Atma Ram Sanatan Dharma College

University of Delhi

Data Structures

Practical File

Submitted By:

Jyotiswaroop Srivastav
College Roll No. 21/18023
Semester III
BSc. (Hons) Computer Science

Submitted To:

Ms. Shalini Gupta

1. Given a list of N elements, which follows no particular arrangement, you are required to search an element x in the list. The list is stored using array data structure. If the search is successful, the output should be the index at which the element occurs, otherwise returns -1 to indicate that the element is not present in the list. Assume that the elements of the list are all distinct. Write a program to perform the desired task.

```cpp
#include <iostream>
#define MAX_SIZE 100
using namespace std;

template <class T> int linearSearch(T *arr, int size, T el)
{
    for (int i = 0; i < size; i++)
    if (arr[i] == el)
        return i;
    return -1;
}

int main(void)
{
    int ch = 1, el, res, N, arr[MAX_SIZE];
    cout << "Enter Number of Elements: ";   cin >> N;
    cout << "Enter Array Elements: ";
    for (int i = 0; i < N; i++)
        cin >> arr[i];

    cout << "Enter Search Element: ";   cin >> el;
    res = linearSearch<int>(arr, N, el);

    if (res != -1)
        cout << "FOUND: Element found at index "<< res << endl;
    else
        cout << "NOT FOUND: Element not found in array"<< endl;

    return 0;
}
```

Output

```
Enter Number of Elements: 4
Enter Array Elements: 1 9 5 2
Enter Search Element: 5
FOUND: Element found at index 2
```

```
Enter Number of Elements: 4
Enter Array Elements: 1 9 5 2
Enter Search Element: 3
NOT FOUND: Element not found in array
```

2. Given a list of N elements, which is sorted in ascending order, you are required to search an element x in the list. The list is stored using array data structure. If the search is successful, the output should be the index at which the element occurs, otherwise returns -1 to indicate that the element is not present in the list. Assume that the elements of the list are all distinct. Write a program to perform the desired task.

```cpp
#include <iostream>
#define MAX_SIZE 100
using namespace std;

template <class T> int binarySearch(T *arr, int left, int right, T el)
{

    if (right >= left)
    {
        int mid = (right + left) / 2;
        if (arr[mid] == el)
            return mid;
        if (arr[mid] > el)
            return binarySearch(arr, left, mid - 1, el);
        return binarySearch(arr, mid + 1, right, el);
    }
    return -1;
}
int main(void)
{
    int ch = 1, el, res, N, arr[MAX_SIZE];
    cout << "Enter Number of Elements: ";   cin >> N;

    cout << "Enter Array Elements: ";
    for (int i = 0; i < N; i++)
        cin >> arr[i];

    cout << "Enter Search Element: ";
    cin >> el;
    res = binarySearch<int>(arr, 0, N - 1, el);
    if (res != -1)
        cout << "FOUND: Element found at index "<< res << endl;
```

```
    else
        cout << "NOT FOUND: Element not found in array"<< endl;
    return 0;
}
```

Output

```
Enter Number of Elements: 4
Enter Array Elements: 1 2 3 4
Enter Search Element: 4
FOUND: Element found at index 3

Enter Number of Elements: 4
Enter Array Elements: 1 2 3 4
Enter Search Element: 5
NOT FOUND: Element not found in array
```

3. Write a program to implement singly linked list which supports the following operations:

(i) Insert an element x at the beginning of the singly linked list

(ii) Insert an element x at position in the singly linked list

(iii)Remove an element from the beginning of the singly linked list

(iv) Remove an element from position in the singly linked list.

(v) Search for an element x in the singly linked list and return its pointer

(vi) Concatenate two singly linked lists

```
#include <iostream>
using namespace std;


void getch();
void clrscr();

template <class T> class Node {
    public:   T info;
    Node *ptr;
};

template <class T> class SinglyLinkedList
```

```
{
    protected:
    Node<T> *head, *tail;
    public:
    // Constructor
    SinglyLinkedList()
    {
        head = tail = NULL;
    }

    // Destructor
    ~SinglyLinkedList()
    {
        if (this->isEmpty())
            return;
        Node<T> *ptr, *temp = head;
        while (temp != NULL)
        {
            ptr = temp->ptr;
            delete temp;
            temp = ptr;
        }
        head = tail = NULL;
        return;
    }

    // Checks if the list is empty - O(1)   bool isEmpty()   {     return
(head == NULL || tail == NULL);
}

  // Inserts a node at the beginning - O(1)
void insertFront(T info)
{
    Node<T> *temp = new Node<T>();
    temp->info = info;
    temp->ptr = head;
    if (this->isEmpty())
        tail = temp;
    head = temp;
    cout << "Inserted " << info << " at front...";
    this->display();
    return;
}

// Inserts a node at a specified location - O(n)
void insertAtLoc(int loc, T info)
{
    if (loc == 1)
```

```
    {
        this->insertFront(info);
        return;
    }
    Node<T> *temp = head;
    for (int i = 1; temp != NULL && i < loc - 1; i++)
    temp = temp->ptr;
    if (temp == NULL)
    {
        cout << "Invalid location...\n";
        return;
    }
    if (temp == tail)
    {
        this->insertBack(info);       return;
    }
    Node<T> *node = new Node<T>();
    node->info = info;
    node->ptr = temp->ptr;
    temp->ptr = node;
    cout << "Inserted node " << info << " at location " << loc << "...";
    this->display();
    return;
}

  // Inserts a node at the end - O(1)
void insertBack(T info)
{
    Node<T> *temp = new Node<T>();
    temp->info = info;
    temp->ptr = NULL;
    if (this->isEmpty())
        head = tail = temp;
    else
        tail->ptr = temp;
    tail = temp;
    cout << "Inserted " << info << " at back...";
    this->display();
    return;
}

  // Removes a node from the beginning - O(1)
void deleteFront()
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return;
```

```
    }
    Node<T> *temp = head;
    head = temp->ptr;
    delete temp;
    if (this->isEmpty())
        tail = NULL;
    cout << "\nDeleted node at front...";
    this->display();
    return;
}

  // Removes a node at a specified location - O(n)
void deleteAtLoc(int loc)
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return;
    }
    if (loc == 1)
    {
        this->deleteFront();
        return;
    }
    Node<T> *node, *temp = head;
    for (int i = 1; temp != NULL && i < loc - 1; i++)
        temp = temp->ptr;
    if (temp == NULL || temp->ptr == NULL)
    {
        cout << "Invalid location...\n";
        return;     }
    if (temp == tail)
    {
        this->deleteBack();
        return;
    }
    node = temp->ptr->ptr;
    delete temp->ptr;
    temp->ptr = node;
    cout << "Deleted node "<< "at location " << loc << "...";
    this->display();
    return;
}

// Removes a node at the end - O(n)
void deleteBack()
{
    if (this->isEmpty())
```

```
    {
        cout << "\nList is empty...\n";
        return;
    }
    if (head == tail)
    {
        this->deleteFront();
        return;
    }
    else
    {
        Node<T> *temp = head;
        while (temp->ptr->ptr != NULL)
        temp = temp->ptr;
        delete temp->ptr;
        temp->ptr = NULL;
        tail = temp;
    }
    cout << "\nDeleted node at back...";
    this->display();
    return;
}

  // Reverses the linked list - O(n)
void reverse()
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return;
    }
    Node<T> *temp = head,*prev = NULL,
    *next = NULL;
    tail = temp;
    while (temp != NULL)
    {

        next = temp->ptr;
        temp->ptr = prev;
        prev = temp;
        temp = next;
    }

    head = prev;
    cout << "\nList reversed...";
    this->display();
    return;
  }
```

```
  // Concatenates two lists - O(n)
void concat(SinglyLinkedList<T> &list)
{
    if (!list.isEmpty() && !this->isEmpty())
    {
        Node<T> *node,
        *temp = tail,
        *temp1 = list.head;
        while (temp1 != NULL)
        {
            node = new Node<T>();
            node->info = temp1->info;
            node->ptr = NULL;
            temp->ptr = node;
            temp = temp->ptr;
            temp1 = temp1->ptr;
        }
        tail = node;
        cout << "Concatenated two lists...\n";
        this->display();
    }
    else
        cout << "\nOne of the lists is empty...\n";
        return;
}

  // Overloads the + operator - O(n)
void operator+(SinglyLinkedList<T> &list)
{
    this->concat(list);
    return;
}

  // Searches for an element - O(n)
Node<T> *search(T ele)
{
    if (this->isEmpty())
    return nullptr;
    Node<T> *temp = head;
    while (temp != NULL)
    {
        if (temp->info == ele)
        return temp;
        temp = temp->ptr;
    }
    return nullptr;
}
```

```
  // Calculates the number of nodes - O(n)
int count()
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return -1;
    }
    int count = 0; Node<T> *temp;
    for (temp = head; temp != NULL;
        temp = temp->ptr, count++);
    return count;
}

// Traverses the list and prints all nodes - O(n)
void display()
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return;
    }
    Node<T> *temp = head;
    cout << "\nList: ";
    while (temp->ptr != NULL)
    {
        cout << temp->info << " -> ";
        temp = temp->ptr;
    }
    cout << temp->info << endl;
    return;
  }
};

int main(void)
{
    int choice, ele, info, loc, count;
    SinglyLinkedList<int> list, list2;
    do
    {
        cout << "\tSingly Linked List\n"
        << "==================================\n"
        << "  (1)  Search      (2)  InsertFront\n"
        << "  (3)  InsertBack  (4)  InsertAtLoc\n"
        << "  (5)  DeleteFront (6)  DeleteBack\n"
        << "  (7)  DeleteAtLoc (8)  Display\n"
        << "  (9)  Count       (10) Reverse\n"
```

```
        << "  (11) Concat      (0)  Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;
        switch (choice)
        {
            case 1:
                cout << "\nEnter Search Element: ";
                cin >> ele;
                if (list.search(ele) != nullptr)
                    cout << "Element " << ele << " found...\n";
                else
                    cout << "Element not found or List is Empty...\n";
                break;
            case 2:
                cout << "\nEnter Element: ";
                cin >> info;
                list.insertFront(info);
                break;
            case 3:
                cout << "\nEnter Element: ";
                cin >> info;
                list.insertBack(info);
                break;
            case 4:
                cout << "\nEnter Location: ";
                cin >> loc;
                cout << "Enter Element: ";
                cin >> info;
                list.insertAtLoc(loc, info);
                break;
            case 5:
                list.deleteFront();
                break;
            case 6:
                list.deleteBack();
                break;
            case 7:
                cout << "\nEnter Location: ";
                cin >> loc;
                list.deleteAtLoc(loc);
                break;
            case 8:
                list.display();
                break;
            case 9:
                count = list.count();
                if (count != -1)
                cout << "\nNumber of Nodes: " << count << endl;
```

```
                break;
            case 11:
                if (!list2.isEmpty())
                {
                    cout << "\nList B:";
                    list2.display();
                }
                cout << "\nNumber of Nodes to add in List B: ";
                cin >> count;
                if (count)
                {
                    cout << "Enter Elements to List B: ";
                    for (int i = 0; i < count; i++)
                    {
                        cin >> info;
                        list2.insertBack(info);
                    }
                    list + list2;
                }
                break;
            case 10:
                list.reverse();
                break;
            case 0:
                default:
            break;
        }
        getch();
        clrscr();
    }
    while (choice != 0);
        return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}

void clrscr()
{
    #ifdef _WIN32   system("cls");
    #elif __unix__    system("clear");
    #endif   return;
}
```

Output

```
        Singly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 2

Enter Element: 10
Inserted 10 at front...
List: 10

Press any key to continue...
```

```
        Singly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 4

Enter Location: 2
Enter Element: 20
Inserted 20 at back...
List: 10 -> 20

Press any key to continue...
```

```
        Singly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 7

Enter Location: 2
Deleted node at location 2...
List: 15

Press any key to continue...
```

```
        Singly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 5

Deleted node at front...
List: 15 -> 20

Press any key to continue...
```

```
        Singly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 4

Enter Location: 2
Enter Element: 15
Inserted node 15 at location 2...
List: 10 -> 15 -> 20

Press any key to continue...
```

```
        Singly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 1

Enter Search Element: 15
Element 15 found...

Press any key to continue...
```

```
        Singly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 1

Enter Search Element: 10
Element not found or List is Empty...

Press any key to continue...
```

```
        Singly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 11

Number of Nodes to add in List B: 3
Enter Elements to List B: 1 2 3
Inserted 1 at back...
List: 1
Inserted 2 at back...
List: 1 -> 2
Inserted 3 at back...
List: 1 -> 2 -> 3
Concatenated two lists...

List: 15 -> 1 -> 2 -> 3

Press any key to continue...
```

4. Write a program to implement doubly linked list which supports the following operations:

(i) Insert an element x at the beginning of the doubly linked list

(ii) Insert an element x at position in the doubly linked list

(iii)Insert an element x at the end of the doubly linked list

(iv) Remove an element from the beginning of the doubly linked list

(v) Remove an element from position in the doubly linked list.

