1. Introduction

Computer Science: Definition and History, Algorithms, Turing Machine, Higher Level Programming Languages, Tools, The first C++ Program and its Syntactic and Semantic Ingredients

What is Computer Science?

- The science of **systematic processing of informations**, . . .
- . . . particularly the automatic processing using digital computers.

(Wikipedia, according to “Duden Informatik”)

Informatics $\neq$ Science of Computers

*Computer science is not about machines, in the same way that astronomy is not about telescopes.*

Computer Science ⊆ Informatics

- Computer science is also concerned with the development of fast computers and networks...
- ...but not as an end in itself but for the systematic processing of informations.

This course

- Systematic problem solving with algorithms and the programming language C++.
- Hence: *not only* but also programming course.

Algorithm: Fundamental Notion of Computer Science

- Algorithm:
  - Instructions to solve a problem step by step
  - Execution does not require any intelligence, but precision (even computers can do it)
  - according to *Muhammed al-Chwarizmi*, author of an arabic computation textbook (about 825)

Computer Science ≠ Computer Literacy

- Computer literacy: *user knowledge*
  - Handling a computer
  - Working with computer programs for text processing, email, presentations...

Computer Science *Fundamental knowledge*

- How does a computer work?
- How do you write a computer program?
Oldest Nontrivial Algorithm

Euclidean algorithm (from the *elements* from Euklid, 3. century B.C.)

- Input: integers $a > 0$, $b > 0$
- Output: gcd of $a$ und $b$

While $b \neq 0$
  - If $a > b$ then $a \leftarrow a - b$
  - else: $b \leftarrow b - a$

Result: $a$.

Live Demo: Turing Machine

Euklid in the Box

Computers – Concept

A bright idea: universal Turing machine (Alan Turing, 1936)
Computer – Implementation

- Z1 – Konrad Zuse (1938)
- ENIAC – John Von Neumann (1945)

Von Neumann Architecture

- Memory (RAM) for programs and data
- Processor (CPU) to process programs and data
- I/O components to communicate with the world

Memory for data and program

- Sequence of bits from \{0, 1\}.
- Program state: value of all bits.
- Aggregation of bits to memory cells (often: 8 Bits = 1 Byte)
- Every memory cell has an address.
- Random access: access time to the memory cell is (nearly) independent of its address.

Processor

- The processor (CPU)
  - executes instructions in machine language
  - has an own "fast" memory (registers)
  - can read from and write to main memory
  - features a set of simplest operations = instructions (e.g. adding to register values)
Computing speed

In the time, on average, that the sound takes to travel from my mouth to you ...

\[30 \text{ m} \cong \text{more than 100,000,000 instructions}\]

A contemporary desktop PC can process more than 100 millions instructions \(^1\)

---

Programming

- With a *programming language* we issue commands to a computer such that it does exactly what we want.
- The sequence of instructions is the *(computer) program*

---

Why programming?

- Do I study computer science or what ...
- There are programs for everything ...
- I am not interested in programming ...
- Because computer science is a mandatory subject here, unfortunately...
- . . .

---

*Mathematics used to be the lingua franca of the natural sciences on all universities. Today this is computer science.*

*Lino Guzzella, president of ETH Zurich, NZZ Online, 1.9.2017*
**This is why programming!**

- Any understanding of modern technology requires knowledge about the fundamental operating principles of a computer.
- Programming (with the computer as a tool) is evolving a cultural technique like reading and writing (using the tools paper and pencil).
- Programming is *the* interface between engineering and computer science – the interdisciplinary area is growing constantly.
- Programming is fun!

---

**Programming Languages**

- The language that the computer can understand (machine language) is very primitive.
- Simple operations have to be subdivided into many single steps.
- The machine language varies between computers.

---

**Higher Programming Languages**

- Can be represented as program text that can be *understood* by humans.
- Is *independent* of the computer model → Abstraction!

---

**Programming languages – classification**

Differentiation into

- Compiled vs. interpreted languages
  - *C++*, *C#*, *Pascal*, *Modula*, *Oberon*, *Java* vs. *Python*, *Tcl*, *Matlab*
- *Higher* programming languages vs. *Assembler*
- *Multi-purpose* programming languages vs. single purpose programming languages
- *Procedural*, *object oriented*, functional and logical languages.
Why **C++**?

Other popular programming languages: Java, C#, Objective-C, Modula, Oberon, Python . . .

General consensus:
- "The" programming language for systems programming: C
- C has a fundamental weakness: missing (type) safety

---

**C++**

- **C++** equips C with the power of the abstraction of a higher programming language
- In this course: C++ introduced as high level language, not as better C
- Approach: traditionally procedural → object-oriented.

