DAY – 13 Assignment and LABs

// Assignment -1 Create a class and use vector for different ' API ' calls.

```
#include<iostream>
#include<algorithm>   // find and sort funct.
#include<vector>        // provides std::vector
using namespace std;

/* class VectorTask {
private:
    vector<int> data;

public:
    // Adding an element to the vector
    void addElement(int value) {
        data.push_back(value);
    }

    // Remove an element from the vector
    void removeElement(int value) {
        auto i = find(data.begin(), data.end(), value);
        if (i != data.end()) {
            data.erase(i);
        } else {
            cout << "Element not found." <<endl;
        }
    }

    // Access an element by index
    int getElement(int index) const {
        if (index >= 0 && index < data.size()) {
            return data[index];
        } else {
           // Using 'cerr' instead of 'cout' helps distinguish between regular output and error
messages.
            cerr << "Index out of bounds." << endl;
            return -1;
        }
    }

    // Print all elements in the vector
    void printElements() const {
        for (int value : data) {
            cout << value << " ";
        }
             cout << endl;
    }

    // Get the size of the vector
    size_t getSize() const {
        return data.size();
    }

    // Clear all elements in the vector
    void clearElements() {
        data.clear();
    }

      // Sort the vector
       void sortElements() {
        // sort(data.end(),data.begin());
        sort(data.begin(),data.end());
    }
};

int main() {
    VectorTask vec;

    // Adding elements
    vec.addElement(12);
    vec.addElement(13);
    vec.addElement(18);
    vec.addElement(11);
```

```
cout << "Elements after adding: ";
vec.printElements();

// Removing an element
vec.removeElement(13);
cout << "Elements after removing 13: ";
vec.printElements();

// Accessing an element by index
cout << "Element at index 1: " << vec.getElement(1) << endl;
cout << "Element at index 2: " << vec.getElement(2) << endl;
cout << "Element at index 3: " << vec.getElement(3) << endl;

// Sorting elements
vec.sortElements();
cout << "Elements after sorting: ";
vec.printElements();

// Getting the size of the vector
cout << "Size of the vector: " << vec.getSize() << endl;

// Clearing all elements of the vector
vec.clearElements();
cout << "Elements after clearing: ";
vec.printElements();
```

```
    return 0;
} */

// **Queue** : Code - 1  { Implement. of push() operation. }
// A queue in data structures is a linear collection of elements that follows
// the FIFO (First In, First Out) principle .

# include<queue>

/* int main() {

    //Empty Queue
    queue<int>myqueue;
    myqueue.push(0); myqueue.push(1); myqueue.push(2); myqueue.push(3);

    // Printing content of queue
    while (!myqueue.empty()) {
        cout << ' ' <<myqueue.front();
           myqueue.pop();
    }
} */
```

// **Queue** : Code - 2 { Implement. of pop() operation }

```
/* int main()
{
    // Empty queue
    queue<int> myqueue;
    myqueue.push(0);
    myqueue.push(1);
    myqueue.push(2);
    myqueue.push(3);

    // now, queue becomes 0,1,2,3
    myqueue.pop();
    myqueue.pop();

    // now, queue becomes 2,3

    // Printing content of queue
     while (!myqueue.empty()) {
        cout << ' ' <<myqueue.front();
           myqueue.pop();
    }
} */
```

```
// **Stack** LIFO Code - 1 on working of STL Stack.


#include<bits/stdc++.h>


/* void showstack(stack <int> s)
{
    while(!s.empty())
    {
        cout<<'\t'<<s.top()<<endl;
        s.pop();
    }
}


int main(){
    stack <int> s;


    s.push(10);
    s.push(30);
    s.push(20);
    s.push(5);
    s.push(1);


    cout<<"The stack is : ";
    showstack(s);
    cout<<"\n s.size():"<<s.size();
    cout<<"\n s.top():"<<s.top();

    cout<<"\n s.pop():";
    s.pop();
    showstack(s);
    return 0;
} */
```