(vi) Remove an element from the end of the doubly linked list

(vii) Search for an element x in the doubly linked list and return its pointer

(viii) Concatenate two doubly linked lists

```
#include <iostream>
using namespace std;


void getch();
void clrscr();

template <class T> class Node {
    public:
    T info;
    Node *prev;
    Node *next;
};
template <class T> class DoublyLinkedList
{
    protected:
    Node<T> *head, *tail;
    public:
       // Constructor
    DoublyLinkedList()
    {
        head = tail = NULL;
    }

    // Destructor
    ~DoublyLinkedList()
    {
        if (this->isEmpty())
            return;
        Node<T> *ptr;
        for (; !isEmpty();)
        {
            ptr = head->next;
            delete head;
            head = ptr;
        }
        head = tail = ptr;
        return;
    }

    // Checks if the list is empty - O(1)
    bool isEmpty()
    {
```

```
        return (head == NULL || tail == NULL);
    }

  // Inserts a node at the beginning - O(1)
    void insertFront(T info)
    {
        Node<T> *temp = new Node<T>();
        temp->info = info;
        temp->next = head;
        temp->prev = NULL;
        if (this->isEmpty())
            tail = temp;
        else
            head->prev = temp;
        head = temp;
        cout << "Inserted " << info << " at front...";     this-
>display();
        return;
    }

  // Inserts a node at a specified location - O(n)
    void insertAtLoc(int loc, T info)
    {
        if (loc == 1)
        {
            this->insertFront(info);
            return;
        }
        Node<T> *temp = head;
        for (int i = 1; temp != NULL && i < loc - 1; i++)
            temp = temp->next;
        if (temp == NULL)
        {
            cout << "Invalid location...\n";
            return;
        }
        if (temp == tail)
        {
            this->insertBack(info);
            return;
        }
        Node<T> *node = new Node<T>();
        node->info = info;
        node->next = temp->next;
        node->prev = temp;
        temp->next->prev = node;
        temp->next = node;
```

```
        cout << "Inserted node " << info << " at location " << loc <<
"...";
        this->display();
        return;
    }

    // Inserts a node at the end - O(1)
    void insertBack(T info)
    {
        Node<T> *temp = new Node<T>();
        temp->info = info;
        temp->next = NULL;
        temp->prev = tail;
        if (this->isEmpty())
            head = tail = temp;
        else
            tail->next = temp;
        tail = temp;
        cout << "Inserted " << info << " at back...";
        this->display();
        return;
    }

    // Removes a node from the beginning - O(1)
    void deleteFront()
    {
        if (this->isEmpty())
        {
            cout << "\nList is empty...\n";
            return;
        }
        Node<T> *temp = head;
        head = temp->next;
        if (this->isEmpty())
            tail = NULL;
        else
            head->prev = NULL;
        delete temp;

        cout << "\nDeleted node at front...";
        this->display();
        return;
    }

    // Removes a node at a specified location - O(n)
    void deleteAtLoc(int loc)
    {
        if (this->isEmpty())
```

```
        {
            cout << "\nList is empty...\n";
            return;
        }     i
        f (loc == 1)
        {
            this->deleteFront();
            return;
        }
        Node<T> *node, *temp = head;
        for (int i = 1; temp != NULL && i < loc - 1; i++)
            temp = temp->next;
        if (temp == NULL || temp->next == NULL)
        {
            cout << "Invalid location...\n";
            return;
        }
        if (temp->next == tail)
        {
            this->deleteBack();
            return;
        }
        node = temp->next->next;
        node->prev = temp;
        delete temp->next;
        temp->next = node;
        cout << "Deleted node "<< "at location " << loc << "...";     this-
>display();     return;
    }

    // Removes a node at the end - O(1)
    void deleteBack()
    {
        if (this->isEmpty())
        {
            cout << "\nList is empty...\n";
            return;
        }
        Node<T> *temp = tail;
        tail = temp->prev;
        if (this->isEmpty())
            head = NULL;
        else
            tail->next = NULL;
        delete temp;
        cout << "\nDeleted node at back...";
        this->display();
        return;
```

```
    }

    // Reverses the linked list - O(n)
    void reverse()   {
        if (this->isEmpty())
        {
            cout << "\nList is empty...\n";
            return;
        }
        Node<T> *temp = head,
        *temp1 = NULL;
        tail = temp;
        while (temp != NULL)
        {
            temp1 = temp->prev;

            temp->prev = temp->next;
            temp->next = temp1;
            temp = temp->prev;
        }
        if (temp1 != NULL)
            head = temp1->prev;
            cout << "\nList reversed...";
            this->display();
            return;
    }

    // Concatenates two lists - O(n)
    void concat(DoublyLinkedList<T> &list)
    {
        if (!list.isEmpty() && !this->isEmpty())
        {
            Node<T> *node,
            *temp = tail,
            *temp1 = list.head;
            while (temp1 != NULL)
        {
            node = new Node<T>();
            node->info = temp1->info;
            node->next = NULL;
            node->prev = temp;
            temp->next = node;
            temp = temp->next;
            temp1 = temp1->next;
        }
        tail = node;
        cout << "Concatenated two lists...\n";
        this->display();
```

```
        }
        else
            cout << "\nOne of the lists is empty...\n";
        return;
    }

    // Overloads the + operator - O(n)
    void operator+(DoublyLinkedList<T> &list)
    {
        this->concat(list);     r
        eturn;
    }

    // Searches for an element - O(n)
    Node<T> *search(T ele)
    {
        if (this->isEmpty())
        return nullptr;
        Node<T> *temp = head;
        while (temp != NULL)
        {
            if (temp->info == ele)
            return temp;
            temp = temp->next;
        }
        return nullptr;
    }

    // Calculates the number of nodes - O(n)
    int count()   {
        if (this->isEmpty())
        {
            cout << "\nList is empty...\n";
            return -1;
        }
        int count = 0;
        Node<T> *temp;
        for (temp = head; temp != NULL;
            temp = temp->next, count++);

        return count;
    }

    // Traverses the list and prints all nodes - O(n)
    void display()   {
        if (this->isEmpty())
        {
            cout << "\nList is empty...\n";
```

```
            return;
        }
        Node<T> *temp = head;
        cout << "\nList: ";
        while (temp->next != NULL)
        {
            cout << temp->info << " -> ";
            temp = temp->next;
        }
        cout << temp->info << endl;
        return;
    }
};

int main(void) {
    int info, ele, choice, loc, count;   DoublyLinkedList<int> list, list2;
    do
    {
        cout << "\tDoubly Linked List\n"
        << "==================================\n"
        << "  (1)  Search      (2)  InsertFront\n"
        << "  (3)  InsertBack  (4)  InsertAtLoc\n"
        << "  (5)  DeleteFront (6)  DeleteBack\n"
        << "  (7)  DeleteAtLoc (8)  Display\n"
        << "  (9)  Count       (10) Reverse\n"
        << "  (11) Concat      (0)  Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;
        switch (choice)
        {
            case 1:
                cout << "\nEnter Search Element: ";
                cin >> ele;
                if (list.search(ele) != nullptr)
                    cout << "Element " << ele << " found...\n";
                else
                    cout << "Element not found or List is Empty...\n";
                break;
            case 2:
                cout << "\nEnter Element: ";
                cin >> info;
                list.insertFront(info);
                break;
            case 3:
                cout << "\nEnter Element: ";
                cin >> info;
                list.insertBack(info);
                break;
```

```
            case 4:
                cout << "\nEnter Location: ";
                cin >> loc;
                cout << "Enter Element: ";
                cin >> info;
                list.insertAtLoc(loc, info);
                break;
            case 5:
                list.deleteFront();
                break;
            case 6:
                list.deleteBack();
                break;
            case 7:
                cout << "\nEnter Location: ";
                cin >> loc;
                list.deleteAtLoc(loc);
                break;
            case 8:
                list.display();
                break;
            case 9:
                count = list.count();
                if (count != -1)
                    cout << "\nNumber of Nodes: " << count << endl;
                break;
            case 10:
                list.reverse();
                break;
            case 11:
                if (!list2.isEmpty())
                {
                    cout << "\nList B:";
                    list2.display();
                }
                cout << "\nNumber of Nodes to add in List B: ";
                cin >> count;
                if (count)
                {
                    cout << "Enter Elements to List B: ";
                    for (int i = 0; i < count; i++)
                    {
                        cin >> info;
                        list2.insertBack(info);
                    }
                    list + list2;
                }
                break;
```

```
            case 0:
            default:
            break;
        }
        getch();
        clrscr();

        while (choice != 0);
            return 0;
    }
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
    #ifdef _WIN32   system("cls");
    #elif __unix__    system("clear");
    #endif   return;
}
```

Output

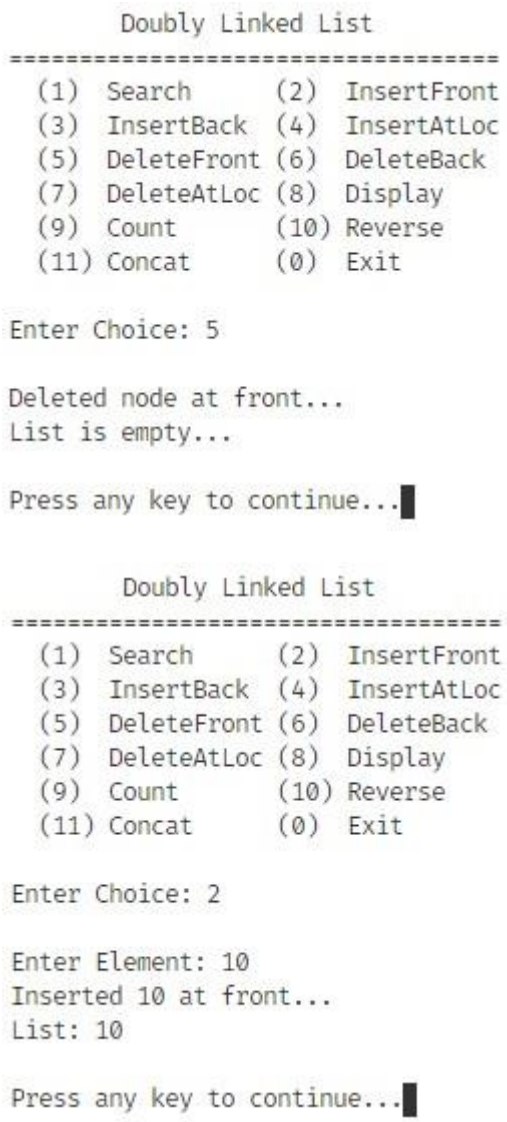
```
        Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 2

Enter Element: 10
Inserted 10 at front...
List: 10

Press any key to continue...
```

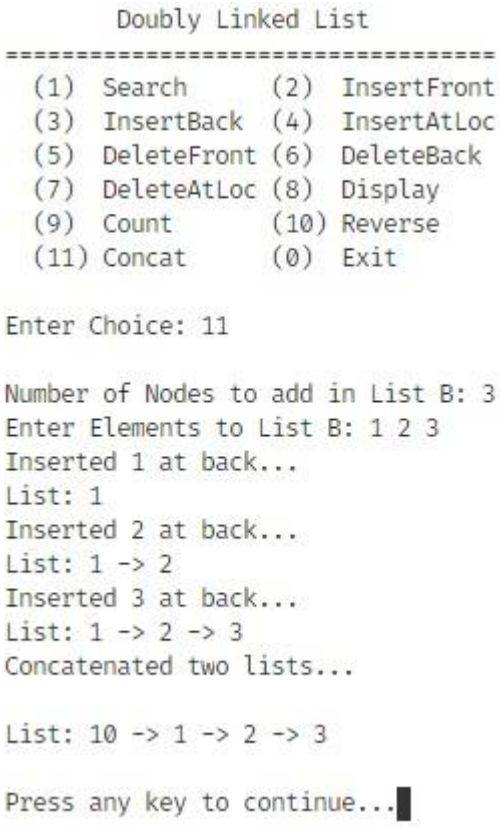
```
        Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 3

Enter Element: 30
Inserted 30 at back...
List: 10 -> 30

Press any key to continue...
```

```
        Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 4

Enter Location: 2
Enter Element: 20
Inserted node 20 at location 2...
List: 10 -> 20 -> 30

Press any key to continue...
```

```
        Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 7

Enter Location: 2
Deleted node at location 2...
List: 10 -> 30

Press any key to continue...
```

```
        Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 6

Deleted node at back...
List: 10

Press any key to continue...
```





5. Write a program to implement circularly linked list which supports the following operations:

(i) Insert an element x at the front of the circularly linked list

(ii) Insert an element x after an element y in the circularly linked list

(iii)Insert an element x at the back of the circularly linked list

(iv) Remove an element from the back of the circularly linked list

(v) Remove an element from the front of the circularly linked list

(vi) remove the element x from the circularly linked list

(vii)Search for an element x in the circularly linked list and return its pointer

(viii) Concatenate two circularly linked lists

```cpp
#include <iostream>
using namespace std;

void getch();
void clrscr();

template <class T> class Node {
    public:
    T info;
    Node *prev;
    Node *next;
};

template <class T>
class CircularDoublyLinkedList
{
    protected:
    Node<T> *tail;
    public:
    // Constructor
    CircularDoublyLinkedList()
    {
        tail = NULL;
    }

    // Destructor
    ~CircularDoublyLinkedList()
    {
        if (this->isEmpty())
            return;
        Node<T> *ptr, *temp = tail->next;
        while (temp != tail)
        {
            ptr = temp;
            temp = ptr->next;
            delete ptr;
        }
        delete temp;
        tail = NULL;
        return;
    }

    // Checks if the list is empty - O(1)
    bool isEmpty()   {
        return tail == NULL;
    }

    // Inserts a node at the beginning - O(1)
```

```
    void insertFront(T info)
    {
        Node<T> *temp = new Node<T>();
        temp->info = info;
        if (this->isEmpty())
        {
            temp->next = temp;
            temp->prev = temp;
            tail = temp;
        }

        else
        {
            temp->prev = tail;
            temp->next = tail->next;
            tail->next->prev = temp;
            tail->next = temp;
        }
        cout << "Inserted " << info << " at front...";     this-
>display();
        return;
    }

    // Inserts a node at a specified location - O(n)
    void insertAtLoc(T searchEle, T info)
    {
        int loc = 0;
        if (this->isEmpty())
        {
            cout << "List Empty...\n";
            return;
        }
        int i = 0;
        Node<T> *temp = tail->next;
        do
        {
            ++i;
            if (temp->info == searchEle)
                loc = i;
                temp = temp->next;
        }
        while (temp != tail->next);

        if (loc == 0)     {
            cout << "Search Element Not Found...\n";       return;
        }
        loc++;
        if (loc == 1)
```

```
        {
            this->insertFront(info);
            return;
        }
        int size = this->count();
        if (loc > size + 1 || loc < 1)
        {
            cout << "Invalid location...\n";
            return;
        }
        if (loc == size + 1)
        {
            this->insertBack(info);
            return;
        }
        temp = tail->next;
        for (int i = 1; temp->next != tail && i < loc - 1;i++)
            temp = temp->next;
        Node<T> *node = new Node<T>();
        node->info = info;
        node->next = temp->next;
        temp->next->prev = node;
        node->prev = temp;
        temp->next = node; cout << "Inserted node " << info << " at location "
<< loc << "...";
        this->display();
        return;
    }

    // Inserts a node at the end - O(1)
    void insertBack(T info)
    {
        Node<T> *temp = new Node<T>();
        temp->info = info;
        if (this->isEmpty())
        {
            temp->next = temp;
            temp->prev = temp;
        }
        else
        {
            temp->next = tail->next;
            temp->prev = tail;
            tail->next = temp;
            temp->next->prev = temp;
        }
        tail = temp;
        cout << "Inserted " << info << " at back...";
```

```
        this->display();
        return;
    }

    // Removes a node from the beginning - O(1)   void deleteFront()
    {
        if (this->isEmpty())
        {
        cout << "\nList is empty...\n";       return;     }     if (tail->next
== tail)
        {       delete tail;       tail = NULL;
        }     else
        {
        Node<T> *temp = tail->next;       tail->next = temp->next;       temp-
>next->prev = tail;       delete temp;
    } cout << "\nDeleted node at front..."; this->display();
        return;
    }

    // Removes a node at a specified location - O(n)   void deleteAtLoc(T ele)
    {     int loc = 0;
        if (this->isEmpty())
        {       cout << "List Empty...\n";       return;
        }      int i = 0;
        Node<T> *temp = tail->next;     do     {       ++i;       if (temp-
>info == ele)         loc = i;       temp = temp->next;
        } while (temp != tail->next);

        if (loc == 0)     {       cout << "Search Element Not
Found...\n";       return;
        }      int size = this->count();     if (loc > size || loc < 1)
        {       cout << "Invalid location...\n";       return;     }
        if (loc == size)
        {
        this->deleteBack();       return;     }     temp = tail->next;     for
(int i = 1; temp->next != tail && i < loc; i++)       temp = temp->next; temp-
>prev->next = temp->next;
    temp->next->prev = temp->prev; delete temp; cout << "Deleted node
"          << "at location " << loc << "...";     this->display();     return;
    }

    // Removes a node at the end - O(1)   void deleteBack()   {
        if (this->isEmpty())
        {       cout << "\nList is empty...\n";       return;     }
        if (tail->next == tail)
        {       delete tail;       tail = NULL;     }     else
        {
```

```
        Node<T> *temp = tail;       tail = temp->prev;       temp->next->prev
= tail;       tail->next = temp->next;       delete temp;     }     cout <<
"\nDeleted node at back...";     this->display();     return;
    }

    // Reverses the linked list - O(n)   void reverse()   {
        if (this->isEmpty())
        {       cout << "\nList is empty...\n";       return;
        }
        Node<T> *temp = tail->next,
                *headRef = tail->next,             *temp1 =
NULL;     do     {       temp1 = temp->prev;       temp->prev = temp-
>next;       temp->next = temp1;       temp = temp->prev;     } while (temp !=
headRef);     tail = headRef;     cout << "\nList reversed...";     this-
>display();     return;
    }

    // Concatenates two lists - O(n)   void concat(CircularDoublyLinkedList<T>
&list)
    {     if (!list.isEmpty() && !this->isEmpty())
        {
        tail->next->prev = list.tail;       Node<T> *temp = tail-
>next;       tail->next = list.tail->next;       list.tail->next =
temp;       tail = list.tail;       cout << "Concatenated two
lists...\n";       this->display();
        }     else       cout << "\nOne of the lists is
empty...\n";     return;
    }

    // Overloads the + operator - O(n)   void
operator+(CircularDoublyLinkedList<T> &list)
    {     this->concat(list);     return;
    }

    // Searches for an element - O(n)
    Node<T> *search(T ele)
    {     if (this->isEmpty())       return nullptr;     Node<T> *temp = tail-
>next;     do     {       if (temp->info == ele)         return
temp;       temp = temp->next;
        } while (temp != tail->next); return nullptr;
    }

    // Calculates the number of nodes - O(n)   int count()   {
        if (this->isEmpty())
        {       cout << "\nList is empty...\n";       return -1;     }     int
count = 0;     Node<T> *temp = tail->next;     do     {       temp = temp-
>next;       count++;
        } while (temp != tail->next);     return count;
```

```
    }

    // Traverses the list and prints all nodes - O(n)   void display()   {
        if (this->isEmpty())
        {       cout << "\nList is empty...\n";       return;
        }
        Node<T> *temp = tail->next;     cout << "\nList: ";     while (temp !=
tail)
        {
        cout << temp->info << " -> ";       temp = temp->next;
        }     cout << temp->info << endl;     return;
    }
};  int main(void) {
  int info, ele, choice, loc, count;   CircularDoublyLinkedList<int> list,
list2;   do   {     cout << "\tCircular Doubly Linked List\n"          <<
"==================================\n"          <<
"  (1)  Search      (2)  InsertFront\n"
         << "  (3)  InsertBack  (4)  InsertAtLoc\n"
         << "  (5)  DeleteFront (6)  DeleteBack\n"
         << "  (7)  DeleteAtLoc (8)  Display\n"
         << "  (9)  Count       (10) Reverse\n"          << "  (11)
Concat      (0)  Exit\n\n";     cout << "Enter Choice: ";     cin >>
choice;     switch (choice)
    {     case 1:
      cout << "\nEnter Search Element: ";       cin >> ele;       if
(list.search(ele) != nullptr)         cout << "Element " << ele << "
found...\n";       else         cout << "Element not found or List is
Empty...\n";       break;     case 2:
      cout << "\nEnter Element: ";       cin >>
info;       list.insertFront(info);       break;     case 3:
      cout << "\nEnter Element: ";       cin >>
info;       list.insertBack(info);       break;     case 4:
      cout << "\nInsert After: ";       cin >> ele;       cout << "Enter
Element: ";       cin >> info;       list.insertAtLoc(ele,
info);       break;     case
5:       list.deleteFront();       break;     case
6:       list.deleteBack();       break;     case 7:
      cout << "\nEnter Element: ";       cin >>
ele;       list.deleteAtLoc(ele);       break;     case 8:
      list.display();
      break;     case 9:
      count = list.count();       if (count != -1)         cout << "\nNumber
of Nodes: " << count << endl;       break;     case
10:       list.reverse();       break;     case 11:       if
(!list2.isEmpty())
      {         cout << "\nList B:";         list2.display();
```

```
      }       cout << "\nNumber of Nodes to add in List B: ";       cin >>
count;       if (count)       {         cout << "Enter Elements to List B:
";         for (int i = 0; i < count; i++)
        {           cin >> info;           list2.insertBack(info);
        }         list + list2;       }       break;     case
0:     default:       break;     }     getch();     clrscr();   } while
(choice != 0);   return 0;
}  void getch() {
  cout << "\nPress any key to
continue...";   cin.ignore();   cin.get();   return;
}  void clrscr()
{
#ifdef _WIN32   system("cls"); #elif __unix__   system("clear");
#endif   return; }
```

