C++ Programming Methods

Assignment 4, Large Number

Bas Terwijn <b.terwijn@uva.nl>

In this assignment we will compute large Fibonacci and factorial numbers by overloading an existing Number class. Here we practice using strings, classes, inheritance, pointers and dynamic memory allocation.

Math Library

In order to compute Fibonacci and factorial numbers you are provided with the source of a library and a test application that already works in file “smallNumber.cpp”. Compile and run it, you will notice that it works for small “n” but for large “n” (for example 100) SmallNumber will overflow and therefore give wrong results.

SmallNumber

The SmallNumber class is a subclass of (inherits from) the abstract Number class and represent its value as an “int”. The methods of the Number class overload operators such as “==”, “+”, “*”, these operators are chosen in an attempt to increase code readability. Operator overloading is explained in Chapter 8.

Math

The Math class only takes pointers to the Number class as arguments, this (or alternatively reference arguments) and the fact that most methods of Number are virtual allows for late binding. Therefore we can substitute the SmallNumber class by our own class to solve the overflow problem without modifying the Math class. This is important because in many cases you won't have access to the source of a library you are using and making the right modification likely is difficult.

LargeNumber

To compute large Fibonacci and factorial numbers we ask you to create class LargeNumber that inherits from the Number class and implements its methods. Your LargeNumber class should be able to represent non-negative whole numbers of arbitrary length by storing the number in a double linked list. Each element of the double linked list should have an “int” that holds “k” (use global k=4 as maximum) digits of the number and a pointer to the next and previous element. This double linked list should be updated accordingly when the arithmetic methods are called. Use an efficient implementation for these methods to avoid long waiting times for large numbers. For addition and multiplication take inspiration of how you were taught to do them in elementary school. Be careful to free (using the “delete” expression) any memory that is allocated by the “new” expression,

especially for intermediate results. Otherwise you run the risk of running out of memory when computing with large numbers. Note thereby that methods “factoryMethod”, “+”, and “*” each allocate memory for a new Number that they return. Optionally use program “valgrind” to detect memory leaks:

http://valgrind.org/docs/manual/quick-start.html

Operator Overloading

Be careful with the operators, for example:

```
SmallNumber *n1=new SmallNumber(“111”)
SmallNumber *n2=new SmallNumber(“222”)
SmallNumber *n3;
n3=(*n1)+n2; // will make n3 equal to 333
n3=n1+n2;    // will set pointer n3 to pointer value n1 plus n2, probably
             // not what you want!!
```

Testing

Small Fibonacci and factorial numbers can be found easily to compare your results with. For larger numbers we provide you with file “test.txt” with the output of our program for large n.

Submission

Submit your solution before the deadline to blackboard. Add to each solution:

- your name, student number and the name of the assignment

- references to the source of any algorithm or code that you did not create yourself

- operating system and compiler that was used to test the code