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

Christoffer Holm

Department of Computer and information science

LINKÖPING UNIVERSITY

1 What is TDDD38?
2 About C++
3 How to use C++
4 Basic IO

LINKÖPING UNIVERSITY

What is TDDD38?

Assumptions/prerequisites

In this course I will assume that:

- You can program in a procedural language

What is TDDD38?

Assumptions/prerequisites

In this course I will assume that:

- You can program in a procedural language
- You are familiar with object-oriented programming

What is TDDD38?

Assumptions/prerequisites

In this course I will assume that:

- You can program in a procedural language
- You are familiar with object-oriented programming
- You are **motivated** to become a better programmer

What is TDDD38?

Assumptions/prerequisites

In this course I will assume that:

- You can program in a procedural language
- You are familiar with object-oriented programming
- You are **motivated** to become a better programmer
- You are interested in a career involving programming

What is TDDD38?

Assumptions/prerequisites

In this course I will assume that:

- You can program in a procedural language
- You are familiar with object-oriented programming
- You are **motivated** to become a better programmer
- You are interested in a career involving programming
- **Note:** I'm **NOT** assuming that you know C++ , however I will not teach the basics: that's up to you!

What is TDDD38?

What is programming *about*?

- Computation
- Abstraction
- Communication

What is TDDD38?

What is programming *about*?

If a programmer wrote a solution that no one understands,
is that a good solution?

What is TDDD38?

What is programming *about*?

- Programming isn't *just* the art of computation. If this was the case then we would still be programming in machine code.
- *Abstraction* is one of the most fundamental aspects involved in programming.
- Suitable abstractions allows us *humans* to simplify and reduce problems to a level where we are able to actually *reason* about them.

What is TDDD38?

What is programming *about*?

- Good abstraction should *simplify* and *recontextualize* problems to such a degree that *other people* can understand what happens in the code.
- “If a programmer wrote a solution that no one understands, is that actually a good solution?”
- For most people the answer to the question above is *no*. This means that programming is as much about *communication* as it is about *computation*.

What is TDDD38?

What is programming *about*?

- In this course we will proceed with the assumption that everyone here wants to write code that is easy to understand and easy to use.
- This will usually involve more work for us since we have to build good and *useful* abstractions.
- In this course we will explore the idea of *abstraction* by using C++ as a concrete example.

What is TDDD38?

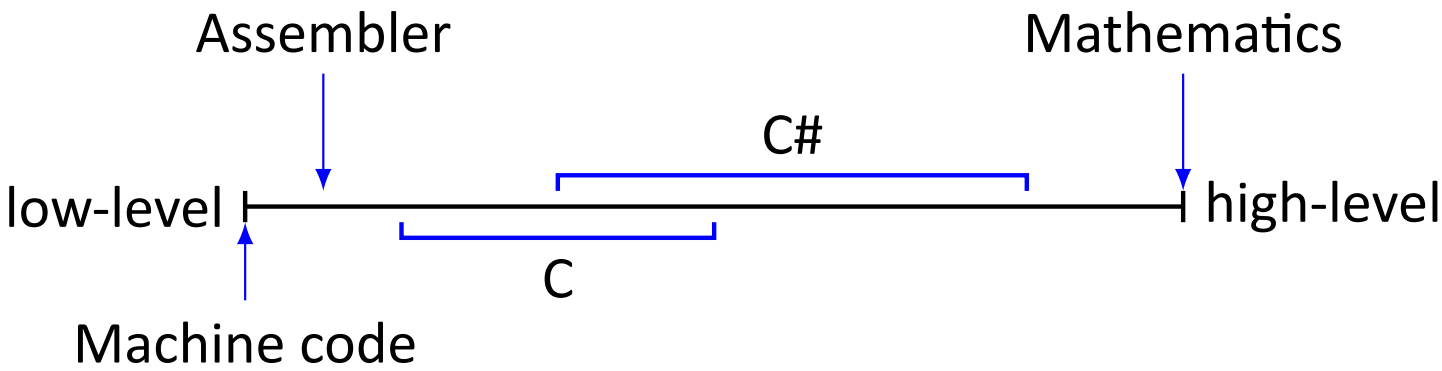
Why C++?

