DM560

Introduction to Programming in C++

Course Organization
Motivations

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

[*Based on slides by Bjarne Stroustrup*]

Outline

Outline

Course Elements

- Lectures: (F) 11 classes
- Exercises: (TE) 4 classes
- Labs: (TL) 7 classes
- Exam: consisting in a number of assignments during the course + a final project
- Teacher Assistant: Troels Risum Vigsøe Frimer

Course Material

- Regularly check the public course web page:
  http://www.imada.sdu.dk/~marco/DM560/
  Slides, assignments, links
- Text book:
  [BS] Bjarne Stroustrup, *Programming – Principles and Practice Using C++ (Second Edition)*. Addison-Wesley 2014.
- You should have received an IMADA account for the ComputerLab

Expected Workload

- Prepare/attend/process lecture: 1+2+1 hour
- Prepare/attend exercise/lab: 2+2 hours
- Total: $11 \cdot 4 + (4 + 7) \cdot 4 + = 88$ hours (5 ECTS)
- Expected: $1/3 \cdot 37$ hours for 7 weeks = 86
- Assignments: $3 \cdot 2$ hours
- Project: 50 hours

Aims of the Course

- Teach/learn
  - Fundamental programming concepts
  - Key useful techniques
  - Basic Standard C++ facilities

- procedural programming
- data abstraction
- object-oriented programming
- generic programming
- (functional programming)

- After the course, you'll be able to
  - Write small C++ programs for scientific computations
  - Learn the basics of many other languages by yourself
  - Read much larger programs
- After the course, you will *not* (yet) be
  - An expert programmer
  - A C++ language expert
  - An expert user of advanced libraries

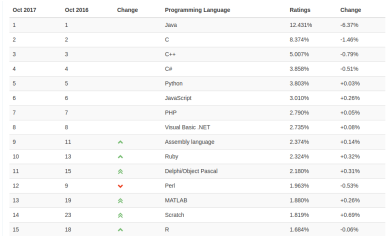
Motivations

Why would you want to program?

- Our civilization runs on software. The book provides a realm of examples from the engineering sector.
- Most programs do not run on things that look like a PC a screen, a keyboard, a box under the table
- In your case: programming as a tool for science, to carry out complex computations and simulations, to analyze large amount of data, eg from LHC.
- Foster understanding of problems and their solutions in problem solving.
  Only by expressing a correct program and constructing and testing a program can you be certain that your understanding is complete.
- Like Mathematics it can be an intellectual exercise that sharpens our ability to think.
- Programming is more concrete than most forms of math. It is a way to reach out and change the world.
- It can be fun

Why C++?

- Performance (aka, efficiency)
- Cross platform
- Expressiveness
- Correctness?
- TIOBE index



Oct 2017	Oct 2016	Change	Programming Language	Ratings	Change
1	1		Java	12.431%	-6.37%
2	2		C	8.374%	-1.46%
3	3		C++	5.007%	-0.79%
4	4		C#	3.858%	-0.51%
5	5		Python	3.803%	+0.03%
6	6		JavaScript	3.010%	+0.26%
7	7		PHP	2.790%	+0.05%
8	8		Visual Basic .NET	2.735%	+0.08%
9	11	⌃	Assembly language	2.374%	+0.14%
10	13	⌃	Ruby	2.324%	+0.32%
11	15	⏶⏶	Delphi/Object Pascal	2.180%	+0.31%
12	9	⌄	Perl	1.963%	-0.53%
13	19	⏶⏶	MATLAB	1.880%	+0.26%
14	23	⏶⏶	Scratch	1.819%	+0.69%
15	18	⌃	R	1.684%	-0.06%

Aspects of C++

- procedural
- object oriented
- functional
- generic

- statically typed
- natively compiled
- deterministic object lifetime
- pay for what you use
- compile time computation

Most of the programming concepts in C++ can be used directly in other languages, such as C, C#, Fortran, and Java.

