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 ≠ 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**.
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?

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)

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: $a$. 

"Dixit algorizmi..." (Latin translation)
**Live Demo: Turing Machine**

1. Move right:
   - [9]  → 

2. Move left:
   - L

**Euklid in the Box**

```
Programmcode

While b ≠ 0
  If a > b then
    a ← a − b
  else:
    b ← b − a

Ergebnis: a.
```

**ETH: pioneer of modern computer science**

1950: ETH rents the Z4 from Konrad Zuse, the only working computer in Europe at that time.

1956: 

http://de.wikipedia.org/wiki/ERMETH
1958–1963: Entwicklung von ALGOL 60 (der ersten formal definierten Programmiersprache), unter anderem durch Heinz Rutishauer, ETH

1964: Erstmals können ETH-Studierende selbst einen Computer programmieren (die CDC 1604, gebaut von Seymour Cray).

Computer – Implementation

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

Ingredients of a Von Neumann Architecture

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

Computer

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 m ≈ more than 100,000,000 instructions

a contemporary desktop PC can process more than 100 millions instructions

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...
- ...
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).
- Most qualified jobs require at least elementary programming skills.
- 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 ...  
- C++ is practically relevant.  
- For the computational computing (as required in math and physics), C++ offers a lot of useful concepts.  
- C++ is widespread and “runs everywhere”  
- C++ is standardized i.e. there is an “official” C++.  
- The lecturer likes 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
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?

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.

Syntax and semantics of C++
The ISO/IEC Standard 14822 (1998, 2011,...) is the “law” 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

C++: Kinds of errors illustrated with German sentences
- Das Auto fuhr zu schnell.
- DasAuto 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 galoppiert schnell.
- Manche Tiere riechen gut.

Syntaktisch und semantisch korrekt.
Syntaxfehler: Wortbildung.
Syntaxfehler: Satzstellung.
Syntaxfehler: Satzzeichen fehlen.
Syntaktisch korrekt aber mehrdeutig. [kein Analogon]
Syntaktisch korrekt, doch semantisch fehlerhaft: Falscher Artikel. [Typfehler]
Syntaktisch und grammatikalisch korrekt! Semantisch fehlerhaft. [Laufzeitfehler]
Syntaktisch und semantisch korrekt. Semantisch mehrdeutig. [kein Analogon]
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
#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;
}
```

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)

“Accessories:” Comments

```cpp
#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;
}
```
Comments and Layout

Comments
- are contained in every good program.
- document what and how a program does something and how it should be used,
- are ignored by the compiler
- Syntax: “double slash” // until the line ends.

The compiler ignores additionally
- Empty lines, spaces,
- Indendations that should reflect the program logic

#include <iostream>
int main(){std::cout << "Compute a^8 for a =? ";
int a; std::cin >> a; int b = a * a; b = b * b;
std::cout << a << "^8 = " << b*b << "\n";return 0;}

... but we do!

“Accessories:” Include and Main Function

#include <iostream>
int main() { float exp = 8;
// input
std::cout << "Compute a^8 for a =? ";
// computation
int a; std::cin >> a;
int b = a * a; // b = a^2
b = b * b; // b = a^4
// output b * b, i.e., a^8
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

- 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:
  expr;

  where expr is an expression
- Effect is the effect of expr, the value of expr is ignored.

Example: b = b*b;
Return Statements

- Do only occur in functions and are of the form `return expr;`
- Specify the return value of a function
- Example: `return 0;`

Values and Effects

- Determine what a program does,
- Are purely semantical concepts:
  - Symbol 0 means `Value 0 ∈ ℤ`
  - `std::cin >> a;` means effect "read in a number"
- Depend on the program state (memory content, inputs)

Statements – Effects

```cpp
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 – Variable Definitions

```cpp
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;
}
```
**Declaration Statements**

- introduce new names in the program,
- consist of declaration and semicolon
- Example: \texttt{int a;}
- can initialize variables
- Example: \texttt{int b = a * a;}

**Types and Functionality**

- \texttt{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 (\texttt{int})
  - natural numbers (\texttt{unsigned int})
  - real numbers (\texttt{float}, \texttt{double})
  - boolean values (\texttt{bool})
  - ...

**Literals**

- represent constant values
- have a fixed \texttt{type} and \texttt{value}
- are "syntactical values".
- Examples:
  - 0 has type \texttt{int}, value 0.
  - 1.2e5 has type \texttt{double}, value $1.2 \cdot 10^5$. 
Variables

- represent (varying) values,
- have
  - name
  - type
  - value
  - address
- are "visible" in the program context.

Beispiel

```plaintext
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.

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.

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)

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)

L-Values and R-Values

int a;
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;

Examples

have type, value und effect (potentially).

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

Example
b = b * 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

L-Wert (“Left of the assignment operator”)

- Expression with address
- 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 cannot change its value

Operators and Operands Building Blocks

```cpp
// 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^8, i.e., a^8
std::cout << a << "^8 = " << b * b << "\n";
return 0;
```
Operators

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

Multiplication Operator *

- Expects 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 * a and b * 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 * 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 `<<`

- 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 << a` (mostly console output)

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

Why returning the output stream?

- allows bundling of output

`std::cout << a << "^8 = " << b * b << "\n"

is parenthesized as follows

```
((((std::cout << a) << "^8 = ") << b * b) << "\n")
```

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

---

**power8_exact.cpp**

```
// Program: power8_exact.cpp
// Raise a number to the eighth power, using integers of arbitrary size

#include <iostream>
#include <IFMP/integer.h>

int main()
{
  // input
  std::cout << "Compute a^8 for a =? ";
  ifmp::integer a;
  std::cin >> a;

  // computation
  ifmp::integer 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;
}
```

Problem with `power8.cpp`: large input values are not correctly handled

- reason: domain of the type int is limited
- solution: use a different type e.g. `ifmp::integer`