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

Workshop 8 Reflection

Name : Hashmeet Singh Saini

Seneca ID : hsaini28

Student No. : 153070214

Date : November 16th, 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 the concept of smart pointers and their advantages over raw pointers. This workshop was relatively easier because of the smaller concept and less coding requirements.

1. Over OOP244 and OOP345, the toughest part of the workshop is always managing memory to prevent memory leaks. Since every resource allocated dynamically(using new keyword) must be deleted (with delete or delete[]). This greatly reduces the chances of possible memory leaks and makes it easier for the coder to use dynamic memory. In my code I have used smart pointer in the writeSmart function at line 49:

std::unique\_ptr<EmployeeWage> empWage(new EmployeeWage(emp[i].name, sal[j].salary));

the empWage pointer created here destructs itself as soon as the scope ends(end of function).

1. The first difference between a raw and smart pointer is the same as mentioned above, i.e. a smart pointer reduces the chances of memory leaks due to its feature of destroying itself at the end of the scope. This couldn’t be done with raw pointers which had to be manually destroyed.

The second difference is that a smart pointer can’t be copied. This means that it must be passed as reference when passing to a function. That is the same reason why we had to create two different += operators in the GeneratingList.h file. One overload dealing with raw pointers looks like this and doesn’t deal with references. Only the pointer itself is passed.

void operator+=(T\* object) {

list.push\_back(\*object);

}

On the other hand the overload dealing with a smart pointer receives its pointer as reference (since copy can’t be created).

void operator+=(T& object) {

list.push\_back(object);

}