Master TSI
C++ Programming
2016-17

Exercises 1

The exercises have to be performed on Linux OS using G++ (the free C++ compiler suite)[1].

1. Open a console and your preferred text editor (if you don't have any preferred editor, try with `gedit`).
2. Type the C++ program and save it to a file, for example `your_file.cpp`.
3. Compile it with `g++` command:

   ```
   g++ your_file.cpp -o your_file
   ```
4. If there is some error, fix it by restarting at step 2.
5. Run the obtained program:

   ./your_file

In this first suite of exercise, you have to use a minimal version C++ (shown during the source) and made of C statements and C++ input/output (`cout`, `cerr`, `cin`). It is recalled that, in order to use C++ input/output, you have to prepend your program with:

```
#include <iostream>
using namespace std;
```

1. Basic Algorithms

1. Using a loop, write a program that computes the sum of n first integers (n is provided by the user). To test your program, uses several values n recalling that:

   $$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$
2. Write a program that computes $n!$, n given by the user. The following tests may be applied to validate it:

   0! = 1, 1! = 1, 4! = 24, 10! = 3,638,800
3. Write a program that computes the maximum and the minimum of n integer values provided by the user (n is also provided by the user). To test it, you can use the sets of values below:

   n = 5, [3, -5, 10, 0, 4], max = 10, min = -5

   n = 1, [0], max = 0, min = 0

1 Remark that all tools we will use during this course are either free, or open source and can be installed and run on your own laptop / desktop. I also advise you to install a version of Linux on your PC or to run it from a virtual machine like VirtualBox or VMWare (that are also available for free).

4. We want to approximate the sin function using Mac Laurin series. Write a program that asks the user for a number of iteration *n* and value *x* and that computes sin(*x*) using the `math.h` trigonometric functions and the Mac Laurin series. Test your program with several values of *n* and *x* and use the `math.h`'s sin(*x*) to validate your results.

   $$\sin(x)=\sum_{i=0}^{n}\frac{(-1)^i}{(2i+1)!}x^{2i+1}$$

5. If needed, improve the previous version by remarking that :

   $(-1)^{i+1} = (-1)^i * (-1)$

   $(2(i+1) + 1)! = (2i + 3)! = (2i + 1)! \times (2i + 2) \times (2i + 3)$

   $x^{2(i+1)+1} = x^{2i+3} = x^{2i+1} \times x \times x$

6. Modify the previous program to display the difference between `math.h`'s sinus and Mac Laurin sinus between [-π, π] with a step of 0.2. Test your program.

2. Array Management

1. Write a program that declares an array `t` of integers with a size of 5 and that is then filled with values provided by the user.
2. Add to the previous program some code to compute and display the average of elements of `t`. Test it.
3. Modify the previous program to display also the number of elements of `t` that are higher or equal to the average. Test it.
4. Modify the previous program to change the size of array `t` to 1,000 and to let the user enter the number of elements `n` to work with, under the assumption that `n` ≤ 1000. According to `n`, only a part of `t` will be used but your program will become much more general.

   Test it with `n` = 8 and values 5, -2, 4, 10, 11, 0, 6, 7.

   Test it with `n` = 3 and values 11, 12, 10.

3. Functions

1. Write a function `int fact(int n)` that computes *n*!. Then write a main program that computes the number of *k* combinations over *n*, C_n^k , *k* and *n* provided by the user.

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

2. Write a sub-program called `divmod` that takes as input parameters two integers, *x* and *y* and returns two values: the integer quotient *q* and the remainder *r*.
3. Uses the `divmod` function, defined previously, to write a program that displays the digits composing an integer number *n* in reverse order (*n* is provided by the user). One may

observe that, for example, starting from 123, the division by 10 gives 12 and remainder 3. Then 12 divided by 10 gives 1 and remainder 2 and, finally, 1 divided by 10, gives 0 and remainder 1.

4. Use the `divmod` function to compute the sum of squares of decimal digits composing a number *n*.
5. Put the algorithm computing the sums of decimal digit square sum inside a function called `digit_square`. Which are the input parameters, the output parameters and the result of such a function?
6. (for fun) Use the functions previously defined to compute the display, for each number between 1 and 200:

   x `->` *y* `;`
   Where x is the considered number and y the sum of squares of its digit.
7. (for fun) Copy and paste the resulting lines on the page http://ushiroad.com/jsviz between tags:

   ```
   digraph testgraph {
         …
   }
   ```
   And click on the button at the bottom-right to get the display.

   The iterative computation of digit square sum of a number is called the numbers of Prabhekar and the iterative computation always ends either on the fix point 1, or in the cycle 4, 16, 37, 58, 89, 145, 42, 20. Surprising… isn't it?