CMPT 212 (2008-3) Assignment 1*
Due online Tuesday February 5, 2008, 11pm

Ján Maňuch
jmanuch@sfu.ca

1 Programming Assignment

Write a C++ program that acts as a simple "printing" calculator which work with arbitrary large integer values. Your program must compile under Visual Studio C++ 2005 and must run under Microsoft Windows as a console application (at the so called "MS-DOS" or "command" prompt).

All your code must be located in a file called a1.cpp, which will be the only file submitted for the assignment. You will be expected to use C++-style input/output (`cin` and `cout`, etc).

Information about evaluating and grading the assignment can be found at the course website.

If you have any questions about the assignment, do not hesitate to send me an email. I might revise the text of the assignment based on your questions, if I find out that some parts are difficult to understand.

2 Problem Specification

The submitted programs must behave as follows:

2.1 Overview

The program acts as a simple calculator but can work with arbitrary large integers. It can perform the following operations: `+`, `-`, `*`, `/`, `C` and `E`. It remembers one value shown on the display (called "accumulator"). Initially, the value of accumulator is set to `0`.

The program repeatedly performs the following tasks:

- Displays `"Value: "`, the value of accumulator.
- Displays `"Enter: "` and waits until user enters an input and presses ENTER.
- Reads the input from the user. (Note! The input can be arbitrary long and you have to make sure your program will read all of it until the end.)
- If the input was `E`, exits the program. Otherwise, it modifies the values of accumulator and memory based on the input.

2.2 Valid inputs

The following is the list of possible inputs and expected actions of the program. In the inputs, `<number>` stands for an arbitrary long sequence of digits `0`–`9`, possibly prefixed with the "minus" sign `-`.

* *Revision* : 1.0 (Tuesday 22nd January, 2008,11:26); Details in this document may change before the date of the submission of the assignment. Please make sure you have the latest version.

`=<number>`	set the value of accumulator to the number
`+<number>`	add the number to the accumulator
`-<number>`	subtract the number from the accumulator
`*<number>`	multiply the accumulator by the number
`/<number>`	divide the accumulator by the number; take only the integer part of the division; for instance: if the accumulator contains value `100` and the user enters `/11`, the new value of the accumulator is `9`
`C`	reset the value of accumulator to `0` (note: the input `0` would have the same effect)
`E`	exit the program

On any other input than above, the program should respond with the text `"Invalid input"` and the value of accumulator should stay unchanged. Also note that the above commands cannot be combined, i.e., the input `+2*3` should not add value `2` to the accumulator and then multiply the accumulator by `3`, but **it should** just print `"Invalid input"` message.

2.3 Sample run

The following is the interaction of the program and a user. The strings entered by the user are printed in green.

```
Value: 0
Enter: =12345678901
Value: 12345678901
Enter: *98765432109
Value: 1219326311336229232209
Enter: +10E5
Invalid input
Value: 1219326311336229232209
Enter: 105
Invalid input
Value: 1219326311336229232209
Enter: +4564564564564554564564
Value: 5783890875900783796773
Enter: /777777777
Value: 7436431133594
Enter: -12345678901234
Value: -4909247767640
Enter: *-2
Value: 9818495535280
Enter: =101
Value: 101
Enter: ++2
Invalid input
Value: 101
Enter: E
```

2.4 Remarks and hints

To successfully complete the assignment you will have to deal with the following problems:

- Store the values of accumulator, memory and other temporary values in a suitable data structure (dynamic arrays, linked lists, etc.). Do not try to store one value in one variable of type `long long`. Type `long long` can

work with values up to 20 digits which would not be enough in the above example (and will be not enough in the test files).

- On the other hand, you can assume that the number of digits of the values needed to be stored in accumulator, memory or other temporary variables is at most **`LONG_MAX`** (defined in **`climits`** header file), i.e., you can store the number of digits in a variable of type **`long`**.
- Be careful when reading the input from the user. Make sure you don't overwrite some undeclared memory. For example, if you declare array **`input`** of characters of size 100 and use **`cin >> input`** to get the input from the user, and the user enters 200 characters without space in it, it will happen and your program will probably crash.
- Also make sure that you read the whole input from the user. For instance if the user enters 100 digits followed by anything else than digit, it's not a valid input, and your program should just print error message and leave the values of accumulator and memory unchanged.
- You need to figure out a way how to perform arithmetic operations with numbers stored in the data structure you have chosen. While addition and subtraction might be easy, multiplication and division are a bit more complex (hint: you can check wikipedia for help with algorithms for multiplication and division or you can check here). If you have any problem, in particular with the division, please, come to discuss the algorithm to my office hours.
- Remember, your program should be also able to deal with negative integers.

2.5 Testing

Your program will be tested using test files. The test file contains the list of inputs. For example, for the above sample run, the test file would look like this:

```
=12345678901
*98765432109
+10E5
105
+4564564564564554564564
/777777777
-12345678901234
*-2
=101
++2
E
```

To run the program with a test file `a1_test1.txt`, the following command will be used

```
a1 < a1_test1.txt >a1_test1-result.txt
```

in the command prompt, where **`a1.exe`** contains compiled executable code of your program. This command redirects the input from the file to the program. It will print the output to the file **`a1_test1-result.txt`**, which will be compared with **`a1_test1-output.txt`**. To get the full score from the accuracy part, the two files (for this test and one other test which is not available to you) have to be identical.

3 Programming Contest

Small size of executable code used to be very important in the design of software, this also being one of the driving forces behind the creation of the C programming language. Small sizes are still important for embedded applications, or modules/utilities of operating systems with required small footprint.

Assignment 1 will have an associated programming contest. Only assignments that are considered to have adequate implementations, as outlined in Section 2 will be entered into the contest.

All qualifying entries will be compiled and tested on Linux, using the GCC compiler, similar with the one installed on the MTF Linux lab machines (e.g. `tomato.csil.sfu.ca`). The winners will be entries that result in the *smallest size of executable*.

Up to three entries will be selected. In case of ties that result in more than three top entries of equal size, none of the tied entries will be considered.

The winners will be publicly acknowledged on the course website and during lecture. The source code of winning assignments will be uploaded to and linked from the course web site. Students who are not comfortable with these conditions should indicate that through email at time of submission of their assignment and will not be considered for the contest.