Constructors and Destructors

Constructors

Recall that when defining a class, the programmer may choose to write special member functions, called **constructors**. Syntactically speaking, constructors are member functions whose names are the same name as the class they belong to. Constructors are called automatically under the following circumstances.

1) When a class instance is declared. For example, "String S;", where "String" is a class.
2) When an instance of a class is passed by value to a function.
3) When an instance of a class is returned as a function value.

When working with static classes, the compiler automatically generates constructors. When implementing classes with dynamically created fields, it is **vital** that the programmer define their own constructors. If not, the compiler will generate constructors *that do not work* because they make a **shallow copy** of the class instance, i.e. they do not copy the dynamically created members of the class. A constructor that copies all the data of a class, including dynamically created data fields is said to make a **deep copy**.

A constructor that has no formal parameters, such as "`String::String()`" is called the **default constructor**. The default constructor would be called in case 1) above. The **copy constructor** is called in situations 2) and 3) above. It is also called when a class instance is declared and initialized to another class member, for example, "String S2 = S1;".

Constructors have many of the characteristics of normal member functions.

1) You declare and define them within the class, or declare them within the class, via a prototype, and define them outside.
2) Constructors, like most C++ functions, can have default arguments or use member initialization lists.

Constructors also have some unique features.

1) They do not have return value declarations, not even void.
2) You can't take their addresses.
3) Constructors are generated by the compiler if they haven't been explicitly defined. A constructor generated by the compiler will be public and make shallow copies if the class has dynamic members.
4) You can't call constructors the way you call a normal function. They are invoked implicitly as well as explicitly. The compiler automatically calls constructors when defining objects.

Destructors

- A class can have only one destructor. The name of a destructor is the same as the name of the class, but it is preceded by a tilde, **~**. For example, "~String()" is the name of the String class destructor. The destructor cannot have any parameters.
- If you do not write a destructor, the compiler will generate one for you automatically. However, if your class has any dynamically created fields, this destructor *will not work*!
- The class destructor is called automatically when a class instance goes out of scope.
- When writing a destructor and deallocating a dynamic array of standard types, the call "`delete A`" works. However, if A points to an array of objects, you should use "`delete [] A`".