CECS 282 - Lab 4

Reading

Reading from *C++ How to Program*:

1. Skim Chapter 9 for review of declaring classes and objects
2. Chapter 10.1, 10.3, 10.4, 10.5, 10.6, 10.7 (operator overloading)

Assignment

There are two ways to get credit for this Lab. Either **demo your Project 1 code and pass the demo**; or **complete the questions below**. If you have not finished Project 1, you can also demo your `PrintBoard` and `InBounds` functions to get a 1-week extension on the due date.

1. Suppose a C++ class has a field (instance variable) of type `string`. You want to add a method to the class to access that string by returning it. You have four options for how to return the string; for each option, **justify** why you *might* choose to return in that way. One example is given:
   (a) `string *`
   Answer: We might return a pointer if we want someone else to be able to mutate our field, and they should know the field might sometimes be null.
   (b) `string &`
   (c) `string`
   (d) `const string &`
2. Answer True or False for each of these questions about operator overloading. Give a **one-sentence explanation** for each ***false*** answer.
   (a) The precedence (order of operations priority) of an operator cannot be changed by overloading.
   (b) If you overload `operator==`, the compiler automatically knows how to evaluate the `!=` operator.
   (c) An operator can be a **member operator** only if the left-hand side operand is of a different type than the class it is defined in.
   (d) You can modify the behavior of an operator that operates solely on primitive types, e.g., you can change the behavior of `+` when used with `int`s.
   (e) An arithmetic operator like `operator+` **must** return an object of the same type as the parameters.
   (f) You *should* overload every operator for every class you write.
3. Suppose you have two variables `a` and `b` of some arbitrary class `C`, and that `C` provides correct overloads for operators `<` and `==`. Using only those two operators, and other boolean logic operators (and, or, not), show how to accomplish the equivalent of these conditional expressions. One example is given.
   (a) `a <= b`
   Ans: `(a < b || a == b)`
   (b) `a != b`
   (c) `a > b`
   (d) `a >= b`
4. Practice writing C++ classes by creating the following class. Separate the declaration into a .h file and the implementation of any non-inline methods into a .cpp file:

   The class is named `BankAccount` and represents an account of money at an arbitrary bank. Satisfy these requirements:
   (a) Two fields: an account number (integer) and a balance (the amount of money in the account; a floating-point number).
   (b) One constructor, taking parameters for the accout number and balance of the account, and initializing the object appropriately.
   (c) One accessor method for the account balance, `GetBalance()`.

(d) A method `GetInterestRate()`, which returns the interest rate of the account. Interest rate is equal to 0.001 if the account balance is less than 10,000; equal to 0.003 if balance is between 10,000 and 100,000 (inclusive); and equal to 0.005 if greater than 100,000.

(e) A method `ApplyInterest()`, which multiplies the current balance by the account's interest rate, and then increases the balance by that product.

(f) A member operator `operator<`, which takes another `BankAccount` as a parameter and returns whether the context `BankAccount` (`this`) has a smaller balance than the parameter.

You **must** mark methods as `const` when appropriate, and must justify whether each parameter to your methods should be a copy, a reference, a pointer, or a const reference.

How to Get Credit

See above.