Course Outline

Part I: The basics

- Types, variables, strings, console I/O, arithmetic operations, vectors functions, source files, classes
- control structures, error handling, design, implementation, and use of functions and user-defined types
- debugging and testing

Part II: Input and Output (I/O)

- File I/O, I/O streams
- Graphical output (2D)
- Graphical User Interface

Part III: Data structures and algorithms

- Memory management: free store, pointers, and arrays
- Data structures: lists, maps, sorting and searching, vectors. Templates
- The STL: containers and algorithms

Part IV: Broadening the view

- Software ideals and history
- Text processing, regular expression matching, numerics, embedded systems programming, testing, C, etc.

Course Outline

Appendices

A: C++ language summary

B: C++ standard library summary

C: Integrated development environment (IDE) and

D,E: Graphical user interface (GUI) library.

- Index (extensive)
- Glossary (short)

Exercises and Assignments

Designed according to:

- Realism: The concepts, constructs, and techniques can be used to build “industrial strength” programs
- Simplicity: The examples used are among the simplest realistic ones that illustrate the concepts, constructs, and techniques
- Your exercises and projects will provide more complex examples
- Scalability: The concepts, constructs, and techniques can be used to construct large, reliable, and efficient programs

Mutual Expectations

The teacher provides:

1. introduction to topics and concepts (as often as needed)
2. answers to your questions (in class and during breaks)
3. guidance to your learning by selecting topics and assigning exercises

The students provide:

1. questions, when something is unclear
2. seek contact to the TA when in need for help
3. preparation for lectures and exercise/lab sections

Your Tasks

- Read chapters before the lectures
- Review questions: good to learn terminology, and articulate ideas and concepts. Terminology is important to search on google and communication
- Drills: do all
- Exercises: the ones recommended
- Assignments
- Final Project

"Programming is learned by writing programs." —Brian Kernighan

Cooperate on Learning

Except for the work you hand in as individual contributions, you are strongly encouraged to collaborate and help each other (If in doubt if a collaboration is legitimate: ask!)

- Don't claim to have written code that you copied from others
- Don't give anyone else your code (to hand in for a grade)
- When you rely on the work of others, explicitly list all your sources – i.e. give credit to those who did the work
- Don't study alone when you don't have to: Form study groups
- Do help each other (without plagiarizing)
- Prepare questions
- The only stupid questions are the ones you wanted to ask but didn't

False Myths about Programmers

Programmer $\equiv$ Lonely male person

Instead:

Working in teams, social and communication skills are essential

Being a programmer requires having an intellectually challenging set of skills that are part of many important and interesting technical disciplines.

Someone totally ignorant of programming is reduced to believing in magic and is dangerous in many technical roles

Further Remarks

- Consider every web resource highly suspect until you have reason to believe better of it
- C++ has taken distance from C. This course is not C-first
- We use ISO standard C++
- Consider portability and the use of a variety of machine architectures and operating systems
- Command line; Example: compile, link, and execute a simple program consisting of two source files, `my_file1.cpp` and `my_file2.cpp`, using the GNU C++ compiler on a Unix or Linux system:
  ```
  bash c++ -o my_program my_file1.cpp my_file2.cpp ./my_program
  ```
- Knowing "why" is important for programming skills.
  Conversely, just memorizing lots of poorly understood rules and language facilities is limiting, a source of errors, and a massive waste of time: Manuals are there for that.
- Programming $\subset$ Computer Science
  CS is the systematic study of computing systems and computation
- We want correctness, reliability, affordability and maintainability

Outline

The Process of Developing a Program

- Analysis: What's the problem?
- Design: How do we solve the problem?
- Programming: Express the solution to the problem (the design) in code. Make sure that the code is correct and maintainable.
- Testing: Make sure the system works correctly under all circumstances required by systematically trying it out.

Feedback is an important element of the process

Discuss designs and programming techniques with friends, colleagues, potential users, and so on before you head for the keyboard.

