

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

1. Course Organization
2. Developing a program

1. Course Organization

2. Developing a program

Introfase					1	1	1	1	1	2	1	1	1	1		<b>11</b>
Træningsfase (eks)						1		1			1				1	<b>4</b>
Træningsfase (lab)					1		1		1	1	1	1	1	1	1	<b>9</b>

- Lectures: (F) 11 classes
- Exercises: (TE) 4 classes
- Labs: (TL) 9 classes
- You have been registered as user of the IMADA ComputerLab
- Assessment: consisting of two assignments in class during the course + a final project
- Teacher Assistant: Troels Risum Vigsøe Frimer

- Regularly check the public course web page:  
<http://www.imada.sdu.dk/~marco/DM560/>  
Slides, exercises, links
- Text book:  
[BS] Bjarne Stroustrup, *Programming – Principles and Practice Using C++ (Second Edition)*.  
Addison-Wesley 2014.

# 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

- Teach/learn
  - Fundamental programming concepts
  - Key useful techniques
  - Basic Standard C++ facilities
- 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
- imperative/procedural programming
- data abstraction
- object-oriented programming
- generic programming
- (functional programming)

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

Sep 2018	Sep 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.430%	+4.75%
2	2		C	15.447%	+8.06%
3	5	▲	Python	7.653%	+4.67%
4	3	▼	C++	7.394%	+1.83%
5	8	▲	Visual Basic .NET	5.308%	+3.33%
6	4	▼	C#	3.295%	-1.48%
7	6	▼	PHP	2.775%	+0.57%
8	7	▼	JavaScript	2.131%	+0.11%
9	-	▲	SQL	2.062%	+2.06%
10	18	▲	Objective-C	1.509%	+0.00%
11	12	▲	Delphi/Object Pascal	1.292%	-0.49%
12	10	▼	Ruby	1.291%	-0.64%
13	16	▲	MATLAB	1.276%	-0.35%

## Aspects of C++

- imperative/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.

## 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
- Classes and Polymorphism

## 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.

## 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)

Exercise 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
- Scalability: The concepts, constructs, and techniques can be used to construct large, reliable, and efficient programs

Don't be too impatient for “realistic” examples.

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

- **Recommended reading** is material you are expected to know about the next time we meet  
**Additional reading** is material you can read already to get acquainted with concepts we will discuss later in the course
  - Review questions: good to learn terminology, and articulate ideas and concepts.  
Terminology is important to ask peers and communication
  - Drills: do all
  - Exercises: the ones recommended
  - Exercises in class
  - Assignments in class
  - Final Project
- “Programming is learned by writing programs.” — Brian Kernighan

# False Myths about Programmers

Programmer  $\equiv$  Lonely 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

- 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++ 14
- Consider portability and the use of a variety of machine architectures and operating systems
- 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](#)



- Use the lectures to understand which parts are relevant and essential and focus on those when reading the book afterwards
- Exercises are not published before the lectures because you need to learn how to design programs in class. Use after class to work on the exercises for the training session
- It is good to sketch first the program on a piece of paper. This is a useful step in solving a programming program. The design is a human brain activity not a computer activity and the best way not to skip this stage is to stay away from computer
- There will not be published solutions. This is mainly because there are many different solutions that work.
- We do not use an advanced IDE because our focus is on developing programming skills. When you have acquired them it will be easier to move to an advanced IDE.

1. Course Organization
2. Developing a program

# 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
```

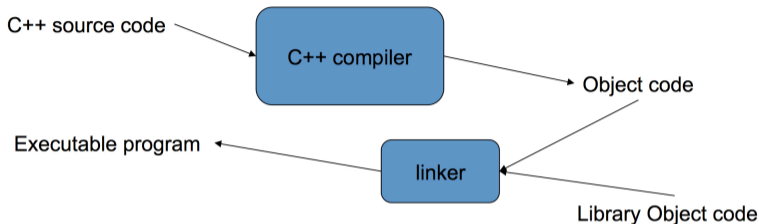
- Its purpose is to help you get used to your tools
  - Compiler
  - Program development environment
  - Program execution environment
- Type in the program carefully
  - After you get it to work, please make a few mistakes to see how the tools respond; for example:
    - Forget the header
    - Forget to terminate the string
    - Misspell return (e.g., retrun)
    - Forget a semicolon
    - Forget { or }
    - ...

- 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

## 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.





- 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

- 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 command line interface (CLI) as opposed to graphical user interface (GUI)

```
bash$ /usr/bin/g++ -c my_file1.cpp my_file2.cpp
bash$ g++ -o my_program my_file1.o my_file2.o
bash$ ./my_program
```

```
bash$ g++ -o my_program my_file1.cpp my_file2.cpp
bash$ ./my_program
```

- Example: our “Hello World”

```
bash$ g++ -o hello hello_world.cpp
bash$ ./my_program
```

- Viewing compiled code in readable assembly language:

```
objdump -d hello
```

```
bash$ g++ -S -o hello hello_world.cpp
```

```
bash$ less hello
```

# 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 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.