---

**Deutsch vs. C++**

**Deutsch**

*Es ist nicht genug zu wissen, man muss auch anwenden.*  
*(Johann Wolfgang von Goethe)*

**C++**

// computation
```cpp
int b = a * a;  // b = a^2
b = b * b;      // b = a^4
```
Syntax and Semantics

Like our language, programs have to be formed according to certain rules.
- **Syntax:** Connection rules for elementary symbols (characters)
- **Semantics:** interpretation rules for connected symbols.

Corresponding rules for a computer program are simpler but also more strict because computers are relatively stupid.

Syntax and Semantics of C++

**Syntax**
- What is a C++ program?
- Is it grammatically correct?

**Semantics**
- What does a program mean?
- What kind of algorithm does a program implement?

C++: Kinds of errors illustrated with German sentences

- Das Auto fuhr zu schnell.
- Rot das Auto ist.
- Man empfiehlt dem Dozenten nicht zu widersprechen
- Sie ist nicht gross und rothaarig.
- Die Auto ist rot.
- Das Fahrrad gallopiert schnell.
- Manche Tiere riechen gut.

Syntax and semantics of C++


- is the “law” of C++
- defines the grammar and meaning of C++ programs
- contains new concepts for advanced programming . . .
- . . . which is why we will not go into details of such concepts
**Programming Tools**

- **Editor**: Program to modify, edit and store C++ program texts
- **Compiler**: program to translate a program text into machine language
- **Computer**: machine to execute machine language programs
- **Operating System**: program to organize all procedures such as file handling, editor-, compiler- and program execution.

**Language constructs with an example**

- **Comments/layout**
- **Include directive**
- **the main function**
- **Values effects**
- **Types and functionality**
- **literals**
- **variables**
- **constants**
- **identifiers, names**
- **objects**
- **expressions**
- **L- and R- values**
- **operators**
- **statements**

**The first C++ program Most important ingredients...**

```cpp
// Program: power8.cpp
// Raise a number to the eighth power.
#include <iostream>
int main(){
    // input
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a; // statements: Do something (read in a)
    // computation
    int b = a * a; // b = a^2 // expressions: Compute a value (a^2)
    b = b * b; // b = a^4
    // output b * b, i.e., a^8
    std::cout << a << "^8 = " << b * b << "\n";
    return 0;
}
```

**Behavior of a Program**

At compile time:
- program accepted by the compiler (syntactically correct)
- Compiler error

During runtime:
- correct result
- incorrect result
- program crashes
- program does not terminate (endless loop)
// Program: power8.cpp
// Raise a number to the eighth power.
#include <iostream>
int main()
{
    // input
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a;
    // computation
    int b = a * a; // b = a^2
    b = b * b; // b = a^4
    // output b * b, i.e., a^8
    std::cout << a << ^8 = " << b * b << "\n";
    return 0;
}
Include Directives

C++ consists of
- the core language
- standard library
  - in-/output (header iostream)
  - mathematical functions (cmath)
  - ...

#include <iostream>

- makes in- and output available

The main Function

the main-function
- is provided in any C++ program
- is called by the operating system
- like a mathematical function ...
  - arguments
  - return value
- ... but with an additional effect
  - Read a number and output the 8th power.

Statements: Do something!

int main() {
  // input
  std::cout << "Compute a^8 for a =? ";
  int a;
  std::cin >> a;
  // computation
  int b = a * a; // b = a^2
  b = b * b; // b = a^4
  // output b * b, i.e., a^8
  std::cout << a << "^8 = " << b * b << "\n";
  return 0;
}

Statements

- building blocks of a C++ program
- are executed (sequentially)
- end with a semicolon
- Any statement has an effect (potentially)
Expression Statements

- have the following form:
  
  \[
  \text{expr;}
  \]

  where \text{expr} is an expression

- Effect is the effect of \text{expr}, the value of \text{expr} is ignored.