A first program

```
// ...
int main()                              // main() is where a C++ program starts
{
    cout << "Hello, world!\n";          // output the 13 characters Hello, world!
                                        // followed by a new line
    return 0;                           // return a value indicating success
}
// quotes delimit a string literal
// NOTE: "smart" quotes ‘‘ ’’ will cause compiler problems.
//         so make sure your quotes are of the style " "
// \n is a notation for a new line
```

A first program

```
// a first program:
#include "std_lib_facilities.h"      // get the library facilities needed for now
int main()                           // main() is where a C++ program starts
{
    cout << "Hello, world!\n";       // output the 13 characters Hello, world!
                                     // followed by a new line
    return 0;                        // return a value indicating success
}
    // note the semicolons; they terminate statements
    // braces { } group statements into a block
    // main( ) is a function that takes no arguments ( )
    //     and returns an int (integer value) to indicate success or
    failure
```

A first program

```
// modified for Windows console mode:
#include "std_lib_facilities.h"      // get the facilities for this course
int main()                           // main() is where a C++ program starts
{
    cout << "Hello, world!\n";       // output the 13 characters Hello, world!
                                     // followed by a new line
    keep_window_open();              // wait for a keystroke
    return 0;                        // return a value indicating success
}
// without keep_window_open() the output window will be closed immediately
// before you have a chance to read the output (on Visual C++ 20xx)
```

Hello, world!

- Its purpose is to help you get used to your tools
  - Compiler
  - Program development environment
  - Program execution environment

- Type in the program carefully
  - After you get it to work, please make a few mistakes to see how the tools respond; for example:
    - Forget the header
    - Forget to terminate the string
    - Misspell return (e.g., retrun)
    - Forget a semicolon
    - Forget { or }
    - ...

Hello, world!

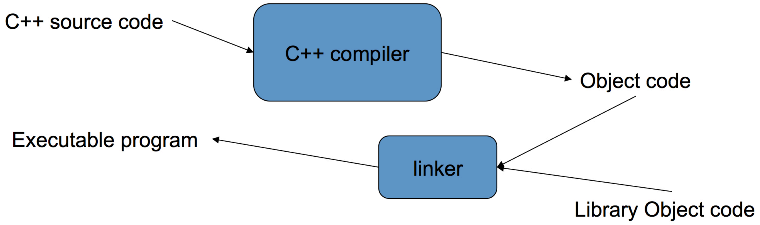
- Only `cout << ‘‘Hello, world!n’’;` directly does anything
- That’s normal
  Most of our code, and most of the systems we use simply exist to make some other code elegant and/or efficient
- Notation, libraries, and other support is what makes our code simple, comprehensible, trustworthy, and efficient
  the alternative is writing 1,000,000 lines of machine code
- This implies that we should not just “get things done” we should take great care that things are done elegantly, correctly, and in ways that ease the creation of more/other software

Code Style

Coding Style Matters!

- Code is read and modified repeatedly by others.
- Make their job more manageable by using good style.
- Remember, one of those “other people” might be you.

Compilation and linking



- You write C++ source code. Source code is (in principle) human readable
- The compiler translates what you wrote into object code (sometimes called machine code)
  Object code is simple enough for a computer to "understand"
- The linker links your code to system code needed to execute
  E.g., input/output libraries, operating system code, and windowing code
- The result is an executable program: E.g., a .exe file on windows or an a.out file on Unix

So what is programming?

- Conventional definitions
  - Telling a very fast moron exactly what to do
  - A plan for solving a problem on a computer
  - Specifying the order of a program execution
    - But modern programs often involve millions of lines of code
    - And manipulation of data is central
- Definition from another domain (academia)
  - A . . . program is an organized and directed accumulation of resources to accomplish specific . . . objectives . . .
    - Good, but no mention of actually doing anything
- The definition we will use
  - Specifying the structure and behavior of a program, and testing that the program performs its task correctly and with acceptable performance
    - Never forget to check that it works
- Software $\equiv$ one or more programs

Programming

Programming is fundamentally simple:
Just state what the machine is to do

So why is programming hard?

- We want "the machine" to do complex things
    - And computers are nitpicking, unforgiving, dumb beasts
- The world is more complex than we'd like to believe
    - So we don't always know the implications of what we want
- "Programming is understanding"
    - When you can program a task, you understand it
    - When you program, you spend significant time trying to understand the task you want to automate
- Programming is part practical, part theory
    - If you are just practical, you produce non-scalable unmaintainable hacks
    - If you are just theoretical, you produce toys

The next lecture

- Will talk about types, values, variables, declarations, simple input and output, very simple computations, and type safety.