if (node->left->left) {
            return rightRotate(node);
        } else {
            node->left = leftRotate(node->left);
            return rightRotate(node);
        }
    }
    if (balance < -1) {
        if (node->right->right) {
            return leftRotate(node);
        } else {
            node->right = rightRotate(node->right);
            return leftRotate(node);
        }
    }

    return node;
}

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

struct Node* findMin(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

struct Node* deleteNode(struct Node* root, int value) {
    if (root == NULL)
        return root;
    if (value < root->data)
        root->left = deleteNode(root->left, value);
    else if (value > root->data)
        root->right = deleteNode(root->right, value);

```

```
else {
    if (root->left == NULL) {
        struct Node* temp = root->right;
        free(root);
        return temp;
    } else if (root->right == NULL) {
        struct Node* temp = root->left;
        free(root);
        return temp;
    }

    struct Node* temp = findMin(root->right);
    root->data = temp->data;
    root->right = deleteNode(root->right, temp->data);
}
updateHeight(root);
return balanceNode(root);
}

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

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

int main() {
    struct Node* root = NULL;
    int choice, value;
    do {
        printf("\nBalanced Binary Search Tree Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Search\n");
        printf("4. In-order Traversal\n");
        printf("5. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
```

```
case 1:
    printf("Enter value to insert: ");
    scanf("%d", &value);
    if (search(root, value) == NULL)
        root = insert(root, value);
    else
        printf("%d is already in the tree.\n", value);
    break;
case 2:
    printf("Enter value to delete: ");
    scanf("%d", &value);
    if (search(root, value) != NULL)
        root = deleteNode(root, value);
    else
        printf("%d not found in the tree.\n", value);
    break;
case 3:
    printf("Enter value to search: ");
    scanf("%d", &value);
    struct Node* result = search(root, value);
    if (result != NULL)
        printf("%d found in the tree.\n", value);
    else
        printf("%d not found in the tree.\n", value);
    break;
case 4:
    printf("In-order Traversal: ");
    inOrderTraversal(root);
    printf("\n");
    break;
case 5:
    printf("Exiting...\n");
    break;
default:
    printf("Invalid choice. Please try again.\n");
    break;
}
} while (choice != 5);
while (root != NULL) {
    struct Node* temp = root;
    root = deleteNode(root, temp->data);
}
return 0;
}
```


Output:

```
mca@HP-Z238: ~/abhirami/c_labcycle_2
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc balanced.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 1
Enter value to insert: 30

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 1
Enter value to insert: 20

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 1
Enter value to insert: 40

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 4
In-order Traversal: 20 30 40

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 3
```

```
mca@HP-Z238: ~/abhirami/c_labcycle_2

1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 4
In-order Traversal: 20 30 40

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 3
Enter value to search: 50
50 not found in the tree.

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 2
Enter value to delete: 40

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 4
In-order Traversal: 20 30

Balanced Binary Search Tree Operations:
1. Insert
2. Delete
3. Search
4. In-order Traversal
5. Quit
Enter your choice: 5
Exiting...
mca@HP-Z238:~/abhirami/c_labcycle_2$
```

15. Implement set operations (union, intersection, difference).**Program:**

```
#include <stdio.h>

#define MAX_SIZE 100

int readSet(int set[]) {
    int size, i;

    printf("Enter the size of the set: ");
    scanf("%d", &size);

    printf("Enter elements of the set:\n");
    for (i = 0; i < size; i++) {
        scanf("%d", &set[i]);
    }
    return size;
}

void displaySet(int set[], int size) {
    int i;

    printf("Set: { ");
    for (i = 0; i < size; i++) {
        printf("%d ", set[i]);
    }
    printf("}\n");
}

int setUnion(int set1[], int size1, int set2[], int size2, int result[]) {
    int i, j, k = 0;
    for (i = 0; i < size1; i++) {
        result[k++] = set1[i];
    }
```

```
    for (i = 0; i < size2; i++) {
        int found = 0;
        for (j = 0; j < size1; j++) {
            if (set2[i] == set1[j]) {
                found = 1;
                break;
            }
        }
        if (!found) {
            result[k++] = set2[i];
        }
    }
    return k;
}

int setIntersection(int set1[], int size1, int set2[], int size2, int result[]) {
    int i, j, k = 0;
    for (i = 0; i < size1; i++) {
        for (j = 0; j < size2; j++) {
            if (set1[i] == set2[j]) {
                result[k++] = set1[i];
                break;
            }
        }
    }

    return k;
}

int setDifference(int set1[], int size1, int set2[], int size2, int result[]) {
    int i, j, k = 0;
    for (i = 0; i < size1; i++) {
```

```
int found = 0;
for (j = 0; j < size2; j++) {
    if (set1[i] == set2[j]) {
        found = 1;
        break;
    }
}
if (!found) {
    result[k++] = set1[i];
}
}
return k;
}

int main() {
    int set1[MAX_SIZE], set2[MAX_SIZE], result[MAX_SIZE];
    int size1, size2, resultSize;
    int choice;
    size1 = readSet(set1);
    size2 = readSet(set2);
    do {
        printf("\nSet Operations:\n");
        printf("1. Union\n");
        printf("2. Intersection\n");
        printf("3. Difference (set1 - set2)\n");
        printf("4. Display Sets\n");
        printf("5. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
```

```
        resultSize = setUnion(set1, size1, set2, size2, result);
        displaySet(result, resultSize);
        break;
    case 2:
        resultSize = setIntersection(set1, size1, set2, size2, result);
        displaySet(result, resultSize);
        break;
    case 3:
        resultSize = setDifference(set1, size1, set2, size2, result);
        displaySet(result, resultSize);
        break;
    case 4:
        printf("\nSets:\n");
        printf("Set 1: ");
        displaySet(set1, size1);
        printf("Set 2: ");
        displaySet(set2, size2);
        break;
    case 5:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
        break;
}
} while (choice != 5);
return 0;
}
```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_2
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc setoperations.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out
Enter the size of the set: 4
Enter elements of the set:
2
4
3
1
Enter the size of the set: 3
Enter elements of the set:
4
2
5
Set Operations:
1. Union
2. Intersection
3. Difference (set1 - set2)
4. Display Sets
5. Quit
Enter your choice: 1
Set: { 2 4 3 1 5 }

Set Operations:
1. Union
2. Intersection
3. Difference (set1 - set2)
4. Display Sets
5. Quit
Enter your choice: 2
Set: { 2 4 }

Set Operations:
1. Union
2. Intersection
3. Difference (set1 - set2)
4. Display Sets
5. Quit
Enter your choice: 3
Set: { 3 1 }

Set Operations:
1. Union
2. Intersection
3. Difference (set1 - set2)
4. Display Sets

```

```

mca@HP-Z238: ~/abhirami/c_labcycle_2
2. Intersection
3. Difference (set1 - set2)
4. Display Sets
5. Quit
Enter your choice: 1
Set: { 2 4 3 1 5 }

Set Operations:
1. Union
2. Intersection
3. Difference (set1 - set2)
4. Display Sets
5. Quit
Enter your choice: 2
Set: { 2 4 }

Set Operations:
1. Union
2. Intersection
3. Difference (set1 - set2)
4. Display Sets
5. Quit
Enter your choice: 3
Set: { 3 1 }

Set Operations:
1. Union
2. Intersection
3. Difference (set1 - set2)
4. Display Sets
5. Quit
Enter your choice: 4
Sets:
Set 1: Set: { 2 4 3 1 }
Set 2: Set: { 4 2 5 }

Set Operations:
1. Union
2. Intersection
3. Difference (set1 - set2)
4. Display Sets
5. Quit
Enter your choice: 5
Exiting...
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc treetraversal.c