Output

```
       Circular Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 2

Enter Element: 10
Inserted 10 at front...
List: 10

Press any key to continue...
```

```
       Circular Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 3

Enter Element: 30
Inserted 30 at back...
List: 10 -> 30

Press any key to continue...
```

```
       Circular Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 4

Insert After: 10
Enter Element: 20
Inserted node 20 at location 2...
List: 10 -> 20 -> 30

Press any key to continue...
```

```
       Circular Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 7

Enter Element: 20
Deleted node at location 2...
List: 10 -> 30

Press any key to continue...
```

```
       Circular Doubly Linked List
==================================
  (1)  Search      (2)  InsertFront
  (3)  InsertBack  (4)  InsertAtLoc
  (5)  DeleteFront (6)  DeleteBack
  (7)  DeleteAtLoc (8)  Display
  (9)  Count       (10) Reverse
  (11) Concat      (0)  Exit

Enter Choice: 6

Deleted node at back...
List: 10

Press any key to continue...
```

```
        Circular Doubly Linked List
 ==================================
   (1)  Search      (2)  InsertFront
   (3)  InsertBack  (4)  InsertAtLoc
   (5)  DeleteFront (6)  DeleteBack
   (7)  DeleteAtLoc (8)  Display
   (9)  Count       (10) Reverse
   (11) Concat      (0)  Exit

 Enter Choice: 5

 Deleted node at front...
 List is empty...

 Press any key to continue...
```

```
        Circular Doubly Linked List
 ==================================
   (1)  Search      (2)  InsertFront
   (3)  InsertBack  (4)  InsertAtLoc
   (5)  DeleteFront (6)  DeleteBack
   (7)  DeleteAtLoc (8)  Display
   (9)  Count       (10) Reverse
   (11) Concat      (0)  Exit

 Enter Choice: 2

 Enter Element: 10
 Inserted 10 at front...
 List: 10

 Press any key to continue...
```

```
        Circular Doubly Linked List
 ==================================
   (1)  Search      (2)  InsertFront
   (3)  InsertBack  (4)  InsertAtLoc
   (5)  DeleteFront (6)  DeleteBack
   (7)  DeleteAtLoc (8)  Display
   (9)  Count       (10) Reverse
   (11) Concat      (0)  Exit

 Enter Choice: 1

 Enter Search Element: 10
 Element 10 found...

 Press any key to continue...
```

```
        Circular Doubly Linked List
 ==================================
   (1)  Search      (2)  InsertFront
   (3)  InsertBack  (4)  InsertAtLoc
   (5)  DeleteFront (6)  DeleteBack
   (7)  DeleteAtLoc (8)  Display
   (9)  Count       (10) Reverse
   (11) Concat      (0)  Exit

 Enter Choice: 11

 Number of Nodes to add in List B: 3
 Enter Elements to List B: 1 2 3
 Inserted 1 at back...
 List: 1
 Inserted 2 at back...
 List: 1 -> 2
 Inserted 3 at back...
 List: 1 -> 2 -> 3
 Concatenated two lists...

 List: 10 -> 1 -> 2 -> 3

 Press any key to continue...
```

6. Implement a stack using Array representation

```cpp
#include <iostream>
using namespace std;

#define n 100

class Stack{
    int *arr;
    int top;

    public:
    Stack(){
        arr = new int[n];
        top = -1;
    }

    void Push(int data){
        if(top==n-1)
            cout<<"Stack Overflow"<<endl;

        top++;
        arr[top]=data;
    }
    void Pop(){
        if(top==-1)
            cout<<"Stack Underflow"<<endl;
        top--;
    }
    void Top(){
        cout<<arr[top]<<endl;
    }
    bool isEmpty(){
        return top==-1;
    }
    int isFull(){
        return top==n-1;
    }

};

int main(){

    Stack s;
    s.Push(1);
    s.Push(2);
    s.Push(3);
    s.Push(4);
    s.Push(5);
```

```
    s.Top();

    s.Pop();

    s.Top();

    cout<<s.isFull()<<endl;

    s.Pop();
    s.Pop();
    s.Pop();
    s.Pop();

    cout<<s.isEmpty()<<endl;
}
```

Output:

```
5
4
0
1
```

7. Implement a stack using Linked representation

```
#include <iostream>
using namespace std;

class Node{
    public:
    int val;
    Node * next;

    Node(){
        val = 0 ;
        next = NULL;
    }

    Node(int data){
        val = data;
        next = NULL;
    }
};

class LL{
    public:
    Node * head;
```

```
    LL(){
        head = NULL;
    }

    void Push(int);

    void Display();

    void Pop();
};

void LL::Push(int data){
    Node *temp = new Node(data);
    if(head == NULL){
        head = temp;
        return;
    }
    Node *cur = head;
    while(cur->next!=NULL){
        cur = cur->next;
    }
    cur->next = temp;
}

void LL::Pop(){
    if(head == NULL){
        cout<<"Stack Underflow\n";
        return;
    }
    Node * cur = head;
    Node *temp = cur->next;
    while(temp->next!=NULL){
        temp = temp->next;
    }
    cur->next = NULL;
    free(temp);

}

void LL::Display()
{
    Node* temp = head;
    if (head == NULL) {
        cout << "List empty" << endl;
        return;
    }
```

```
    while (temp != NULL) {
        cout << temp->val << " -> ";
        temp = temp->next;
    }cout<<"NULL\n";
}

int main(){
    LL l;
    l.Push(11);
    l.Push(22);
    l.Push(33);
    l.Push(44);
    l.Push(55);

    l.Display();

    l.Pop();

    l.Display();
}
```

Output:

```
11 -> 22 -> 33 -> 44 -> 55 -> NULL
11 -> NULL
```

8. Implement Queue using Circular Array representation

```
#include <iostream>
using namespace std;
#define SIZE 5
int A[SIZE];
int front = -1;
int rear = -1;

// Function to check if queue is empty or not
bool isempty()
{
    if (front == -1 && rear == -1)
        return true;
    else
        return false;
}

// function to enter elements in queue
void enqueue(int value)
{
    // queue is full
```

```
    if ((rear + 1) % SIZE == front)
        cout << "Queue is full \n";
    else
    {
        // first element inserted
        if (front == -1)
            front = 0;
        // insert element at rear
        rear = (rear + 1) % SIZE;
        A[rear] = value;
    }
}

// function to delete/remove element from queue
void dequeue()
{
    if (isempty())
        cout << "Queue is empty\n";
    else
        // only one element
        if (front == rear)
            front = rear = -1;
        else
            front = (front + 1) % SIZE;
}

// function to show the element at front
void showfront()
{
    if (isempty())
        cout << "Queue is empty\n";
    else
        cout << "element at front is:" << A[front];
}

// function to display queue
void displayQueue()
{
    if (isempty())
        cout << "Queue is empty\n";
    else
    {
        int i;
        if (front <= rear)
        {
            for (i = front; i <= rear; i++)
                cout << A[i] << " ";
        }
```

```
        else
        {
            i = front;
            while (i < SIZE)
            {
                cout << A[i] << " ";
                i++;
            }
            i = 0;
            while (i <= rear)
            {
                cout << A[i] << " ";
                i++;
            }
        }
    }
}

// Main Function
int main()
{
    int choice, flag = 1, value;
    while (flag == 1)
    {
        cout << "\n1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit\n";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "Enter Value:\n";
            cin >> value;
            enqueue(value);
            break;
        case 2:
            dequeue();
            break;
        case 3:
            showfront();
            break;
        case 4:
            displayQueue();
            break;
        case 5:
            flag = 0;
            break;
        }
    }
```

```
    return 0;
}
```

Output :

```
1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit
1
Enter Value:
1

1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit
1
Enter Value:
4

1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit
1
Enter Value:
6

1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit
2

1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit
3
element at front is:4
1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit
4
4 6
1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit
5
```

9. Implement Queue using Circular linked list representation

```
// circularSinglyLinkedList.hpp
#include <iostream>

using namespace std;
template <class T>
class Node
{
public:
    T info;
    Node *ptr;
};
template <class T>
class CircularSinglyLinkedList
{
public:
    Node<T> *tail;
```

```
    // Constructor
    CircularSinglyLinkedList()
    {
        tail = NULL;
    }

    // Destructor
    ~CircularSinglyLinkedList()
    {
        if (this->isEmpty())
            return;
        Node<T> *ptr, *temp = tail->ptr;
        while (temp != tail)
        {
            ptr = temp;
            temp = ptr->ptr;
            delete ptr;
        }
        delete temp;
        tail = NULL;
        return;
    }

    // Checks if the list is empty - O(1)   bool isEmpty()   {     return tail
== NULL;
}

// Inserts a node at the beginning - O(1)   void insertFront(T info)
{
    Node<T> *temp = new Node<T>();
    temp->info = info;
    if (this->isEmpty())
    {
        temp->ptr = temp;
        tail = temp;
    }
    else
    {
        temp->ptr = tail->ptr;
        tail->ptr = temp;
    }
    return;
}

// Inserts a node at a specified location - O(n)   void insertAtLoc(int loc, T
info)
{
    if (loc == 1)
```

```
    {
        this->insertFront(info);
        return;
    }
    int size = this->count();
    if (loc > size + 1 || loc < 1)
    {
        cout << "Invalid location...\n";
        return;
    }
    if (loc == size + 1)
    {
        this->insertBack(info);
        return;
    }
    Node<T> *temp = tail->ptr;
    for (int i = 1; temp->ptr != tail && i < loc - 1; i++)
        temp = temp->ptr;
    Node<T> *node = new Node<T>();
    node->info = info;
    node->ptr = temp->ptr;
    temp->ptr = node;
    return;
}

// Inserts a node at the end - O(1)   void insertBack(T info)
{
    Node<T> *temp = new Node<T>();
    temp->info = info;
    if (this->isEmpty())
        temp->ptr = temp;
    else
    {
        temp->ptr = tail->ptr;
        tail->ptr = temp;
    }
    tail = temp;
    return;
}

// Removes a node from the beginning - O(1)   void deleteFront()
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return;
    }
    else if (tail->ptr == tail)
```

```
    {
        delete tail;
        tail = NULL;
    }
    else
    {
        Node<T> *temp;
        temp = tail->ptr->ptr;
        delete tail->ptr;
        tail->ptr = temp;
    }
    return;
}

// Removes a node at a specified location - O(n)   void deleteAtLoc(int loc)
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return;
    }
    int size = this->count();
    if (loc > size || loc < 1)
    {
        cout << "Invalid location...\n";
        return;
    }
    if (loc == size)
    {
        this->deleteBack();
        return;
    }
    Node<T> *node, *temp = tail->ptr;
    for (int i = 1; temp->ptr != tail && i < loc - 1; i++)
        temp = temp->ptr;
    node = temp->ptr->ptr;
    delete temp->ptr;
    temp->ptr = node;
    return;
}

// Removes a node at the end - O(n)   void deleteBack()
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return;
    }
```

```
    else if (tail->ptr == tail)
    {
        delete tail;
        tail = NULL;
    }
    else
    {
        Node<T> *temp = tail->ptr;
        while (temp->ptr != tail)
            temp = temp->ptr;
        temp->ptr = tail->ptr;
        delete tail;
        tail = temp;
    }
    return;
}

// Traverses the list and prints all nodes - O(n)   void display()   {
if (this->isEmpty())
{
    cout << "\nList is empty...\n";
    return;
}
Node<T> *temp = tail->ptr;
while (temp != tail)
{
    cout << temp->info << " -> ";
    temp = temp->ptr;
}
cout << temp->info << endl;
return;
}
}
;

// main.cpp
#include "circularSinglyLinkedList.hpp"

using namespace std;
void getch();
void clrscr();
template <class T>
class Queue
{
protected:
    Node<T> *front, *rear;
    CircularSinglyLinkedList<T> list;
```

```
public:
    Queue()
    {
        this->front = this->list.tail;
        this->rear = this->list.tail;
    }
    bool enqueue(T ele)
    {
        this->list.insertBack(ele);
        this->front = this->list.tail->ptr;
        this->rear = this->list.tail;
        return true;
    }

    T dequeue()
    {
        if (this->isEmpty())
        {
            cout << "ERROR: Queue Empty\n";
            return (T)(NULL);
        }
        T temp = this->front->info;
        this->list.deleteFront();
        if (this->isEmpty())
            this->front = this->list.tail;
        else
            this->front = this->list.tail->ptr;
        this->rear = this->list.tail;
        return temp;
    }

    T frontEl()
    {
        if (this->isEmpty())
        {
            cout << "Queue Empty";
            return (T)(NULL);
        }
        return this->front->info;
    }
    bool isEmpty()
    {
        return this->list.isEmpty();
    }
    void clear()
    {
        while (!this->isEmpty())
            this->dequeue();
```

```
    }
    void display()
    {
        if (this->isEmpty())
        {
            cout << "Queue Empty";
            return;
        }
        this->list.display();
        return;
    }
};
int main(void)
{
    int el, res, choice;
    Queue<int> q;
    do
    {
        cout << "\tCircular Queue - CSLList\n"
             << "==================================\n"
             << "  (1) Enqueue  (2) Dequeue\n"
             << "  (3) Front    (4) Clear\n"
             << "  (5) Display  (0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "\nEnter Element: ";
            cin >> el;
            res = q.enqueue(el);
            if (res)
            {
                cout << "\nEnqueued " << el << "...\n";
                cout << "Queue: ";
                q.display();
            }
            break;
        case 2:
            res = q.dequeue();
            if (res)
            {
                cout << "\nDequeued " << res << "...\n";
                cout << "Queue: ";
                q.display();
            }
            break;
        case 3:
```

```
            cout << "\nFront Element: " << q.frontEl() << endl;
            break;
        case 4:
            q.clear();
            break;
        case 5:
            cout << "\nQueue: ";
            q.display();
        default:
            break;
        }
        getch();
        clrscr();
    } while (choice != 0);
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