Example: \text{b = b*b;}

Return Statements

- do only occur in functions and are of the form
  
  \[
  \text{return expr;}
  \]

  where \text{expr} is an expression

- specify the return value of a function

Example: \text{return 0;}

Statements – Effects

```c
int main() {
  // input
  std::cout << "Compute a^8 for a =? ";
  int a;
  std::cin >> a;
  // computation
  int b = a * a; // b = a^2
  b = b * b; // b = a^4
  // output b * b, i.e., a^8
  std::cout << a << "^8 = " << b * b << "\n";
  return 0;
}
```

Values and Effects

- determine what a program does,
- are purely semantical concepts:
  - Symbol 0 means Value 0 ∈ \mathbb{Z}
  - \text{std::cin >> a;} means effect “read in a number”
- depend on the program state (memory content, inputs)
int main() {
    // input
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a;
    // computation
    int b = a * a;  // b = a^2
    b = b * b;  // b = a^4
    // output b, i.e., a^8
    std::cout << a << "^8 = " << b * b << "\n";
    return 0;
}

Declaration Statements
- introduce new names in the program,
- consist of declaration and semicolon

Example: int a;

- can initialize variables

Example: int b = a * a;

Types and Functionality
- int: C++ integer type
- corresponds to \((\mathbb{Z}, +, \times)\) in math

In C++ each type has a name and
- a domain (e.g. integers)
- functionality (e.g. addition/multiplication)

Fundamental Types
- C++ comprises fundamental types for
  - integers (int)
  - natural numbers (unsigned int)
  - real numbers (float, double)
  - boolean values (bool)
  - ...

Statements – Variable Definitions

// input
std::cout << "Compute a^8 for a =? ";
int a;
std::cin >> a;

// computation
int b = a * a;  // b = a^2
b = b * b;  // b = a^4

// output b, i.e., a^8
std::cout << a << "^8 = " << b * b << "\n";
### Literals

- Represent constant values
- Have a fixed *type* and *value*
- Are "syntactical values".

Examples:
- `0` has type `int`, value `0`.
- `1.2e5` has type `double`, value `1.2 \cdot 10^5`.

### Variables

- Represent (varying) values,
- Have
  - *name*
  - *type*
  - *value*
  - *address*
- Are "visible" in the program context.

**Example**

```cpp
int a; defines a variable with
- name: `a`
- type: `int`
- value: (initially) undefined
- Address: determined by compiler
```

### Objects

- Represent values in main memory
- Have *type*, *address* and *value* (memory content at the address)
- Can be named (variable) ...
- ... but also anonymous.

### Identifiers and Names

(Variable-)names are identifiers
- Allowed: `A,...,Z; a,...,z; 0,...,9;_`
- First symbol needs to be a character.

There are more names:
- `std::cin` (Qualified identifier)

### Remarks

A program has a *fixed* number of variables. In order to be able to deal with a variable number of value, it requires "anonymous" addresses that can be address via temporary names.
Expressions: compute a value!

- represent *Computations*
- are either primary (b)
- or composed (b*b)...
- ...from different expressions, using operators
- have a type and a value

Analogy: building blocks

---

**Expressions**

- represent *computations*
- are primary or composite (by other expressions and operations)

```
int a; 
// computation
int b = a * a; // b = a^2
b = b * b; // Two times composed expression

// output b + b, i.e., a^8
std::cout << a^8 = b * b << ".n";
```

- can be put into parantheses

- a * a is equivalent to (a * a)

---

**Expressions have type, value und effect (potentially).**

**Example**

```
int a; 
int b;
```

- type: int (type of the operands)
- Value: product of a and a
- Effect: none.

---

**Example**

```
int a; 
int b;
```

- type: int (Typ der Operanden)
- Value: product of b and b
- Effect: assignment of the product value to b

The type of an expression is fixed but the value and effect are only determined by the evaluation of the expression.
L-Values and R-Values

// input
std::cout << "Compute \texttt{a}^8 \text{ for } a =? \";
int a;
std::cin >> a; // L-value (expression + address)

// computation
int b = a * a; // \texttt{b} = \texttt{a}^2
b = b * b; // \texttt{b} = \texttt{a}^4

// output \texttt{b} * \texttt{b}, i.e., \texttt{a}^8
std::cout << a << "^8 = " << b * b << "\n";
return 0;

L-Values and R-Values

L-Wert ("Left of the assignment operator")
- Expression with \textit{address}
- \textit{Value} is the content at the memory location according to the type of the expression.
- L-Value can change its value (e.g. via assignment)

Example: variable name

R-Wert ("Right of the assignment operator")
- Expression that is no L-value
  - Example: literal 0
- Any L-Value can be used as R-Value (but not the other way round)
- An R-Value \textit{cannot change} its value

Operators and Operands

// input
std::cout << "Compute \texttt{a}^8 \text{ for } a =? \";
int a;
std::cin >> a; // right operand (variable name)

// computation
int b = \texttt{a}^2
b = \texttt{b} * \texttt{b}; // \texttt{b} = \texttt{a}^4

// output \texttt{\texttt{b} \ast \texttt{b}}, i.e., \texttt{a}^8
std::cout << a << "^8 = " << b * b << "\n";
return 0;
Operators

- Operators combine expressions (operands) into new composed expressions.
- They specify for the operands and the result the types and if the have to be L- or R-values.
- They have an arity.