```

16. Implement disjoint set operations.**Program:**

```
#include <stdio.h>

#define MAX 100

int parent[MAX], rank[MAX], n;

// Function to find the representative of the set containing element x
int find(int x) {
    if (x != parent[x])
        parent[x] = find(parent[x]); // Path Compression
    return parent[x];
}

int main() {
    printf("Enter the number of elements: ");
    if (scanf("%d", &n) != 1 || n <= 0 || n > MAX) {
        printf("Invalid input. Please enter a positive integer less than or equal to %d.\n",
MAX);
        return 1;
    }
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
    int choice, x, y;
    while (1) {
        printf("\nOperations:\n1. Union\n2. Find\n3. Display Set Representatives\n4.
Exit\nEnter your choice: ");
        if (scanf("%d", &choice) != 1) {
            printf("Invalid input. Please enter an integer.\n");
            continue;
        }
        switch (choice) {
```

case 1:

```
// Union operation
```

```
printf("Enter elements to perform union: ");
```

```
if (scanf("%d %d", &x, &y) != 2 || x < 0 || x >= n || y < 0 || y >= n) {
```

```
    printf("Invalid input. Please enter valid elements.\n");
```

```
} else {
```

```
    int rootX = find(x);
```

```
    int rootY = find(y);
```

```
    if (rootX == rootY) {
```

```
        printf("%d and %d are already in the same set.\n", x, y);
```

```
    } else {
```

```
        // Merge sets
```

```
        if (rank[rootX] > rank[rootY]) {
```

```
            parent[rootY] = rootX;
```

```
        } else if (rank[rootX] < rank[rootY]) {
```

```
            parent[rootX] = rootY;
```

```
        } else {
```

```
            parent[rootY] = rootX;
```

```
            rank[rootX]++;
```

```
        }
```

```
        printf("Union of %d and %d is performed.\n", x, y);
```

```
    }
```

```
}
```

```
break;
```

case 2:

```
// Find operation
```

```
printf("Enter element to find its set: ");
```

```
if (scanf("%d", &x) != 1 || x < 0 || x >= n) {
```

```
    printf("Invalid input. Please enter a valid element.\n");
```

```
} else {
```



```

        printf("Set representative of %d is %d\n", x, find(x));
    }

    break;

case 3:

    // Display set representatives

    printf("Set Representatives:\n");

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

        printf("Element %d belongs to set with representative %d\n", i, find(i));

    }

    break;

case 4:

    return 0;

default:

    printf("Invalid choice. Please enter a valid option.\n");

    break;

}

}

}

```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_2
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc 9disjoint.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out
Enter the number of elements: 3

Operations:
1. Union
2. Find
3. Display Set Representatives
4. Exit
Enter your choice: 1
Enter elements to perform union: 2
1
Union of 2 and 1 is performed.

Operations:
1. Union
2. Find
3. Display Set Representatives
4. Exit
Enter your choice: 2
Enter element to find its set: 2
Set representative of 2 is 2

Operations:
1. Union
2. Find
3. Display Set Representatives
4. Exit
Enter your choice: 3
Set Representatives:
Element 0 belongs to set with representative 0
Element 1 belongs to set with representative 2
Element 2 belongs to set with representative 2

Operations:
1. Union
2. Find
3. Display Set Representatives
4. Exit
Enter your choice: 4
mca@HP-Z238:~/abhirami/c_labcycle_2$

```

17. Implement tree traversal methods DFS (In-order, Pre-Order, Post-Order), and BFS.**Program:**

```
#include <stdio.h>

#include <stdlib.h>

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

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

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

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

```
}

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

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

void breadthFirstSearch(struct Node* root) {
    if (root == NULL)
        return;
    struct Node* queue[100];
    int front = -1, rear = -1;
    queue[++rear] = root;
    while (front < rear) {
        struct Node* current = queue[++front];
        printf("%d ", current->data);
        if (current->left != NULL)
            queue[++rear] = current->left;

        if (current->right != NULL)
            queue[++rear] = current->right;
    }
}
```

```
    }  
}  
  
int main() {  
    struct Node* root = NULL;  
  
    int choice, value;  
  
    do {  
  
        printf("\nTree traversal Operations:\n1. Insert\n2. In-order Traversal\n3. Pre-order  
Traversal\n4. Post-order Traversal\n5. Breadth-First Search (BFS)\n6. Quit\n");  
  
        printf("Enter your choice: ");  
  
        scanf("%d", &choice);  
  
        switch (choice) {  
  
            case 1:  
  
                printf("Enter value to insert: ");  
  
                scanf("%d", &value);  
  
                root = insert(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("Breadth-First Search (BFS): ");

            breadthFirstSearch(root);

            printf("\n");

            break;

        case 6:

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

            break;

        default:

            printf("Invalid choice. Please try again.\n");

            break;

    }

    } while (choice != 6);

    return 0;

}

```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_2
mca@HP-Z238:~/abhirami/c_labcycle_2$ gcc treetraversal.c
mca@HP-Z238:~/abhirami/c_labcycle_2$ ./a.out

Binary Tree Operations:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 1
Enter value to insert: 2

Binary Tree Operations:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 1
Enter value to insert: 5

Binary Tree Operations:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 1
Enter value to insert: 4

Binary Tree Operations:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 1
Enter value to insert: 7

Binary Tree Operations:
1. Insert
2. In-order Traversal

```

```
mca@HP-Z238: ~/abhiram/c_labcycle_2
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 2
In-order Traversal: 2 4 5 7

Binary Tree Operations:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 3
Pre-order Traversal: 2 5 4 7

Binary Tree Operations:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 4
Post-order Traversal: 4 7 5 2

Binary Tree Operations:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 5
Breadth-First Search (BFS): 2 5 4 7

Binary Tree Operations:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Breadth-First Search (BFS)
6. Quit
Enter your choice: 6
Exiting...
```

18. Implement Binomial Heaps and operations (Create, Insert, Delete).**Program:**

```

#include<stdio.h>

#include<malloc.h>

struct node {
    int n;
    int degree;
    struct node* parent;
    struct node* child;
    struct node* sibling;
};

struct node* MAKE_bin_HEAP();
int bin_LINK(struct node*, struct node*);
struct node* CREATE_NODE(int);
struct node* bin_HEAP_UNION(struct node*, struct node*);
struct node* bin_HEAP_INSERT(struct node*, struct node*);
struct node* bin_HEAP_MERGE(struct node*, struct node*);
struct node* bin_HEAP_EXTRACT_MIN(struct node*);
int REVERT_LIST(struct node*);
int DISPLAY(struct node*);
struct node* FIND_NODE(struct node*, int);
int bin_HEAP_DECREASE_KEY(struct node*, int, int);
int bin_HEAP_DELETE(struct node*, int);
int count = 1;
struct node* MAKE_bin_HEAP() {
    struct node* np;
    np = NULL;
    return np;
}
struct node * H = NULL;

```

```
struct node *Hr = NULL;

int bin_LINK(struct node* y, struct node* z) {
    y->parent = z;
    y->sibling = z->child;
    z->child = y;
    z->degree = z->degree + 1;
}

struct node* CREATE_NODE(int k) {
    struct node* p;//new node;
    p = (struct node*) malloc(sizeof(struct node));
    p->n = k;
    return p;
}

struct node* bin_HEAP_UNION(struct node* H1, struct node* H2) {
    struct node* prev_x;
    struct node* next_x;
    struct node* x;
    struct node* H = MAKE_bin_HEAP();
    H = bin_HEAP_MERGE(H1, H2);
    if (H == NULL)
        return H;
    prev_x = NULL;
    x = H;
    next_x = x->sibling;
    while (next_x != NULL) {
        if ((x->degree != next_x->degree) || ((next_x->sibling != NULL)
            && (next_x->sibling->degree == x->degree)) {
            prev_x = x;
            x = next_x;
        }
    }
```



```

    else {
        if (x->n <= next_x->n) {
            x->sibling = next_x->sibling;
            bin_LINK(next_x, x);
        }
        else {
            if (prev_x == NULL)
                H = next_x;
            else
                prev_x->sibling = next_x;
            bin_LINK(x, next_x);
            x = next_x;
        }
    }
    next_x = x->sibling;
}
return H;
}

struct node* bin_HEAP_INSERT(struct node* H, struct node* x) {
    struct node* H1 = MAKE_bin_HEAP();
    x->parent = NULL;
    x->child = NULL;
    x->sibling = NULL;
    x->degree = 0;
    H1 = x;
    H = bin_HEAP_UNION(H, H1);
    return H;
}

struct node* bin_HEAP_MERGE(struct node* H1, struct node* H2) {
    struct node* H = MAKE_bin_HEAP();

```

```
struct node* y;
struct node* z;
struct node* a;
struct node* b;
y = H1;
z = H2;
if (y != NULL) {
    if (z != NULL && y->degree <= z->degree)
        H = y;
    else if (z != NULL && y->degree > z->degree)
        /* need some modifications here;the first and the else conditions
        can be merged together!!!! */
        H = z;
    else
        H = y;
} else
    H = z;
while (y != NULL && z != NULL) {
    if (y->degree < z->degree) {
        y = y->sibling;
    }
    else if (y->degree == z->degree) {
        a = y->sibling;
        y->sibling = z;
        y = a;
    } else {
        b = z->sibling;
        z->sibling = y;
        z = b;
    }
}
```

