Programming Assignment 4
(Virtual Functions, Templates)
Due date: November 12, 2013

Instructions:

- This assignment is to be turned in via Blackboard before **11:59 PM** on **November 12, 2013**. You should submit '.cpp' files with names mentioned after each problem.

- Make sure your programs compile (without any compiler errors). You will not receive any credit if your program does not compile. If you are unable to complete any of the programs, submit the parts that work (with no compiler errors) for partial credit.

- Your grade will be based on functionality (does the program do what it is suppose to do), readability (is the code nicely formatted), and understandability (are the literals meaningful and is the code modular and well documented with appropriate comments). Try to incorporate all the good programming practices and styles. E.g., **WRITE COMMENTS**, declare all constants and variables at the top of a function, store quantities using the appropriate data types, declare function prototypes, etc.

---

1. Virtual Functions (20 points)

In this problem you have to write class definitions and a main program to simulate different pricing plans for a cellular telephone provider. Suppose you work in the marketing department of the Cellular Two telecommunications company. Your boss wants you to analyze a proposed plan that offers customers a Basic plan and a Premium plan. The Premium plan is designed so that customers who make many calls save money by paying a higher monthly fee. Write class definition for Customer (that represents a basic customer) and for Premium_Customer. Customer should contain data members common to all customers, like the number of calls made in a month, the customers name, and so on. Also implement a virtual member function for computing a bill, Compute_Bill( ) which uses the appropriate algorithm (mentioned below) and data members to compute a monthly charge for each type of customer. In all classes be sure to also include any constructors and other member functions that you think are needed especially if you use dynamic memory allocation in your class. The Premium_Customer class should inherit all attributes of the basic customer, contain data members and initializations specific to its payment plan, and a specific implementation for the Compute_Bill() member function which overrides the default method. Demonstrate programmatically which plan is better for a customer who makes 30 calls per month averaging 3 minutes per call. What about 60 calls?

```
// This is a sample template for a main program, your version may be different
int main () {
    Customer* list[6] ;
    list[0] = new Customer("John Dough", 20);
    list[1] = new Customer("Bob Dough", 50);
      // and so on...
    list[6] = new Premium_Customer("Jane Doe", 100);

    for(int i=0; i<6; i++) {
        cout << "Customer " << list[i]->getName() << " owes "
             << list[i].Compute_Bill() << " dollars." << endl;
    }

    // delete all the customers
    return 1;
}
```

Compute a basic customer's bill as:

Bill = monthlyfee + (percall X numcalls)

using a monthly fee of $10 and a per call charge of $0.50.

Compute a premium customer's bill as:

Bill = monthlyfee + (percall X numcalls) + (permin X nummins)

using a monthly fee of $20, a per call rate of $0.05, and a per minute call rate of $0.10

Bonus Part (Optional) - 10 points

Make this program a simulation to compare the two plans. Make a larger population of customers half of whom use each plan. Vary the number of calls from 10 to 100 (by ten) per month and use a random number generator to pick different numbers of minutes between 2 and 8 for each premium customer. Which plan is better for most customers in the study? For the customers who do get a savings with the premium plan, what is the average savings?

Filename: phoneBill.cpp

---

2. Templates & Operator Overloading (30 points)

Design a generalized template class named *SimpleVector* - an array that can hold any specified data type. Use a dynamic array for the implementation of SimpleVector. This class should have the following:

- private data members - a pointer to the specific template data type, and the array size
- a default no-arg constructor that sets the pointer to null and array size to 0
- a single argument constructor that takes the array size as argument and creates a dynamic array using the specified size.
- your own copy constructor that performs a deep copy, i.e., create a dynamic array and copy elements from the other array that was passed as an argument to the copy constructor
- a destructor that deallocates the memory allocated to the dynamic array
- accessor method to return the array size
- a public method called getElementAt that takes position as argument and returns the element from the array at that specified position
- Overload the [] operator. The argument is a subscript. This function returns a reference to the element in the array indexed by the subscript.

The main() should test all of these functions. It should perform the following steps in a loop:

- ask the user what type of data do they want to enter (this would be the data type you would use to create an object of the SimpleVector class). You can ask the user to enter 1 for integer, 2 for double and 3 for strings.
- ask the user how many data inputs do they have (this would be the size of your dynamic array).
- ask the user to enter the data (this is the data that you would enter in the dynamic array).
- ask the user to enter an index to retrieve the data at that index (use 'getElementAt' member function for this). Display the retrieved data with the index in a nicely formatted way.
- ask the user again to enter an index to retrieve the data at that index (use the overloaded '[]' operator for this). Display the retrieved data with the index in a nicely formatted way.
- create a new SimpleVector object using the copy constructor by passing the old object to the constructor. Display to the user the data in this new object.
- ask the user if they want to enter the data again (y/n).
- if 'n', then the program ends, otherwise it should clear the SimpleVector object (i.e., deallocate the dynamic array, set it to NULL and set the size to 0) and repeat the loop (ask the user to enter new data...).

Filename: simpleVector.cpp