“C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off”

– Bjarne Stroustrup

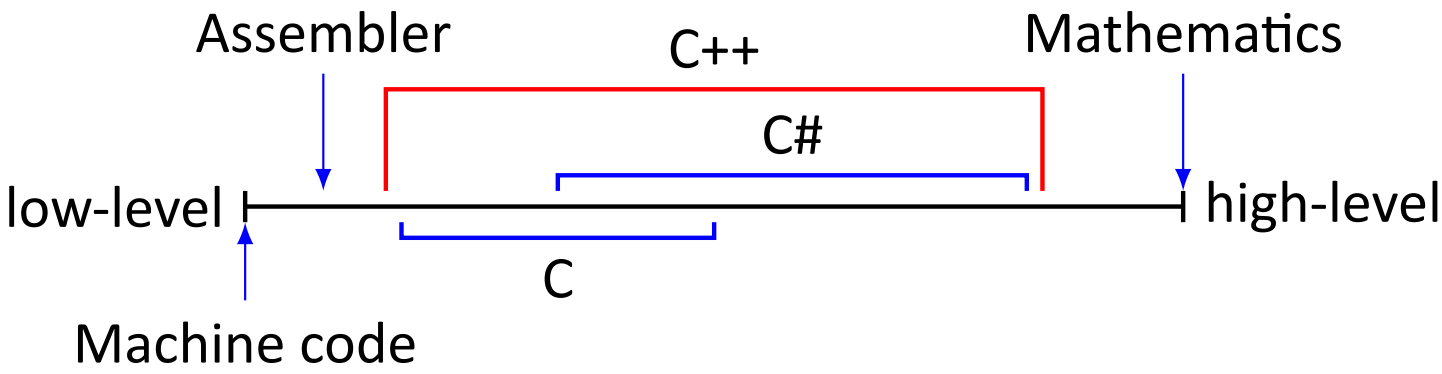
What is TDDD38?

Why C++?



What is TDDD38?

Why C++?



What is TDDD38?

Why C++?

- The aim for this course is to study C++ *in depth*. There are many reasons for doing this type of deep-dive:
  - It allows us to expand our perspectives on what is actually possible to do through abstractions.
  - Many of the ideas in C++ are present in other programming languages which means we can more easily learn new languages.
  - C++ forces you to understand and solve common compuation and abstraction problems, which means it is a great tool for evolving as a programmer.
  - And lastly: It is fun!

What is TDDD38?

Course information

- self-study
- lectures & seminars
- office hours
- Course web page:
  http://www.ida.liu.se/~TDDD38/
- E-mail: TDDD38@ida.liu.se
- Optional midterm test
- Examination

What is TDDD38?

Course information

- This course is a *self-study* course. This means it is *your* responsibilty to put in the needed work and time to learn the material.
- There are no deadlines during the course and the only *mandatory* aspect of the course is the exam.
- You have to manage your time and *utilize* the resources provided in the course.

What is TDDD38?

Course information

- During the course there are lectures & seminars booked in the schedule. These are essentially the same thing, but seminars tend to be a bit more interactive.
- It is *highly* recommended to attend these sessions since they are the primary source of information during the course (and it is more fun for me!)
- The slides published on the course web page have additional slides/information compared to the stripped versions used during the lectures/seminars.

What is TDDD38?

Course information

- There are also a lot of *office hours* sessions (supervision/handledningspass in TimeEdit).
- These are times during which I am available for drop-in visits in my office. This is your opportunity to get direct feedback on your work or to ask questions. Even if you don't have explicit questions you can still swing by and we'll probably find something to discuss!
- I also welcome questions not related to the course, but I can't guarantee that I'll be able answer those :)

What is TDDD38?

