Einführung in die Programmierung für Physiker

SoSe 2024 – Marc Wagner

Michael Eichberg: eichberg@itp.uni-frankfurt.de

Exercise sheet 5

To be handed in on 24.05.2024.

Exercise 1 [*Leaplings*] (2 + 2 + 2 = 6 pts.)

A person born in a leap year on February 29 may be called a leapling. Technically, leaplings will have fewer birthday anniversaries than their age in years.

In Gilbert and Sullivan's 1879 comic opera "The Pirates of Penzance", Frederic the leapling pirate apprentice, born in 1856, discovers, at the age of 21, that he is bound to serve the pirates until his 21st birthday rather than for 21 years, meaning he has to serve for much much longer. We want to help Frederic to find out in which calendar year he will be allowed to retire according to his indenture, and how old he will really be at his retirement. Write a program to

(i) Establish whether any year provided as input by the user is or is *not* a leap year (input in this case is just a year),

(ii) Count leap years within any range of years provided as input by the user (input are the first and last year of the range of years),

(iii) Establish the year at which a leapling, born in some user provided year, will have celebrated the number of birthdays provided by the user and what their real age in years will be by then (input are a year and a number of birthdays).

Hint: You can first ask the user in which mode they want to run the code and use

```
if ( /* < condition to enter mode 1 > */ ) {
/* ... */
} else if ( /* < condition to enter mode 2 > */ ) {
/* ... */
} else if ( /* < condition to enter mode 3 > */ ) {
/* ... */
} else {
/* ... */
}
```

to enter various modes, e.g. reading a `char`.

Hint: If you write a function `bool IsYearLeapYear(unsigned int year)` not only can you use it in the solution of task (i), but you could reuse it in the solution of task (ii). Note that, strictly speaking, the `bool` data type did not originally belong to the `C` language, but was introduced in `C++` (The `C99` standard of `C` introduced the `stdbool.h` header to allow the user to use `bool` as in `C++`.). However, it is very handy and we encourage you to use it (remember to use, then, e.g., `g++` as compiler). A `bool` variable can either contain the logical value `true` or `false` and statements like `bool isLeap=false;` or `return true;` are perfectly allowed. Moreover, you can test the content of a bool variable `x` with an if-clause as `if(x==true)`, or simply `if(x)`. Also a function `unsigned int CountLeapYearsInRange( unsigned int year1, unsigned int year2)` might turn out to be useful more than once!

Time to Test! Note that you can test your solution for task (iii) against your solution for task (ii).

Exercise 2 [*Random walk in 8 dimensions*] (2 + 2 + 2 = 6 pts.)

Consider a particle at position $\boldsymbol{n}$ on a discrete, infinite grid with 8 dimensions. We want to simulate the random walk of this particle by randomly choosing a sign $\pm$ and a direction $i \in \{0, \ldots, 7\}$, then adding ± 1 to n_i in each step.

(i) Write a program which performs $N = 10^7$ steps, starting at $\boldsymbol{n} = 0$, where the coordinate n_i is changed using `if`-statements. At the end, print the distance from the origin $|\boldsymbol{n}|_2$.

Hint: To generate random numbers, you can use the `rand()` function from `stdlib.h`.

Hint: Be careful with your choice of data types, 10^7 and $(10^7)^2$ are large numbers.

(ii) Repeat task (i), replacing `if`-statements with a `switch`.

(iii) We would like to compare execution times between the `if` and `switch` variant. Change your functions, such that no random numbers are generated, but instead, fixing $i = 7$ and increasing n_7 by $+1$ in each step. Comment on your observation. Use the `time.h` library and the `clock()` function to measure and compare computation times:

```
#include <time.h>

/* ... */
clock_t begin = clock();
/* <tasks to perform> */
clock_t end = clock();
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
/* ... */
```

Hint: Refactor your code, such that both tasks are performed in seperate functions.

Exercise 3 [*Prime numbers*] (2 + 3 + 2 + 1 = 8 pts.)

Write a program which asks the user for a number N and

(i) checks wether it is a prime number,

(ii) if it is not a prime number, prints its prime decomposition,

(iii) counts the number of prime numbers up to the given number.

To check if N is a prime number and how to perform a prime decomposition, do not use a library, but create your own implementation.

Optional: In the prime decomposition, group the same factors together. The program should print something like `1000 = 2^3 * 5^3`.

Structure your code: Although you did not discuss in detail the functions syntax, it should not be difficult to structure your code writing simple functions. Refer to the lecture notes in case of doubt. Implement the check of task (i) in a `bool IsPrime(unsigned int inputNumber)` function and write a function which, given a number, prints its prime decomposition to the standard output. A possible signature for the latter could be `void PrintPrimeDecomposition(unsigned int inputNumber)`.

Time to Test! What happens if you give 1 or a negative number to your code? Run your code with several integers, like `17`, `40309`, `65536` and `100003`. For $N = 1000$ there are 168 prime numbers up to N.

(iv) Remove any `scanf` function and fix the number to $N = 10^7$. Run your code and measure the execution time using the `time.h` library. Alternatively, you can do it from your shell using the `time` command, which has to be placed before your executable (e.g. `time ./myEx`). The `time` command gives you more time measurements, consider only the *real* one. Try to recompile your code with different optimization levels (here we refer to the names of the `gcc` compiler options – if you use a different compiler you have to check which are the equivalent names), e.g. `-O1`, `-O2`, `-O3`, and measure the execution time again. How significant is the change?

Remark: Never use optimization flags like `-Ofast` which imply `-ffast-math`, because they will probably make your *floating point* result inaccurate and then wrong!