```
        Circular Queue - CSLList
==================================
  (1) Enqueue  (2) Dequeue
  (3) Front    (4) Clear
  (5) Display  (0) Exit

Enter Choice: 1

Enter Element: 1

Enqueued 1...
Queue: 1

Press any key to continue...
```

```
        Circular Queue - CSLList
==================================
  (1) Enqueue  (2) Dequeue
  (3) Front    (4) Clear
  (5) Display  (0) Exit

Enter Choice: 1

Enter Element: 2

Enqueued 2...
Queue: 1 → 2

Press any key to continue...
```

```
        Circular Queue - CSLList
==================================
  (1) Enqueue  (2) Dequeue
  (3) Front    (4) Clear
  (5) Display  (0) Exit

Enter Choice: 1

Enter Element: 2

Enqueued 3...
Queue: 1 → 2 → 3

Press any key to continue...
```

```
        Circular Queue - CSLList
==================================
  (1) Enqueue  (2) Dequeue
  (3) Front    (4) Clear
  (5) Display  (0) Exit

Enter Choice: 2

Dequeued 1...
Queue: 2 → 3

Press any key to continue...
```

```
        Circular Queue - CSLList
==================================
  (1) Enqueue  (2) Dequeue
  (3) Front    (4) Clear
  (5) Display  (0) Exit

Enter Choice: 3

Front Element: 2

Press any key to continue...

        Circular Queue - CSLList
==================================
  (1) Enqueue  (2) Dequeue
  (3) Front    (4) Clear
  (5) Display  (0) Exit

Enter Choice: 2

Dequeued 2...
Queue: 3

Press any key to continue...

        Circular Queue - CSLList
==================================
  (1) Enqueue  (2) Dequeue
  (3) Front    (4) Clear
  (5) Display  (0) Exit

Enter Choice: 2
Enqueued 90...
Queue: 90

Press any key to continue...

        Circular Queue - CSLList
==================================
  (1) Enqueue  (2) Dequeue
  (3) Front    (4) Clear
  (5) Display  (0) Exit

Enter Choice: 2

Dequeued 90...
Queue: Queue Empty
Press any key to continue...
```

10. Implement Double-ended Queues using Linked list representation

```
// doublyLinkedList.hpp
#include <iostream>

using namespace std;
void getch();
void clrscr();
template <class T>
class Node
{
public:
    T info;
    Node *prev;
    Node *next;
};
template <class T>
class DoublyLinkedList
{
public:
    Node<T> *head, *tail;

    // Constructor
    DoublyLinkedList()
    {
        head = tail = NULL;
    }

    // Destructor
    ~DoublyLinkedList()
    {
        if (this->isEmpty())
            return;
        Node<T> *ptr;
        for (; !isEmpty();)
        {
            ptr = head->next;
            delete head;
            head = ptr;
        }
        head = tail = ptr;
        return;
    }
```

```
    // Checks if the list is empty - O(1)   bool isEmpty()   {     return
(head == NULL || tail == NULL);
}

// Inserts a node at the beginning - O(1)   void insertFront(T info)
{
    Node<T> *temp = new Node<T>();
    temp->info = info;
    temp->next = head;
    temp->prev = NULL;
    if (this->isEmpty())
        tail = temp;
    else
        head->prev = temp;
    head = temp;
    return;
}

// Inserts a node at the end - O(1)   void insertBack(T info)
{
    Node<T> *temp = new Node<T>();
    temp->info = info;
    temp->next = NULL;
    temp->prev = tail;
    if (this->isEmpty())
        head = tail = temp;
    else
        tail->next = temp;
    tail = temp;
    return;
}

// Removes a node from the beginning - O(1)   void deleteFront()   {
if (this->isEmpty())
{
    cout << "\nList is empty...\n";
    return;
}
Node<T> *temp = head;
head = temp->next;
if (this->isEmpty())
    tail = NULL;
else
    head->prev = NULL;
delete temp;
return;
}
```

```
// Removes a node at the end - O(1)   void deleteBack()
{
    if (this->isEmpty())
    {
        cout << "\nList is empty...\n";
        return;
    }
    Node<T> *temp = tail;
    tail = temp->prev;
    if (this->isEmpty())
        head = NULL;
    else
        tail->next = NULL;
    delete temp;
    return;
}

// Traverses the list and prints all nodes - O(n)   void display()   {
if (this->isEmpty())
{
    cout << "\nList is empty...\n";
    return;
}
Node<T> *temp = head;
while (temp->next != NULL)
{
    cout << temp->info << " -> ";
    temp = temp->next;
}
cout << temp->info << endl;
return;
}
}
;

// main.cpp
#include "doublyLinkedList.hpp"
using namespace std;
void getch();
void clrscr();
template <class T>
class DoublyEndedQueue
{
protected:
    Node<T> *front, *rear;
    DoublyLinkedList<T> list;

public:
```

```
    DoublyEndedQueue()
    {
        this->front = this->list.head;
        this->rear = this->list.tail;
    }
    void enqueueFront(T ele)
    {
        this->list.insertFront(ele);
        this->front = this->list.head;
        this->rear = this->list.tail;
    }
    void enqueueRear(T ele)
    {
        this->list.insertBack(ele);
        this->front = this->list.head;
        this->rear = this->list.tail;
    }

    T dequeueFront()
    {
        if (this->isEmpty())
        {
            cout << "ERROR: Queue Empty\n";
            return (T)(NULL);
        }
        T temp = this->front->info;
        this->list.deleteFront();
        this->front = this->list.head;
        this->rear = this->list.tail;
        return temp;
    }

    T dequeueRear()
    {
        if (this->isEmpty())
        {
            cout << "ERROR: Queue Empty\n";
            return (T)(NULL);
        }
        T temp = this->rear->info;
        this->list.deleteBack();
        this->front = this->list.head;
        this->rear = this->list.tail;
        return temp;
    }

    T frontEl()
    {
```

```
        if (this->isEmpty())
        {
            cout << "Queue Empty";
            return (T)(NULL);
        }
        return this->front->info;
    }
    bool isEmpty()
    {
        return this->list.isEmpty();
    }
    void clear()
    {
        while (!this->isEmpty())
            this->dequeue();
    }
    void display()
    {
        if (this->isEmpty())
        {
            cout << "Queue Empty";
            return;
        }
        this->list.display();
        return;
    }
};
int main(void)
{
    int el, res, choice;
    DoublyEndedQueue<int> q;
    do
    {
        cout << "\tDoubly Ended Queue - Deque\n"
             << "=================================\n"
             << "  (1) EnqueueBack   (2) DequeueRear\n"
             << "  (3) EnqueueFront  (4) DequeueFront\n"
             << "  (5) Front         (6) Display\n"
             << "  (0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "\nEnter Element: ";
            cin >> el;
            q.enqueueRear(el);
            cout << "\nEnqueued " << el << " at rear...\n";
```

```
            cout << "Queue: ";
            q.display();
            break;
        case 2:
            res = q.dequeueRear();
            if (res)
            {
                cout << "\nDequeued " << res << " from rear...\n";
                cout << "Queue: ";
                q.display();
            }
            break;
        case 3:
            cout << "\nEnter Element: ";
            cin >> el;
            q.enqueueFront(el);
            cout << "\nEnqueued " << el << " at front...\n";
            cout << "Queue: ";
            q.display();
            break;
        case 4:
            res = q.dequeueFront();
            if (res)
            {
                cout << "\nDequeued " << res << " from front...\n";
                cout << "Queue: ";
                q.display();
            }
            break;
        case 5:
            cout << "\nFront Element: " << q.frontEl() << endl;
            break;
        case 6:
            cout << "\nQueue: ";
            q.display();
        default:
            break;
        }
        getch();
        clrscr();
    } while (choice != 0);
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
```

```
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

```
        Doubly Ended Queue - Deque
====================================
  (1) EnqueueBack   (2) DequeueRear
  (3) EnqueueFront  (4) DequeueFront
  (5) Front         (6) Display
  (0) Exit

Enter Choice: 1

Enter Element: 10

Enqueued 10 at rear...
Queue: 10

Press any key to continue...
```

```
        Doubly Ended Queue - Deque
====================================
  (1) EnqueueBack   (2) DequeueRear
  (3) EnqueueFront  (4) DequeueFront
  (5) Front         (6) Display
  (0) Exit

Enter Choice: 3

Enter Element: 20

Enqueued 20 at front...
Queue: 20 -> 10

Press any key to continue...
```

```
        Doubly Ended Queue - Deque
====================================
  (1) EnqueueBack   (2) DequeueRear
  (3) EnqueueFront  (4) DequeueFront
  (5) Front         (6) Display
  (0) Exit

Enter Choice: 2

Dequeued 10 from rear...
Queue: 20

Press any key to continue...
```

```
        Doubly Ended Queue - Deque
====================================
  (1) EnqueueBack   (2) DequeueRear
  (3) EnqueueFront  (4) DequeueFront
  (5) Front         (6) Display
  (0) Exit

Enter Choice: 4

Dequeued 20 from front...
Queue: Queue Empty
Press any key to continue...
```

11. Write a program to implement Binary Search Tree which supports the following operations:

(i) Insert an element x

(ii) Delete an element x

(iii) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position in the BST

(iv) Display the elements of the BST in preorder, inorder, and postorder traversal