Course information

- The course web page contains *all* material.
- The primary resource on the web page is the “Seminars and exercises” page: there I publish all slides as well as exercises, reading material and other useful things for each lecture/seminar.
- The idea isn’t that you should solve all exercises. Instead it is up to you to identify what you need to practice and do the exercises that cover those things. You can of course discuss it with me as well!

What is TDDD38?

Course information

- There are also quizzes for most seminars on the web page which you can do to test how much you understood from the seminar.
- **Note:** These aren't all up-to-date with the current material, so don't expect all questions to be answerable by *just* attending the seminars. See it as practice for finding reliable information on your own!
- There is also the page "Examination" which contains information about the exam, as well as *most* exams given since 2016 with solutions.

What is TDDD38?

Course information

The primary learning resources during the course are:

- The lectures/seminars where we cover the most important/challenging topics of the course.
- The office hours where you are free to get help from a teacher (me!). People who come to these session never regret it! It is fun and helpful for everyone!
- You are always welcome to E-mail me any questions and I'll try to give a thorough answer as soon as possible.

What is TDDD38?

Course information

- After the first period of the semester there is an optional midterm test (Swedish: dugga) in the schedule.
- This midterm test serves two purposes: 1) it is an opportunity for you to test how far along you are in the course and 2) you get a chance to test out the exam system before the final exam.
- (There is also a “secret” purpose: it serves as a motivator for students to actually *study*)

What is TDDD38?

Course information

- The midterm test contains two questions from previous exams. These questions cover *roughly* the same topic.
- Exactly what the topic is varies from year to year, but it will be something we've covered at that point.
- The test is 1 hour and 45 minutes in length and you need to solve *one* assignment to pass.
- If you pass the test you automatically get full marks on one of the assignments in the final exam (this assignment will be clearly *marked* during the exam).

What is TDDD38?

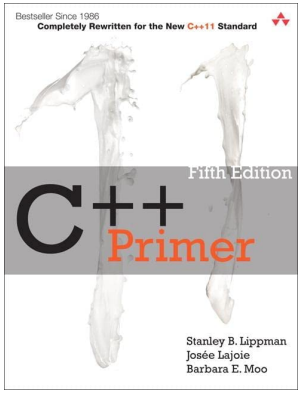
Course information

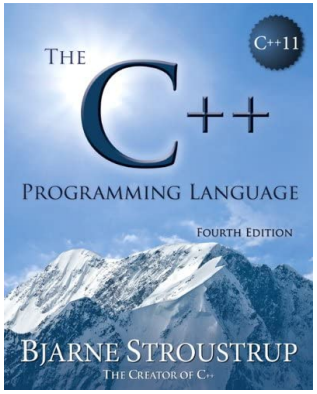
- The exam is a practical exam done in the SU computer labs (In the B-building).
- The exam *usually* consists of 4 programming assignments and 3 to 4 discussion assignments.
- The exam is based on *points*. Programming problems are worth between 4 to 6 points while discussion assigmments are worth 1 to 4 points.
- The total points of the exam sums to 25 points. A passing grade is given at 11 points (rounded *up*).

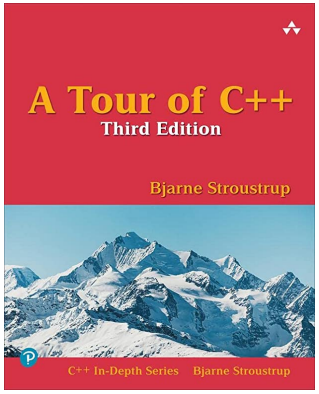
What is TDDD38?

