---------------------------------------------------------------------------------------------------------

REFLECTION

Name: Hashmeet Singh Saini

Seneca ID : hsaini28

Student No. : 153070214

Date : November 9th, 2022

I have done all the coding by myself and only copied the code

that my professor provided to complete my project milestones.

---------------------------------------------------------------------------------------------------------

In this week’s workshop I learnt how to use the different algorithmic libraries. I learnt how to use numeric library for accumulating data values, create lambda expression on the go to be supplied to different algorithms as predicate, and I also used the algorithm library to sort values. There ere other libraries used mentioned ahead.

1. First and foremost I used the for\_each algorithm from the algorithm library in the display function as shown in line 110 of CrimeStatistics.h. It’s the best option because all I have to do is loop through the different elements of the m\_crimes vector and call the <<operator on each element.

In the same function accumulate from the numeric library has been used twice (line 113 and 116) for the purpose of looping through the same elements and pass them to a lambda function that calculates a total number of cases and total number of resolved cases by accessing the specific attributes. It the best option because I want to calculate sum of a particular attribute of the m\_crime struct and accumulate does exactly that.

In the sort function of the CrimeStatistics class I have used the sort algorithm four times that loops through the m\_crimes vector and pass the elements to a binary function that compares which of the two operands is greater by comparing the corresponding attributes. It’s the best option because it can easily loop through the elements, somp[are them and organize them without the need of writing a lengthy logic.

In the cleanList() function I have used the transform algorithm because I intend to loop through the vector, make changes to it and store it back into the same vector(updating). transform does this and passes every single element to a predicate that contains the logic for updation.

The inCollection intends to return true/ false by deciding if there is even a single crime in the vector that has the matching crime as the argument passed. For this used any\_of algorithm as it loops through the elements, and passes each element to a predicate function that implements the condition. The flag returned by the algorithm is returned by the function.

The getListForProvince() uses two such algorithms. The purpose of the function is to create a new list that only contains the elements that belong to a particular province (user entered). To do this, first I have used the count\_if function that loops through the elements to count the number of elements that satisfy the condition in the predicate which is the province of each crime match the user-entered province. This returns a count which is actually used as the length of new list we need to create. Once this is done, the next thing is to populate the list with elements with the objects that satisfy the same above mentioned condition. To do this I have used the copy\_if algorithm that takes in the beginning and end of the m\_crimes, beginning of the target list and a predicate that every element is subjected to.

1. Algorithms have several advantages over implementing the functionality by ourselves because of various reasons. The most important thing is that all algorithms loop through the elements of an STL container, but when using the algorithms, there is no need to handle the construction of loop because it is entirely managed by itself. Additionally the majority of the logic that resides inside the loop is also covered by algorithms leaving only the smaller parts of the logic to be coded by the user(usually conditions and function through lambda expression). This saves user a lot of coding time, complexity and reduces chances of possible errors. For example to code the sort() function, (assuming bubble sort) I would have to create two loops, a condition for comparison and a logic for swapping the values.

void bubbleSort()

{

int i, j;

for (i = 0; i < n - 1; i++)

for (j = 0; j < n - i - 1; j++)

if (m\_crimes[j].province > m\_crimes[j+1].province)

Crime tempCrime=m\_crime[j];

m\_crime\_j=m\_crime[j+1];

m\_crime[j+1]=tempCrime;

}

When the same code is implemented through the sort algorithm it can be written as:

if (strcmp(field,"Province")==0) {

std::sort(m\_crimes.begin(), m\_crimes.end(), [](const Crime& crime1, const Crime& crime2) {

return crime2.m\_province > crime1.m\_province;

});

}

1. In my opinion the best approach for this would be a menu driven approach where instead of receiving a char string, the sort function receives an integer number between 1 and 4, each standing for a particular field for the sorting logic. This eliminates the probability of receiving incorrectly spelled field names. But if the function is intended to receive the const char string for field, we can use exception handling. This means that if the entered field doesn’t match with any fields we are looking for, an exception would be thrown stating that the sort parameter is invalid. The call for this function on the other hand, would be placed inside a try block and the catch block would let the user know what went wrong by displaying the message.

void CrimeStatistics::sort(const char\* field)

{

if (strcmp(field,"Province")==0) {

std::sort(m\_crimes.begin(), m\_crimes.end(), [](const Crime& crime1, const Crime& crime2) {

return crime2.m\_province > crime1.m\_province;

});

else if (strcmp(field, "Crime")==0) {

std::sort(m\_crimes.begin(), m\_crimes.end(), [](const Crime& crime1, const Crime& crime2) {

return crime2.m\_crime > crime1.m\_crime;

});

}

else if (strcmp(field, "Cases")==0) {

std::sort(m\_crimes.begin(), m\_crimes.end(), [](const Crime& crime1, const Crime& crime2) {

return crime2.m\_numOfCases > crime1.m\_numOfCases;

});

}

else if (strcmp(field, "Resolved")==0) {

std::sort(m\_crimes.begin(), m\_crimes.end(), [](const Crime& crime1, const Crime& crime2) {

return crime2.m\_resolved > crime1.m\_resolved;

});

}

else{

std::string err “invalid sort parameter entered”;

throw err;

}

}