Lab 9A

An Employee Class Hierarchy

Introduction

In these exercises you will work with the Employee class hierarchy to reinforce basic inheritance concepts. You will practice the initialization of both base class and embedded class objects and verify the order of invocation of constructors and destructors. You will define a function in the base class that is overridden differently in derived classes.

Suggested Time: 20 minutes.

Instructions

1. In the working directory there are files for an "Employee" project demonstrating inheritance from an **Employee** class and composition with the **String** class. Study the code in the files **Employee.h** and in the demo program **DemoEmp.cpp**. Make sure you understand the syntax used, including initializer lists with inline code in the constructors. Then answer the following questions. Then build and run the program to verify your answers. You may want to redirect the output to a file, which you can save and examine.

a. In the constructors for **Employee** a character pointer is used as an input argument. As the code is written, what member of the **String** class is used to initialize **m_name**?

b. If instead the following code was used, how would your answer change? Which code is better and why?

```
Employee(const char *name = "")
{
      Trace("Employee::Employee(const char*)");
      m_name = name;
}
```

c. In the **GetName** function of the **Employee** class, a **String** object is used as the return value.. What member of the **String** class is used to convert the **String** object to a character pointer?

d. What is the order of constructor and destructor calls when the test program is run?

2. Implement a **GetPay** function for each employee class that will calculate the pay appropriately for an employee. Add code to **DemoEmp.cpp** to calculate and print out the pay for the employees Sally and Wally. What is an appropriate way to implement **GetPay** in the base class? (This is a preview of some ideas we will discuss in the next chapter!)

Lab 9B

More Practice with Inheritance

Introduction

In this lab you will gain some more practice working with inheritance. You will work with a bare bones example in which there is a base class, a derived class, and an embedded member object.

Suggested Time: 15 minutes.

Instructions

1. Run the starter project and observe the order of constructors and destructors for **Base**, **Derived** and **Member**. Make sure you understand the operation of this program.

2. Add code to initialize the embedded **Member** object in **Derived** to be initialized to 99.

3. Modify the implementation of **Derived::Print** so that you do not call **Base::Print** but instead access **xx** directly. What other change do you have to make to enable this solution? Do you think this solution is better than the original?

Lab 9A Answers

1. a) The constructor **String::String(const char*)** does the initialization.

   b) The overloaded operator **String::operator=(const char *)** does the initialization. The code as originally written is better, because the embedded String object gets initialized directly by one call to a constructor. The second version is less efficient, because first the default String constructor is invoked to create an "empty" String object, and then the overloaded assignment operator is called.

   c) The overloaded cast operator **String::operator const char* () const** does the conversion.

   d) The order of constructors is embedded member, base class, derived class. The destructors are invoked in the reverse order. Sally (**SalaryEmployee**) is constructed first and destroyed last.

2. If you have a **GetPay** member function in the base class, about all you can do is assign some default pay to a "generic" employee. In the next chapter we will see that Employee can be an "abstract" base class, and we can specify a function that we don't have to implement. The highlighted code shows implementation of **GetPay** for the two "concrete" classes **SalaryEmployee** and **WageEmployee**, and the usage in the demo program.