TDDD38/726G82:
Adv. Programming in C++
Course Introduction

Christoffer Holm

Department of Computer and information science

LINKÖPING UNIVERSITY

1. Course organization
2. C++
3. How to succeed

LINKÖPING UNIVERSITY

1 Course organization
2 C++
3 How to succeed

Course organization

Assumptions

In this course we assume that that you are:

- comfortable with procedural programming

Course organization

Assumptions

In this course we assume that that you are:

- comfortable with procedural programming
- familiar with object-oriented programming

Course organization

Assumptions

In this course we assume that that you are:

- comfortable with procedural programming
- familiar with object-oriented programming
- **motivated** to become a better programmer

Course organization

Assumptions

In this course we assume that that you are:

- comfortable with procedural programming
- familiar with object-oriented programming
- **motivated** to become a better programmer
- interested in a career involving programming

Course organization

Prerequisites

- No prior knowledge of C++?

Course organization

Prerequisites

- No prior knowledge of C++?
- OK if you are willing to learn basics on your own!

Course organization

Setting the stage

What is programming?

Course organization

What is programming?

- Programming isn't *just* about **computation** and computers. If that was the case, why would programming languages exist? Why would we care about code quality?
- A large part of programming is **abstraction** and the formal expression of ideas. Humans are good at intuition and language, but not so good at computation and details, while computers are the complete opposite. Programming is as much about bridging the gap between computation and human thinking as it is about computation.
- Code is usually read and modified more times than it is written. More often than not, programming is a collaborative effort and for large software projects to succeed, planning and **communication** is key. A good programmer doesn't just implement computational solutions to problems, they also write code which communicates the intent, purpose and execution of code to other programmers.

Course organization

What is this course about?

- Expand our toolbox

Course organization

What is this course about?

- Expand our toolbox
- Build solid abstractions

Course organization

What is this course about?

- Expand our toolbox
- Build solid abstractions
- Sharpen our understanding

Course organization

What is this course about?

- Expand our toolbox
- Build solid abstractions
- Sharpen our understanding
- Becoming the “guru”

Course organization

What this course is **not** about?

- Learning tools

Course organization

What this course is **not** about?

- Learning tools
- Memorizing solutions

Course organization

What this course is **not** about?

- Learning tools
- Memorizing solutions
- Writing specialized software

Course organization

What this course is **not** about?

- Learning tools
- Memorizing solutions
- Writing specialized software
- Why C++ is the best language ever

Course organization

What this course is **not** about?

- Learning tools
- Memorizing solutions
- Writing specialized software
- Why C++ is the best language ever
- Why other languages are “better” than C++

Course organization

The course

- A self-study course
- Lectures & seminars
- Office hours & teaching sessions
- http://www.ida.liu.se/~TDDD38/
- E-mail: TDDD38@ida.liu.se
- Examination

Course organization

The course

- A self-study course
  - Nothing is mandatory (except the examination)
  - It is *your* responsibility to learn
  - How you use the resources provided by the course is up to you
- Lectures & seminars
- Office hours & teaching sessions
- http://www.ida.liu.se/~TDDD38/
- E-mail: TDDD38@ida.liu.se
- Examination

Course organization

The course

- A self-study course
- Lectures & seminars
  - This course only has one session called a lecture (this one!)
  - All other lectures are called *seminars*
  - The reason for this is due to the slightly more interactive nature of my teaching style, and to break student assumptions about what a lecture is
- Office hours & teaching sessions
- `http://www.ida.liu.se/~TDDD38/`
- E-mail: `TDDD38@ida.liu.se`
- Examination

Course organization

The course

- A self-study course
- Lectures & seminars
- Office hours & teaching sessions
  - Both types of sessions are meant for: questions, discussion and help
  - Office hours are drop-in times when I am available at my office
  - Teaching sessions are sessions in classrooms where I am present
  - The teaching sessions is student-oriented, so questions and requests from students drives what happens.
- `http://www.ida.liu.se/~TDDD38/`
- E-mail: `TDDD38@ida.liu.se`
- Examination