```
    }  
    return H;  
}  
  
int DISPLAY(struct node* H) {  
    struct node* p;  
    if (H == NULL) {  
        printf("\nHEAP EMPTY");  
        return 0;  
    }  
    printf("\nTHE ROOT NODES ARE:-\n");  
    p = H;  
    while (p != NULL) {  
        printf("%d", p->n);  
        if (p->sibling != NULL)  
            printf("-->");  
        p = p->sibling;  
    }  
    printf("\n");  
}  
  
struct node* bin_HEAP_EXTRACT_MIN(struct node* H1) {  
    int min;  
    struct node* t = NULL;  
    struct node* x = H1;  
    struct node *Hr;  
    struct node* p;  
    Hr = NULL;  
    if (x == NULL) {  
        printf("\nNOTHING TO EXTRACT");  
        return x;  
    }  
}
```

```
// int min=x->n;

p = x;
while (p->sibling != NULL) {
    if ((p->sibling)->n < min) {
        min = (p->sibling)->n;
        t = p;
        x = p->sibling;
    }
    p = p->sibling;
}

if (t == NULL && x->sibling == NULL)
    H1 = NULL;
else if (t == NULL)
    H1 = x->sibling;
else if (t->sibling == NULL)
    t = NULL;
else
    t->sibling = x->sibling;
if (x->child != NULL) {
    REVERT_LIST(x->child);
    (x->child)->sibling = NULL;
}
H = bin_HEAP_UNION(H1, Hr);
return x;
}

int REVERT_LIST(struct node* y){
    if (y->sibling != NULL) {
        REVERT_LIST(y->sibling);
        (y->sibling)->sibling = y;
    } else {
```

```

        Hr = y;
    }
}

struct node* FIND_NODE(struct node* H, int k) {
    struct node* x = H;
    struct node* p = NULL;
    if (x->n == k) {
        p = x;
        return p;
    }
    if (x->child != NULL && p == NULL) {
        p = FIND_NODE(x->child, k);
    }
    if (x->sibling != NULL && p == NULL){
        p = FIND_NODE(x->sibling, k);
    }
    return p;
}

int bin_HEAP_DECREASE_KEY(struct node* H, int i, int k) {
    int temp;
    struct node* p;
    struct node* y;
    struct node* z;
    p = FIND_NODE(H, i);
    if (p == NULL){
        printf("\nINVALID CHOICE OF KEY TO BE REDUCED");
        return 0;
    }
    if (k > p->n){
        printf("\nSORRY!THE NEW KEY IS GREATER THAN CURRENT ONE");

```

```
        return 0;
    }
    p->n = k;
    y = p;
    z = p->parent;
    while (z != NULL && y->n < z->n) {
        temp = y->n;
        y->n = z->n;
        z->n = temp;
        y = z;
        z = z->parent;
    }
    printf("\nKEY REDUCED SUCCESSFULLY!");
}

int bin_HEAP_DELETE(struct node* H, int k){
    struct node* np;
    if (H == NULL){
        printf("\nHEAP EMPTY");
        return 0;
    }
    bin_HEAP_DECREASE_KEY(H, k, -1000);
    np = bin_HEAP_EXTRACT_MIN(H);
    if (np != NULL)
        printf("\nNODE DELETED SUCCESSFULLY");
}

int main(){
    int i, n, m, l;
    struct node* p;
    struct node* np;
    char ch;
```

```
printf("\nENTER THE NUMBER OF ELEMENTS:");
scanf("%d", &n);
printf("\nENTER THE ELEMENTS:\n");
for (i = 1; i <= n; i++){
    scanf("%d", &m);
    np = CREATE_NODE(m);
    H = bin_HEAP_INSERT(H, np);
}
DISPLAY(H);
do{
    printf("\nMENU:-\n");
    printf("\n1)INSERT AN ELEMENT\n2)EXTRACT THE MINIMUM KEY
NODE\n3)DECREASE A NODE KEY\n4)DELETE A NODE\n5)QUIT\n");
    scanf("%d", &l);
    switch (l) {
    case 1:
        do{
            printf("\nENTER THE ELEMENT TO BE INSERTED:");
            scanf("%d", &m);
            p = CREATE_NODE(m);
            H = bin_HEAP_INSERT(H, p);
            printf("\nNOW THE HEAP IS:\n");
            DISPLAY(H);
            printf("\nINSERT MORE(y/Y)= \n");
            fflush(stdin);
            scanf("%c", &ch);
        } while (ch == 'Y' || ch == 'y');
        break;
    case 2:
        do{
            printf("\nEXTRACTING THE MINIMUM KEY NODE");
```

```
p = bin_HEAP_EXTRACT_MIN(H);
if (p != NULL)
    printf("\nTHE EXTRACTED NODE IS %d", p->n);
printf("\nNOW THE HEAP IS:\n");
DISPLAY(H);
printf("\nEXTRACT MORE(y/Y)\n");
fflush(stdin);
scanf("%c", &ch);
} while (ch == 'Y' || ch == 'y');
break;
case 3:
do{
    printf("\nENTER THE KEY OF THE NODE TO BE DECREASED:");
    scanf("%d", &m);
    printf("\nENTER THE NEW KEY : ");
    scanf("%d", &l);
    bin_HEAP_DECREASE_KEY(H, m, l);
    printf("\nNOW THE HEAP IS:\n");
    DISPLAY(H);
    printf("\nDECREASE MORE(y/Y)\n");
    fflush(stdin);
    scanf("%c", &ch);
} while (ch == 'Y' || ch == 'y');
break;
case 4:
do{
    printf("\nENTER THE KEY TO BE DELETED: ");
    scanf("%d", &m);
    bin_HEAP_DELETE(H, m);
    printf("\nDELETE MORE(y/Y)\n");
```



```

        fflush(stdin);

        scanf("%c", &ch);

    } while (ch == 'y' || ch == 'Y');

    break;

case 5:

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

    break;

default:

    printf("\nINVALID ENTRY...TRY AGAIN. ..\n");

}

} while (1 != 5);

}

```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_3
mca@HP-Z238:~/abhirami/c_labcycle_3$ gcc heap1.c
mca@HP-Z238:~/abhirami/c_labcycle_3$ ./a.out

ENTER THE NUMBER OF ELEMENTS:5

ENTER THE ELEMENTS:
2
6
7
2
9

THE ROOT NODES ARE:-
9-->2

MENU:-
1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
4)DELETE A NODE
5)QUIT
1

ENTER THE ELEMENT TO BE INSERTED:3

NOW THE HEAP IS:

THE ROOT NODES ARE:-
3-->2

INSERT MORE(y/Y)=

MENU:-
1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
4)DELETE A NODE
5)QUIT
2

EXTRACTING THE MINIMUM KEY NODE
THE EXTRACTED NODE IS 2
NOW THE HEAP IS:

```

```
mca@HP-Z238: ~/abhirami/c_labcycle_3

MENU: -
1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
4)DELETE A NODE
5)QUIT
2

EXTRACTING THE MINIMUM KEY NODE
THE EXTRACTED NODE IS 2
NOW THE HEAP IS:

THE ROOT NODES ARE:-
3

EXTRACT MORE(y/Y)

MENU: -
1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
4)DELETE A NODE
5)QUIT
4

ENTER THE KEY TO BE DELETED: 7

INVALID CHOICE OF KEY TO BE REDUCED
NODE DELETED SUCCESSFULLY
DELETE MORE(y/Y)

MENU: -
1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
4)DELETE A NODE
5)QUIT
5

Exiting....
mca@HP-Z238:~/abhirami/c_labcycle_3$ 3~
```

19. Implement B Trees and its operations.**Program:**

```

#include <stdio.h>

#include <stdlib.h>

#define MAX 4

#define MIN 2

struct btreeNode{
    int val[MAX + 1], count;
    struct btreeNode *link[MAX + 1];
};

struct btreeNode *root;

struct btreeNode * createNode(int val, struct btreeNode *child) {
    struct btreeNode *newNode;
    newNode = (struct btreeNode *)malloc(sizeof(struct btreeNode));
    newNode->val[1] = val;
    newNode->count = 1;
    newNode->link[0] = root;
    newNode->link[1] = child;
    return newNode;
}

void addValToNode(int val, int pos, struct btreeNode *node, struct btreeNode *child){
    int j = node->count;
    while (j > pos){
        node->val[j + 1] = node->val[j];
        node->link[j + 1] = node->link[j];
        j--;
    }
    node->val[j + 1] = val;
    node->link[j + 1] = child;
    node->count++;
}

