Programming Assignment #0
Due Date: Jan 30th 2024
Smart Pointers

Description:
For this programming assignment, you will implement a **Smart Pointer.** Oftentimes, we allocate pointers to objects on the heap that we forget to deallocate and free up memory for. On certain occasions, forgetting to free up heap memory can lead to program crashes. In order to avoid such a situation, it is important that we devise a mechanism by which we are guaranteed to deallocate the heap memory that we allocate in our program. Languages like Java and C# have inbuilt garbage collection mechanisms that deallocate memory so that the programmer does not have to concern themselves with memory leaks. In C++, we will use Smart Pointers to achieve the same objective. A Smart Pointer is a wrapper class around a pointer with operators like * and → overloaded. It is a stack allocated object that immediately frees up the heap memory it holds after it goes out of scope.

starter.zip (Starter Code):
The following is a brief description of the starter code provided to you:

- **main.cpp**
  - **SmartPointer** class: is a **non-functional** class that accepts a pointer (of templated type) as a parameter in its constructor. This SmartPointer class implements overloaded * and → operators.
  - **void memoryLeakCheck()** function: This function uses the SmartPointer class you have implemented to instantiate a few SmartPointer objects and is called in main().

Contract (TASK):
The current SmartPointer class provided to you does not manage memory correctly and as a result the program contains memory leaks. **Your task is to fix the SmartPointer class so that there are no memory leaks. This assignment will be graded based on whether or not your code produces memory leaks.**

Deliverables and Submissions:

In a zip folder named using the format **<FirstName>-<LastName>-<UIN>-<PA0>.zip** you must submit the following file(s) to Canvas:

- **main.cpp (**containing your functional implementation of the SmartPointer)

Grading:
This assignment is worth 10 points. If your program has no memory leaks, you will receive 10 points. If there are memory leaks, you will receive no points.

The grade in this assignment serves as an extra credit opportunity to be utilized as an adder worth 1% of the final grade in case you are at the borderline of the next grade.

testInfrastructure.zip (Test Methodology):
In order to familiarize you with the auto-grading environment on our end we are providing you with an automated test infrastructure that can be used to check the correctness of your code. The testing infrastructure is constructed as follows:

- **smart_test.cpp**: This file contains the memoryLeakCheck function which also instantiates SmartPointer objects that you have implemented in main.cpp
    - **Note: The provided memoryLeakCheck function in main.cpp is for you to write your own tests and play around with the SmartPointer object to ensure that things are working as expected. When auto-grading your code, the smart_test.cpp script will overwrite the main() and memoryLeakCheck() functions in main.cpp that you have implemented in the following manner -**
        - You will notice that there is a **#ifndef TEST** guard around the memoryLeakCheck and main functions in the given main.cpp starter file. You will also have noticed that there is a **#define TEST** guard present on top of smart_test.cpp. When smart_test.cpp includes main.cpp, the memoryLeak() and main() functions as part of main.cpp will not be included in the smart_test.cpp. This is because TEST has already been defined, and therefore the compiler does not include the code present within the #ifndef TEST guard thereby preventing multiple redefinition of these functions. This allows the smart_test.cpp file to execute its own memoryLeakCheck function without any issues.
- **test.py**: This file automatically compiles the smart_test.cpp file and runs valgrind over the compiled executable to check whether there are memory leaks.

To check whether your code has any memory leaks, you can simply include these two files into your directory (containing main.cpp) and check the result of running the test.py script using the command: **python3 test.py**
Alternatively, you can also use the makefile provided to compile the program and then run valgrind on the command line on the resulting executable using the command: **valgrind –leak-check=full ./<executable name>**

[Advisory]:

1. You are encouraged to instantiate your own SmartPointer objects in main.cpp and test for memory leaks by yourself before you use the testing script. Also note that valgrind may not be available on Mac M1/M2 which might result in the test script not working on these machines - you may use Leaks on Mac M1/M2s as an alternative to valgrind.
2. **DO NOT** submit the smart_test.cpp or the test.py files associated with the testing infrastructure. These are solely provided to you as an aid and with the objective to familiarize you with the auto-grading

environment on our end. You must strictly adhere to the submission format mentioned above.