Course organization

The course

- A self-study course
- Lectures & seminars
- Office hours & teaching sessions
- http://www.ida.liu.se/~TDDD38/
  - All material relevant for the course is found on the course website
  - Slides, exercises, reading material, recommended reading etc.
  - This website is completely open (and I will fight to keep it that way), since this material might be useful for you later on in your careers
- E-mail: TDDD38@ida.liu.se
- Examination

Course organization

The course

- A self-study course
- Lectures & seminars
- Office hours & teaching sessions
- http://www.ida.liu.se/~TDDD38/
- E-mail: TDDD38@ida.liu.se
  - This E-mail address leads to me
  - You are free to ask anything via E-mail, although the teaching quality over E-mail is not as good as in-person, so prefer office hours and teaching sessions for questions.
- Examination

Course organization

The course

- A self-study course
- Lectures & seminars
- Office hours & teaching sessions
- http://www.ida.liu.se/~TDDD38/
- E-mail: TDDD38@ida.liu.se
- Examination
  - This course consists of three modules
  - Each module is examined in isolation
  - There are two ways to complete a module:
    1. Pass the in-course assessment (DAK<n>) for module <n>
    2. Pass the specific module during the exam (DAT3)

Course organization

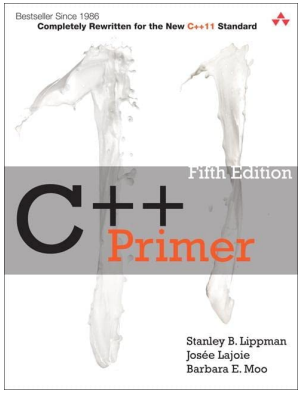
The course

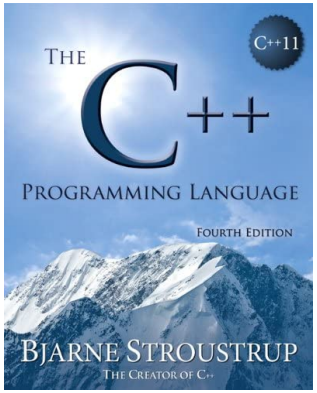
- A self-study course
- Lectures & seminars
- Office hours & teaching sessions
- http://www.ida.liu.se/~TDDD38/
- E-mail: TDDD38@ida.liu.se
- Examination
  - The DAK sessions are booked throughout the semester
  - DAT3 has one part per module
  - You only have to pass each module once
  - You choose how to pass a module (DAK<n> or DAT3)
  - Each module is graded by points
  - The final grade is decided by these points
  - see the course website for more information

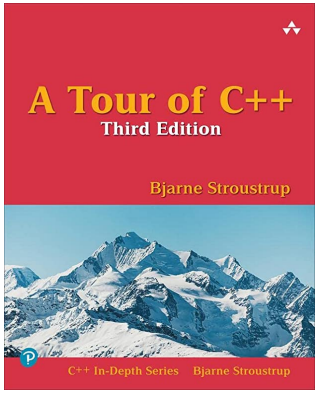
Course organization

Optional Literature

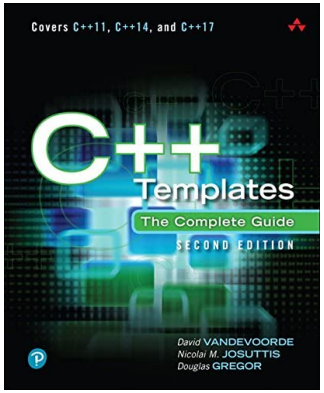
General books:

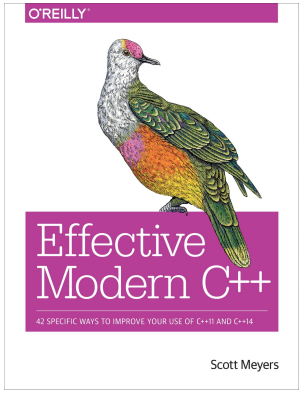


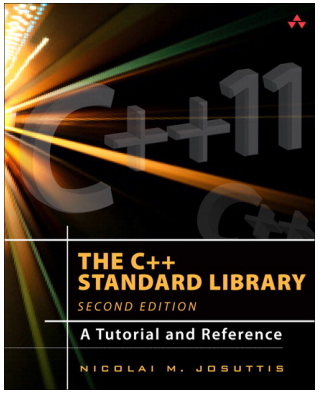




Specific books:







Course organization

Optional Literature

General books

- C++ Primer, 5th edition, Lippman, Lajoie, Moo
- The C++ Programming Language, 4th edition, Stroustrup
- A Tour of C++, 3rd edition, Stroustrup

Course organization

Optional Literature

Specific books

- C++ Templates: The Complete Guide, 2nd edition, Vandevoorde, Josuttis, Gregor
- Effective Modern C++, 1st edition, Meyers
- The C++ Standard Library: A Tutorial and Reference, 2nd edition, Josuttis

Course organization

Literature

https://en.cppreference.com/w/

Course organization

The course philosophy

- Programming is a *skill*

Course organization

The course philosophy

- Programming is a *skill*
- Skills develop with practice

Course organization

The course philosophy

- Programming is a *skill*
- Skills develop with practice
- Improvement does not happen in a vacuum

Course organization

The course philosophy

- Programming is a *skill*
- Skills develop with practice
- Improvement does not happen in a vacuum
- Understanding leads to efficiency

1 Course organization
2 C++
3 How to succeed

LINKÖPING UNIVERSITY

C++

Why C++?

- A complex language

C++

Why C++?

- A complex language
  - C++ is old (first releasted 1985) and ever-growing.
  - Because of this C++ has a lot of complicated rules that are not always intuitive at first inspection.
  - Understanding these rules requires us to understand programming, differences in hardware, various fields of programming and the challenges presented by designing a language.
  - The design of many other languages are based on C++ (many language also tries to improve the design, to make it more intuitive), so understanding C++ will generally help you understand other languages.

C++

Why C++?

- A complex language
- Both broad and deep

C++

Why C++?

- A complex language
- Both broad and deep
  - C++ can work very close to the hardware, but it also supports very high-level paradigms such as: functional programming, object-oriented programming, generic programming and procedural programming.
  - So C++ is quite a broad language, you can do most things in it.
  - These features are also highly customizable which means we can dig quite deep into how they work and how they interact with other features.

C++

Why C++?

- A complex language
- Both broad and deep
- Flexible

C++

Why C++?

- A complex language
- Both broad and deep
- Flexible
  - C++ has a big focus on freedom, meaning the language itself avoids forcing programmers to do things in specific ways.
  - C++ aims to support as many styles of programming as possible, and in particular: it is designed to allow for mixing idioms, paradigms and abstractions.
  - C++ is also designed to work independent of hardware, operating system and platform.

C++

Why C++?

- A complex language
- Both broad and deep
- Flexible
- High potential for efficiency

C++

Why C++?

- A complex language
- Both broad and deep
- Flexible
- High potential for efficiency
  - C++ compiles directly to assembly, which means it operates natively.
  - The language is also designed with the goal of keeping runtime costs as low as possible.
  - Most modern compilers have extremely mature and compentent optimization modules, partly because C++ is designed to be possible to optimize, and partly because C++ is mature and popular.
  - Note however that this doesn't mean C++ is *always* faster than other alternatives. To write efficient C++ code the programmer must be quite competent.

C++

Why C++?

- A complex language
- Both broad and deep
- Flexible
- High potential for efficiency
- Requires deep understanding

C++

Why C++?

- A complex language
- Both broad and deep
- Flexible
- High potential for efficiency
- Requires deep understanding
  - The language is *not* designed to be easy. It is designed to be efficient and flexible.
  - This means that in order to use C++ properly you must have a deep understanding of many things: memory and hardware, how high-level abstractions interact with each other and with the hardware, formal specifications of the language and software, and what costs are associated with each decision.