```

```

    }

void splitNode (int val, int *pval, int pos, struct btreeNode *node,
    struct btreeNode *child, struct btreeNode **newNode) {
    int median, j;
    if (pos > MIN)
        median = MIN + 1;
    else
        median = MIN;
    *newNode = (struct btreeNode *)malloc(sizeof(struct btreeNode));
    j = median + 1;
    while (j <= MAX){
        (*newNode)->val[j - median] = node->val[j];
        (*newNode)->link[j - median] = node->link[j];
        j++;
    }
    node->count = median;
    (*newNode)->count = MAX - median;
    if (pos <= MIN) {
        addValToNode(val, pos, node, child);
    }
    else{
        addValToNode(val, pos - median, *newNode, child);
    }

    *pval = node->val[node->count];
    (*newNode)->link[0] = node->link[node->count];
    node->count--;
}

int setValueInNode(int val, int *pval, struct btreeNode *node, struct btreeNode **child){
    int pos;
    if (!node){

```

```
        *pval = val;
        *child = NULL;
        return 1;
    }
    if (val < node->val[1]){
        pos = 0;
    }
    else{
        for (pos = node->count;
            (val < node->val[pos] && pos > 1); pos--);
        if (val == node->val[pos]){
            printf("Duplicates not allowed\n");
            return 0;
        }
    }
    if (setValueInNode(val, pval, node->link[pos], child)){
        if (node->count < MAX){
            addValToNode(*pval, pos, node, *child);
        } else{
            splitNode(*pval, pval, pos, node, *child, child);
            return 1;
        }
    }
    return 0;
}

void insertion(int val){
    int flag, i;
    struct btreeNode *child;
    flag = setValueInNode(val, &i, root, &child);
    if (flag)
```

```
        root = createNode(i, child);
    }

void copySuccessor(struct btreeNode *myNode, int pos){
    struct btreeNode *dummy;
    dummy = myNode->link[pos];
    for (;dummy->link[0] != NULL;)
        dummy = dummy->link[0];
    myNode->val[pos] = dummy->val[1];
}

void removeVal(struct btreeNode *myNode, int pos){
    int i = pos + 1;
    while (i <= myNode->count) {
        myNode->val[i - 1] = myNode->val[i];
        myNode->link[i - 1] = myNode->link[i];
        i++;
    }
    myNode->count--;
}

void doRightShift(struct btreeNode *myNode, int pos){
    struct btreeNode *x = myNode->link[pos];
    int j = x->count;
    while (j > 0) {
        x->val[j + 1] = x->val[j];
        x->link[j + 1] = x->link[j];
    }
    x->val[1] = myNode->val[pos];
    x->link[1] = x->link[0];
    x->count++;
    x = myNode->link[pos - 1];
    myNode->val[pos] = x->val[x->count];
}
```

```
    myNode->link[pos] = x->link[x->count];
    x->count--;
    return;
}

void doLeftShift(struct btreeNode *myNode, int pos){
    int j = 1;
    struct btreeNode *x = myNode->link[pos - 1];
    x->count++;
    x->val[x->count] = myNode->val[pos];
    x->link[x->count] = myNode->link[pos]->link[0];
    x = myNode->link[pos];
    myNode->val[pos] = x->val[1];
    x->link[0] = x->link[1];
    x->count--;
    while (j <= x->count) {
        x->val[j] = x->val[j + 1];
        x->link[j] = x->link[j + 1];
        j++;
    }
    return;
}

void mergeNodes(struct btreeNode *myNode, int pos){
    int j = 1;
    struct btreeNode *x1 = myNode->link[pos], *x2 = myNode->link[pos - 1];
    x2->count++;
    x2->val[x2->count] = myNode->val[pos];
    x2->link[x2->count] = myNode->link[0];
    while (j <= x1->count){
        x2->count++;
        x2->val[x2->count] = x1->val[j];
```

```
        x2->link[x2->count] = x1->link[j];

        j++;
    }
    j = pos;
    while (j < myNode->count){
        myNode->val[j] = myNode->val[j + 1];
        myNode->link[j] = myNode->link[j + 1];
        j++;
    }
    myNode->count--;
    free(x1);
}

void adjustNode(struct btreeNode *myNode, int pos){
    if (!pos) {
        if (myNode->link[1]->count > MIN){
            doLeftShift(myNode, 1);
        } else{
            mergeNodes(myNode, 1);
        }
    } else{
        if (myNode->count != pos){
            if(myNode->link[pos - 1]->count > MIN){
                doRightShift(myNode, pos);
            } else{
                if (myNode->link[pos + 1]->count > MIN){
                    doLeftShift(myNode, pos + 1);
                } else{
                    mergeNodes(myNode, pos);
                }
            }
        }
    }
}
```



```
    } else{
        if (myNode->link[pos - 1]->count > MIN)
            doRightShift(myNode, pos);
        else
            mergeNodes(myNode, pos);
    }
}

int delValFromNode(int val, struct btreeNode *myNode){
    int pos, flag = 0;
    if (myNode) {
        if (val < myNode->val[1]) {
            pos = 0;
            flag = 0;
        } else{
            for (pos = myNode->count;
                (val < myNode->val[pos] && pos > 1); pos--);
            if (val == myNode->val[pos]){
                flag = 1;
            }
            else{
                flag = 0;
            }
        }
    }
    if (flag){
        if (myNode->link[pos - 1]){
            copySuccessor(myNode, pos);
            flag = delValFromNode(myNode->val[pos], myNode->link[pos]);
            if (flag == 0){
                printf("Given data is not present in B-Tree\n");
            }
        } else{
            removeVal(myNode, pos);
        }
    }
}
```

```
        }
    } else{
        flag = delValFromNode(val, myNode->link[pos]);
    }
    if (myNode->link[pos]){
        if (myNode->link[pos]->count < MIN)
            adjustNode(myNode, pos);
    }
}
return flag;
}

void deletion(int val, struct btreeNode *myNode){
    struct btreeNode *tmp;
    if (!delValFromNode(val, myNode)){
        printf("Given value is not present in B-Tree\n");
        return;
    } else{
        if (myNode->count == 0){
            tmp = myNode;
            myNode = myNode->link[0];
            free(tmp);
        }
        root = myNode;
        return;
    }
}

void searching(int val, int *pos, struct btreeNode *myNode){
    if (!myNode){
        return;
    }
    if (val < myNode->val[1])
    {
```

```
        *pos = 0;
    } else{
        for (*pos = myNode->count;
            (val < myNode->val[*pos] && *pos > 1); (*pos)--);
        if (val == myNode->val[*pos]) {
            printf("Given data %d is present in B-Tree", val);
            return;
        }
        searching(val, pos, myNode->link[*pos]);
        return;
    }
/* B-Tree Traversal */
void traversal(struct btreeNode *myNode){
    int i;
    if (myNode){
        for (i = 0; i < myNode->count; i++){
            traversal(myNode->link[i]);
            printf("%d ", myNode->val[i + 1]);
        }
        traversal(myNode->link[i]);
    }
}
int main(){
    int val, ch;
    while (1){
        printf("\n1. Insertion\n2. Deletion\n3. Searching\n4. Traversal\n5. Exit\n");
        printf("Enter your choice:\n");
        scanf("%d", &ch);
        switch (ch){
            case 1:
                printf("Enter your element:");
```

```
        scanf("%d", &val);
        insertion(val);
        break;
    case 2:
        printf("Enter the element to delete:");
        scanf("%d", &val);
        deletion(val, root);
        break;
    case 3:
        printf("Enter the element to search:");
        scanf("%d", &val);
        searching(val, &ch, root);
        break;
    case 4:
        traversal(root);
        break;
    case 5:
        exit(0);
    default:
        printf("U have entered wrong option!!\n");
        break;
    }
    printf("\n");
}
}
```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_3
mca@HP-Z238:~/abhirami/c_labcycle_3$ gcc btree2.c
mca@HP-Z238:~/abhirami/c_labcycle_3$ ./a.out
1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
1
Enter your element:4

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
1
Enter your element:7

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
1
Enter your element:2

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
4
2 4 7

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:

```

```

mca@HP-Z238: ~/abhirami/c_labcycle_3
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
4
2 4 7