Optional Literature

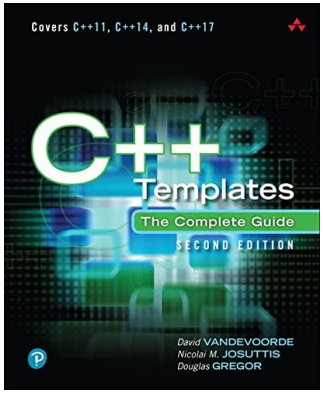
General books:

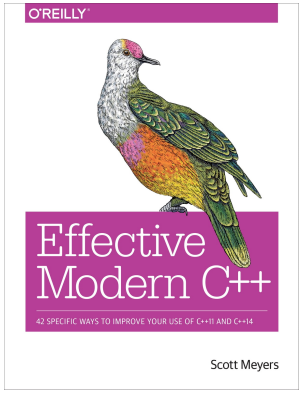


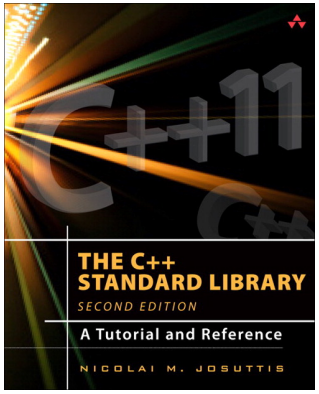




Specific books:







What is TDDD38?

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

What is TDDD38?

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

What is TDDD38?

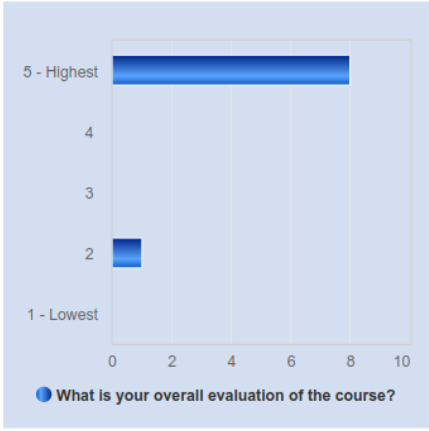
Literature

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

What is TDDD38?

Evalulation from last time

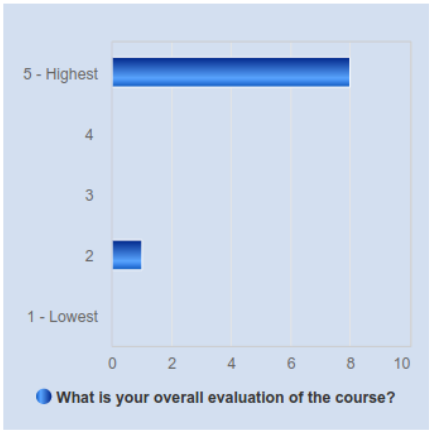
- Students want mandatory labs
- **Answer:** I'm the only teacher so it's not possible due to time constraints



What is TDDD38?

Evalulation from last time

- Students want the recorded lectures
- **Answer:** I never recorded the lectures during distance mode so *there are no* recordings



What is TDDD38?

Changes from last term

- updated course goals
- Reduced time spent on basic object orientation
- A lot of updates to the slides and seminars/lectures

What is TDDD38?

How to study for this course

- Go to lectures and seminars
- Read all exercises for a given seminar and solve the one(s) you find difficult.
- Make an *honest* attempt at solving those exercises after corresponding lecture/seminar
- If you get stuck: resist the urge to peek at the solution. Instead write down your question and either 1) try to find an answer yourself, or 2) talk to me!

What is TDDD38?

How to study for this course

- Spend consistent time *actually* programming. Theoretical/passive knowledge will not take you far enough to pass the exam.
- Most people are unable to learn how to apply programming concepts by *only* reading about it. You need to experiment with them yourself.
- *Be critical* about everything you read or hear (this includes info from me...). Make sure that you can **explain** why something is done.

1 What is TDDD38?
2 About C++
3 How to use C++
4 Basic IO

LINKÖPING UNIVERSITY

About C++

What is C++?

- general-purpose programming language

About C++

What is C++?

- general-purpose programming language
- compiled language

About C++

What is C++?

- general-purpose programming language
- compiled language
- based on C

About C++

What is C++?

- general-purpose programming language
- compiled language
- based on C
- powerful tools for abstractions

About C++

What is C++?

- general-purpose programming language
- compiled language
- based on C
- powerful tools for abstractions
- Designed by committee

About C++

What is C++?

