MCIS-4130
Homework Assignment 5

Purpose:

This exercise provides opportunities to explore both the user and supplier sides of class programming—that is, both the use of existing classes and writing your own.

What to hand in:

For part A, hand in your test program for the Date class; include your header file (in standard form), your implementation file(s), and test run output of the program.

For part B, hand in your test program for the Trace class; include your header file, your implementation files, and test run output of the program.

Note that each part requires its own main program.

Problems:

A. Make *Dat* a class *e*

Convert your *Date* struct into a class. Be sure to make the proper parts *public* and *private.* Consider what the proper names of your interfaces are, and whether you need any special constructors or special versions of any of the built-in operations.

Provide overloaded versions of << and >> for input and output; these may not be either friend (allowed to break encapsulation) or member functions . See homework problem 2, and the sample solution to see how Print() and Read() can be used to implement this..

Add appropriate operator overloading to your *Date* class. You should overload at least the following operators:

`+` and `-` (`int` argument), `>`, `<`, `>=`, `<=`, `!=`, `==`.

Optional: operator - (`const Date&` argument).

Make functions and parameters *const* where this is appropriate.

Do not remove any existing interfaces from the Date class!

Modify your test program to test the *Date* class. Be sure to test all interfaces, including the new ones!

Note: The Date class will be used in HW 6 & 7!

B. A Trace class

Write a *Trace* class with the following responsibilities

1 The constructor requires a user-provided string to associate with each instance

1 Upon construction, the instance prints *"Entered"* and its string

1 Upon destruction, the instance prints *"Exiting"* and its string (the one provided in the constructor).

Build a main program which creates instances of *Trace* at various parts of your program.

Be sure to include at least one example of an instance in global scope (extern), file scope (static), local (static) scope, local (auto) scope, nested local scope (inside a block or function), dynamically allocated ("new"ed) with and without deletion.

See if the sequence of messages matches your expectation.

Notes:

For part A, use existing functions to implement the new forms whereever possible. Did your test program change more, less, or the same as you expected?

For part B, you may find more reliable results with `printf()` than using the object `cout` (and the overloaded `operator <<).` Why? (Hint: How is `cout` created?)

Evaluation criteria:

Programs should be sure to cover all the possibilities requested.

For user interface, be sure that premature EOF or invalid input does not cause the program to perform improperly or go into an infinite loop.

For class definitions, be sure to separate the definition into header and body parts. Please do not provide inline function definitions in the header. Ensure that class functions have the proper visibility (*public* or *private*), and that proper constructors, destructors, and other special functions are provided where necessary.

15	Part A: Header File
15	Part A: Implementation files
20	Part A: Test files and output ***Note: If your program doesn't compile, hand in the error listing as the results!***
15	Part B: Header File
15	Part B: Implementation files
20	Part B: Test files and output ***Note: If your program doesn't compile, hand in the error listing as the results!***