Multiplication Operator *

- Expect two R-values of the same type as operands (arity 2).
- "Returns the product as R-value of the same type", that means formally:
  - The composite expression is an R-value; its value is the product of the value of the two operands.

Examples: \( a \ast a \) and \( b \ast b \)

Assignment Operator =

- Left operand is L-value,
- Right operand is R-value of the same type.
- Assigns to the left operand the value of the right operand and returns the left operand as L-value.

Examples: \( b = b \ast b \) and \( a = b \)

Attention, Trap!

The operator \( = \) corresponds to the assignment operator of mathematics (:=), not to the comparison operator (=).

Input Operator >>

- Left operand is L-Value (input stream).
- Right operand is L-Value.
- Assigns to the right operand the next value read from the input stream, *removing it from the input stream* and returns the input stream as L-value.

Example `std::cin >> a` (mostly keyboard input)

- Input stream is being changed and must thus be an L-Value.
Output Operator $\ll$

- left operand is L-Value (output stream)
- right operand is R-Value
- outputs the value of the right operand, appends it to the output stream and returns the output stream as L-Value

Example: `std::cout $\ll$ a` (mostly console output)

- The output stream is being changed and must thus be an L-Value.

Output Operator $\ll$

Why returning the output stream?
- allows bundling of output

```cpp
std::cout $\ll$ a $\ll$ "^8 = " $\ll$ b * b $\ll$ "\n"
```

is parenthesized as follows

```cpp
((((std::cout $\ll$ a) $\ll$ "^8 = ") $\ll$ b * b) $\ll$ "\n"
```

- `std::cout $\ll$ a` is the left hand operand of the next $\ll$ and is thus an L-Value that is no variable name

Structure

Events
- Tuesday 13:15 - 17:00, Einführung.
- Thursday 13:15 - 17:00, Self Study with Tutoring
- Friday 13:15 - 17:00, Self Study with Tutoring

Course web page
https://lec.inf.ethz.ch/mavt/et

2. Organization of the Engineering Tool 1
No lacking resources!

For the exercises we use an online development environment that requires only a browser, internet connection and your ETH login.

If you do not have access to a computer: there are a lot of computers publicly accessible at ETH.

Online Tutorial

For a smooth course entry we provide an online C++ tutorial
Goal: leveling of the different programming skills.
Written mini test for your self assessment in the first recitation session of the course “Informatik”

Online Tutorial URL

Enrollment to the Tutorial
https://frontend-1.et.ethz.ch/sc/RowC4K2euLYbJ6es4

Performance Assessment

This course is graded pass/fail. You are working online on a small C++ project, which will be graded automatically.

- You can try as many times as you want until the due date.
- You get direct feedback, which helps you to solve the task
- The task has been made accessible now. You have time until March 4th, 23:59
**Academic integrity**

The ETH Zurich Ordinance on performance assessments applies the rule:

**Rule:** You submit solutions that you have written yourself and that you have understood.

We check this (partially automatically) and reserve our rights to adopt disciplinary measures.

---

**Codeboard**

*Codeboard* is an online IDE: programming in the browser.

- Bring your laptop / tablet / ... along, if available.
- You can try out examples in class without having to install any tools.

---

**Code Expert**

Our exercise system consists of two independent systems that communicate with each other:

- **The ETH submission system:** Allows us to evaluate your tasks.
- **The online IDE:** The programming environment

---

**Enrollment for the project**

**Codeboard.io Registration**

Go to https://codeboard.io and create an account, stay logged in.

**Registration for exercises**

Go to https://expert.ethz.ch/mavt_et1_2018 and inscribe for one of the exercise groups there.
**Codeboard.io Registration**

If you do not yet have an **Codeboard.io** account ...

- We use the online IDE **Codeboard.io**
- Create an account to store your progress and be able to review submissions later on
- Credentials can be chose arbitrarily **Do not use the ETH password.**

**Codeboard.io Login**

If you have an account, log in:

**Opening the Project**

- Visit [https://expert.ethz.ch/mavt_et1_2018](https://expert.ethz.ch/mavt_et1_2018)
- Log in with your nethz account.

**Opening the Project**

Register with this dialog for (the only possible) exercise group.
The Project

You are now registered and the first exercise is loaded. Follow the instructions in the yellow box.

The Project – codeboard.io login

Attention If you see this message, click on Sign in now and register with your codeboard.io account.

The Project – store progress

Attention! Store your progress regularly. So you can continue working at any different location.