- general-purpose programming language
- compiled language
- based on C
- powerful tools for abstractions
- Designed by committee
- Has a standard (ISO/IEC 14882:2020)

What is C++?

General rules

- C++ must be useful *now*
- Support different styles
- Forcing the programmer is bad

What is C++?

General rules

- All features must be affordable
- Usefulness > misuse prevention
- Composition of different parts

What is C++?

Technical rules

- No implicit violations of the type system
- User-defines types = built-in types
- Locality is good
- When in doubt, choose the easiest alternative

What is C++?

Low-level rules

- Leave only assembler below C++
- Don’t pay for what you don’t use (zero-overhead)

What is C++?

Style

- No standardized style
- I will use a style, but you may use your own
- Cpp Core Guidelines
- This course will focus on C++17 and later
- Usage of C features will be penalized

1 What is TDDD38?
2 About C++
3 How to use C++
4 Basic IO

How to use C++

Tools required

- Operating System (duh!)

How to use C++

Tools required

- Operating System (duh!)
- Editor

How to use C++

Tools required

- Operating System (duh!)
- Editor
- Compiler

How to use C++

Additional tools

- Debugger

How to use C++

Additional tools

- Debugger
- Static analyzers

How to use C++

Additional tools

- Debugger
- Static analyzers
- Runtime analyzers

How to use C++

Compilers

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

[1] Ported as MinGW

How to use 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

How to use 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

How to use C++

Compilers

About MSVC:

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

How to use C++

Editors

- Emacs
- Vim
- Visual Studio Code
- etc.

How to use C++

Editors

- Emacs
- Vim
- Visual Studio Code
- etc.

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

How to use C++

IDEs

- CLion (All platforms)
- Eclipse (All platforms)
- Visual Studio (Windows)
- XCode (Mac)

How to use C++

IDEs

- CLion (All platforms)
- Eclipse (All platforms)
- Visual Studio (Windows)
- XCode (Mac)

IDEs can be nice, but you shouldn't rely on them *too much*

How to use C++

How to compile

GCC:

```
$ emacs file.cc                          # write code
$ g++ -o myprogram file.cc               # compile
$ ./myprogram                            # run program
```

How to use C++

How to compile

clang:

```
$ emacs file.cc                          # write code
$ clang++ -o myprogram file.cc           # compile
$ ./myprogram                            # run program
```

How to use C++

How to compile

MSVC:

```
$ emacs file.cc                          # write code
$ cl.exe /Fe myprogram.exe file.cc # compile
$ myprogram.exe                          # run program
```

How to use C++

Important terms

- Compile time (static)
- Runtime (dynamic)
- Implementation defined behaviour
- Undefined behaviour

How to use C++

Compile time

- C++ is a *compiled* language. This means that the *compiler* will translate your code into machine code.
- During this step the compiler might perform pre-calculations or make certain decisions (sometimes called *static behaviour*) based on your source code.
- These calculations are done during *compile time*

How to use C++

Runtime

- Once your compiler has transformed your code into machine code it is possible to actually *run* the program.
- Any calculations and decisions (sometimes called *dynamic behaviour*) done in this step happen during *runtime*.
- In C++ the line between *compile time* and *runtime* is somewhat blurred.

How to use C++

Example

- It is important to understand the difference between compile time and runtime.
- A typical example would be that the concept of *types*.
- Data types enforce how the compiler generates the machine code, but the machine code itself doesn't contain any information about data types.
- This means that during runtime, all information about types has been lost.

How to use C++

Implementation defined behaviour

- The C++ *standard* wants C++ to be *efficient* on *all* platforms/architectures
- This means that certain things might need to differ depending on the architecture.
- *Implementation defined behaviour* refers to such cases where the standard says it is up to the implementors to define the behaviour.
- **Example:** On most modern systems a *byte* is 8 bits, but there are older embedded systems where a byte is 7 bits. This means that it is *implementation defined* how many bytes are stored in a byte.

