AlPro
Tp1 - Introduction to C and big integers

1 Debugging

The file `debug.c` contains a number of bugs that can be fixed with the help of the compiling errors. To compile the file, go to the folder that contains it and use the following command: `gcc -o debug debug.c`.

When all bugs are fixed, this will create an executable file `debug` that you can run using `./debug`.

2 Fibonacci

In the file `firstProgram.c` you will find the following algorithm to compute n Fibonacci numbers (defined by the induction $u_{n+1} = u_n + u_{n-1}$) starting with 1 and $r = \frac{1-\sqrt{5}}{2}$. We can show by induction that for all n we have $u_n = r^n$. The goal of this program is to highlight the effects of the propagation of rounding errors in the recurrence calculation of the sequence (u_n).

Algorithm 1 Fibonacci sequence with $r = \frac{1-\sqrt{5}}{2}$

```
1: procedure FIBONACCI(n)    ▷ Return the n-th Fibonacci number starting with u_0 = 1 and u_1 = (1-√5)/2
2:     if n == 0 OR n == 1 then
3:         return n
4:     end if
5:     return Fibonacci(n − 1) + Fibonacci(n − 2)
6: end procedure
```

Use the command `gcc -o first firstProgram.c -lm` with option `-lm` to be able to use the `math.h` library.

3 manipulation of large integers

3.1 Introduction

The goal of this tutorial is to be able to perform calculations on integers with a larger number of digits than the standard integer types. Each integer will be represented by its digits (in base 10), the number of digits being limited by the constants DIGITSMAX. We define the following types:

- `intArray : array` that is a fixed length array of integer;
-
```
bigInt : record
  boolean : negative
  intArray : digits
end record
```

that corresponds to the big integer

For an integer x, `negative` is 1 (`true`) if the number is negative. The values in the array `digits` are the values of the $(a_i) \in \{0, \ldots, 9\}$ (listed from left to right) such that: $|x| = \sum_{i=0}^{DIGITSMAX-1} a_i 10^i$.

This way the number 54559983401 will be represented in the array `digits` as follows:

index	0	1	2	3	4	5	6	7	8	9	10	11	...	19
digit	1	0	4	3	8	9	9	5	5	4	5	0	...	0

3.2 Useful files (on hippocampus or the box)

3.2.1 Types

The file `big_int.h` contains the definition of the structured type `bigInt`. You have to include this file (**#include** "`big_int.h`") in every source file (`.c`) in which you want to use this type `bigInt`. Just remember, arrays start at 0 and end at length-1 in C.

3.2.2 Read and write functions for a big int

To get started, you will need to manipulate this big integers. For that you will use two functions:

- `readBigInt()` that reads a big integer from the keyboard (with less than DIGITSMAX) and return a `bigInt n`.
- `printBigInt(n)` that writes a big integer n on the screen (in the terminal).

On hippocampus you will find the following files (that you should not modify):

- The file `read_write.c` containing the definition for both functions.
- The file `read_write.h` containing the declaration for both functions. You have to include this file (**#include** ``` `big_int.h` ```) in every source file in which you want to use these functions.

3.3 Work to be done

Design and program in C the following functions:

- In the file `utilities.c`:
  - conversion of a standard integer to a big integer;
  - equality test between to big integer;
  - comparison in absolute value: for two big integers a and b, do we have $|a| \leq |b|$?
- In the file `utilities.h`: the declarations of the functions contained in the file `utilities.c`.
- In the file `operations.c`:
  - addition $a + b$ of two big integer a and b **of same sign**;
  - subtraction $a - b$ of two big integers a and b of same sign and such that $|a| \leq |b|$;
  - addition and subtraction of two big integers of any signs, using the previous functions;
  - Optional: the multiplication and euclidean division of two positive long integers.
- In the file `operations.h`: the declarations of the functions contained in the file `operations.c`.

Simultaneously, in a file `main.c`, write a main function that you will modify progressively to test the different programmed functions.

4 Optional: Report

If you want, you could submit some code and/or a report containing for example:

- For every function:
  - the specification;
  - a brief description of the principle;
  - simple tests to illustrate its proper functioning;
- The algorithms of the functions for the "same-sign addition", the "any-sign addition" and, if you have done so, those for the multiplication and euclidean division.
- For the test sets you may use:
  - simple examples that cover all cases to illustrate the proper functioning of the operations,
  - the computation of fibonacci numbers: $u_0 = 0, u_1 = 1, \forall n \geq 2, u_n = u_{n-1} + u_{n-2}$ for different values $n \in \{100, 1000, 10000\}$, print the n-th fibonacci number.
    Verify for example that $u_{60} - u_{59} = u_{58}$ using the operations functions over big integers.