Submitted by: Muhammad Ghanayem , 207965922

Additional questions:

smart pointers:

Q1:

Pro:

- **Automatic Memory Management**: One of the primary advantages of smart pointers over raw pointers is that they automatically manage memory, reducing the risk of memory leaks and dangling pointers. They automatically deallocate memory when the object they point to is no longer needed which simplifies code maintenance and safety.

Con:

- **Performance Overhead**: Smart pointers introduce a small overhead due to their automatic memory management functionalities and, in the case of std::shared_ptr, the additional overhead of maintaining reference counts. This can impact performance, particularly in highly optimized scenarios where the overhead of these operations is significant compared to the simplicity and speed of raw pointers.

Q2:

Code:

```cpp
void useRawPointers() {
    int iterations = 1000000;
    for (int i = 0; i < iterations; ++i) {
        int* ptr = new int(i);
        *ptr *= 2;
        delete ptr;
    }
}

void useSmartPointers() {
    int iterations = 1000000;
    for (int i = 0; i < iterations; ++i) {
        std::unique_ptr<int> ptr(new int(i));
        *ptr *= 2;
    }
}

int main() {
    auto start = std::chrono::high_resolution_clock::now();
    useRawPointers();
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = end - start;
    std::cout << "Raw Pointers: " << elapsed.count() << " seconds.\n";

    start = std::chrono::high_resolution_clock::now();
    useSmartPointers();
    end = std::chrono::high_resolution_clock::now();
    elapsed = end - start;
    std::cout << "Smart Pointers: " << elapsed.count() << " seconds.\n";

    return 0;
}
```

Output:

```
muhammadgha@DESKTOP-RN6FR8J:~/parallel_programming_hw0$ ./smartvsrawpts
Raw Pointers: 0.0270365 seconds.
Smart Pointers: 0.0889467 seconds.
muhammadgha@DESKTOP-RN6FR8J:~/parallel_programming_hw0$ ./smartvsrawpts
Raw Pointers: 0.0153334 seconds.
Smart Pointers: 0.0824433 seconds.
muhammadgha@DESKTOP-RN6FR8J:~/parallel_programming_hw0$ ./smartvsrawpts
Raw Pointers: 0.0189029 seconds.
Smart Pointers: 0.0865401 seconds.
muhammadgha@DESKTOP-RN6FR8J:~/parallel_programming_hw0$
```

Explanation:

These results align with the expected outcomes. The raw pointers are faster in this case due to their minimal overhead compared to smart pointers, which include additional management functionalities like automatic memory deallocation.

Q3:

Consider a sorting algorithm that not only needs to compare elements based on their values but also needs to count the number of comparisons made. A functor can maintain a counter as an instance variable, updating it upon each invocation of its operator(). a task that neither a function pointer nor a standard lambda function can perform easily without additional external variables or captures.

Q4:

When performing a simple transformation or operation on elements of a container, a lambda function can be more easy to read. For example, using std::transform to square each element in a vector. In this case, using a functor would require defining an extra class or struct outside of the local scope where the operation is needed, which could complicate the code unnecessarily for simple tasks.