(v) Display the elements of the BST in level-by-level traversal

(vi) Display the height of the BST

```
#include <iostream>
#define MAX_SIZE 100
using namespace std;
template <class T>
class Stack
{
protected:
    int tos, size;
    T arr[MAX_SIZE];

public:
    Stack(int size = 30)
    {
        this->tos = -1;
        this->size = size;
    }
    bool push(T ele)
    {
        if (this->tos >= (this->size - 1))
        {
            cerr << "ERROR: Stack Overflow\n";
            return false;
        }
        this->arr[++(this->tos)] = ele;
        return true;
    }

    T pop()
    {
        if (this->isEmpty())
        {
            cout << "ERROR: Stack Underflow\n";
            return (T)(NULL);
        }
        return this->arr[(this->tos)--];
    }

    T top()
    {
```

```
        if (this->isEmpty())
        {
            cout << "Stack Empty";
            return (T)(NULL);
        }
        return this->arr[this->tos];
    }
    bool isEmpty()
    {
        return this->tos == -1;
    }
    void clear()
    {
        while (!this->isEmpty())
            this->pop();
    }
};

// queue.hpp
#include <iostream>
#define MAX_SIZE 100

using namespace std;
template <class T>
class Queue
{
protected:
    T arr[MAX_SIZE];
    int front, rear, size;

public:
    Queue(int size = 100)
    {
        this->front = -1;
        this->rear = -1;
        this->size = size;
    }
    bool enqueue(T ele)
    {
        if (this->rear >= (this->size - 1))
        {
            cerr << "ERROR: Queue Filled\n";
            return false;
        }
        else if (this->isEmpty())
        {
            this->rear++;
            this->front++;
```

```
            this->arr[this->front] = ele;
        }
        else
            this->arr[++(this->rear)] = ele;
        return true;
    }

    T dequeue()
    {
        if (this->front >= this->size)
        {
            cout << "ERROR: Queue Finished\n";
            return (T)(NULL);
        }
        else if (this->isEmpty())
        {
            cout << "ERROR: Queue Empty\n";
            return (T)(NULL);
        }
        else if (this->front == this->rear)
        {
            T temp = this->arr[this->front];
            this->clear();
            return temp;
        }
        return this->arr[(this->front)++];
    }

    T frontEl()
    {
        if (this->isEmpty())
        {
            cout << "Queue Empty";
            return (T)(NULL);
        }
        return this->arr[this->front];
    }
    bool isEmpty()
    {
        return this->front == -1;
    }
    void clear()
    {
        this->front = this->rear = -1;
    }
    void display()
    {
        if (this->isEmpty())
```

```
        {
            cout << "Queue Empty";
            return;
        }
        int i;
        for (i = this->front; i < this->rear; i++)
            cout << this->arr[i] << " <- ";
        cout << this->arr[i] << endl;
        return;
    }
};

// main.cpp
#include "stack.hpp"
#include "queue.hpp"
void getch();
void clrscr();
template <class T>
class Node
{
public:
    T data;
    Node *left, *right;
    Node()
    {
        left = nullptr;
        right = nullptr;
    }
};

class BinarySearchTree
{
public:
    Node<int> *root;
    Stack<Node<int> *> stack;
    Queue<Node<int> *> queue;
    int countLeaf, countNonLeaf;

    BinarySearchTree()
    {
        root = nullptr;
    }
    void insert(int data, Node<int> *current)
    {
        Node<int> *temp;
        if (root == nullptr)
        {
            root = new Node<int>;
```

```
            root->data = data;
            root->left = root->right = nullptr;
        }
        else
        {
            if ((data < current->data) &&
                (current->left == nullptr))
            {
                temp = new Node<int>;
                temp->data = data;
                temp->left = temp->right = nullptr;
                current->left = temp;
            }
            else if ((data >= current->data) &&
                     (current->right == nullptr))
            {
                temp = new Node<int>;
                temp->data = data;
                temp->left = temp->right = nullptr;
                current->right = temp;
            }
            else
            {
                if (data < current->data)
                    insert(data, current->left);
                else
                    insert(data, current->right);
            }
        }
    }
    bool search(Node<int> *node, int key)
    {
        if (node == nullptr)
            return false;
        if (node->data == key)
            return true;
        bool left = search(node->left, key);
        if (left)
            return true;
        bool right = search(node->right, key);
        return right;
    }
    void inOrderRecursive(Node<int> *root)
    {
        if (root != nullptr)
        {
            inOrderRecursive(root->left);
            cout << root->data << " ";
```

```
            inOrderRecursive(root->right);
        }
    }
    void preOrderRecursive(Node<int> *root)
    {
        if (root != nullptr)
        {
            cout << root->data << " ";
            preOrderRecursive(root->left);
            preOrderRecursive(root->right);
        }
    }
    void postOrderRecursive(Node<int> *root)
    {
        if (root != nullptr)
        {
            postOrderRecursive(root->left);
            postOrderRecursive(root->right);
            cout << root->data << " ";
        }
    }
    void inOrderIterative()
    {
        Node<int> *current = root;
        while (current != nullptr || stack.isEmpty() == false)
        {
            while (current != nullptr)
            {
                stack.push(current);
                current = current->left;
            }
            current = stack.pop();
            cout << current->data << " ";
            current = current->right;
        }
    }
    void preOrderIterative()
    {
        Node<int> *node, *temp = root;
        if (temp == nullptr)
            return;
        stack.push(temp);
        while (!stack.isEmpty())
        {
            node = stack.pop();
            cout << node->data << " ";
            if (node->right)
                stack.push(node->right);
```

```
            if (node->left)
                stack.push(node->left);
        }
    }
    void postOrderIterative()
    {
        Node<int> *temp = root;
        if (temp == nullptr)
            return;
        do
        {
            while (temp)
            {
                if (temp->right)
                    stack.push(temp->right);
                stack.push(temp);
                temp = temp->left;
            }
            temp = stack.pop();
            if (temp->right && !stack.isEmpty() && stack.top() == temp->right)
            {
                stack.pop();
                stack.push(temp);
                temp = temp->right;
            }
            else
            {
                cout << temp->data << " ";
                temp = nullptr;
            }
        } while (!stack.isEmpty());
    }
    void levelByLevelTraversal()
    {
        Node<int> *current = root;
        if (current == nullptr)
            return;
        queue.enqueue(current);
        while (!queue.isEmpty())
        {
            current = queue.dequeue();
            cout << current->data << " ";
            if (current->left)
                queue.enqueue(current->left);
            if (current->right)
                queue.enqueue(current->right);
        }
        cout << endl;
```

```
}
void mirror(Node<int> *current)
{
    if (current == nullptr)
        return;
    else
    {
        mirror(current->left);
        mirror(current->right);

        Node<int> *temp = current->left;

        current->left = current->right;
        current->right = temp;
    }
}
int height(Node<int> *current)
{
    if (current == nullptr)
        return 0;
    else
    {
        int leftHeight = height(current->left);
        int rightHeight = height(current->right);
        if (leftHeight > rightHeight)
            return (leftHeight + 1);
        else
            return (rightHeight + 1);
    }
}
void countNodes(Node<int> *current)
{
    if (current == nullptr)
        return;
    if (current->left != nullptr || current->right != nullptr)
        countNonLeaf++;
    if (current->left == nullptr && current->right == nullptr)
        countLeaf++;
    countNodes(current->left);
    countNodes(current->right);
}
void deleteByMerging(Node<int> *temp, int key)
{
    Node<int> *prev = nullptr;

    while (temp != nullptr)
    {
        if (temp->data == key)
```

```
                break;
            prev = temp;
            if (temp->data < key)
                temp = temp->right;
            else
                temp = temp->left;
        }
        if (temp != nullptr && temp->data == key)
        {
            if (temp == root)
                mergeHelper(root);
            else if (prev->left == temp)
                mergeHelper(prev->left);
            else
                mergeHelper(prev->right);
        }
        else if (root != nullptr)
            cout << "\nNode Not Found...";
        return;
    }
    void mergeHelper(Node<int> *&node)
    {
        Node<int> *temp = node;
        if (node == nullptr)
            return;

        // no right child - single child     if (node->right ==
nullptr)       node = node->left;

        // no left child - single chold     else if (node->left ==
nullptr)       node = node->right;

        // node has both children     else
        {
            // find in-order predecessor       temp = node->left;       while
(temp->right != nullptr)         temp = temp->right;       // merge subtree to
predecessor       temp->right = node->right;       temp = node;       node =
node->left;
        }

        // delete the node     delete temp;      return;
    }
    void deleteByCopying(Node<int> *temp, int key)
    {
        Node<int> *prev = nullptr;
        while (temp != nullptr && temp->data != key)
        {
            prev = temp;
```

```
            if (temp->data < key)
                temp = temp->right;
            else
                temp = temp->left;
        }
        if (temp != nullptr && temp->data == key)
        {
            if (temp == root)
                copyHelper(root);
            else if (prev->left == temp)
                copyHelper(prev->left);
            else
                copyHelper(prev->right);
        }
        else if (root != nullptr)
            cout << "\nNode Not Found...";
        return;
    }
    void copyHelper(Node<int> *&node)
    {
        Node<int> *prev, *temp = node;

        // no right child - single child     if (node->right ==
nullptr)       node = node->left;

        // no left child - single chold     else if (node->left ==
nullptr)       node = node->right;

        // node has both children     else     {       prev = node;
        // find the in-order predecessor       temp = node->left;       while
(temp->right != nullptr)
        {
            prev = temp;
            temp = temp->right;
        }
        // copy the prdecessor key       node->data = temp->data;       //
handle dangling subtrees       if (prev == node)         prev->left = temp-
>left;       else
        prev->right = temp->left;
    }

    // delete the node     delete temp;      return;
} void searchAndReplace(int key, int newKey)
{
    if (search(root, key))
    {
        deleteByMerging(root, key);
        insert(newKey, root);
```

```
    }
    else
    {
        cout << "Node Not Found...";
    }
}
}
;
int main(void)
{
    BinarySearchTree tree;
    int choice, data, data2;
    do
    {
        cout << "      MENU      \n"
             << "================\n"
             << "(1) Insertion\n"
             << "(2) Searching a node\n"
             << "(3) Display its preorder, postorder and inorder traversals.
(recu rsive)\n"
             << "(4) Display its preorder, postorder and inorder traversals.
(iter ative)\n"
             << "(5) Display level-by-level traversal. (BFS)\n"
             << "(6) Create a mirror image of tree\n"
             << "(7) Count the non-leaf, leaf and total number of nodes  \n"
             << "(8) Search for an element x in the BST and change its value
to y \n"
             << "    and then place the node with value y at its appropriate
posit ion\n"
             << "(9) Display height of tree\n"
             << "(10) Perform deletion by merging\n"
             << "(11) Perform deletion by copying\n"
             << "(0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "\nEnter Node Data: ";
            cin >> data;
            tree.insert(data, tree.root);
            break;
        case 2:
            cout << "\nEnter Search Data: ";
            cin >> data;
            cout << "Search Result: ";
            if (tree.search(tree.root, data))
                cout << "Found";
```

```
            else
                cout << "Not Found";
            cout << endl;
            break;
        case 3:
            cout << endl;
            cout << "In-Order Recursive Traversal: ";
            tree.inOrderRecursive(tree.root);
            cout << endl;
            cout << "Pre-Order Recursive Traversal: ";
            tree.preOrderRecursive(tree.root);
            cout << endl;
            cout << "Post-Order Recursive Traversal: ";
            tree.postOrderRecursive(tree.root);
            cout << endl;
            break;
        case 4:
            cout << endl;
            cout << "In-Order Iterative Traversal: ";
            tree.inOrderIterative();
            cout << endl;
            cout << "Pre-Order Iterative Traversal: ";
            tree.preOrderIterative();
            cout << endl;
            cout << "Post-Order Iterative Traversal: ";
            tree.postOrderIterative();
            cout << endl;
            break;
        case 5:
            cout << endl;
            cout << "Level-by-level Traversal: \n";
            tree.levelByLevelTraversal();
            break;
        case 6:
            cout << endl;
            tree.mirror(tree.root);
            cout << "Tree converted to its Mirror Tree..."
                 << endl;
            break;
        case 7:
            tree.countLeaf = tree.countNonLeaf = 0;
            tree.countNodes(tree.root);
            cout << endl;
            cout << "Leaf Nodes: " << tree.countLeaf << endl;
            cout << "Non-Leaf Nodes: " << tree.countNonLeaf << endl;
            cout << "Total Nodes: " << tree.countNonLeaf + tree.countLeaf <<
endl;
            break;
```

```
        case 8:
            cout << "\nEnter Search Data: ";
            cin >> data;
            cout << "Enter Replacement: ";
            cin >> data2;
            tree.searchAndReplace(data, data2);
            break;
        case 9:
            cout << endl;
            cout << "Height of Tree: "
                 << tree.height(tree.root)
                 << endl;
            break;
        case 10:
            cout << "\nEnter Node to Delete: ";
            cin >> data;
            tree.deleteByMerging(tree.root, data);
            break;
        case 11:
            cout << "\nEnter Node to Delete: ";
            cin >> data;
            tree.deleteByCopying(tree.root, data);
            break;
        case 0:
        default:
            break;
        }
        getch();
        clrscr();
    } while (choice != 0);
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 1

Enter Node Data: 10

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 1

Enter Node Data: 5

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 1

Enter Node Data: 14

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 1

Enter Node Data: 0

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 1

Enter Node Data: 6

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree

(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 1

Enter Node Data: 10

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y
and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 1

Enter Node Data: 14

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree

(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 2

Enter Search Data: 14

Search Result: Found

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 2

Enter Search Data: 2

Search Result: Not Found

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 0 5 6 10 10 14 14

Pre-Order Recursive Traversal: 10 5 0 6 14 10 14

Post-Order Recursive Traversal: 0 6 5 10 14 14 10

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 4

In-Order Iterative Traversal: 0 5 6 10 10 14 14

Pre-Order Iterative Traversal: 10 5 0 6 14 10 14

Post-Order Iterative Traversal: 0 6 5 10 14 14 10

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 5

Level-by-level Traversal:

10 5 14 0 6 10 14

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree

(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 7

Leaf Nodes: 4

Non-Leaf Nodes: 3

Total Nodes: 7

Press any key to continue...

MENU

==========

(1) Insertion
(2) Searching a node
(3) Display its preorder, postorder and inorder traversals. (recursive)
(4) Display its preorder, postorder and inorder traversals. (iterative)
(5) Display level-by-level traversal. (BFS)
(6) Create a mirror image of tree
(7) Count the non-leaf, leaf and total number of nodes
(8) Search for an element x in the BST and change its value to y and then place the node with value y at its appropriate position
(9) Display height of tree
(10) Perform deletion by merging
(11) Perform deletion by copying
(0) Exit

Enter Choice: 0

Press any key to continue...

12. Write a program, using templates, to sort a list of n elements. Give user the option to perform sorting using Insertion sort, Bubble sort or Selection sort.

```
#include <iostream>
#define MAX_SIZE 100
using namespace std;
```

```
void getch();
void clrscr();
template <class T>
void swapElements(T &a, T &b)
{
    T t = a;
    a = b;
    b = t;
}
template <class T>
T *bubbleSort(T *arr, int size)
{
    for (int i = 0; i < size - 1; i++)
        for (int j = 0; j < size - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swapElements<T>(arr[j], arr[j + 1]);
    return arr;
}
template <class T>
T *insertionSort(T *arr, int size)
{
    T key;
    int i, j;
    for (i = 1; i < size; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j--];
        }
        arr[j + 1] = key;
    }
    return arr;
}
template <class T>
T *selectionSort(T *arr, int size)
{
    int min;
    for (int i = 0; i < size - 1; i++)
    {
        min = i;
        for (int j = i + 1; j < size; j++)
            if (arr[j] < arr[min])
                min = j;
        swapElements<T>(arr[min], arr[i]);
    }
    return arr;
```

```
}
template <class T>
void display(T *arr, int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
}
int main(void)
{
    int ch = 1, size, arr[MAX_SIZE];
    cout << "Enter Number of Elements: ";
    cin >> size;

    cout << "Enter Array Elements: ";
    for (int i = 0; i < size; i++)
        cin >> arr[i];
    clrscr();
    do
    {
        cout << "\t\tMenu\n=======================================\n"
             << " (1) Bubble Sort     (2) Insertion Sort\n"
             << " (3) Selection Sort  (0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> ch;
        switch (ch)
        {
        case 1:
            cout << endl;
            cout << "Original Array: ";
            display<int>(arr, size);
            cout << endl;
            cout << "Bubble Sort: ";
            display<int>(bubbleSort<int>(arr, size), size);
            cout << endl;
            break;
        case 2:
            cout << endl;
            cout << "Original Array: ";
            display<int>(arr, size);
            cout << endl;
            cout << "Insertion Sort: ";
            display<int>(insertionSort<int>(arr, size), size);
            cout << endl;
            break;
        case 3:
            cout << endl;
            cout << "Original Array: ";
            display<int>(arr, size);
```

```
            cout << endl;
            cout << "Selection Sort: ";
            display<int>(selectionSort<int>(arr, size), size);
            cout << endl;
            break;
        default:
            break;
        }
        getch();
        clrscr();
    } while (ch != 0);
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

```
Enter Number of Elements: 5
Enter Array Elements: 1 9 8 2 3

                Menu
 ======================================
  (1) Bubble Sort     (2) Insertion Sort
  (3) Selection Sort  (0) Exit

 Enter Choice: 1

 Original Array: 1 9 8 2 3
 Bubble Sort: 1 2 3 8 9

 Press any key to continue...

Enter Number of Elements: 5
Enter Array Elements: 1 9 8 2 3

                Menu
 ======================================
  (1) Bubble Sort     (2) Insertion Sort
  (3) Selection Sort  (0) Exit

Enter Choice: 2

Original Array: 1 9 8 2 3
Insertion Sort: 1 2 8 2 3

Press any key to continue...

Enter Number of Elements: 5
Enter Array Elements: 1 9 8 2 3

                Menu
 ======================================
  (1) Bubble Sort     (2) Insertion Sort
  (3) Selection Sort  (0) Exit

Enter Choice: 3

Original Array: 1 9 8 2 3
Selection Sort: 1 2 3 8 9

Press any key to continue...
```

