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 overriding an existing Number class. Here we practice using strings, classes, inheritance, polymorphism, 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 test application that already works in file “SmallNumber.cpp”. Compile and run it, you will notice that it works fine with small values for its single command line argument “n” but with large “n” (for example 100) SmallNumber objects will overflow and therefore give wrong results.

SmallNumber

The SmallNumber class is a subclass of (inherits from) the abstract Number class that defines the interface for a number. The SmallNumber represents its value using an “int” and overrides the methods of Number to manipulate this “int”. These methods include the familiar “==”, “++”, “+”, and “*” operators that were chosen in an attempt to increase code readability. The “int” representation is not able to represent very large numbers and this of course is the reason for the overflow problem that we will solve in this assignment.

Math

The Math class that computes the Fibonacci and factorial numbers takes references to the Number interface as arguments to its methods. This and the fact that the methods of the Number class are “virtual”, allows for late binding. This late binding allows us to substitute the SmallNumber class by our own class and in doing so change the representation of a number to solve the overflow problem. Notice that we don't need to, and will not, modifying the Math class to accomplish this (assume we don't even have access to its implementation code).

In general this technique using late binding, called “polymorphism”, is a powerful mechanism that allows us to create software who's behavior can be changed by future users only by overriding the methods of a virtual class.

LargeNumber

To compute large Fibonacci and factorial numbers that don’t fit in an “int”, this assignment asks you to create a LargeNumber class that can represent very large numbers and that, similar to the class SmallNumber, inherits from the Number class and overrides its methods. Your LargeNumber

class should represent a non-negative whole number of arbitrary length by representing the number as a double-linked list of “ints” instead of a single “int”. Each node in the double-linked list should have an “int” that holds at most “k” (use k=4 as maximum) digits of the number . In addition each node should have a pointer to the next and previous node. When a method like “++” of LargeNumber is called its double-linked list should be updated so that it represents the correct result. Use an efficient implementation for these methods to avoid low performance when calculating with LargeNumber. For addition and multiplication take inspiration of how you were (hopefully) taught to do them in elementary school.

You have to implement your own double-linked list, you may not use an existing implementation but of course you are allowed to look at descriptions of how best to implement them. It is best to first implement and test your double-linked list on its own before using it in your LargeNumber class implementation.

Memory management

For the double-linked list you will have to use dynamic memory allocation. This can best be done using the “new” keyword. Be careful to then also use the “delete” keyword to deallocate the memory when it is no longer needed. If you don't free memory that is no longer needed your program might run slow or even crash when all memory is used up. It can be difficult to do memory management correctly. Modern C++ has smart-pointers (unique_ptr, shared_ptr) to make this easier, but for this assignment you are not allowed to use these. We ask you to use normal pointers so you get familiar with these first. Some options and tools that can help you with memory management and improving your code generally are:

extra warnings
Turn on extra warnings when compiling:

```
$ g++ -Wall -Wextra -pedantic LargeNumber.cpp  -o LargeNumber
```

clang-tidy: https://clang.llvm.org/extra/clang-tidy
Makes suggestions to improve your code following the latest C++ guidelines.

```
$ clang-tidy -checks=* ./LargeNumber.cpp
```

or add “-fix” to let it edit your code for you (make a backup first).

```
$ clang-tidy -checks=* ./LargeNumber.cpp -fix
```

sanitiser: https://en.wikipedia.org/wiki/AddressSanitizer
Adds memory checks to the executable that do their work at run time:

```
$ g++ -g -fsanitize=address LargeNumber.cpp  -o LargeNumber
$ ./LargeNumber 10 # now look for warning
```

valgrind: http://valgrind.org/docs/manual/quick-start.html
Checks for memory leaks at run time:

```
$ g++ -g LargeNumber.cpp  -o LargeNumber
$ valgrind ./LargeNumber 10 # now read the log
```

Testing

Small Fibonacci and factorial numbers can easily be found to compare your results with. For some larger numbers we provide you with file “test.txt”. The larger numbers in this file were computed by a particularly fast implementation of LargeNumber. We don't require you to compute the larger of these numbers as this might take a long time.

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