How to use C++

Undefined behaviour

- There are syntactically valid statements/expressions in C++ which have no well-behaved semantics.
- These cases are refered to as *undefined behaviour*.
- **Example:** If we have an array of data with 3 elements. It is syntactically correct to access *any* index, but if we try to access for example index 537 there is no way for us or the compiler to know what will happen.
- This means that doing out-of-bounds accesses is *undefined behaviour*.

How to use C++

Undefined behaviour vs. Implementation defined behaviour

- Undefined behaviour usually means that *this behaviour is nonsense* meaning it is not something the compiler doesn't have to handle gracefully.
- Implementation defined behaviour is something that the standard doesn't specify, but it *forces* the compiler implementor to actually do something *well behaved*.
- **Note:** Based on the definition, invoking *undefined behaviour* can result in anything while invoking *implementation defined behaviour* will result in consistent behaviour *while operating on the same CPU*.

1 What is TDDD38?
2 About C++
3 How to use C++
4 Basic IO

LINKÖPING UNIVERSITY

Basic IO

NOT Hello World!

```
1 #include <iostream>
2 int main()
3 {
4   std::cout << "NOT Hello World!" << std::endl;
5   return 0;
6 }
```

Basic IO

```
1  #include <iostream>
2  #include <string>
3  using std::cout;
4  using std::endl;
5  int main()
6  {
7    cout << "What is your name? ";
8
9    std::string name{};
10   std::cin >> name;
11
12   cout << "Your name is " << name << endl;
13 }
```

Basic IO

```
1  #include <iostream>
2  #include <string>
3  using std::cout;
4  using std::endl;
5  int main()
6  {
7    int number{};
8    cout << "Enter a number: ";
9    std::cin >> number;
10   if (number >= 0)
11   {
12     cout << "Your number is positive!" << endl;
13   }
14 }
```

Basic IO

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5    int counter{0}, sum{}, number{};
6    cout << "Enter your numbers: ";
7    while (counter < 5)
8    {
9      cin >> number;
10     sum += number;
11     ++counter;
12   }
13   cout << "The sum is: " << sum << endl;
14 }
```

Basic IO

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5    int sum{}, number{};
6    cout << "Enter your numbers: ";
7    for (int i{0}; i < 5; ++i)
8    {
9      cin >> number;
10     sum += number;
11   }
12   cout << "The sum is: " << sum << endl;
13 }
```

Basic IO

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5    int number{};
6    do
7    {
8      cout << "Enter number [0-10]: ";
9      cin >> number;
10   } while (number < 0 || number > 10);
11 }
```

Basic IO

Buffered input

```
1 int x;
2 std::cout << "Enter integer: ";
3 std::cin >> x;
4 std::cout << "You entered: " << x << std::endl;
5 std::cout << "Enter another integer: ";
6 std::cin >> x;
7 std::cout << "You entered: " << x << std::endl;
```

Basic IO

Buffered input