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
3
Enter the element to search:4
Given data 4 is present in B-Tree

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
2
Enter the element to delete:7

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
4
2 4

1. Insertion
2. Deletion
3. Searching
4. Traversal
5. Exit
Enter your choice:
5
mca@HP-Z238:~/abhirami/c_labcycle_3$

```

20. Implement Red Black Trees and its operations.**Program:**

```
#include <stdio.h>

#include <stdlib.h>

// Node structure for Red-Black Tree
typedef struct Node {
    int key;
    int color; // 0 for black, 1 for red
    struct Node* parent;
    struct Node* left;
    struct Node* right;
} Node;

// Sentinel node to represent NULL (external nodes in the tree)
Node* NIL;

// Function to create a new node with given key and color
Node* createNode(int key, int color) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->key = key;
    newNode->color = color;
    newNode->parent = NIL;
    newNode->left = NIL;
    newNode->right = NIL;
    return newNode;
}

// Function to perform left rotation on the given node
void leftRotate(Node** root, Node* x) {
    Node* y = x->right;
    x->right = y->left;

    if (y->left != NIL)
```

```
    y->left->parent = x;

    y->parent = x->parent;

    if (x->parent == NIL)
        *root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;

    y->left = x;
    x->parent = y;
}

// Function to perform right rotation on the given node
void rightRotate(Node** root, Node* y) {
    Node* x = y->left;
    y->left = x->right;

    if (x->right != NIL)
        x->right->parent = y;

    x->parent = y->parent;

    if (y->parent == NIL)
        *root = x;
    else if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
```

```
x->right = y;
y->parent = x;
}

// Function to fix the Red-Black Tree after insertion
void insertFixup(Node** root, Node* z) {
    while (z->parent->color == 1) {
        if (z->parent == z->parent->parent->left) {
            Node* y = z->parent->parent->right;
            if (y->color == 1) {
                z->parent->color = 0;
                y->color = 0;
                z->parent->parent->color = 1;
                z = z->parent->parent;
            } else {
                if (z == z->parent->right) {
                    z = z->parent;
                    leftRotate(root, z);
                }
                z->parent->color = 0;
                z->parent->parent->color = 1;
                rightRotate(root, z->parent->parent);
            }
        } else {
            Node* y = z->parent->parent->left;
            if (y->color == 1) {
                z->parent->color = 0;
                y->color = 0;
                z->parent->parent->color = 1;
                z = z->parent->parent;
            }
        }
    }
}
```



```
    } else {
        if (z == z->parent->left) {
            z = z->parent;
            rightRotate(root, z);
        }
        z->parent->color = 0;
        z->parent->parent->color = 1;
        leftRotate(root, z->parent->parent);
    }
}

}

(*root)->color = 0;
}

// Function to insert a key into the Red-Black Tree
void insert(Node** root, int key) {
    Node* z = createNode(key, 1);
    Node* y = NIL;
    Node* x = *root;

    while (x != NIL) {
        y = x;
        if (z->key < x->key)
            x = x->left;
        else
            x = x->right;
    }
    z->parent = y;

    if (y == NIL)
        *root = z;
```

```
    else if (z->key < y->key)
        y->left = z;
    else
        y->right = z;

    z->left = NIL;
    z->right = NIL;
    z->color = 1; // Red

    insertFixup(root, z);
}

// Function to print the Red-Black Tree (in-order traversal)
void inOrderTraversal(Node* root) {
    if (root != NIL) {
        inOrderTraversal(root->left);
        printf("%d (%s) ", root->key, root->color == 0 ? "Black" : "Red");
        inOrderTraversal(root->right);
    }
}

int main() {
    NIL = createNode(0, 0);
    Node* root = NIL;
    int choice, key;
    do {
        printf("\n1. Insert\n2. Display\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```

        printf("Enter key to insert: ");

        scanf("%d", &key);

        insert(&root, key);

        break;

    case 2:

        printf("Red-Black Tree (in-order traversal): \n");

        inOrderTraversal(root);

        printf("\n");

        break;

    case 3:

        exit(0);

    default:

        printf("Invalid choice!\n");

    }

} while (1);

return 0;

}

```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_3
mca@HP-Z238:~/abhirami/c_labcycle_3$ gcc red3.c
mca@HP-Z238:~/abhirami/c_labcycle_3$ ./a.out

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter key to insert: 5

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter key to insert: 8

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter key to insert: 20

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter key to insert: 6

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter key to insert: 12

1. Insert
2. Display
3. Exit
Enter your choice: 2
Red-Black Tree (in-order traversal):
5 (Black) 6 (Red) 8 (Black) 12 (Red) 20 (Black)

1. Insert
2. Display
3. Exit
Enter your choice: 3
mca@HP-Z238:~/abhirami/c_labcycle_3$

```

21. Graph Traversal techniques (DFS and BFS) and Topological Sorting.**Program:**

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX_VERTICES 5

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

typedef struct Graph {
    Node* adjList[MAX_VERTICES];
    int numVertices;
} Graph;

Graph* createGraph(int numVertices) {
    if (numVertices <= 0 || numVertices > MAX_VERTICES) {
        printf("Invalid number of vertices. Exiting...\n");
        exit(EXIT_FAILURE);
    }

    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->numVertices = numVertices;
    for (int i = 0; i < numVertices; ++i)
        graph->adjList[i] = NULL;

    return graph;
}

void addEdge(Graph* graph, int src, int dest) {
    if (src < 0 || src >= graph->numVertices || dest < 0 || dest >= graph->numVertices) {
        printf("Invalid source or destination vertex. Ignoring edge...\n");
        return;
    }
```

```
    }  
    Node* newNode = (Node*)malloc(sizeof(Node));  
    newNode->vertex = dest;  
    newNode->next = graph->adjList[src];  
    graph->adjList[src] = newNode;  
}  
void DFS(Graph* graph, int vertex, bool visited[]) {  
    visited[vertex] = true;  
    printf("%d ", vertex);  
    Node* adjNode = graph->adjList[vertex];  
    while (adjNode != NULL) {  
        int adjVertex = adjNode->vertex;  
        if (!visited[adjVertex])  
            DFS(graph, adjVertex, visited);  
        adjNode = adjNode->next;  
    }  
}  
void BFS(Graph* graph, int start) {  
    if (start < 0 || start >= graph->numVertices) {  
        printf("Invalid starting vertex. Exiting...\n");  
        exit(EXIT_FAILURE);  
    }  
    bool visited[MAX_VERTICES] = { false };  
    int queue[MAX_VERTICES];  
    int front = 0, rear = -1;  
    visited[start] = true;  
    queue[++rear] = start;  
    while (front <= rear) {  
        int vertex = queue[front++];  
        printf("%d ", vertex);
```

```
Node* adjNode = graph->adjList[vertex];
while (adjNode != NULL) {
    int adjVertex = adjNode->vertex;
    if (!visited[adjVertex]) {
        visited[adjVertex] = true;
        queue[(++rear) % MAX_VERTICES] = adjVertex
    }
    adjNode = adjNode->next;
}
}

void topologicalSortUtil(Graph* graph, int vertex, bool visited[], int stack[], int*
stackIndex) {
    visited[vertex] = true;
    Node* adjNode = graph->adjList[vertex];
    while (adjNode != NULL) {
        int adjVertex = adjNode->vertex;
        if (!visited[adjVertex])
            topologicalSortUtil(graph, adjVertex, visited, stack, stackIndex);
        adjNode = adjNode->next;
    }

    stack[++(*stackIndex)] = vertex;
}

void topologicalSort(Graph* graph) {
    bool visited[MAX_VERTICES] = { false };
    int stack[MAX_VERTICES];
    int stackIndex = -1;
    for (int i = 0; i < graph->numVertices; ++i) {
        if (!visited[i])
            topologicalSortUtil(graph, i, visited, stack, &stackIndex);
    }
}
```

```
    }

    printf("Topological Sorting: ");
    while (stackIndex >= 0)
        printf("%d ", stack[stackIndex--]);
    }

