Seneca | SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

FINAL EXAMINATION - Part A

TERM	COURSE NAME	COURSE CODE	VERSION
Summer 2020	Object-Oriented Software Development using C++	OOP345	A

PROFESSORS: Hossein Pourmodheji, Mufleh Al-Shatnawi, Asam Gulaid

SPECIAL INSTRUCTIONS:

Test Component	Availability	Time Limit
Part A and B	Aug 11th, 8 AM to Aug 12th, 8 AM	No Time Limit
Part C	Aug 11th, 8 AM to Aug 12th, 8 AM	3 hours
Part D	Aug 12th, 8 AM to Aug 13th, 8 AM	3 hours
Part E	Aug 13th, 8 AM to Aug 14th, 8 AM	4 hours

- For Part C, D and E, 80% of the mark is dedicated to the coding part, and 20% is dedicated to the explanation.

SENECA'S ACADEMIC HONESTY POLICY

As a Seneca student, you must conduct yourself in an honest and trustworthy manner in all aspects of your academic career. A dishonest attempt to obtain an academic advantage is considered an offense, and will not be tolerated by the College.

APPROVED BY:
Kathy Dumanski, Chair, School of SDDS

Part A: Concept Questions (20%)

1. What is a **race condition** in the context of multi-threading?

   Multi-threading is the approach towards programming where the program uses multiple threads to do the job and processing of the instructions thus making it more efficient and faster. The race condition is a condition that adversely affects the multi-threading and multi-processing of the programs. Race conditions occur in the program or multi-threaded environment where the multiple threads share the data and then they try to change it by accessing the data at the same time.

2. Explain how **future** facilitates transfer values between tasks through **shared states**?

   A future retrieves a value from a shared state in which a provider has stored the value. Each provider-future pair establishes one synchronization point for two tasks that execute concurrently. The provider creates an empty shared state on initialization. Once the provider has supplied value to that shared state, that state is ready for access by the future associated with that provider

3. Explain the concept of the **ragged arrays**. Support your answer with an example.

   A ragged array is also known as a jagged array. It is an array of arrays of which the member arrays can be of various sizes and producing rows of ragged edges when envisioned as output. Interestingly , two-dimensional arrays are always rectangular so jagged arrays should not be confused with multidimensional arrays, but the former is often used to emulate the latter.

   example:

```
int[][]r;
r = new int[3][]; // creates 3 rows
r[0] = new int[4]; // create 4 columns for row 0
r[1] = new int[6]; // create 6 columns for row 1
```

4. What is the benefit of using a file object in a **binary mode**? Support your answer with an example.

   So far, we have learned file operations on text files, what if the files are binary (such as .exe file). The above programs will not work for binary files, however there is a minor change in handling Binary files. The main difference is the file name & modes. Let's understand this with the help of an example. Let's say I have two binary files bin1.exe & bin2.exe – I want to copy content of bin1.exe to bin2.exe:

```
#include <stdio.h>
int main()
{
   char ch;

   /* Pointers for both binary files*/
   FILE *fpbr, *fpbw;

   /* Open for bin1.exe file in rb mode */
   fpbr = fopen("bin1.exe", "rb");

   /* test logic for successful open*/
```

```
    if (fpbr == NULL)
    {
       puts("Input Binary file is having issues while opening");
    }

    /* Opening file bin2.exe in “wb” mode for writing*/
    fpbw= fopen("bin2.exe", "wb");

    /* Ensure bin2.exe opened successfully*/
    if (fpbw == NULL)
    {
      puts("Output binary file is having issues while opening");
    }

    /*Read & Write Logic for binary files*/
    while(1)
    {
       ch = fgetc(fpbr);
       if (ch==EOF)
           break;
        else
           fputc(ch, fpbw);
     }

     /* Closing both the binary files */
     fclose(fpbr);
     fclose(fpbw);

     return 0;
}
```

5. The algorithm category of the **STL** consists of three distinct libraries. Name the three libraries and describe the functionality of each one in a short phrase.

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators.

Algorithms: The header calculation characterizes an assortment of capacities particularly intended to be utilized on scopes of elements. They follow up on containers and give intends to different activities for the contents of the containers

Containers: Containers or container classes store objects and data. There are in total seven standard “first-class” container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors. There are four types of containers: 1. Sequence Containers 2. Container Adaptors 3.Associative Containers 4.Unordered Associative Containers.

Functions: The STL incorporates classes that over-burden the capacity call operator. Examples of such classes are called work objects or functors. Functors permit the working of the related capacity to be modified with the assistance of parameters. to be passed

Iterators: As the name proposes, iterators are utilized for working upon an arrangement of qualities. They are the significant element that permit simplification in STL.

6. Explain the benefit of using **smart pointers** in comparison to the raw pointers.

Indeed, even before bouncing into the advantages of Smart Pointers, I would suggest understanding for what reason are they even required. Above all else, smart pointers were never intended to supplant the utilization of every single Raw pointer. They were planned with an extremely explicit reason: Memory the board for example cleaning up/liberating dynamically allocated memory. One difference is that raw pointer adaptation doesn't work. You allot a pointer to a temporary variable, which dies when constructor is finished. Which features a significant distinction among raw and smart pointers - smart pointers keep you from the greater part of such mix-ups.

7. Explain the difference between **std::unique_ptr** and **std::shared_ptr**.

When using unique_ptr, there can be at most one unique_ptr pointing at any one resource. When that unique_ptr is destroyed, the resource is automatically reclaimed. Because there can only be one unique_ptr to any resource, any attempt to make a copy of a unique_ptr will cause a compile-time error. shared_ptr, on the other hand, allows for multiple pointers to point at a given resource. When the very last shared_ptr to a resource is destroyed, the resource will be deallocated.
Use unique_ptr when you want a single pointer to an object that will be reclaimed when that single pointer is destroyed. While use shared_ptr when you want multiple pointers to the same resource.

8. In C++, a thread object can be joinable or not joinable. When is a thread object **not joinable**?

Thread::joinable is an in-built capacity in C++ std::thread. It is an onlooker work which implies it watches a state and afterward restores the comparing output and checks whether the string object is joinable or not.

A string object is supposed to be joinable on the off chance that it distinguishes/speak to a functioning string of execution.

A thread is not joinable if:

1. **It was default-constructed.**
2. **If either of its member join or detach has been called.**
3. **It has been moved elsewhere**

9. What does the acronym **RAII** stand for? Name an STL class that implements RAII in C++?

RAII stand for Resource Acquisition Is Initialization. The colloquialism highlights epitomizing assets (pieces of memory, open documents, opened mutexes, and so on) in the neighborhood, programmed protests and having the destructor of that article delivering the asset when the item is devastated toward the finish of the extension it has a place with:

{

raii obj(acquire_resource());

/...

}/obj's dtor will call release_resource()

Obviously, objects aren't generally neighborhood, programmed objects. They could be individuals from a class, as well:

class something {

private:

raii obj_;/will live amazing examples of the class

/...

};

10. Discuss the advantages of using **Function-Like Macros** in C++.

A large scale is a name given to a square of the code which can be subbed where the code scrap is to be utilized for more than once.

- The speed of the execution of the program is the significant favourable position of utilizing a large scale.

- It spares a great deal of time that is spent by the compiler for conjuring/calling the capacities.

- It decreases the length of the program.