13. Write a program to implement:

(a) Diagonal Matrix using one-dimensional array

```cpp
#include <climits>
#include <iostream>
#define MAX_SIZE 10

using namespace std;
```

```
void getch();
void clrscr();
class DiagonalMatrix
{
public:
    int *arr;
    int size;
    int nrows;
    int ncols;

    DiagonalMatrix(int rows = MAX_SIZE, int cols = MAX_SIZE)
    {
        if (rows != cols)
            cerr << "ERROR: Invalid Dimensions" << endl;
        else
        {
            size = rows;
            nrows = ncols = rows;
            arr = new int[size];
            for (int i = 0; i < size; i++)
                arr[i] = 0;
        }
    }

    ~DiagonalMatrix()
    {
        delete[] arr;
    }
    void store(int data, int row, int col)
    {
        if (row != col || row >= nrows || col >= ncols)
            cerr << "ERROR: Invalid Location" << endl;
        else
            arr[row] = data;
    }
    int retrieve(int row, int col)
    {
        if (row >= nrows || col >= ncols)
            return INT_MIN;
        if (row != col)
            return 0;
        else
            return arr[row];
    }
    void display()
    {
        for (int i = 0; i < nrows; i++)
        {
```

```
            for (int j = 0; j < ncols; j++)
                if (i == j)
                    cout << arr[i] << " ";
                else
                    cout << 0 << " ";
            cout << endl;
        }
    }
};
int main(void)
{
    int rows, cols, data, choice = 1;
    cout << "Enter Number of Rows: ";
    cin >> rows;
    cout << "Enter Number of Columns: ";
    cin >> cols;
    clrscr();

    DiagonalMatrix matrix(rows, cols);
    do
    {
        cout << "      MENU      \n"
             << "================\n"
             << "(1) Display\n"
             << "(2) Store\n"
             << "(3) Retrieve\n"
             << "(0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "\nMatrix:\n";
            matrix.display();
            break;
        case 2:
            cout << "Enter Data: ";
            cin >> data;
            cout << "Enter Position: ";
            cin >> rows >> cols;
            matrix.store(data, rows, cols);
            break;
        case 3:
            cout << "Enter Position: ";
            cin >> rows >> cols;
            data = matrix.retrieve(rows, cols);
            if (data != INT_MIN)
```

```
                cout << "Retrieved " << data << endl;
            else
                cerr << "ERROR: Invalid Location" << endl;
            break;
        case 0:
        default:
            break;
        }
        getch();
        clrscr();

    } while (choice != 0);
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

Enter Number of Rows: 3

Enter Number of Columns: 3

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit

Enter Choice: 2

Enter Data: 8

Enter Position: 0 1

ERROR: Invalid Location

Press any key to continue...

MENU

==========

(1) Display
(2) Store
(3) Retrieve
(0) Exit

Enter Choice: 2

Enter Data: 5

Enter Position: 1 1

Press any key to continue...

MENU

==========

(1) Display (2) Store (3) Retrieve
(0) Exit

Enter Choice: 1

Matrix:

0 0 0

0 5 0

0 0 0

Press any key to continue...

MENU

==========

(1) Display
(2) Store (3) Retrieve
(0) Exit

Enter Choice: 2

Enter Data: 4

Enter Position: 2 2

Press any key to continue..

MENU

==========

(1) Display (2) Store (3) Retrieve
(0) Exit
Enter Choice: 3

Enter Position: 2 2

Retrieved 4

Press any key to continue...

MENU

==========

(1) Display (2)
Store (3) Retrieve
(0) Exit
Enter Choice: 3

Enter Position: 0 2

Retrieved 0

Press any key to continue...

MENU

==========

(1) Display (2)
Store (3) Retrieve
(0) Exit

Enter Choice: 3

Enter Position: 5 5

ERROR: Invalid Location

Press any key to continue...

(b) Lower Triangular Matrix using one-dimensional array

```
#include <climits>
#include <iostream>
#define MAX_SIZE 10

using namespace std;
void getch();
void clrscr();
class LowerTriangularMatrix
{
public:
    int *arr;
    int size;
    int nrows;
    int ncols;

    LowerTriangularMatrix(int rows = MAX_SIZE, int cols = MAX_SIZE)
    {
        if (rows != cols)
            cerr << "ERROR: Invalid Dimensions" << endl;
        else
        {
            size = rows * (rows + 1) / 2;
            nrows = ncols = rows;
            arr = new int[size];
            for (int i = 0; i < size; i++)
                arr[i] = 0;
        }
    }

    ~LowerTriangularMatrix()
    {
        delete[] arr;
    }
    void store(int data, int row, int col)
    {
        if (col > row || row >= nrows || col >= ncols)
            cerr << "ERROR: Invalid Location" << endl;
        else
            arr[row * (row + 1) / 2 + col] = data;
    }
    int retrieve(int row, int col)
    {
        if (row >= nrows || col >= ncols)
            return INT_MIN;
        if (col > row)
            return 0;
```

```
        else
            return arr[row * (row + 1) / 2 + col];
    }
    void display()
    {
        for (int i = 0; i < nrows; i++)
        {
            for (int j = 0; j < ncols; j++)
                if (i >= j)
                    cout << arr[i * (i + 1) / 2 + j] << " ";
                else
                    cout << 0 << " ";
            cout << endl;
        }
    }
};
int main(void)
{
    int rows, cols, data, choice = 1;
    cout << "Enter Number of Rows: ";
    cin >> rows;
    cout << "Enter Number of Columns: ";
    cin >> cols;
    clrscr();

    LowerTriangularMatrix matrix(rows, cols);
    do
    {
        cout << "      MENU      \n"
             << "================\n"
             << "(1) Display\n"
             << "(2) Store\n"
             << "(3) Retrieve\n"
             << "(0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "\nMatrix:\n";
            matrix.display();
            break;
        case 2:
            cout << "Enter Data: ";
            cin >> data;
            cout << "Enter Position: ";
            cin >> rows >> cols;
```

```
            matrix.store(data, rows, cols);
            break;
        case 3:
            cout << "Enter Position: ";
            cin >> rows >> cols;
            data = matrix.retrieve(rows, cols);
            if (data != INT_MIN)
                cout << "Retrieved " << data << endl;
            else
                cerr << "ERROR: Invalid Location" << endl;
            break;
        case 0:
        default:
            break;
        }
        getch();
        clrscr();

    } while (choice != 0);
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

Enter Number of Rows: 3

Enter Number of Columns: 3

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit
Enter Choice: 2

Enter Data: 4

Enter Position: 1 0

Press any key to continue...

MENU

==========

(1) Display
(2) Store
(3) Retrieve
(0) Exit

Enter Choice: 1

Matrix:

0 0 0

4 0 0

0 0 0

Press any key to continue...

MENU

==========

(1) Display
(2) Store
(3) Retrieve
(0) Exit

Enter Choice: 2

Enter Data: 2

Enter Position: 0 1

ERROR: Invalid Location

Press any key to continue...

MENU

==========

(1) Display
(2) Store
(3) Retrieve
(0) Exit
Enter Choice: 2

Enter Data: 8

Enter Position: 2 1

Press any key to continue...

MENU

==========

(1) Display
(2) Store (3) Retrieve
(0) Exit

Enter Choice: 1

Matrix:

0 0 0

4 0 0

0 8 0

Press any key to continue...

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit

Enter Choice: 3

Enter Position: 1 0

Retrieved 4

Press any key to continue...

(c) Upper Triangular Matrix using one-dimensional array

```
#include <climits>
#include <iostream>
#define MAX_SIZE 10

using namespace std;
void getch();
void clrscr();
class UpperTriangularMatrix
{
public:
    int *arr;
    int size;
    int nrows;
    int ncols;

    UpperTriangularMatrix(int rows = MAX_SIZE, int cols = MAX_SIZE)
    {
        if (rows != cols)
            cerr << "ERROR: Invalid Dimensions" << endl;
        else
        {
            size = cols * (cols + 1) / 2;
            nrows = ncols = rows;
            arr = new int[size];
            for (int i = 0; i < size; i++)
                arr[i] = 0;
        }
    }

    ~UpperTriangularMatrix()
    {
        delete[] arr;
    }
    void store(int data, int row, int col)
    {
        if (row > col || row >= nrows || col >= ncols)
            cerr << "ERROR: Invalid Location" << endl;
        else
            arr[col * (col + 1) / 2 + row] = data;
    }
    int retrieve(int row, int col)
    {
        if (row >= nrows || col >= ncols)
            return INT_MIN;
        if (row > col)
            return 0;
        else
```

```
            return arr[col * (col + 1) / 2 + row];
    }
    void display()
    {
        for (int i = 0; i < nrows; i++)
        {
            for (int j = 0; j < ncols; j++)
                if (j >= i)
                    cout << arr[j * (j + 1) / 2 + i] << " ";
                else
                    cout << 0 << " ";
            cout << endl;
        }
    }
};
int main(void)
{
    int rows, cols, data, choice = 1;
    cout << "Enter Number of Rows: ";
    cin >> rows;
    cout << "Enter Number of Columns: ";
    cin >> cols;
    clrscr();

    UpperTriangularMatrix matrix(rows, cols);
    do
    {
        cout << "      MENU      \n"
             << "================\n"
             << "(1) Display\n"
             << "(2) Store\n"
             << "(3) Retrieve\n"
             << "(0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "\nMatrix:\n";
            matrix.display();
            break;
        case 2:
            cout << "Enter Data: ";
            cin >> data;
            cout << "Enter Position: ";
            cin >> rows >> cols;
            matrix.store(data, rows, cols);
```

```
            break;
        case 3:
            cout << "Enter Position: ";
            cin >> rows >> cols;
            data = matrix.retrieve(rows, cols);
            if (data != INT_MIN)
                cout << "Retrieved " << data << endl;
            else
                cerr << "ERROR: Invalid Location" << endl;
            break;
        case 0:
        default:
            break;
        }
        getch();
        clrscr();

    } while (choice != 0);
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

Enter Number of Rows: 3

Enter Number of Columns: 3

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit

Enter Choice: 2

Enter Data: 3

Enter Position: 0 2

Press any key to continue...

MENU

==========

(1) Display (2) Store (3) Retrieve
(0) Exit

Enter Choice: 2

Enter Data: 5

Enter Position: 1 1

Press any key to continue...

MENU

==========

(1) Display
(2) Store
(3) Retrieve
(0) Exit

Enter Choice: 1

Matrix:

0 0 3

0 5 0

0 0 0

Press any key to continue...

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit
Enter Choice: 2

Enter Data: 7

Enter Position: 2 0

ERROR: Invalid Location

Press any key to continue...

MENU

==========

(1) Display
(2) Store
(3) Retrieve (0) Exit

Enter Choice: 3

Enter Position: 0 2

Retrieved 3

Press any key to continue...

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit

Enter Choice: 3

Enter Position: 1 1

Retrieved 5

Press any key to continue...

(d) Symmetric Matrix using one-dimensional array

```
#include <climits>
#include <iostream>
#define MAX_SIZE 10
```

```
using namespace std;
void getch();
void clrscr();
class SymmetricMatrix
{
public:
    int *arr;
    int size;
    int nrows;
    int ncols;

    SymmetricMatrix(int rows = MAX_SIZE, int cols = MAX_SIZE)
    {
        if (rows != cols)
            cerr << "ERROR: Invalid Dimensions" << endl;
        else
        {
            size = rows * (rows + 1) / 2;
            nrows = ncols = rows;
            arr = new int[size];
            for (int i = 0; i < size; i++)
                arr[i] = 0;
        }
    }

    ~SymmetricMatrix()
    {
        delete[] arr;
    }
    void store(int data, int row, int col)
    {
        if (row >= nrows || col >= ncols)
            cerr << "ERROR: Invalid Location" << endl;
        else if (col > row)
            arr[col * (col + 1) / 2 + row] = data;
        else
            arr[row * (row + 1) / 2 + col] = data;
    }
    int retrieve(int row, int col)
    {
        if (row >= nrows || col >= ncols)
            return INT_MIN;
        if (col > row)
            return arr[col * (col + 1) / 2 + row];
        else
            return arr[row * (row + 1) / 2 + col];
    }
```

```
    void display()
    {
        for (int i = 0; i < nrows; i++)
        {
            for (int j = 0; j < ncols; j++)
                if (i > j)
                    cout << arr[i * (i + 1) / 2 + j] << " ";
                else
                    cout << arr[j * (j + 1) / 2 + i] << " ";
            cout << endl;
        }
    }
};
int main(void)
{
    int rows, cols, data, choice = 1;
    cout << "Enter Number of Rows: ";
    cin >> rows;
    cout << "Enter Number of Columns: ";
    cin >> cols;
    clrscr();

    SymmetricMatrix matrix(rows, cols);
    do
    {
        cout << "      MENU      \n"
             << "================\n"
             << "(1) Display\n"
             << "(2) Store\n"
             << "(3) Retrieve\n"
             << "(0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "\nMatrix:\n";
            matrix.display();
            break;
        case 2:
            cout << "Enter Data: ";
            cin >> data;
            cout << "Enter Position: ";
            cin >> rows >> cols;
            matrix.store(data, rows, cols);
            break;
        case 3:
```

```
            cout << "Enter Position: ";
            cin >> rows >> cols;
            data = matrix.retrieve(rows, cols);
            if (data != INT_MIN)
                cout << "Retrieved " << data << endl;
            else
                cerr << "ERROR: Invalid Location" << endl;
            break;
        case 0:
        default:
            break;
        }
        getch();
        clrscr();

    } while (choice != 0);

    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

Enter Number of Rows: 3

Enter Number of Columns: 3

MENU

==========

(1) Display
(2) Store
(3) Retrieve
(0) Exit

Enter Choice: 2

Enter Data: 8

Enter Position: 1 2

Press any key to continue...

MENU

==========

(1) Display
(2) Store
(3) Retrieve
(0) Exit

Enter Choice: 1

Matrix:

0 0 0

0 0 8

0 8 0

Press any key to continue...

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit
Enter Choice: 2

Enter Data: 9

Enter Position: 0 2

Press any key to continue...

MENU

==========

(1) Display (2)
Store (3) Retrieve
(0) Exit

Enter Choice: 1

Matrix:

0 0 9

0 0 8

9 8 0

Press any key to continue...

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit
Enter Choice: 2

Enter Data: 5

Enter Position: 0 0

Press any key to continue...

MENU

==========

(1) Display
(2) Store
(3) Retrieve
(0) Exit

Enter Choice: 1

Matrix:

5 0 9

0 0 8

9 8 0

Press any key to continue...

MENU

==========

(1) Display (2) Store
(3) Retrieve

(0) Exit

Enter Choice: 3

Enter Position: 2 0

Retrieved 9

Press any key to continue...