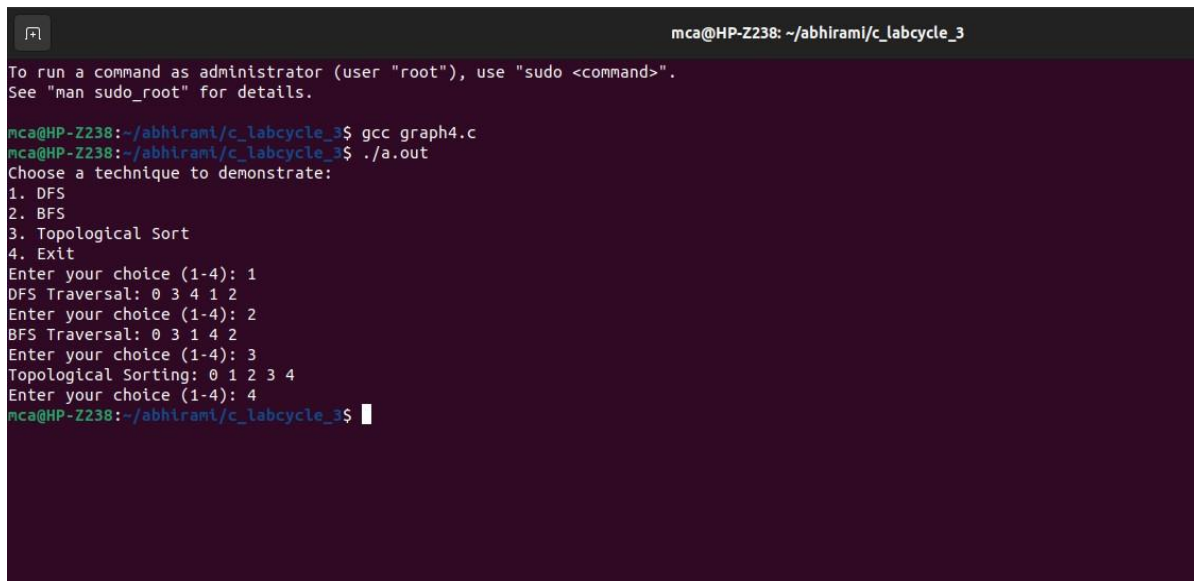
int main() {
    Graph* graph = createGraph(MAX_VERTICES);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 3);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);

    int choice;

    printf("Choose a technique to demonstrate:\n");
    printf("1. DFS\n2. BFS\n3. Topological Sort\n4. Exit\n");
    while (1) {
        printf("Enter your choice (1-4): ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("DFS Traversal: ");
                DFS(graph, 0, (bool[MAX_VERTICES]){ false });
                printf("\n");
                break;
            case 2:
                printf("BFS Traversal: ");
                BFS(graph, 0);
                printf("\n");
                break;
            case 3:
```

```
        topologicalSort(graph);  
        printf("\n");  
        break;  
    case 4:  
        exit(EXIT_SUCCESS);  
    default:  
        printf("Invalid choice!\n");  
    }  
}  
return 0;  
}
```

Output:



```
mca@HP-Z238: ~/abhirami/c_labcycle_3  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
mca@HP-Z238:~/abhirami/c_labcycle_3$ gcc graph4.c  
mca@HP-Z238:~/abhirami/c_labcycle_3$ ./a.out  
Choose a technique to demonstrate:  
1. DFS  
2. BFS  
3. Topological Sort  
4. Exit  
Enter your choice (1-4): 1  
DFS Traversal: 0 3 4 1 2  
Enter your choice (1-4): 2  
BFS Traversal: 0 3 1 4 2  
Enter your choice (1-4): 3  
Topological Sorting: 0 1 2 3 4  
Enter your choice (1-4): 4  
mca@HP-Z238:~/abhirami/c_labcycle_3$
```


22. Finding the Strongly connected Components in a directed graph.**Program:**

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX_VERTICES 100

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

typedef struct Graph {
    Node* adjList[MAX_VERTICES];
    Node* revAdjList[MAX_VERTICES];
    int numVertices;
} Graph;

Graph* createGraph(int numVertices) {
    if (numVertices <= 0 || numVertices > MAX_VERTICES) {
        printf("Invalid number of vertices. Exiting...\n");
        exit(EXIT_FAILURE);
    }

    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->numVertices = numVertices;
    for (int i = 0; i < numVertices; ++i) {
        graph->adjList[i] = NULL;
        graph->revAdjList[i] = NULL;
    }
    return graph;
}

void addEdge(Graph* graph, int src, int dest) {
    if (src < 0 || src >= graph->numVertices || dest < 0 || dest >= graph->numVertices) {
```

```
    printf("Invalid source or destination vertex. Ignoring edge...\n");
    return;
}
Node* newNode = (Node*)malloc(sizeof(Node));
newNode->vertex = dest;
newNode->next = graph->adjList[src];
graph->adjList[src] = newNode;
// Reverse graph for Kosaraju's algorithm
newNode = (Node*)malloc(sizeof(Node));
newNode->vertex = src;
newNode->next = graph->revAdjList[dest];
graph->revAdjList[dest] = newNode;
}
void DFSUtil(Graph* graph, int vertex, bool visited[]) {
    visited[vertex] = true;
    printf("%d ", vertex);
    Node* adjNode = graph->adjList[vertex];
    while (adjNode != NULL) {
        int adjVertex = adjNode->vertex;
        if (!visited[adjVertex])
            DFSUtil(graph, adjVertex, visited);
        adjNode = adjNode->next;
    }
}
void fillOrder(Graph* graph, int vertex, bool visited[], int stack[], int* stackIndex) {
    visited[vertex] = true;
    Node* adjNode = graph->revAdjList[vertex];
    while (adjNode != NULL) {
        int adjVertex = adjNode->vertex;
        if (!visited[adjVertex])
```

```

        fillOrder(graph, adjVertex, visited, stack, stackIndex);
        adjNode = adjNode->next;
    }
    stack[++(*stackIndex)] = vertex;
}

Graph* getTranspose(Graph* graph) {
    Graph* transposedGraph = createGraph(graph->numVertices);
    for (int i = 0; i < graph->numVertices; ++i) {
        Node* current = graph->adjList[i];
        while (current != NULL) {
            addEdge(transposedGraph, current->vertex, i);
            current = current->next;
        }
    }
    return transposedGraph;
}

void printSCCs(Graph* graph) {
    int stack[MAX_VERTICES];
    int stackIndex = -1;
    bool visited[MAX_VERTICES] = { false };
    for (int i = 0; i < graph->numVertices; ++i) {
        if (!visited[i])
            fillOrder(graph, i, visited, stack, &stackIndex);
    }
    Graph* transposedGraph = getTranspose(graph);
    for (int i = 0; i < graph->numVertices; ++i)
        visited[i] = false;
    while (stackIndex >= 0) {
        int vertex = stack[stackIndex--];
        if (!visited[vertex]) {

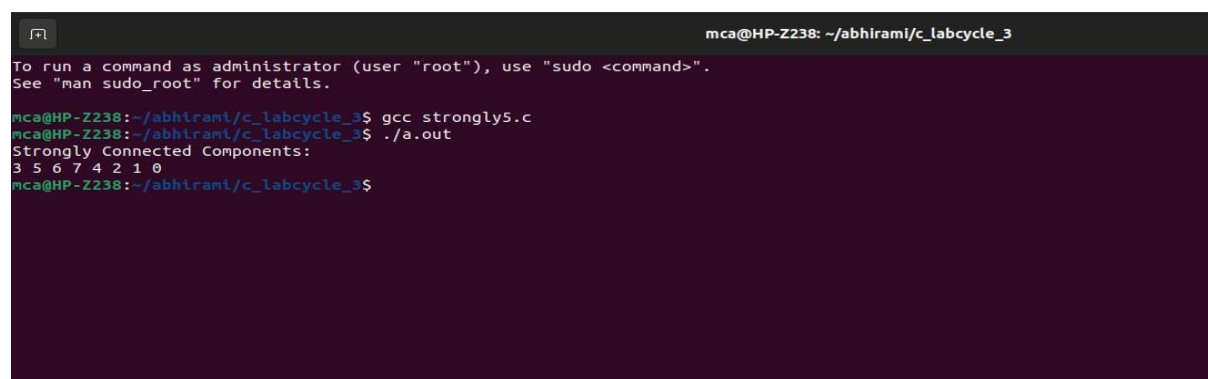
```

```
        DFSUtil(transposedGraph, vertex, visited);
        printf("\n");
    }
}

free(transposedGraph);
}

int main() {
    Graph* graph = createGraph(8);
    // Define edges for demonstration
    addEdge(graph, 0, 1);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 0);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);
    addEdge(graph, 4, 5);
    addEdge(graph, 5, 3);
    addEdge(graph, 6, 5);
    addEdge(graph, 6, 7);
    addEdge(graph, 7, 6);
    printf("Strongly Connected Components:\n");
    printSCCs(graph);
    return 0;
}
```

