COP3330 Test 1 Review

1. Intro to Object-Oriented Programming
    - Machine languages – this is what is necessary for the computer. Hard for people to read
    - Assembly languages – these are symbolic translations of machine code, easier for people to read
    - Object – an encapsulation of data and functions that act upon that data. Three aspects of an object
        - Name – variable name
        - Attributes (member data) – the data that describes the what the object is
        - Behavior (member functions) – behavior aspects of the object (functions that describe what the object does
    - Class – a blueprint for objects. A class is a user-defined type that describes what a certain type of object will look like. A class description consists of a declaration and a definition (usually split into separate files i.e. header file)
    - An object is a single instance of a class.
    - You can create many objects from the same class type.
2. Classes and Objects
    - Declare – A declaration gives an interface. A variable declaration gives the type. A function declaration tells how to use it, without bothering with how it works. A class declaration shows what an object will look like and what its available functions are.
    - Define – A definition usually consists of the implementation details. This part doesn't need to be seen by the user of the interface. A function definition is the code that makes a function work (function body). A class definition consists of definitions of its members.
    - Use – The use of an item, through its interface. The user of an executable program uses the graphic interface, keyboard, and mouse. The user of a function is a programmer, who makes calls to the function (without needing to know the implementation details). The user of a class is also a programmer, who uses the class by creating objects and calling the available functions for those objects.
    - Public – Can be accessed from inside or outside of the object. Essentially the interface of the object. The user of an object is some other portion of code.
    - Private – Can only be used by the object itself. A general rule: Make member data private to protect it. If there are functions that do not need to be part of the interface, then these can be private, as well.
    - Reasons for hiding data:
        - Makes interface simpler for user.
        - Principle of least privilege
        - More secure. Less chance of misuse
        - Class implementation easy to change without affecting other modules that use it.
    - To declare a class:

```
class <ClassName>
{
public:
        (public member data and functions go here)
private:
        (private member data and functions go here)
}; //Do not forget the semicolon at the end
```

3. Constructor – A special member function of a class whose purpose is usually to initialize the members of an object. You do not explicitly call the constructor function; it is automatically called when you declare an object.
    - A constructor is easy to recognize because:
        - It has the same name as the class
        - It has no return type
            - **Example:**
                - Circles circ1;
        - This creates an object named circ1 and causes the Circles constructor function to be run for the circ1 object.
    - The scope resolution operator is used for specifying which class a member belongs to when we define it separately from its declaration inside the class. The format is: className::memberName
    - To class a member function of an existing object, we use the dot-operator. The syntax format is: objectName.memberName
    - memberName can be a member function call or a member data item
        - **Examples:**
            - f1.Show(); //f1 is the object, Show() is the member function
            - cout << f2.Evaluate(); //f2 is the object, Evaluate() is the member function
4. Constructor with parameters – To utilize a constructor with parameters, we can pass arguments in when the object is declared.
    - Fraction(); //default constructor
    - Fraction(int n, int d = 1); //constructor with parameters
5. Default constructor – A constructor with no parameters. When we declare objects and do not pass in any parameters, the default constructor will be used.
    - **Examples**:
    - Fraction f1, f2; // fraction objects that will use the default constructor
    - **Examples with parameters:**

```
Fraction f1(2,3); // passes in 2 and 3 as parameters
Int x = 4, y = 8;
Fraction f2(x,y); // passes in the values stored in x and y
```

- Default parameters can be used with constructors as well. The constructor Fraction(int n, int d = 1); has a default parameter of d = 1, so if you do: Fraction f3(6); // the first parameter initializes to 6 and the second one initializes to 1.

6. Compilation and Debugging
   - Multiple file projects – Most full programs are not contained in a single file.
     - A class module normally consists of:
       i. A header file – Contains the declaration of the class (without implementation details). Usually in the format *filename.h*
       ii. An implementation file – Contains implementations of the class members. Usually in the format *filename.cpp*
       iii. **Examples**:
         a. circle.h //header file for a class called Circle
         b. circle.cpp //implementation file for Circle class
       iv. Filenames do not have to be the same as the class name, but it helps to identify the contents or purpose of a file.
       v. #include the header files NOT the cpp files
   - Compilation – Compile stage and linking stage
     - Compile Stage
       i. Syntax checked for correctness.
       ii. Variables and function calls checked to unsure that correct declarations were made and that they match.
       iii. Translation into object code.
     - Linking Stage
       i. Links the object code into an executable program
       ii. May involve one of more object code files.
       iii. The linking stage is the time when function calls are matched up with their definitions, and the compiler checks to make sure it has one, and only one, definition for every function that is called.
       iv. The end result of linking is usually an executable program.
       v. **Examples of linking**:
       vi. g++ -c frac.cpp main.cpp //translates into object code
       vii. g++ -o sample frac.o main.o //links object code into executable called sample, to run this you would simply type in the executable's name
7. Debugging – Compile stage errors, linking stage errors, and run-time errors
   - Compile stage errors – These are errors that will be reported by the compiler, which usually provides a filename and line number indicating where it ran into trouble.
     - Always start at the top of the error list.
     - Compile as you go.
   - Linking stage errors – These errors do not usually contain line numbers, since the linker works on object code, not the original source code.
     - Usually caused by agreement between definitions and calls
     - Usually specify a symbol which resembles a function or variable name.