```
$ ./a.out
Enter integer: 3
You entered: 3
Enter another integer: 5
You entered: 5
```

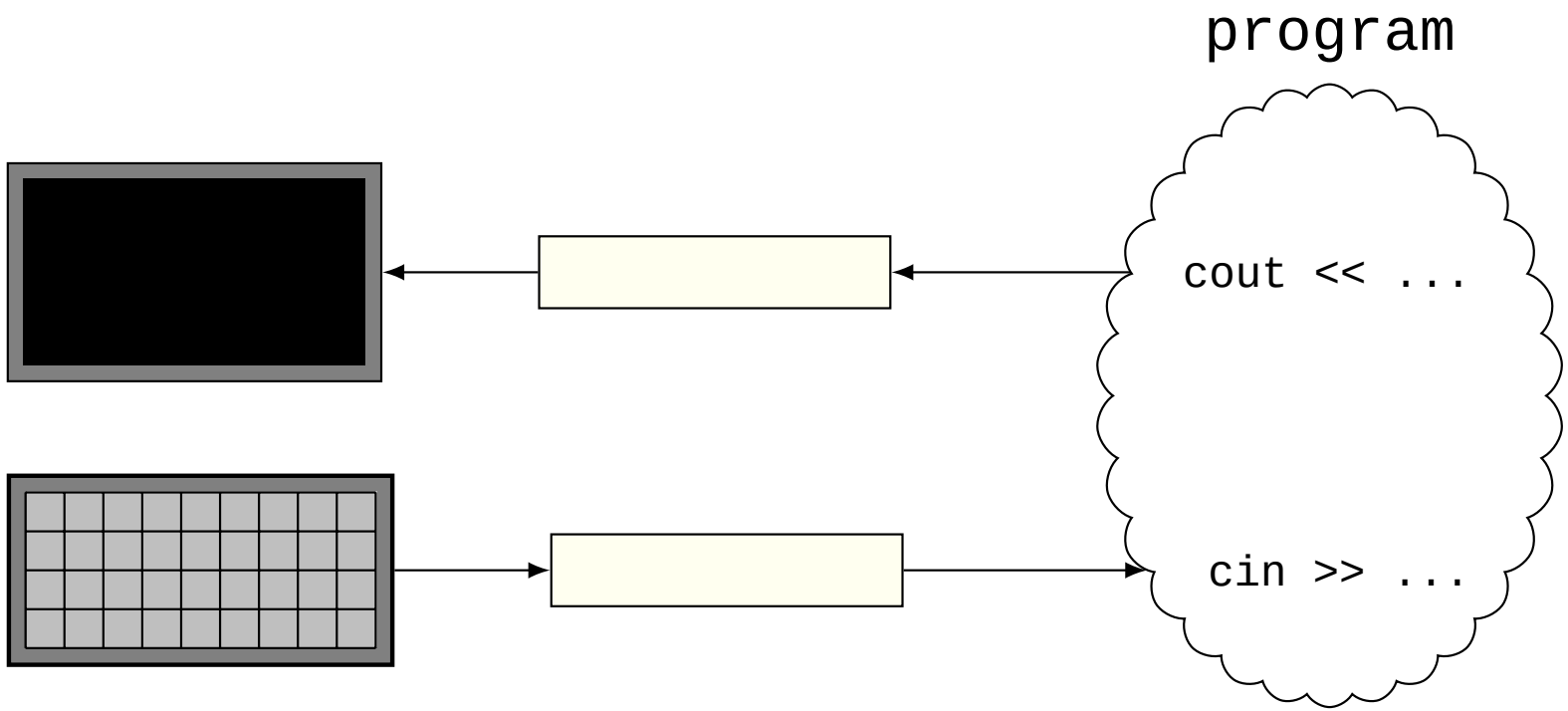
Basic IO

Buffered input

```
$ ./a.out
Enter integer: 3 5
You entered: 3
Enter another integer: You entered: 5
```

Basic IO

The complete picture



Basic IO

Buffered input

```
1 int x;
2 std::cout << "Enter integer: ";
3 std::cin >> x;
4 std::cout << "You entered: " << x << std::endl;
5 std::cin.ignore(1000, '\n');
6 std::cout << "Enter another integer: ";
7 std::cin >> x;
8 std::cout << "You entered: " << x << std::endl;
```

Basic IO

Buffered input

```
$ ./a.out
Enter integer: 3
You entered: 3
Enter another integer: 5
You entered: 5
```

Basic IO

Buffered input

```
$ ./a.out
Enter integer: 3 5
You entered: 3
Enter another integer: 7 8 9
You entered: 7
```

Basic IO

The complete picture

- Both `std::cout` and `std::cin` are *buffered*.
- This means that whenever we do an IO operation it is performed on the *buffer* first, and if the buffer is empty or incomplete *then and only then* will we perform an actual IO operation.
- Everyting entered during a *read* operation will be stored in the input buffer.
- Then, as long as there are things in the buffer, `std::cin` will operate on the buffer.
- You can clear the buffer with `std::cin.ignore(...)`.

www.liu.se

LINKÖPING UNIVERSITY