Report Lab 1

Tatiana Lopez Guevara

10 de octubre de 2013

1. Variables

1.1. Local vs Global

A global variable can be accessed from any part of the program and will be alive during the execution of the whole program. A local variable is only accessible inside the scope its declared and will be alive during the execution of the function or scope it lives in.

The section 3 shows the use of the global variable φ that holds the known value of the golden ratio. Inside the function `golden_ratio` the value *err* is a local variable which is only accessible inside that function.

1.2. Elementary functions

In this section I created the functions `mean, min, max` which received 2 integer parameters. In this case the input variables do not need to be modified and therefore I learned how to help a little bit the compiler by explicitly specifying the read only property with `const` keyword and also by not creating a copy by passing it a reference with `&` .

Also, in the function `mean` instead of dividing by 2, a multiplication for 0.5 was used because it uses less clock cycles in the execution.

2. Combination

2.1. Factorial

In this function we noticed 2 things besides the optimizations mentioned int the previous section:

- The Fibonacci is only defined for values in $\mathbb{N}^0$ and therefore is better to declare it as unsigned. Due to the extremely fast growing of the factorial function we declared the type as `short`.
- The return type must be able to hold a very large number. At the begining we declared it as `long long` but then we realized that it overflowed with

the factorial of numbers greater than 21 and therefore we had to change it to `double`.

2.2. Combinations

In order to follow the KISS phylosphy, I just reused the `factorial` function created in the previous step to calculate $n!, k!$ and $(n-k)!$ used in the formula.

The number of combinations of a lottery game is C_6^{49} and the program gave the result 1.39838e+07.

2.3. Combinations with repetitions

Same as before. I just reused the `combinations` function to get the term:

$$\binom{n+k-1}{k}$$

2.4. Permutations

In this function I reused the two previous functions to get the value $P_5^{54} = 3{,}79501e+08$ from the formula:

$$P_k^n = \frac{n!}{(n-k)!}$$

3. Fibonacci

The Fibonacci function has a recursive definition, however the implementation realized here was not, first because for printing each one of the terms it was more easy and second because it is faster.

3.1. Golden Ratio

The same logic of the previous section was implemented in this function and the ratio between the current value and the previous value was calculated and printed at each step. When the variable of the error and the rate was declared as `float` the loop went to an infinite cycle because of precision [1] when calculating it with the $\epsilon = 10^{-9}$. The results obtained with the `double` declaration where:

- $\epsilon = 10^{-6}$ stopped with $F_{16} = 987$ and $F_{17} = 1597$.
- $\epsilon = 10^{-9}$ stopped with $F_{23} = 28657$ and $F_{24} = 46368$.
- $\epsilon = 10^{-12}$ stopped with $F_{30} = 832040$ and $F_{31} = 1346269$.
- $\epsilon = 10^{-15}$ stopped with $F_{37} = 24157817$ and $F_{38} = 39088169$.

4. Pascal's Triangle

One of the most simple and effective ways to print the Pascal's triangle is using arrays to calculate the current value only through sums and based only on 2 previous factos. However, this was implemented for Lab2 and here I just used another simple approach by just reusing the `combinations` function for each calculation.

5. Other things

5.1. String

I used 5 ways to create strings: the default constructor, passing a string, assigning a value with the `=` operator, and repeating a character constructor [2]. The empty string was then assigned to the concatenated value of the first 2 strings using the `+` operator. The last string was used to play with the `append` and `insert` functions. All values were shown in the output ussing the `<<` operator and `c_str()`

5.2. Main

The parameters of the main are used for accessing the information of the parameters passed into the program on the invocation. More specifically, `argc` is the number of parameters passed where the 1st one is the program invocation by itself. `argv` is a vector of `argc` possitions, each one corresponding to the value of the parameters for the program.

To illustrate this, at the begining of the `main`, I reused the proposed code in the last point of the lab.

Referencias

[1] Yohan FOUGEROLLE. Software engineering fundamentals of c++. University Lecture, 2013.

[2] C Plus Plus. C++ std::string description, October 2013.