C++

Why C++?

- A complex language
- Both broad and deep
- Flexible
- High potential for efficiency
- Requires deep understanding
- Very popular

C++

Why C++?

- A complex language
- Both broad and deep
- Flexible
- High potential for efficiency
- Requires deep understanding
- Very popular
  - C++ has been used for many years, in many different fields and industries.
  - C++ is common in telecommunications, high-frequency trading, audio processing, the vehicle industry, the gaming industry, medical software, visualization and many many more.
  - Having deep knowledge of C++ means you have easier access to all of these fields in your future career.

C++

What is C++?

- Compiled

C++

What is C++?

- Compiled
- Standardized (syntax & semantics)

C++

What is C++?

- Compiled
- Standardized (syntax & semantics)
- Both high- and low-level

C++

What is C++?

- Compiled
- Standardized (syntax & semantics)
- Both high- and low-level
- Platform agnostic

C++

What is C++?

- Compiled
- Standardized (syntax & semantics)
- Both high- and low-level
- Platform agnostic
  The language does not assume anything about hardware or operating system (can even be written without an OS entirely, called the *freestanding* implementation of C++)

C++

Some background

- C++ started as an extension of C (1985)

C++

Some background

- C++ started as an extension of C (1985)
- One of the early adopters of object-oriented programming

C++

Some background

- C++ started as an extension of C (1985)
- One of the early adopters of object-oriented programming
- Became standardized in 1998

C++

Some background

- C++ started as an extension of C (1985)
- One of the early adopters of object-oriented programming
- Became standardized in 1998
- Is designed by committee

C++

Some background

- C++ started as an extension of C (1985)
- One of the early adopters of object-oriented programming
- Became standardized in 1998
- Is designed by committee
- Targets most tech-industries

C++

How to use C++?

What you need:

- An operating system

C++

How to use C++?

What you need:

- An operating system (probably)

C++

How to use C++?

What you need:

- An operating system (probably)
- A text editor

C++

How to use C++?

What you need:

- An operating system (probably)
- A text editor
- A compiler

C++

Compilers

Compilers	Linux	Mac	Windows
clang (`clang++`)	✓	✓	✓
GCC (`g++`)	✓	✓	✓[1]
MSVC (`cl.exe`)	×	×	✓

[1] Ported as MinGW

C++

Compilers

About clang:

- A very portable compiler (works almost everywhere)
- Very good at optimizing (built on LLVM)
- Is compatible with both GCC and MSVC
- Open source, has Apple as primary contributor
- Available during the exam

C++

Compilers

About GCC:

- The oldest C++ compiler still in wide use today
- Has a *very* good standard library implementation
- Is (usually) installed by default on Linux
- Open source, part of GNU
- Available during the exam

C++

Compilers

About MSVC:

- Arguably the easiest to install on Windows
- Part of the Visual Studio IDE
- Has very good support for C++20 modules
- Good at optimizing code for Windows
- Proprietary, owned and maintained by Microsoft

C++

Text Editors

- Emacs
- Vim
- Visual Studio Code
- etc.

C++

Text Editors

- Emacs
- Vim
- Visual Studio Code
- etc.

Use whatever you want, but these are available during the exam

C++

What about IDEs?

- CLion (All platforms)
- Visual Studio (Windows)

C++

What about IDEs?

- IDEs are very convenient and powerful tools that are built for efficiency and optimized workflows.
- However, all IDEs are complex beasts with many features that solve problems related to compilation and building, how header and source files are related, various configurations of the compiler and many other things. Each IDE solves these problems/configurations differently.
- Relying on a specific IDE will hinder you from fully understanding what the problems the IDE solves actually are. It also locks your understanding into one *specific* way of solving certain problems which hinders innovation and personal growth as a programmer.
- Over-reliance on specific tools can easily render you completely unable to be productive in situations when such tools are unavailable.
- Learning one IDE doesn't necessarily translates to productivity in another.
- But being productive without an IDE means you will for sure be productive regardless of which IDE you use.