Output:



```
mca@HP-Z238: ~/abhirami/c_labcycle_3
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mca@HP-Z238:~/abhirami/c_labcycle_3$ gcc strongly5.c
mca@HP-Z238:~/abhirami/c_labcycle_3$ ./a.out
Strongly Connected Components:
3 5 6 7 4 2 1 0
mca@HP-Z238:~/abhirami/c_labcycle_3$
```

23. Prim's Algorithm for finding the minimum cost spanning tree.**Program:**

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX_VERTICES 100

#define INF 999999

typedef struct {

    int parent;

    int key;

    bool inMST;

} Vertex;

typedef struct Graph {

    int numVertices;

    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];

    Vertex vertices[MAX_VERTICES];

} Graph;

Graph* createGraph(int numVertices) {

    if (numVertices <= 0 || numVertices > MAX_VERTICES) {

        printf("Invalid number of vertices. Exiting...\n");

        exit(EXIT_FAILURE);

    }

    Graph* graph = (Graph*)malloc(sizeof(Graph));

    graph->numVertices = numVertices;

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

        graph->vertices[i].parent = -1;

        graph->vertices[i].key = INF;

        graph->vertices[i].inMST = false;

        for (int j = 0; j < numVertices; ++j)

            graph->adjacencyMatrix[i][j] = INF;
```

```

    }

return graph;
}

void addEdge(Graph* graph, int src, int dest, int weight) {
    if (src >= 0 && src < graph->numVertices && dest >= 0 && dest < graph->numVertices) {
        graph->adjacencyMatrix[src][dest] = weight;
        graph->adjacencyMatrix[dest][src] = weight;
    } else {
        printf("Invalid source or destination vertex. Ignoring edge...\n");
    }
}

int findMinKeyVertex(Graph* graph) {
    int minKey = INF;
    int minIndex = -1;
    for (int i = 0; i < graph->numVertices; ++i) {
        if (!graph->vertices[i].inMST && graph->vertices[i].key < minKey) {
            minKey = graph->vertices[i].key;
            minIndex = i;
        }
    }
    return minIndex;
}

void primMST(Graph* graph) {
    graph->vertices[0].key = 0;
    for (int count = 0; count < graph->numVertices - 1; ++count) {
        int u = findMinKeyVertex(graph);
        graph->vertices[u].inMST = true;
        for (int v = 0; v < graph->numVertices; ++v) {
            if (graph->adjacencyMatrix[u][v] != INF && !graph->vertices[v].inMST &&
                graph->adjacencyMatrix[u][v] < graph->vertices[v].key) {
                graph->vertices[v].key = graph->adjacencyMatrix[u][v];
            }
        }
    }
}

```

```

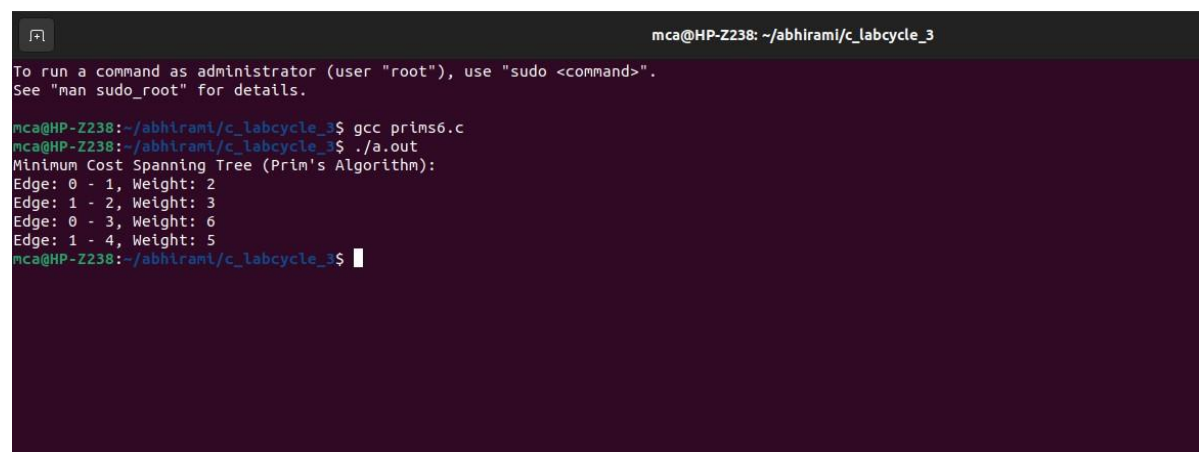
        graph->vertices[v].parent = u;
    }
}

printf("Minimum Cost Spanning Tree (Prim's Algorithm):\n");
for (int i = 1; i < graph->numVertices; ++i)
    printf("Edge: %d - %d, Weight: %d\n", graph->vertices[i].parent, i, graph
        >vertices[i].key);

int main() {
    Graph* graph = createGraph(5);
    // Define edges for demonstration
    addEdge(graph, 0, 1, 2);
    addEdge(graph, 0, 3, 6);
    addEdge(graph, 1, 2, 3);
    addEdge(graph, 1, 3, 8);
    addEdge(graph, 1, 4, 5);
    addEdge(graph, 2, 4, 7);
    addEdge(graph, 3, 4, 9);
    primMST(graph);
    return 0;
}

```

Output:



```

mca@HP-Z238: ~/abhirami/c_labcycle_3
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mca@HP-Z238:~/abhirami/c_labcycle_3$ gcc prims6.c
mca@HP-Z238:~/abhirami/c_labcycle_3$ ./a.out
Minimum Cost Spanning Tree (Prim's Algorithm):
Edge: 0 - 1, Weight: 2
Edge: 1 - 2, Weight: 3
Edge: 0 - 3, Weight: 6
Edge: 1 - 4, Weight: 5
mca@HP-Z238:~/abhirami/c_labcycle_3$

```

24. Kruskal's algorithm using the Disjoint set data structure.**Program:**

```

#include <stdio.h>

#include <stdlib.h>

#define MAX_VERTICES 100

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

typedef struct {
    int parent, rank;
} Subset;

typedef struct {
    int numVertices, numEdges;
    Edge edges[MAX_VERTICES * MAX_VERTICES];
} Graph;

Graph* createGraph(int numVertices, int numEdges) {
    if (numVertices <= 0 || numVertices > MAX_VERTICES || numEdges <= 0 ||
        numEdges > MAX_VERTICES * MAX_VERTICES) {
        printf("Invalid number of vertices or edges. Exiting...\n");
        exit(EXIT_FAILURE);
    }

    Graph* graph = (Graph*)malloc(sizeof(Graph));

    graph->numVertices = numVertices;
    graph->numEdges = numEdges;

    return graph;
}

void addEdge(Graph* graph, int index, int src, int dest, int weight) {
    if (index >= 0 && index < graph->numEdges && src >= 0 && src < graph->numVertices &&
        dest >= 0 && dest < graph->numVertices) {
        graph->edges[index].src = src;
    }
}

```



```
graph->edges[index].dest = dest;
graph->edges[index].weight = weight;
} else {
    printf("Invalid edge information. Exiting...\n");
    exit(EXIT_FAILURE);
}}

int compareEdges(const void* a, const void* b) {
    return ((Edge*)a)->weight - ((Edge*)b)->weight; }

int find(Subset subsets[], int i) {
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);
    return subsets[i].parent;}

void unionSets(Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

void kruskalMST(Graph* graph) {
    Edge result[graph->numVertices];
    int e = 0;
    int i = 0;
    qsort(graph->edges, graph->numEdges, sizeof(graph->edges[0]), compareEdges);
    Subset subsets[graph->numVertices];
    for (i = 0; i < graph->numVertices; ++i) {
```

```

    subsets[i].parent = i;

    subsets[i].rank = 0;
}

i = 0;

while (e < graph->numVertices - 1 && i < graph->numEdges) {
    Edge nextEdge = graph->edges[i++];
    int x = find(subsets, nextEdge.src);
    int y = find(subsets, nextEdge.dest);
    if (x != y) {
        result[e++] = nextEdge;
        unionSets(subsets, x, y);
    }
}

printf("Minimum Cost Spanning Tree (Kruskal's Algorithm):\n");

for (i = 0; i < e; ++i)
    printf("Edge: %d - %d, Weight: %d\n", result[i].src, result[i].dest, result[i].weight); }