14. Write a program to implement AVL Tree.

```
#include <iostream>

using namespace std;
void getch();
void clrscr();
int max(int, int);
template <class T>
class Node
{
public:
    T data;
    int height;
    int balanceFactor;
    Node *left, *right, *parent;
    Node()
    {
        left = nullptr;
        right = nullptr;
        parent = nullptr;
        height = 1;
        balanceFactor = 0;
    }
};
class AVLTree
{
public:
```

```
    Node<int> *root;

    AVLTree()
    {
        root = nullptr;
    }
    int getBalanceFactor(Node<int> *node)
    {
        if (node == nullptr)
            return 0;
        return height(node->right) - height(node->left);
    }

    Node<int> *rightRotate(Node<int> *y)
    {
        Node<int> *x = y->left;
        y->left = x->right;
        if (x->right != nullptr)
            x->right->parent = y;
        x->parent = y->parent;
        if (y->parent == nullptr)
            root = x;
        else if (y == y->parent->right)
            y->parent->right = x;
        else
            y->parent->left = x;
        x->right = y;
        y->parent = x;
        y->height = 1 + max(height(y->left), height(y->right));
        x->height = 1 + max(height(x->left), height(x->right));
        x->balanceFactor = getBalanceFactor(x);
        y->balanceFactor = getBalanceFactor(y);
        return x;
    }

    Node<int> *leftRotate(Node<int> *x)
    {
        Node<int> *y = x->right;
        x->right = y->left;
        if (y->left != nullptr)
            y->left->parent = x;
        y->parent = x->parent;

        if (x->parent == nullptr)
            root = y;
        else if (x == x->parent->left)
            x->parent->left = y;
        else
```

```
            x->parent->right = y;
        y->left = x;
        x->parent = y;
        y->height = 1 + max(height(y->left), height(y->right));
        x->height = 1 + max(height(x->left), height(x->right));
        x->balanceFactor = getBalanceFactor(x);
        y->balanceFactor = getBalanceFactor(y);
        return y;
    }

    Node<int> *insert(int data, Node<int> *current)
    {
        Node<int> *temp;
        if (current == nullptr)
        {
            temp = new Node<int>;
            temp->data = data;
            return temp;
        }
        if (data < current->data)
        {
            current->left = insert(data, current->left);
            current->left->parent = current;
        }
        else
        {
            current->right = insert(data, current->right);
            current->right->parent = current;
        }
        current->height = 1 + max(height(current->left), height(current-
>right));
        current->balanceFactor = getBalanceFactor(current);

        if (current->balanceFactor < -1 && data < current->left->data)
            return rightRotate(current);
        if (current->balanceFactor > 1 && data > current->right->data)
            return leftRotate(current);

        if (current->balanceFactor < -1 && data > current->left->data)
        {
            current->left = leftRotate(current->left);
            return rightRotate(current);
        }
        if (current->balanceFactor > 1 && data < current->right->data)
        {
            current->right = rightRotate(current->right);
            return leftRotate(current);
        }
```

```
        return current;
    }
    bool search(Node<int> *node, int key)
    {
        if (node == nullptr)
            return false;
        if (node->data == key)
            return true;
        bool left = search(node->left, key);
        if (left)
            return true;
        bool right = search(node->right, key);
        return right;
    }
    void inOrderRecursive(Node<int> *root)
    {
        if (root != nullptr)
        {
            inOrderRecursive(root->left);
            cout << root->data << " (" << root->balanceFactor << ") ";
            inOrderRecursive(root->right);
        }
    }
    void preOrderRecursive(Node<int> *root)
    {
        if (root != nullptr)
        {
            cout << root->data << " (" << root->balanceFactor << ") ";
            preOrderRecursive(root->left);
            preOrderRecursive(root->right);
        }
    }
    void postOrderRecursive(Node<int> *root)
    {
        if (root != nullptr)
        {
            postOrderRecursive(root->left);
            postOrderRecursive(root->right);
            cout << root->data << " (" << root->balanceFactor << ") ";
        }
    }
    int height(Node<int> *current)
    {
        if (current == nullptr)
            return 0;
        else
        {
            int leftHeight = height(current->left);
```

```
            int rightHeight = height(current->right);
            if (leftHeight > rightHeight)
                return (leftHeight + 1);
            else
                return (rightHeight + 1);
        }
    }

    Node<int> *deleteByCopying(Node<int> *current, int key)
    {
        if (key < current->data)
            current->left = deleteByCopying(current->left, key);
        else if (key > current->data)
            current->right = deleteByCopying(current->right, key);
        else
        {
            // node with only one child or no child       if ((current->left
== nullptr) ||
          (current->right == nullptr))
          {
              Node<int> *temp = current->left ? current->left : current-
>right;
              if (temp == nullptr)
              {
                  temp = current;
                  current = nullptr;
              }
              else
                  *current = *temp;
              delete temp;
          }
          else
          {
              // copy inorder predecessor         Node<int> *temp = current-
>left;          while (temp->right != nullptr)           temp = temp-
>right;          current->data = temp->data;         current->right =
deleteByCopying(current->right,                                          temp-
>data);
          }
        }
        if (current == nullptr)
            return current;
        current->height = 1 + max(height(current->left), height(current-
>right));
        current->balanceFactor = getBalanceFactor(current);
```

```
        // R(0) and R(-1)     if (current->balanceFactor < -1
&&          getBalanceFactor(current->left) <= 0)       return
rightRotate(current);
        // R(+1)     if (current->balanceFactor < -1
&&          getBalanceFactor(current->left) > 0)
        {
            current->left = leftRotate(current->left);
            return rightRotate(current);
        }

        // L(0) and L(+1)     if (current->balanceFactor > 1
&&          getBalanceFactor(current->right) >= 0)       return
leftRotate(current);
        // L(-1)     if (current->balanceFactor > 1
&&          getBalanceFactor(current->right) < 0)
        {
            current->right = rightRotate(current->right);
            return leftRotate(current);
        }
        return current;
    }
};
int main(void)
{
    AVLTree tree;
    int choice, data, data2;
    do
    {
        cout << "          MENU      \n"
             << "======================\n"
             << "(1) Insert a Node\n"
             << "(2) Search a Node\n"
             << "(3) Display Traversals\n"
             << "(4) Delete a Node\n"
             << "(0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "\nEnter Node Data: ";
            cin >> data;
            tree.root = tree.insert(data, tree.root);
            break;
        case 2:
            cout << "\nEnter Search Data: ";
            cin >> data;
```

```
            cout << "Search Result: ";
            if (tree.search(tree.root, data))
                cout << "Found";
            else
                cout << "Not Found";
            cout << endl;
            break;
        case 3:
            cout << endl;
            cout << "In-Order Recursive Traversal: ";
            tree.inOrderRecursive(tree.root);
            cout << endl;
            cout << "Pre-Order Recursive Traversal: ";
            tree.preOrderRecursive(tree.root);
            cout << endl;
            cout << "Post-Order Recursive Traversal: ";
            tree.postOrderRecursive(tree.root);
            cout << endl;
            break;
        case 4:
            cout << "\nEnter Node to Delete: ";
            cin >> data;
            tree.root = tree.deleteByCopying(tree.root, data);
            break;
        case 0:
        default:
            break;
        }
        getch();
        clrscr();
    } while (choice != 0);
    return 0;
}
int max(int a, int b)
{
    return (a > b) ? a : b;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
```

```
#endif return;
}
```

Output:

MENU

==========

(1) Insert a Node
(2) Search a Node
(3) Display Traversals
(4) Delete a Node
(0) Exit

Enter Choice: 1

Enter Node Data: 10

Press any key to continue...

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node

(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 10 (0)

Pre-Order Recursive Traversal: 10 (0)

Post-Order Recursive Traversal: 10 (0)

Press any key to continue...

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node

(0) Exit

Enter Choice: 1

Enter Node Data: 20

Press any key to continue...

MENU

==========

(1) Insert a Node
(2) Search a Node
(3) Display Traversals
(4) Delete a Node
(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 10 (1) 20 (0)

Pre-Order Recursive Traversal: 10 (1) 20 (0)

Post-Order Recursive Traversal: 20 (0) 10 (1)

Press any key to continue...

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node

(0) Exit
Enter Choice: 1

Enter Node Data: 30

Press any key to continue...

MENU

==========

(1) Insert a Node
(2) Search a Node
(3) Display Traversals
(4) Delete a Node

(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 10 (0) 20 (0) 30 (0)

Pre-Order Recursive Traversal: 20 (0) 10 (0) 30 (0)

Post-Order Recursive Traversal: 10 (0) 30 (0) 20 (0)

Press any key to continue...

MENU

==========

(1) Insert a Node
(2) Search a Node
(3) Display Traversals
(4) Delete a Node
(0) Exit

Enter Choice: 1

Enter Node Data: 40

Press any key to continue...

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node

(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 10 (0) 20 (1) 30 (1) 40 (0)

Pre-Order Recursive Traversal: 20 (1) 10 (0) 30 (1) 40 (0)

Post-Order Recursive Traversal: 10 (0) 40 (0) 30 (1) 20 (1)

Press any key to continue...

MENU

==========

(1) Insert a Node
(2) Search a Node
(3) Display Traversals
(4) Delete a Node
(0) Exit

Enter Choice: 1

Enter Node Data: 50

Press any key to continue...

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node
(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 10 (0) 20 (1) 30 (0) 40 (0) 50 (0)

Pre-Order Recursive Traversal: 20 (1) 10 (0) 40 (0) 30 (0) 50 (0) Post-Order Recursive Traversal: 10 (0) 30 (0) 50 (0) 40 (0) 20 (1)

Press any key to continue..

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node
(0) Exit

Enter Choice: 1

Enter Node Data: 25

Press any key to continue...

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node

(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 10 (0) 20 (0) 25 (0) 30 (0) 40 (1) 50 (0)

Pre-Order Recursive Traversal: 30 (0) 20 (0) 10 (0) 25 (0) 40 (1) 50 (0)

Post-Order Recursive Traversal: 10 (0) 25 (0) 20 (0) 50 (0) 40 (1) 30 (0)

Press any key to continue...

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node

(0) Exit

Enter Choice: 1

Enter Node Data: 5

Press any key to continue..

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node

(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 5 (0) 10 (-1) 20 (-1) 25 (0) 30 (-
1) 40 (1) 50 (0)
Pre-Order Recursive Traversal: 30 (-1) 20 (-1) 10 (-
1) 5 (0) 25 (0) 40 (1) 50 (0)
Post-Order Recursive Traversal: 5 (0) 10 (-1) 25 (0) 20 (-
1) 50 (0) 40 (1) 30 (-1)

Press any key to continue..

MENU

==========

(1) Insert a Node (2) Search a
Node (3) Display Traversals
(4) Delete a Node
(0) Exit

Enter Choice: 1

Enter Node Data: 1

Press any key to continue...

MENU

==========

(1) Insert a Node (2) Search a Node (3) Display Traversals
(4) Delete a Node
(0) Exit

Enter Choice: 3

In-Order Recursive Traversal: 1 (0) 5 (0) 10 (0) 20 (-1) 25 (0) 30 (-

1) 40 (1) 50 (0)

Pre-Order Recursive Traversal: 30 (-1) 20 (-

1) 5 (0) 1 (0) 10 (0) 25 (0) 40 (1) 50 (0)

Post-Order Recursive Traversal: 1 (0) 10 (0) 5 (0) 25 (0) 20 (-

1) 50 (0) 40 (1) 30 (-1)

Press any key to continue...