C++

Other Tools

- Debugger

C++

Other Tools

- Debugger (recommended: gdb)

C++

Other Tools

- Debugger (recommended: gdb)
- Static Analyzer

C++

Other Tools

- Debugger (recommended: gdb)
- Static Analyzer (recommended: cppcheck)

C++

Other Tools

- Debugger (recommended: gdb)
- Static Analyzer (recommended: cppcheck)
- Dynamic Analyzer

C++

Other Tools

- Debugger (recommended: `gdb`)
- Static Analyzer (recommended: `cppcheck`)
- Dynamic Analyzer (recommended: `valgrind`)

C++

Other Tools

- Debugger (recommended: `gdb`)
- Static Analyzer (recommended: `cppcheck`)
- Dynamic Analyzer (recommended: `valgrind`)

There are many other tools, but mastering these will build understanding.

1 Course organization
2 C++
3 How to succeed

LINKÖPING UNIVERSITY

How to succeed

Summary

- Program **a lot**

How to succeed

Summary

- Program **a lot**
- Challenge yourself

How to succeed

Summary

- Program **a lot**
- Challenge yourself
- Question *every* line of code

How to succeed

Summary

- Program **a lot**
- Challenge yourself
- Question *every* line of code
- Do **NOT** take shortcuts

How to succeed

Programming as a skill

- It doesn’t matter how much theory you learn and how many lines of code you study, this will not make you a good programmer in practice.
- The only thing that truly makes you better at programming is actually programming.
- In particular it is important to come up with your own solutions.

How to succeed

Overcoming challenges

- Solving challenging problems is what drives growth.
- Knowing what you find challenging often reveals your gaps in skill and knowledge.
- Improvement means filling in your gaps as much as possible.
- Improvement is not time limited. This course doesn't just teach you C++, it is supposed to teach you *how* to improve.
- Knowing how to improve will be useful your entire career (and probably life).

How to succeed

Habits kills growth

- Software engineering is not about knowing every single pattern, nor is it about memorizing how specific things are done.
- Software engineering is weighing costs versus benefit and producing the best solution given a set of constraints.
- To become a expert engineer you must understand the costs, effects and reasons behind every single line of code you read and write.
- Therefore you should do your best to ensure that you understand every piece of code you see.
- But you must also understand that this isn't always possible, not for anyone, so don't blindly trust someone else's solutions (including mine).

How to succeed

LLM and AI usage

- I do not forbid AI for learning

How to succeed

LLM and AI usage

- I do not forbid AI for learning
- But I have strong reservations regarding its effectiveness as a learning tool

How to succeed

LLM and AI usage

- I do not forbid AI for learning
- But I have strong reservations regarding its effectiveness as a learning tool
- I suggest avoiding it as much as possible

How to succeed

LLM and AI usage

- I do not forbid AI for learning
- But I have strong reservations regarding its effectiveness as a learning tool
- I suggest avoiding it as much as possible
- But when you do use it, please *please* **PLEASE** question everything it produces.

How to succeed

What is expected of you

- Regular practice
  - There are exercises available each week
  - You might have other courses with assignments
  - Personal projects

How to succeed

What is expected of you

- Regular practice
- Identify gaps/weaknesses
  - what issues do you regularly run into?
  - Where do you get stuck?
  - What behaviours surprises you?
  - When do you feel the urge to look at solutions/StackOverflow or ask an LLM? I.e. when do you seek external knowledge?
  - All of these things reveal what you do and do not know

How to succeed

What is expected of you

- Regular practice
- Identify gaps/weaknesses
- Seek explanations (not answers)
  - Looking up *how* to do something will generally not be enough
  - *Why* you don't understand is more important for learning
  - Your goal should not just be: "I want to solve this issue" it should be: "I want to be able to construct my own solution"
  - But to do this you must me able to understand the issue, the mechanics and rules at play and how to work within them.

www.liu.se

LINKÖPING UNIVERSITY