/* Problem 4: Reverse a Queue

Description:

Implement a function to reverse the elements of a queue using a stack. */

```
/* void reverseQueue(queue<int> &q) {
    stack<int> s;

    // Transfer elements from queue to stack
    while (!q.empty()) {
        s.push(q.front());
        q.pop();
    }

    // Transfer elements from stack to queue (reversing the order)
    while (!s.empty()) {
        q.push(s.top());

        s.pop();

    }

}


int main() {

    queue<int> myqueue;

    myqueue.push(10);

    myqueue.push(12);

    myqueue.push(13);

    myqueue.push(14);


    cout << "Original Queue: ";

    while (!myqueue.empty()) {

        cout << myqueue.front() << ' ';

        myqueue.pop();

    }

    cout << endl;


    // Reverse the queue

    reverseQueue(myqueue);


    cout << "Reversed Queue: ";

    while (!myqueue.empty()) {

        cout << myqueue.front() << ' ';
        myqueue.pop();
    }
    cout << endl;

    return 0;
}  */
```

// **<u>Assignment - 2</u>**

/*

1. Implement Queue Using Stacks

2. Maximum Element in Stack

Description:

Design a stack that supports push, pop, and retrieving the maximum element in constant time.

3. Circular Queue Implementation

Description:

Implement a circular queue using an array. The queue should support enqueue, dequeue, and front operations.

4. Sort a Stack

Description:

Write a function to sort a stack such that the smallest items are on the top.*/

//Code - 1

```
/* #include <iostream>
#include <stack>
using namespace std;

class QueueUsingStacks {
    stack<int> s1, s2;

public:
    void enqueue(int x) {
        s1.push(x);
    }

    int dequeue() {
        if (s2.empty()) {
            if (s1.empty()) {
                cout << "Queue is empty" << endl;
                exit(0);
            }
            while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }
        int x = s2.top();
        s2.pop();
        return x;
    }

    bool isEmpty() {
        return s1.empty() && s2.empty();
    }
};

int main() {
    QueueUsingStacks q;
    q.enqueue(1);
    q.enqueue(2);
    q.enqueue(3);
    cout << q.dequeue() << endl; // Output: 1
    cout << q.dequeue() << endl; // Output: 2
    q.enqueue(4);
    cout << q.dequeue() << endl; // Output: 3
    cout << q.dequeue() << endl; // Output: 4
    return 0;
} */
```

// Code - 2

```
/* #include <iostream>
#include <stack>
using namespace std;
class MaxStack {
    stack<int> mainStack;
    stack<int> maxStack;

public:
    void push(int x) {
        mainStack.push(x);
        if (maxStack.empty() || x >= maxStack.top()) {
            maxStack.push(x);
        }
    }

    void pop() {
        if (mainStack.top() == maxStack.top()) {
            maxStack.pop();
        }
        mainStack.pop();
    }
    int top() {
        return mainStack.top();
    }
    int getMax() {
        return maxStack.top();
    }
};
int main() {
    MaxStack s;
    s.push(3);
    s.push(1);
    s.push(5);
    s.push(2);
    cout << "Maximum element: " << s.getMax() << endl; // Output: 5
    s.pop();
    cout << "Maximum element: " << s.getMax() << endl; // Output: 5
    s.pop();
    cout << "Maximum element: " << s.getMax() << endl; // Output: 3
    return 0;
} */
```