15. Write a program to implement a priority queue using heap data structure.

```
#include <iostream>
#define MAX_SIZE 20
using namespace std;
void getch();
void clrscr();
class Heap
{
public:
    int *heap;
    int heapSize;

    Heap(int *&A, int n)
    {
        heap = A;
        heapSize = n;
    }
    int parent(int i)
    {
        return (i - 1) / 2;
    }
    int left(int i)
    {
        return 2 * i + 1;
    }
    int right(int i)
    {
        return 2 * i + 2;
    }
    void maxHeapify(int *&A, int n, int i)
    {
        int temp;
        int largest;
        int l = left(i);
        int r = right(i);
        if (l < n && A[l] > A[i])
        {
            largest = l;
        }
        else
        {
            largest = i;
        }
        if (r < n && A[r] > A[largest])
        {
            largest = r;
        }
        if (largest != i)
```

```
        {
            temp = A[i];
            A[i] = A[largest];
            A[largest] = temp;
            maxHeapify(A, n, largest);
        }
    }
    void buildMaxHeap()
    {
        for (int i = heapSize / 2; i >= 0; i--)
            maxHeapify(heap, heapSize, i);
    }
};
class MaxPriorityQueue
{
public:
    Heap *heap;

    MaxPriorityQueue(int A[], int n)
    {
        heap = new Heap(A, n);
        heap->buildMaxHeap();
    }

    ~MaxPriorityQueue()
    {
        delete heap;
    }
    int size()
    {
        return heap->heapSize;
    }

    void display()
    {
        if (heap->heapSize == 0)
        {
            cerr << "ERROR: Heap Empty";
            return;
        }
        for (int i = 0; i < heap->heapSize; i++)
            cout << heap->heap[i] << " ";
    }
    void heapIncreaseKey(int i, int key)
    {
        int temp;
        if (key < heap->heap[i])
        {
```

```
            cerr << "ERROR: New Key is smaller than Existing Key";
            return;
        }
        heap->heap[i] = key;
        while (i > 0 && heap->heap[heap->parent(i)] < heap->heap[i])
        {
            temp = heap->heap[heap->parent(i)];
            heap->heap[heap->parent(i)] = heap->heap[i];
            heap->heap[i] = temp;
            i = heap->parent(i);
        }
    }
    void maxHeapInsert(int key)
    {
        heap->heapSize++;
        heap->heap[heap->heapSize - 1] = INT8_MIN;
        heapIncreaseKey(heap->heapSize - 1, key);
    }
    int heapMaximum()
    {
        if (heap->heapSize == 0)
        {
            cerr << "ERROR: Heap Empty";
            return -1;
        }
        return heap->heap[0];
    }
    int heapExtractMax()
    {
        if (heap->heapSize < 0)
        {
            cerr << "ERROR: Heap Underflow";
            return -1;
        }
        else if (heap->heapSize == 0)
        {
            cerr << "ERROR: Heap Empty";
            return -1;
        }
        int max = heap->heap[0];
        heap->heap[0] = heap->heap[heap->heapSize];
        heap->maxHeapify(heap->heap, --heap->heapSize, 0);
        return max;
    }
};
int main(void)
{
    int idx, key;
```

```
    int n, choice = 1, A[MAX_SIZE] = {INT8_MAX};
    cout << "Initial Data\n============\n";
    cout << "Enter Number of Nodes: ";
    cin >> n;
    cout << "Enter Keys of the Nodes: ";
    for (int i = 0; i < n; i++)
        cin >> A[i];
    clrscr();

    MaxPriorityQueue queue(A, n);
    do
    {
        cout << "\t   Max Priority Queue\n"
             << "=========================================\n"
             << "  (1) HeapIncreaseKey  (2) MaxHeapInsert\n"
             << "  (3) HeapMaximum      (4) HeapExtractMax\n"
             << "  (5) Display          (0) Exit\n\n";
        cout << "Enter Choice: ";
        cin >> choice;
        cout << endl;
        switch (choice)
        {
        case 1:
            cout << endl;
            cout << "Enter Position: ";
            cin >> idx;
            cout << "Enter New Key: ";
            cin >> key;
            queue.heapIncreaseKey(idx - 1, key);
            cout << endl;
            break;
        case 2:
            cout << endl;
            cout << "Enter Key: ";
            cin >> key;
            queue.maxHeapInsert(key);
            break;
        case 3:
            cout << endl;
            key = queue.heapMaximum();
            if (key != -1)
                cout << "Heap Maximum: " << key << endl;
            break;
        case 4:
            cout << endl;
            key = queue.heapExtractMax();
            if (key != -1)
            {
```

```
                cout << "After Heap Extract Max: ";
                queue.display();
                cout << endl;
            }
            break;
        case 5:
            cout << endl;
            queue.display();
            cout << endl;
            break;
        case 0:
        default:
            break;
        }
        getch();
        clrscr();
    } while (choice != 0);
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
#include <iostream>
#define MAX_SIZE 20
using namespace std;
void getch();
void clrscr();
class Heap
{
public:
    int *heap;
    int heapSize;

    Heap(int *&A, int n)
    {
        heap = A;
        heapSize = n;
    }
```

```
    int parent(int i)
    {
        return (i - 1) / 2;
    }
    int left(int i)
    {
        return 2 * i + 1;
    }
    int right(int i)
    {
        return 2 * i + 2;
    }
    void maxHeapify(int *&A, int n, int i)
    {
        int temp;
        int largest;
        int l = left(i);
        int r = right(i);
        if (l < n && A[l] > A[i])
        {
            largest = l;
        }
        else
        {
            largest = i;
        }
        if (r < n && A[r] > A[largest])
        {
            largest = r;
        }
        if (largest != i)
        {
            temp = A[i];
            A[i] = A[largest];
            A[largest] = temp;
            maxHeapify(A, n, largest);
        }
    }
    void buildMaxHeap()
    {
        for (int i = heapSize / 2; i >= 0; i--)
            maxHeapify(heap, heapSize, i);
    }
};
class MaxPriorityQueue
{
public:
    Heap *heap;
```

```
MaxPriorityQueue(int A[], int n)
{
    heap = new Heap(A, n);
    heap->buildMaxHeap();
}

~MaxPriorityQueue()
{
    delete heap;
}
int size()
{
    return heap->heapSize;
}

void display()
{
    if (heap->heapSize == 0)
    {
        cerr << "ERROR: Heap Empty";
        return;
    }
    for (int i = 0; i < heap->heapSize; i++)
        cout << heap->heap[i] << " ";
}
void heapIncreaseKey(int i, int key)
{
    int temp;
    if (key < heap->heap[i])
    {
        cerr << "ERROR: New Key is smaller than Existing Key";
        return;
    }
    heap->heap[i] = key;
    while (i > 0 && heap->heap[heap->parent(i)] < heap->heap[i])
    {
        temp = heap->heap[heap->parent(i)];
        heap->heap[heap->parent(i)] = heap->heap[i];
        heap->heap[i] = temp;
        i = heap->parent(i);
    }
}
void maxHeapInsert(int key)
{
    heap->heapSize++;
    heap->heap[heap->heapSize - 1] = INT8_MIN;
    heapIncreaseKey(heap->heapSize - 1, key);
```

```
    }
    int heapMaximum()
    {
        if (heap->heapSize == 0)
        {
            cerr << "ERROR: Heap Empty";
            return -1;
        }
        return heap->heap[0];
    }
    int heapExtractMax()
    {
        if (heap->heapSize < 0)
        {
            cerr << "ERROR: Heap Underflow";
            return -1;
        }
        else if (heap->heapSize == 0)
        {
            cerr << "ERROR: Heap Empty";
            return -1;
        }
        int max = heap->heap[0];
        heap->heap[0] = heap->heap[heap->heapSize];
        heap->maxHeapify(heap->heap, --heap->heapSize, 0);
        return max;
    }
};
int main(void)
{
    int idx, key;
    int n, choice = 1, A[MAX_SIZE] = {INT8_MAX};
    cout << "Initial Data\n============\n";
    cout << "Enter Number of Nodes: ";
    cin >> n;
    cout << "Enter Keys of the Nodes: ";
    for (int i = 0; i < n; i++)
        cin >> A[i];
    clrscr();

    MaxPriorityQueue queue(A, n);
    do
    {
        cout << "\t   Max Priority Queue\n"
             << "=========================================\n"
             << "  (1) HeapIncreaseKey  (2) MaxHeapInsert\n"
             << "  (3) HeapMaximum      (4) HeapExtractMax\n"
             << "  (5) Display          (0) Exit\n\n";
```

```
        cout << "Enter Choice: ";
        cin >> choice;
        cout << endl;
        switch (choice)
        {
        case 1:
            cout << endl;
            cout << "Enter Position: ";
            cin >> idx;
            cout << "Enter New Key: ";
            cin >> key;
            queue.heapIncreaseKey(idx - 1, key);
            cout << endl;
            break;
        case 2:
            cout << endl;
            cout << "Enter Key: ";
            cin >> key;
            queue.maxHeapInsert(key);
            break;
        case 3:
            cout << endl;
            key = queue.heapMaximum();
            if (key != -1)
                cout << "Heap Maximum: " << key << endl;
            break;
        case 4:
            cout << endl;
            key = queue.heapExtractMax();
            if (key != -1)
            {
                cout << "After Heap Extract Max: ";
                queue.display();
                cout << endl;
            }
            break;
        case 5:
            cout << endl;
            queue.display();
            cout << endl;
            break;
        case 0:
        default:
            break;
        }
        getch();
        clrscr();
    } while (choice != 0);
```

```
    return 0;
}
void getch()
{
    cout << "\nPress any key to continue...";
    cin.ignore();
    cin.get();
    return;
}
void clrscr()
{
#ifdef _WIN32 system("cls");
#elif __unix__ system("clear");
#endif return;
}
```

Output:

Initial Data

==========

Enter Number of Nodes: 10

Enter Keys of the Nodes: 14 8 10 4 7 9 3 2 1 6

Max Priority Queue

==========

(1) HeapIncreaseKey (2) MaxHeapInsert

(3) HeapMaximum (4) HeapExtractMax

(5) Display (0) Exit

Enter Choice: 5

14 8 10 4 7 9 3 2 1 6

Press any key to continue...

Max Priority Queue

==========

(1) HeapIncreaseKey (2) MaxHeapInsert

(3) HeapMaximum (4) HeapExtractMax

```
 (5) Display         (0) Exit

Enter Choice: 1

Enter Position: 10
Enter New Key: 26

Press any key to continue...

        Max Priority Queue ==========================================
 (1) HeapIncreaseKey  (2) MaxHeapInsert
 (3) HeapMaximum      (4) HeapExtractMax
 (5) Display         (0) Exit

Enter Choice: 5

26 14 10 4 8 9 3 2 1 7
Press any key to continue...
        Max Priority Queue
==========================================
 (1) HeapIncreaseKey  (2) MaxHeapInsert
 (3) HeapMaximum      (4) HeapExtractMax
 (5) Display         (0) Exit

Enter Choice: 2

Enter Key: 40

Press any key to continue...
        Max Priority Queue
==========================================
 (1) HeapIncreaseKey  (2) MaxHeapInsert
 (3) HeapMaximum      (4) HeapExtractMax
 (5) Display         (0) Exit
```

Enter Choice: 5

40 26 10 4 14 9 3 2 1 7 8

Press any key to continue...

Max Priority Queue

==========

(1) HeapIncreaseKey (2) MaxHeapInsert

(3) HeapMaximum (4) HeapExtractMax

(5) Display (0) Exit

Enter Choice: 3

Heap Maximum: 40

Press any key to continue...

Max Priority Queue ==========

(1) HeapIncreaseKey (2) MaxHeapInsert

(3) HeapMaximum (4) HeapExtractMax

(5) Display (0) Exit

Enter Choice: 4

After Heap Extract Max: 26 14 10 4 8 9 3 2 1 7

Press any key to continue...

16. Write a program to evaluate a prefix/postfix expression using stacks.

```
// stack.hpp
#include <iostream>
#define MAX_SIZE 100

using namespace std;
template <class T>
class Stack
{
protected:
    int tos, size;
    T arr[MAX_SIZE];

public:
    Stack(int size = 30)
    {
        this->tos = -1;
        this->size = size;
    }
    bool push(T ele)
    {
        if (this->tos >= (this->size - 1))
        {
            cerr << "ERROR: Stack Overflow\n";
            return false;
        }
        this->arr[++(this->tos)] = ele;
        return true;
    }

    T pop()
    {
        if (this->isEmpty())
        {
            cout << "ERROR: Stack Underflow\n";
            return (T)(NULL);
        }
        return this->arr[(this->tos)--];
    }

    T top()
    {
        if (this->isEmpty())
        {
            cout << "Stack Empty";
            return (T)(NULL);
        }
        return this->arr[this->tos];
```

```
    }
    bool isEmpty()
    {
        return this->tos == -1;
    }
    void clear()
    {
        while (!this->isEmpty())
            this->pop();
    }
};

// main.cpp
#include "stack.hpp"
#include <cstring> #include <string>
#include <cstdlib>
#define MAX_STRLEN 256
class PostfixEvaluator
{
protected:
    Stack<int> stack;

public:
    int evaluate(string &str)
    {
        int val1, val2, temp;
        int size = str.length();
        for (int i = 0; i < size; ++i)
        {
            if (str[i] == ' ')
                continue;
            else if (isdigit(str[i]))
            {
                temp = 0;
                while (isdigit(str[i]))
                    temp = temp * 10 + (int)(str[i++] - '0');
                i--;
                this->stack.push(temp);
            }
            else
            {
                val1 = this->stack.pop();
                val2 = this->stack.pop();
                switch (str[i])
                {
                case '+':
                    this->stack.push(val2 + val1);
                    break;
```

```
                case '-':
                    this->stack.push(val2 - val1);
                    break;
                case '*':
                    this->stack.push(val2 * val1);
                    break;
                case '/':
                    this->stack.push(val2 / val1);
                    break;
                case '%':
                    this->stack.push(val2 % val1);
                    break;
                }
            }
        }
        return this->stack.pop();
    }
};
class PrefixEvaluator
{
private:
    void swap(char &x, char &y)
    {
        char temp = x;
        x = y;
        y = temp;
    }

protected:
    Stack<int> stack;

public:
    int evaluate(string &str)
    {
        string strTemp;
        int val1, val2, temp;
        int size = str.length();
        // parse expression in a reverse fashion     for (int i = size - 1; i
>= 0; i--)
        {
            if (str[i] == ' ')
                continue;
            else if (isdigit(str[i]))
            {
                strTemp = "";
                while (isdigit(str[i]))
                    strTemp.append(string(1, str[i--]));
                i++;
```

```
                for (int i = 0; i < strTemp.length() / 2; i++)
                    swap(strTemp[i], strTemp[strTemp.length() - i - 1]);
                temp = atoi(strTemp.c_str());
                this->stack.push(temp);
            }
            else
            {
                val1 = this->stack.pop();
                val2 = this->stack.pop();
                switch (str[i])
                {
                case '+':
                    this->stack.push(val1 + val2);
                    break;
                case '-':
                    this->stack.push(val1 - val2);
                    break;
                case '*':
                    this->stack.push(val1 * val2);
                    break;
                case '/':
                    this->stack.push(val1 / val2);
                    break;
                case '%':
                    this->stack.push(val1 % val2);
                    break;
                }
            }
        }
        return this->stack.pop();
    }
};
int main(void)
{
    string str;
    PrefixEvaluator preEval;
    PostfixEvaluator postEval;
    cout << "Enter Prefix Expression: ";
    getline(cin, str);
    cout << "Value of Expression: " << preEval.evaluate(str) << endl;
    cout << endl;
    cout << "Enter Postfix Expression: ";
    getline(cin, str);
    cout << "Value of Expression: " << postEval.evaluate(str) << endl;
    return 0;
}
// stack.hpp
#include <iostream>
```

```
#define MAX_SIZE 100

using namespace std;
template <class T>
class Stack
{
protected:
    int tos, size;
    T arr[MAX_SIZE];

public:
    Stack(int size = 30)
    {
        this->tos = -1;
        this->size = size;
    }
    bool push(T ele)
    {
        if (this->tos >= (this->size - 1))
        {
            cerr << "ERROR: Stack Overflow\n";
            return false;
        }
        this->arr[++(this->tos)] = ele;
        return true;
    }

    T pop()
    {
        if (this->isEmpty())
        {
            cout << "ERROR: Stack Underflow\n";
            return (T)(NULL);
        }
        return this->arr[(this->tos)--];
    }

    T top()
    {
        if (this->isEmpty())
        {
            cout << "Stack Empty";
            return (T)(NULL);
        }
        return this->arr[this->tos];
    }
    bool isEmpty()
    {
```

```
        return this->tos == -1;
    }
    void clear()
    {
        while (!this->isEmpty())
            this->pop();
    }
};

// main.cpp
#include "stack.hpp"
#include <cstring> #include <string>
#include <cstdlib>
#define MAX_STRLEN 256
class PostfixEvaluator
{
protected:
    Stack<int> stack;

public:
    int evaluate(string &str)
    {
        int val1, val2, temp;
        int size = str.length();
        for (int i = 0; i < size; ++i)
        {
            if (str[i] == ' ')
                continue;
            else if (isdigit(str[i]))
            {
                temp = 0;
                while (isdigit(str[i]))
                    temp = temp * 10 + (int)(str[i++] - '0');
                i--;
                this->stack.push(temp);
            }
            else
            {
                val1 = this->stack.pop();
                val2 = this->stack.pop();
                switch (str[i])
                {
                case '+':
                    this->stack.push(val2 + val1);
                    break;
                case '-':
                    this->stack.push(val2 - val1);
                    break;
```

```
                case '*':
                    this->stack.push(val2 * val1);
                    break;
                case '/':
                    this->stack.push(val2 / val1);
                    break;
                case '%':
                    this->stack.push(val2 % val1);
                    break;
                }
            }
        }
        return this->stack.pop();
    }
};
class PrefixEvaluator
{
private:
    void swap(char &x, char &y)
    {
        char temp = x;
        x = y;
        y = temp;
    }

protected:
    Stack<int> stack;

public:
    int evaluate(string &str)
    {
        string strTemp;
        int val1, val2, temp;
        int size = str.length();
        // parse expression in a reverse fashion     for (int i = size - 1; i
>= 0; i--)
        {
            if (str[i] == ' ')
                continue;
            else if (isdigit(str[i]))
            {
                strTemp = "";
                while (isdigit(str[i]))
                    strTemp.append(string(1, str[i--]));
                i++;
                for (int i = 0; i < strTemp.length() / 2; i++)
                    swap(strTemp[i], strTemp[strTemp.length() - i - 1]);
                temp = atoi(strTemp.c_str());
```

```
                this->stack.push(temp);
            }
            else
            {
                val1 = this->stack.pop();
                val2 = this->stack.pop();
                switch (str[i])
                {
                case '+':
                    this->stack.push(val1 + val2);
                    break;
                case '-':
                    this->stack.push(val1 - val2);
                    break;
                case '*':
                    this->stack.push(val1 * val2);
                    break;
                case '/':
                    this->stack.push(val1 / val2);
                    break;
                case '%':
                    this->stack.push(val1 % val2);
                    break;
                }
            }
        }
        return this->stack.pop();
    }
};
int main(void)
{
    string str;
    PrefixEvaluator preEval;
    PostfixEvaluator postEval;
    cout << "Enter Prefix Expression: ";
    getline(cin, str);
    cout << "Value of Expression: " << preEval.evaluate(str) << endl;
    cout << endl;
    cout << "Enter Postfix Expression: ";
    getline(cin, str);
    cout << "Value of Expression: " << postEval.evaluate(str) << endl;
    return 0;
}
```

Output:

```
Enter Prefix Expression: - * 5 + 6 2 / 12 4
Value of Expression: 37

Enter Postfix Expression: 5 6 2 + * 12 4 / -
Value of Expression: 37
```