CS2810 OOAIA

STL Part 1
Sequential Containers

Example Code Snippets

1. `vector` Example - Dynamic Array Operations

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    vec.push_back(6);  // Adds an element at the end
    vec.pop_back();     // Removes the last element

    std::cout << "Vector elements: ";
    for (int val : vec)
        std::cout << val << " ";

    std::cout << "\nSize: " << vec.size() << ", Capacity: " <<
vec.capacity() << std::endl;

    vec.insert(vec.begin() + 2, 10); // Insert 10 at index 2

    std::cout << "After Insertion: ";
    for (int val : vec)
        std::cout << val << " ";
```

```
    return 0;
}
```

2. `deque` Example - Fast Insertions & Deletions from Both Ends

```
#include <iostream>
#include <deque>

int main() {
    std::deque<int> dq = {10, 20, 30};

    dq.push_front(5);  // Insert at front
    dq.push_back(40);  // Insert at back
    dq.pop_front();    // Remove from front

    std::cout << "Deque elements: ";
    for (int val : dq)
        std::cout << val << " ";

    std::cout << "\nFront: " << dq.front() << ", Back: " <<
dq.back() << std::endl;

    return 0;
}
```

3. `list` Example - Doubly Linked List Operations

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

int main() {
    std::list<int> lst = {1, 2, 3, 4};

    lst.push_front(0);   // Add at front
```

```
    lst.push_back(5);    // Add at back
    lst.pop_front();     // Remove from front

    std::cout << "List elements: ";
    for (int val : lst)
        std::cout << val << " ";

    std::cout << "\nSize: " << lst.size() << std::endl;

    lst.insert(++lst.begin(), 10);  // Insert 10 after first
element

    std::cout << "After Insertion: ";
    for (int val : lst)
        std::cout << val << " ";

    return 0;
}
```

4. `forward_list` Example - Singly Linked List Operations

```
#include <iostream>
#include <forward_list>

int main() {
    std::forward_list<int> flist = {10, 20, 30, 40};

    flist.push_front(5);  // Insert at front

    std::cout << "Forward List elements: ";
    for (int val : flist)
        std::cout << val << " ";

    // Insert after the first element
    auto it = flist.begin();
```

```
    flist.insert_after(it, 15);

    std::cout << "\nAfter insertion: ";
    for (int val : flist)
        std::cout << val << " ";

    flist.pop_front(); // Removes first element

    std::cout << "\nAfter pop_front: ";
    for (int val : flist)
        std::cout << val << " ";

    return 0;
}
```

5. `stack` Example - LIFO (Last In, First Out)

```
#include <iostream>
#include <stack>

int main() {
    std::stack<int> stk;

    stk.push(10);
    stk.push(20);
    stk.push(30);

    std::cout << "Stack top: " << stk.top() << std::endl;

    stk.pop(); // Removes 30 (LIFO)

    std::cout << "After pop, Stack top: " << stk.top() <<
std::endl;
    std::cout << "Stack size: " << stk.size() << std::endl;
```

```
    return 0;
}
```

6. `queue` Example - FIFO (First In, First Out)

```
#include <iostream>
#include <queue>

int main() {
    std::queue<int> q;

    q.push(1);
    q.push(2);
    q.push(3);

    std::cout << "Front element: " << q.front() << std::endl;
    std::cout << "Back element: " << q.back() << std::endl;

    q.pop();  // Removes 1 (FIFO)

    std::cout << "After pop, Front element: " << q.front() <<
std::endl;
    std::cout << "Queue size: " << q.size() << std::endl;

    return 0;
}
```

7. `priority_queue` Example - Min Heap

```
#include <iostream>
#include <queue>
#include <vector>

int main() {
    std::priority_queue<int, std::vector<int>,
```

```
std::greater<int>> pq; // Min Heap

    pq.push(5);
    pq.push(10);
    pq.push(1);

    std::cout << "Top element (Min Heap): " << pq.top() <<
std::endl; // 1 (min element)

    pq.pop(); // Removes 1

    std::cout << "After pop, Top element: " << pq.top() <<
std::endl;

    return 0;
}
```

Summary of Key Operations

Container	Insert	Remove	Access Top	Iteration	Notes
vector	`push_back()`	`pop_back()`	`vec[i]`	`for (int x : vec)`	Fast random access, slow insert/delete except at the end
deque	`push_back()`, `push_front()`	`pop_back()`, `pop_front()`	`front()`, `back()`	`for (int x : dq)`	Fast insert/delete at both ends

list	`push_back()`, `push_front()`	`pop_back()`, `pop_front()`	No direct indexing	`for (int x : lst)`	Doubly linked list, efficient insertions/deletions anywhere
forward_list	`push_front()`, `insert_after()`	`pop_front()`, `erase_after()`	No direct indexing	`for (int x : flist)`	Singly linked list, forward-only iteration, uses less memory
stack	`push()`	`pop()`	`top()`	No iteration	LIFO structure
queue	`push()`	`pop()`	`front()`, `back()`	No iteration	FIFO structure
priority_queue	`push()`	`pop()`	`top()`	No iteration	Max Heap (default), Min Heap (with `std::greater<int>`)

Practice Problems

1. Problem: Given an array `arr[]` of size `N`, for each element find the next greater element (NGE). The NGE for an element is the first greater element to the right in the array. If no greater element exists, print `-1`.

Example:

```
Input:
arr = [4, 5, 2, 10, 8]

Output:
5 10 10 -1 -1
```

Hint: Use a stack along with a `vector` to solve efficiently in **O(N)** time.

2. Problem: The Josephus problem is a famous theoretical problem:

- There are N people standing in a circle.
- Starting from the first person, every K-th person is eliminated.
- This continues until only one person remains.
- Find the position of the last remaining person.

Example:

```
Input:
N = 7, K = 3

Output:
4
```

Hint: Use `std::list<int>` and simulate the elimination using iterators.

3. Problem: Given an array of `N` integers and a window size **`K`**, print the maximum of each subarray of size **`K`.**

Example:

```
Input:
N = 8, K = 3
arr = [1, 3, -1, -3, 5, 3, 6, 7]

Output:
3 3 5 5 6 7
```

Hint: Use **`deque`** to maintain indices of useful elements.

4. Problem: Given a sorted singly linked list (**`forward_list`**), remove duplicate elements.

Example:

```
Input:
1 -> 1 -> 2 -> 3 -> 3 -> 4

Output:
1 -> 2 -> 3 -> 4
```

5. Problem: Given a string containing only **`(`**, **`)`**, **`{`**, **`}`**, **`[`** and **`]`**, determine if it is balanced.

Example:

```
Input: "{[()]}"
Output: Balanced

Input: "{[(])}"
```

```
Output: Not Balanced
```

Hint: Use `stack` to keep track of opening brackets and pop them when encountering a closing bracket.

6. Problem: Given a queue of size `N` (always even), rearrange it such that the first half and second half are interleaved.

Example:

```
Input:
Queue = [1, 2, 3, 4, 5, 6, 7, 8]

Output:
Queue = [1, 5, 2, 6, 3, 7, 4, 8]
```

7. Problem: Given `K` sorted linked lists, merge them into one sorted list.

Example:

```
Input:
List 1: 1 -> 4 -> 5
List 2: 1 -> 3 -> 4
List 3: 2 -> 6

Output:
1 -> 1 -> 2 -> 3 -> 4 -> 4 -> 5 -> 6
```

Hint: Use a min-heap (`priority_queue`) to always pick the smallest element among the heads of all lists.