Lab 10

Polymorphism in Employee Class Hierarchy

Introduction

In these exercises you will study an enhanced Employee class hierarchy in which **Employee** is now an abstract base class and **GetPay** is a pure virtual function. There is a heterogeneous array of employee pointers and a generic function **PayReport** that polymorphically prepares a report, delegating to each employee's **GetPay** function to calculate pay appropriately. You will look at issues of deleting objects from the collection. Finally you will add a new employee class and observe how easy it is to maintain a program having this kind of structure.

Suggested Time: 45 minutes.

Instructions

1. The file **Employee.h** in the work directory specifies an abstract class of employees. The pure virtual function **GetPay** specifies a function to determine the pay of an employee. There are two concrete derived classes **WageEmployee** and **SalaryEmployee**. A wage employee has an hourly rate of pay and a number of hours worked. Pay is

   ```
       rate * hours.
   ```

   A salary employee has a salary which is equal to the pay. The code file **Employee.cpp** contains an implementation of **GetPay** for the two concrete classes. The code file **DemoPoly.cpp** demonstrates polymorphism and a heterogeneous collection. An array of pointers to **Employee** is declared, and two employee objects are created on the heap. A function **PayReport** takes as an argument an array of pointers to **Employee** and prints a report showing name and pay of each employee, the proper pay being calculated based on the type of employee. Study this code, making sure you understand how this polymorphic behavior is implemented through virtual functions. Then predict what the output will be, including what constructors and destructors are called. Build and run the program to verify your understanding.

2. Why are **Employee** objects not being destroyed? Add code to **DemoPoly.cpp** to cause the **Employee** objects to be destroyed. Again predict the output and build and run the program to verify your understanding.

3. Why are the **SalaryEmployee** and **WageEmployee** destructors not being called? In what file do you need to make a change to fix this problem? Implement the fix. Build and run the program to verify

that it is now behaving as it should. Is this last problem actually an error in this particular case? Under what circumstances would it be an error?

4. Add another concrete class **SalesEmployee** derived from **Employee**. A sales employee has a commission rate and a volume of sales. Pay is

   ```
   commission * sales.
   ```

5. Implement changes in both the header file and code file of the employee classes, and modify the demo program **DemoPoly.cpp** to test the additional class. Does any modification need to be made to the function **PayReport** that prints out a report of pay of all employees stored in an array?

Lab 10 Answers

- **Employee** objects are not being destroyed because **delete** is not called for the objects created by **new**.

- The **SalaryEmployee** and **WageEmployee** destructors are not being called because the destructor in **Employee** was not virtual. Changing the destructor to virtual (in file **Employee.h**) resolves the problem (see **Employee\Step3** directory). The problem is not an error here, because the destructor does nothing (other than print a trace message). It would be an error in a case where the destructor did something substantive, such as deallocate dynamic memory allocated in the constructor.