1. Introduction

Ján Dugáček

September 29, 2017

Obsah

Why do physicists need programming?

- Very basic experiments' results can be computed manually
- Basic experiments' results need a calculator
- Most demonstrative experiments' results can be computed using MS Excel or something similar
- Most data processing needs custom programs (e.g. in Python)
- Often, the amount of computations is so high that the programming language used must be efficient (e.g. C/C++)
- Not knowing to program is a handicap to a physicist

Why C++?

- Most short programs read the data, chew it and spit out the results seemingly in an instant
- However, the computation is not a formula that takes input and outputs the result way too often
- It is very common that the measured data is a non-linear function that depends on values we are trying to obtain and there is no backwards formula
- Very often, it's necessary to try a large number of inputs and compute how well does the result function match with the measured function
- We have 100 points to compare, four parametres, we split each parameter's expected range to 1000 points, we need $100 \cdot 1000^4 = 100\ 000\ 000\ 000\ 000$ operations (processor does only milliards of operations per second)

Why C++?

- The speed of programming languages differs greatly
- The difficulty of programming languages differs greatly, usually difficulty is the price to pay for speed
- Programming languages like R or MatLab are 500 times slower than C, meaning that we compare a day long computation with a year and three months long computation (they have functions written in C++ that can be used, but they are not always available for the task)
- Python is 50 times slower than C
- Java and C# are about 4 times slower than C

Why C++?

- Haskell and Fortran have potential to be marginally faster than C, but Haskell's compiler is not as good as it could theoretically be and Fortran is rather impractical (some call it *obsolete*)
- Plain C is relatively impractical (it used to be the best in its youth, see `https://www.youtube.com/watch?v=1S1fISh-pag`)
- C++ is a superset of C adding features that make it practical
- Many of these features are slower, but we don't have to use them in sections where speed is necessary
- Many programming languages' syntax is based on C or C++

Why *not* C++?

- Learning it takes a lot of time (we'll only learn the basics and the bits that are useful for data processing, don't get scared too bad)
- It took me two hours to learn Python well enough to make a simple game in it
- Writing code to perform a task in C++ usually takes longer than writing it in some other programming languages
- Compilation is annoyingly slow
- It allows for making errors that cannot happen in most programming languages
- Code must be recompiled for every platform
- Error messages can be ludicrously long
- Java is more practical, though the significance of this advantage is often a matter of discussion

Organisation

- Two hours long exercise, begun with a short lecture
- A homework after each lesson
- A larger project

Lectures' contents

- Getting a simple program to work
- Basic constructs
- Memory usage
- Working with text
- Objects
- 2D graphics
- GUIs (going beyond the command line)
- Using more CPU cores
- Installing and using libraries

Recommended software

Neither of these programs are required. If you are used to some alternative and you know to use it sufficiently, you may use it but I might not be able to help you or not have enough time to help you.

- Linux or Windows 10 with WSL (which allows you to run Linux programs on Windows)
- QtCreator
- GNU GCC compiler

Optional exercises

- It is very likely that some of you already know much of this and hearing basic things explained to beginners is useless to you
- If you already know some C++, you may do optional exercises instead of boredly staring or browsing Facebook so that this lecture would have some benefit to you
- There will be optional harder exercises and optional harder homeworks
- If you know how to get C++ programs working (the contents of this lesson), you can do this one right now:

$$\text{Calculate } \int_1^4 \frac{\sin x dx}{x}$$

C

- Created by Dennis Ritchie, released in 1972
- Named C because the author before created a programming language named B with the same idea, but it sucked
- Created to be as close to assembly (the language processors read) as possible but practical to humans
- Because it describes exactly what the processor does, few additional operations are done and the code is very fast
- Has very few functions and syntax constructs (the hard part about it is learning how the processors work)
- Can be run on all kinds of devices, from microcontrollers with several kilobytes of memory to building-sized supercomputers
- While you can do everything in C, many things are lengthy to do without C++ features
- Many programming languages are based on C, like C++, C#, Java, OpenCL, Arnold C, ...

C++

- Created by Bjarne Stroustroup, released in 1983
- Initially named *C with objects*, but later renamed to C++ because objects was not the only addition
- Created to be similar to C but also provide high level features for better program organisation
- Many of these higher level features have the same speed as C, but their relation to assembly is not so clear
- Its standard is over a thousand pages long, learning to use all features is lengthy
- Not all C programs are valid in C++, but these cases are rare

What has to be done

- Code can be written in many text editors, but IDEs (Integrated Development Environment) have the advantage of colouring keywords, completing text and warning about some errors
- The program is compiled into assembly that is readable by the processor
- The compiled program can run alone, aided only by the operating system (to prevent it from misbehaviour), but often some common functions are stored in shared libraries so that they would not be copied in each program
- For debugging purposes, there are programs that execute compiled or uncompiled code (with a significant drawback to execution speed), keeping useful information to track all kinds of information

IDE - QtCreator

- Open source (free full version, no ads), available on Linux, Windows, Mac
- Automatically indents code, highlights keywords, autocompletes function names, autocompletes brackets
- Can find typos, undefined variables, wrong function parametres, mismatched brackets, missing semicolons
- Can manage multi-file programs, controls compilation, allows you to control debugger
- Alternatives are Code::Blocks, MSVC and others
- Failsafe: Gedit, Kate, Notepad++, vim

IDE - QtCreator

- Example C code

```
#include <stdio.h>

int main() {
        printf("Oh yeah, I got it running!\n");
}
```

- `printf` is a function that writes its argument into the command line
- `include` gives access to the `printf` function (at the cost of slightly slower compilation)
- `main` is a function that is called when the program starts

Compilation - GNU GCC

- Reads the code, translates it into C–, translates that into assembly, significantly optimises it and creates an executable file
- Command is `gcc main.c -std=c99 -o main`, where `main.c` is the name of the program you wrote and `main` is the executable it creates
- If the program is in C++, the command is `g++ main.cpp -std=c++11 -o main`, where `main.cpp` is the name of the program you wrote
- Other very useful arguments are `-O3` which makes the try to optimise the program as much as possible (slows down the compilation) and `-g` which makes the program contain line numbers to make it easier to debug
- Failsafe: `http://cpp.sh`

Execution

- Run the program by writing `./itsname` into the command line in the folder where it is, where `itsname` is the program's name
- The program can be given arguments, writing something like `./itsname arg1 arg2`, where `arg1` and any number of further arguments are whatever you made your program read
- The program may expect further input that is written in the command line after the program is started
- The program may open a GUI (Graphical User Interface, or more naturally, a window) to control it; programs like this are usually not run from the command line

Homework

- Finish what you have not managed to do now
- Nothing else (don't get used to it)

Advanced homework

- Do it only if you are more skilled at this!
- Write a program that computes the area of an ellipse if its two semiaxes are given as arguments
- The easiest (and most didactic from the programming point of view) method to do it is to use numeric integration, using the equation $y = \frac{b}{a}\sqrt{1 - x^2}$