OBJECT ORIENTED PROGRAMMING

Lab 5

OBJECT ORIENTED PROGRAMMING WITH C++

- Chapter Objectives
  - Overloading constructor functions
  - Creating and using a copy constructor
  - Using default arguments
  - Overloading and ambiguity

- Overloading constructor functions
  - There are three main reasons why you will want to overload a constructor function:
    - to gain flexibility
    - to support arrays
    - to create copy constructors

- Creating and using a copy constructor
  - When an object is passed to a function, a bitwise (i.e., exact) copy of that object is made and given to the function parameter that receives the object.
  - However, there are cases in which this identical copy is not desirable. For example, if the object contains a pointer to allocated memory, the copy will point to the same memory as does the original object.

- Creating and using a copy constructor
  - A similar situation occurs when an object is returned by a function. The compiler will commonly generate a temporary object that holds copy of the value returned by the function. (This is done automatically and is beyond your control.)
  - This temporary object goes out of scope once the value is returned to the calling routine, causing the temporary object's destructor to be called.
  - However, if the destructor destroys something needed by the calling routine (for example, if it frees dynamically allocated memory), trouble will follow.

- Creating and using a copy constructor
- C++ defines two distinct types of situations in which the value of one object is given to another. The first situation is assignment. The second situation is initialization which can occur three ways:
  - when an object is used to initialize another in a declaration statement
  - when an object is passed as a parameter to a function
  - when a temporary object is created for use as a return value of a function

OBJECT ORIENTED PROGRAMMING WITH C++

- Creating and using a copy constructor
- The copy constructor is called whenever an object is used to initialize another. The most common form of copy constructor is shown here:
    - classname(const classname &obj)
- the following statements would invoke the myclass copy constructor:
    - myclass x = y; // y explicitly initializing x
    - fun1(y); // y passed as a parameter
    - y = func2(); // y receiving a returned object

- Using default arguments
  - This feature is called the ***default argument***, and it allows you to give a parameter a default value when no corresponding argument is specified when the function is called.
  - Using default arguments is a shorthand form of function overloading.
  - To give a parameter a default argument, simply follow that parameter with an equal sign and the value you want it to default to if no corresponding argument is present when the function is called.
  - For example, this function gives its two parameters default values of 0:
    - void f(int a=0, int b=0);

- Using default arguments
  - void f(int a=0, int b=0);
  - Notice that this syntax is similar to variable initialization.
  - This function can now be called three different ways:
    - It can be called with both arguments specified: **f(10, 99);** // a is 10, b is 99
    - It can be called with only the first argument specified. **f(10);** // a is 10, b defaults to 0
    - It can be called with no arguments: **f();** // a and b default to 0
    - In this example, it should be clear that there is no way to default a and specify b.

OBJECT ORIENTED PROGRAMMING WITH C++

- Using default arguments
- When you create a function that has one or more default arguments, those arguments must be specified only once: either in the function's prototype or in its definition if the definition precedes the function's first use.
- The defaults cannot be specified in both the prototype and the definition. This rule applies even if you simply duplicate the same defaults.
- As you can probably guess, all default parameters must be to the right of any parameters that don't have defaults.
- Further, once you begin to define default parameters, you cannot specify any parameters that have no defaults.
- One other point about default arguments: they must be constants or global variables. They **cannot** be local variables or other parameters.

- Using default arguments
  - When you create a function that has one or more default arguments, those arguments must be specified only once: either in the function's prototype or in its definition if the definition precedes the function's first use.
  - The defaults cannot be specified in both the prototype and the definition. This rule applies even if you simply duplicate the same defaults.
  - As you can probably guess, all default parameters must be to the right of any parameters that don't have defaults.
  - Further, once you begin to define default parameters, you cannot specify any parameters that have no defaults.
  - One other point about default arguments: they must be constants or global variables. They **cannot** be local variables or other parameters.

- Overloading and Ambiguity
  - When you are overloading functions, it is possible to introduce ambiguity into your program.
  - Overloading-caused ambiguity can be introduced through type conversions, reference parameters, and default arguments.
  - Further, some types of ambiguity are caused by the overloaded functions themselves.
  - Other types occur in the manner in which an overloaded function is called. Ambiguity must be removed before your program will compile without error.