COMP 322
Winter Semester 2019

INSTRUCTOR: DR. CHAD ZAMMAR
chad.zammar@mcgill.ca

Assignment 3: How Smart is your Pointer?

Due date: April 1st 2019, 11:59 PM.

Before you start:

- Collaboration and research for similar problems on the internet are recommended. However, your submission should reflect individual work and personal effort.
- Some of the topics may not be covered in class due to our limited time. You are encouraged to find answers online. You can also reach to your instructor or TAs for guidance.
- Please do submit your assignment before the due date to avoid penalties or worse risking your assignment being rejected.
- Submit only .cpp and .h files. If some questions require you to develop and explain, you can embed your text in a C++ comment style.
- Make sure your code is clear and readable. Readability of your code as well as the quality of your comments will be graded.
- No submission by email. Submit your work to mycourses.
- Be happy when working on your assignment, because a happy software developer has more inspiration than a sad one :).

Smart Pointer Implementation

Quoting Wikipedia: *In computer science, a smart pointer is an abstract data type that simulates a pointer while providing added features, such as automatic memory management or bounds checking. Such features are intended to reduce bugs caused by the misuse of pointers, while retaining efficiency. Smart pointers typically keep track of the memory they point to, and may also be used to manage other resources, such as network connections and file handles. Smart pointers originated in the programming language C++.*

In this last assignment, we will try to tackle the implementation of a "smart pointer" in its simplest form.

Please note that for all the questions, you are responsible of providing the appropriate test code in your main function. The output of your program should reflect clearly the answers for each question. Failing to do so will be penalized.

Also note that you need to double check and confirm that your code compiles before submitting to myCourses. The best way to minimize compilation issues on our side, is to provide only one cpp file containing all the code.

Question 1 (15 pts)

Research smart pointers provided by the standard library (since C++11). List them and explain the difference between them.

Question 2 (15 pts)

Design and implement your own smart pointer class called **SmartPointer** that will automatically delete the memory for built-in integer type that is allocated through it.

SmartPointer class should be used this way in your main() function:

SmartPointer sPointer(11);

Here we are passing 11 as an argument to the constructor of SmartPointer class. The constructor will allocate an integer variable and will initialize it to 11.

To get the value of your variable, you should provide and implement a method called getValue(). You use it this way:

cout << sPointer.getValue(); // prints 11

You should also be able to differ the initialization of your allocated int variable to a later moment. For example the following code should be supported as well:

SmartPointer sPointer;

sPointer.setValue(133);

cout << sPointer.getValue(); // prints 133

Please note that for this question, SmartPointer class will only handle allocation of integer type variables. Allocating arrays is not supported to keep the class simple. No reference counting is required.

The default constructor should allocate and initialize the allocated integer to zero. The destructor should take care of deleting the allocated memory.

Question 3 (15 pts)

Exceptions are problems encountered during the execution of the program. For example, if the system runs out of memory during a dynamic allocation, it will raise an exception. These exceptions should be treated whenever they are raised in order to avoid the program from yielding undefined behavior or even worse, crashing.

Add appropriate exception handling for the SmartPointer class to treat the following 2 cases:

1. When allocating a new variable, the code should throw an exception if the system runs out of memory. This exception should be treated and a message should be

displayed to the screen warning the user that the variable was not being allocated. I am aware that this cannot really be tested out of the box, so I'm not expecting your main to provide a test case for this situation. However, I want to see that your code "catch" the appropriate exception that the operating system throws when it runs out of memory.

2. Let's assume that we are only interested in positive numbers. The code should throw an exception whenever a user tries to assign a negative number to any variable that was being allocated through SmartPointer class. This exception should be treated and a message should be displayed to the screen warning the user that the class does not handle negative numbers.

Question 4 (15 pts)

Make the class SmartPointer generic using templates to be able to allocate any built-in type through it (int, float, double). Let's ignore char and string types for simplicity.

Use case:

SmartPointer<float> sPointer;

sPointer.setValue(13.31);

cout << sPointer.getValue();

Question 5 (20 pts)

Overload the operators +, - and * to be able to perform arithmetic operations on variables allocated through SmartPointer class. Don't use member methods for the overloaded functions, use friend functions instead. These overloaded functions should be generic as well (using templates).

Use case:

SmartPointer<float> sPointer1;

sPointer1.setValue(1.5);

SmartPointer<float> sPointer2;

sPointer2.setValue(2.5);

SmartPointer<float> sPointer3 = sPointer1 + sPointer2;

cout << sPointer3.getValue() << endl; // prints 4

Question 6 (20 pts)

How can you adapt the class SmartPointer to be able to handle allocation of arrays? Please provide a full implementation. Feel free to choose any suitable signature for your class, including the way it should be used.

Note that the same class should now handle both cases, the array case and the single variable case. You should not provide two separate implementations to handle these 2 cases.

You should also add the use case in the main function.