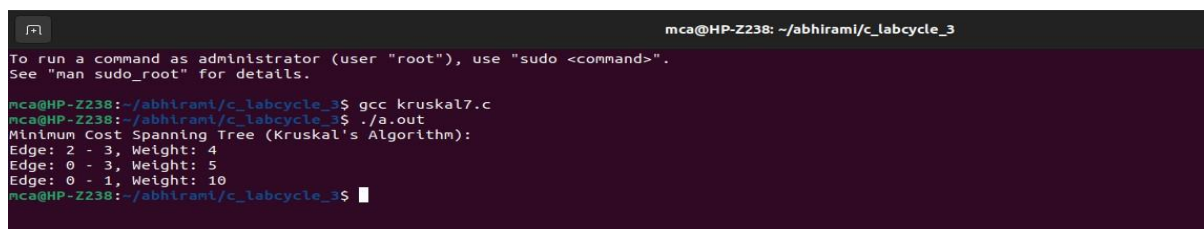
int main() {
    Graph* graph = createGraph(4, 5);
    addEdge(graph, 0, 0, 1, 10);
    addEdge(graph, 1, 0, 2, 6);
    addEdge(graph, 2, 0, 3, 5);
    addEdge(graph, 3, 1, 3, 15);
    addEdge(graph, 4, 2, 3, 4);

    kruskalMST(graph);

    return 0;
}

```

Output:



```

mca@HP-Z238: ~/abhirami/c_labcycle_3
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mca@HP-Z238:~/abhirami/c_labcycle_3$ gcc kruskal7.c
mca@HP-Z238:~/abhirami/c_labcycle_3$ ./a.out
Minimum Cost Spanning Tree (Kruskal's Algorithm):
Edge: 2 - 3, Weight: 4
Edge: 0 - 3, Weight: 5
Edge: 0 - 1, Weight: 10
mca@HP-Z238:~/abhirami/c_labcycle_3$

```

25. Single Source shortest path algorithm using any heap structure that supports mergeable heap operations.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#define MAX_VERTICES 5
#define MAX_DISTANCE 50 // Adjust this based on the expected maximum distance
between nodes
typedef struct {
    int vertex, distance;
} Node;
typedef struct {
    Node* heap;
    int capacity, size;
} MinHeap;
typedef struct {
    int numVertices;
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
} Graph;

Graph* createGraph(int numVertices) {
    if (numVertices <= 0 || numVertices > MAX_VERTICES) {
        printf("Invalid number of vertices. Exiting...\n");
        exit(EXIT_FAILURE);
    }

    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->numVertices = numVertices;
    for (int i = 0; i < numVertices; ++i) {
        for (int j = 0; j < numVertices; ++j) {
            if (i == j) {
                graph->adjacencyMatrix[i][j] = 0; // Distance from a vertex to itself is 0
            } else {
                graph->adjacencyMatrix[i][j] = rand() % MAX_DISTANCE + 1; // Random
                distance between 1 and MAX_DISTANCE
            }
        }
    }

    return graph;
}

MinHeap* createMinHeap(int capacity) {
    MinHeap* heap = (MinHeap*)malloc(sizeof(MinHeap));
    heap->capacity = capacity;
```

```

    heap->size = 0;
    heap->heap = (Node*)malloc(capacity * sizeof(Node));
    return heap;
}
void swap(Node* a, Node* b) {
    Node temp = *a;
    *a = *b;
    *b = temp;
}

void minHeapify(MinHeap* heap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    if (left < heap->size && heap->heap[left].distance < heap->heap[smallest].distance)
        smallest = left;
    if (right < heap->size && heap->heap[right].distance < heap->heap[smallest].distance)
        smallest = right;
    if (smallest != idx) {
        swap(&heap->heap[idx], &heap->heap[smallest]);
        minHeapify(heap, smallest);
    }
}

bool isEmpty(MinHeap* heap) {
    return heap->size == 0;
}

Node extractMin(MinHeap* heap) {
    if (isEmpty(heap))
        exit(EXIT_FAILURE);
    Node root = heap->heap[0];
    heap->heap[0] = heap->heap[heap->size - 1];
    heap->size--;

    minHeapify(heap, 0);
    return root;
}

void decreaseKey(MinHeap* heap, int vertex, int distance) {
    int i;
    for (i = 0; i < heap->size; ++i) {
        if (heap->heap[i].vertex == vertex) {
            heap->heap[i].distance = distance;
            break;
        }
    }
    while (i != 0 && heap->heap[i].distance < heap->heap[(i - 1) / 2].distance) {
        swap(&heap->heap[i], &heap->heap[(i - 1) / 2]);
        i = (i - 1) / 2;
    }
}

```

```

void dijkstra(Graph* graph, int src, int dest) {
    MinHeap* heap = createMinHeap(graph->numVertices);
    Node* result = (Node*)malloc(graph->numVertices * sizeof(Node));
    for (int i = 0; i < graph->numVertices; ++i) {
        heap->heap[i].vertex = i;
        heap->heap[i].distance = MAX_DISTANCE * graph->numVertices + 1; // A value
larger than the sum of all possible distances
        result[i].vertex = -1;
        result[i].distance = MAX_DISTANCE * graph->numVertices + 1; }
    heap->heap[src].distance = 0;
    result[src].distance = 0;
    clock_t start_time = clock();
    while (!isEmpty(heap)) {
        Node current = extractMin(heap);
        int u = current.vertex;

        for (int v = 0; v < graph->numVertices; ++v) {
            if (graph->adjacencyMatrix[u][v] != 0) { // Consider only non-zero distances
                int alt = result[u].distance + graph->adjacencyMatrix[u][v];
                if (alt < result[v].distance) {
                    result[v].distance = alt;
                    result[v].vertex = u;
                    decreaseKey(heap, v, alt);
                }
            }
        }
        clock_t end_time = clock();
        double execution_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
        printf("Paths from source %d to destination %d:\n", src, dest);
        printf("Shortest Distance: %d\n", result[dest].distance);
        printf("Execution Time: %f seconds\n", execution_time);
        printf("Path: ");
        int temp = dest;
        while (temp != -1) {
            printf("%d ", temp);
            temp = result[temp].vertex; }
        printf("\n");
        printf(" All Paths Traversed:\n");
        for (int i = 0; i < graph->numVertices; ++i) {
            if (i == src) continue;
            printf("To vertex %d: Distance = %d, Path = ", i, result[i].distance);
            temp = i;
            while (temp != -1) {
                printf("%d ", temp);
                temp = result[temp].vertex;
            }
            printf("\n"); }
        free(heap->heap);
    }
}

```

```

    free(heap);
    free(result);
}

int main() {
    srand(time(NULL));
    Graph* graph = createGraph(5);
    printf("Random Distance Matrix:\n");
    for (int i = 0; i < graph->numVertices; ++i) {
        for (int j = 0; j < graph->numVertices; ++j) {
            printf("%2d ", graph->adjacencyMatrix[i][j]);
        }
        printf("\n");
    }

    int sourceVertex, destVertex;
    printf("\nEnter source vertex (0-%d): ", graph->numVertices - 1);
    scanf("%d", &sourceVertex);
    printf("Enter destination vertex (0-%d): ", graph->numVertices - 1);
    scanf("%d", &destVertex);
    if (sourceVertex < 0 || sourceVertex >= graph->numVertices ||
        destVertex < 0 || destVertex >= graph->numVertices) {
        printf("Invalid source or destination vertex. Exiting...\n");
        return EXIT_FAILURE;
    }
    dijkstra(graph, sourceVertex, destVertex);

    free(graph);

    return 0;
}

```

Output:

```

mca@HP-Z238: ~/abhirami/c_labcycle_3
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mca@HP-Z238:~/abhirami/c_labcycle_3$ gcc short8.c
mca@HP-Z238:~/abhirami/c_labcycle_3$ ./a.out
Random Distance Matrix:
 0 35 20 29  8
 3  0 49 45  9
22 41  0 15  9
45 15 26  0 14
23 16 45 25  0

Enter source vertex (0-4): 3
Enter destination vertex (0-4): 4
Paths from source 3 to destination 4:
Shortest Distance: 251
Execution Time: 0.000004 seconds
Path: 4
All Paths Traversed:
To vertex 0: Distance = 251, Path = 0
To vertex 1: Distance = 251, Path = 1
To vertex 2: Distance = 251, Path = 2
To vertex 4: Distance = 251, Path = 4
mca@HP-Z238:~/abhirami/c_labcycle_3$

```