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

Workshop 9 Reflection

Name : Hashmeet Singh Saini

Seneca ID : hsaini28

Student No. : 153070214

Date : November 21st, 2022

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

that my professor provided to complete my project milestones.

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

This week’s workshop was interesting as I learned concepts of reading and writing into a binary file. In addition to that I learnt how to divide a complex task into threads using packaged\_tasks and futures so every process occurred in nearly 4th of its original time. I also learnt how to use bind from the functional library to bind some arguments to a function along with the use of placeholders.

1. Even though in this workshop, the original content is stored inside map.txt file, the aim is to store data in the binary file (map.dat) because it is much faster to access. Since computers work in binary, no conversion takes place when writing to a binary file. For the same reason, binary files also take smaller space, so for example an integer number takes 4 bytes of space in a binary file but in case of a text file every integer digit takes up to 1 byte. This also makes binary files a lot more portable and ideal for use in this workshop. An additional benefit is that it doesn’t lose precision while writing as well(because there is no conversion).
2. It is important to bind a function to its arguments along with placeholders for enabling us to manipulate number of values and their positions, to force the function work only in a certain way. It also relieves us from the hassle to pass a certain argument to a function repeatedly. In my code I used to create the task for the thread of calling the digForTreasure function as seen in line 160:

auto task = std::bind(digForTreasure,\_1,mark );

The original function takes two arguments a string reference and a char mark. By using bind, I have created a function called task where the first argument will be any string (placeholder), but the mark will be fixed for all calls to this task. This is how bind became useful to create a threading task in this workshop.

1. Multi-threading is great to increase the overall speed of a program as it divides an overall long task into smaller bits which can simultaneously run to make the process lag-free, fast and responsive. It also has minimized impact on the resources of the system, consuming less processing power and making the program lighter. Since my program is supposed to iterate through every single character of a string one at-a-time, it only makes sense to divide into multiple threads so it consumes less time for longer strings. To achieve this, I created 5 different points on the map array to divide it into 4 chunks. I also created a new function called splitter that takes any two of the above mentioned points and concats the strings in the map array between these points. I use this function to create 4 clean chunks of data I can supply to the threads. Then I creates 4 threads using packaged\_task for the task function, associate every single package with its future using get\_future and then called the package with its own data chunk. Since every thread returns a number(number treasures found), in the end I receive this number from every package’s future and sum to receive the final answer.
2. A packaged\_task and future are related to each other through shared memory. Every packaged\_task shares a memory location with a future, so whatever the formar return is stored in that location and can be accessed with the get() function of the future associated with that package\_task. This helps to add task-based parallelism. A good thing about using packaged\_task is that it reduces the length of code required to create a multi -threaded program as setting up threads, promises and joining the threads in the end all happen internally within the package. But this also means lesser control over the threads. In my code I have created 4 packaged\_tasks and their corresponding futures like std::packaged\_task<size\_t(std::string)>pt1(task);

auto f1 = pt1.get\_future();

As you can see that the wrapper takes in something that looks like function signature, the size\_t here is what is returned and stored in the shared memory. we associate the future f1 to pt1 by using its get\_future() function. We can access the shared memory by using the get() function of the future.