// Code - 3

```
/* #include <iostream>
using namespace std;
class CircularQueue {
    int *arr;
    int front, rear, size, capacity;
```

```
public:
    CircularQueue(int c) {
        capacity = c;
        arr = new int[capacity];
        front = size = 0;
        rear = capacity - 1;
    }

    ~CircularQueue() {
        delete[] arr;
    }

    bool isFull() {
        return (size == capacity);
    }

    bool isEmpty() {
        return (size == 0);
    }
    void enqueue(int x) {
        if (isFull()) {
            cout << "Queue is full" << endl;
            return;
        }
        rear = (rear + 1) % capacity;

        arr[rear] = x;

        size++;

    }


    int dequeue() {

        if (isEmpty()) {

            cout << "Queue is empty" << endl;

            return INT_MIN;

        }

        int item = arr[front];

        front = (front + 1) % capacity;

        size--;

        return item;

    }


    int getFront() {

        if (isEmpty()) {

            cout << "Queue is empty" << endl;

            return INT_MIN;

        }

        return arr[front];

    }

};
```

```
int main() {
    CircularQueue q(5);
    q.enqueue(1);
    q.enqueue(2);
    q.enqueue(3);
    q.enqueue(4);
    q.enqueue(5);
    cout << q.dequeue() << endl; // Output: 1
    q.enqueue(6);
    cout << q.getFront() << endl; // Output: 2
    return 0;
} */
```

// Code - 4

```
/* #include <iostream>
#include <stack>
using namespace std;

void sortedInsert(stack<int> &s, int x) {
    if (s.empty() || x > s.top()) {
        s.push(x);
        return ;
    }
    int temp = s.top();
    s.pop();
    sortedInsert(s, x);
    s.push(temp);
}

void sortStack(stack<int> &s) {
    if (!s.empty()) {
        int x = s.top();
        s.pop();
        sortStack(s);
        sortedInsert(s, x);
    }
}

void printStack(stack<int> s) {
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
    cout << endl;
}
int main() {
    stack<int> s;
    s.push(30);
    s.push(20);
    s.push(50);
    s.push(10);
    s.push(40);

    cout << "Original Stack: ";
    printStack(s);

    sortStack(s);

    cout << "Sorted Stack: ";
    printStack(s);
    return 0;
} */
```

// ** List ** : Code – 5 Program on Implementation of List

```
#include <iostream>
#include <list>

int main() {
    // Create a list
    std::list<int> myList;

    // Insert elements at the end
    myList.push_back(10);
    myList.push_back(20);
    myList.push_back(30);

    // Insert elements at the front
    myList.push_front(5);
    myList.push_front(1);

    // Display elements
    std::cout << "List after push_back and push_front: ";
    for (int val : myList) {
        std::cout << val << " ";
    }
    std::cout << std::endl;

    // Insert element at a specific position
    auto it = myList.begin(); // initializes an iterator to the beginning of the list.
    std::advance(it, 2);  // moves the iterator it two positions forward.
    myList.insert(it, 15); // inserts 15 at the position pointed to by it.

    std::cout << "List after insert: ";
    for (int val : myList) {
        std::cout << val << " ";
    }
    std::cout << std::endl;
```

```
// Erase element at a specific position
it = myList.begin(); // The iterator it is reset to the beginning.
std::advance(it, 3); // moves the iterator three positions forward.
myList.erase(it);  // removes the element at the position pointed to by it.

std::cout << "List after erase: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

// Remove elements by value
myList.remove(10);

std::cout << "List after remove: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

// Remove elements based on a condition
myList.remove_if([](int n) { return n < 10; });

std::cout << "List after remove_if: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

// Sorting the list
myList.sort();

std::cout << "List after sort: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

// Reversing the list
myList.reverse();

std::cout << "List after reverse: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

// Merging two lists
std::list<int> otherList = {40, 50, 60};
myList.merge(otherList);
```

```
std::cout << "List after merge: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

// Clearing the list
myList.clear();
std::cout << "List after clear: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

// Checking if the list is empty
if (myList.empty()) {
    std::cout << "List is empty." << std::endl;
}

// Adding elements again
myList.push_back(100);
myList.push_back(200);
myList.push_back(150);
```

```
    // Accessing front and back elements
    std::cout << "Front element: " << myList.front() << std::endl;
    std::cout << "Back element: " << myList.back() << std::endl;
    std::cout << "Now ,my Final List is: ";
    for (int val : myList) {
        std::cout << val << " ";
    }
    std::cout << std::endl;

    return 0;
}
```