ETH zürich

Felix Friedrich, Malte Schwerhoff

Computer Science

Course at D-MATH/D-PHYS at ETH Zurich

Autumn 2019

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?

What is Computer Science?

- The science of **systematic processing of informations**,…

What is Computer Science?

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

(Wikipedia, according to “Duden Informatik”)

Computer Science vs. Computers

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

Mike Fellows, US Computer Scientist (1991)

Computer Science vs. Computers

- Computer science is also concerned with the development of fast computers and networks…

Computer Science vs. Computers

- 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 $\neq$ Computer Literacy

Computer literacy: *user knowledge*

- Handling a computer
- Working with computer programs for text processing, email, presentations …

Computer Science $\neq$ Computer Literacy

Computer Science *Fundamental knowledge*

- How does a computer work?
- How do you write a computer program?

Back from the past: This course

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

Algorithm: Fundamental in Computer Science

Algorithm:

- Instructions to solve a problem step by step

Algorithm: Fundamental in Computer Science

Algorithm:

- Instructions to solve a problem step by step
- Execution does not require any intelligence, but precision (even computers can do it)

Algorithm: Fundamental in 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)

"Dixit algorizmi…" (Latin translation)

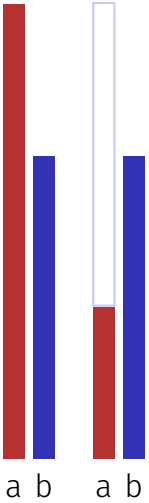
Oldest Nontrivial Algorithm

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

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

Oldest Nontrivial Algorithm

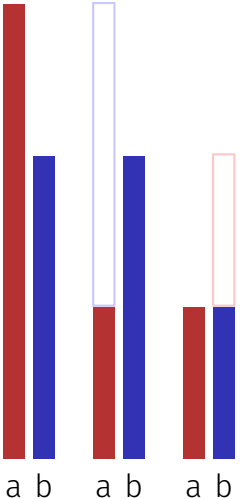
Euclidean algorithm (from the *elements* from Euklid, 3. century B.C.)



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

Oldest Nontrivial Algorithm

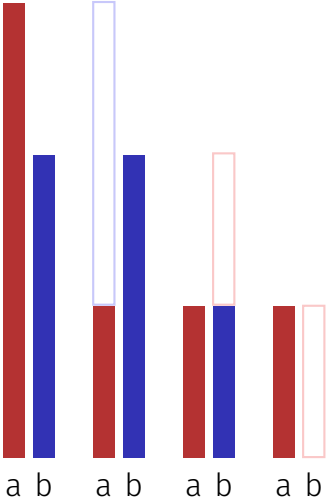
Euclidean algorithm (from the *elements* from Euklid, 3. century B.C.)



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

Oldest Nontrivial Algorithm

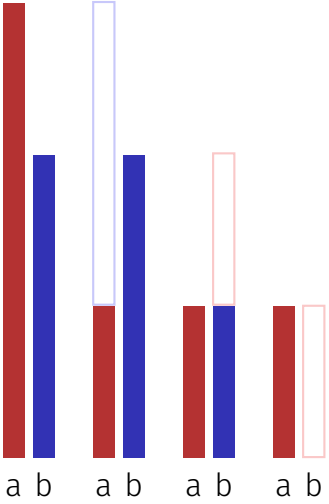
Euclidean algorithm (from the *elements* from Euklid, 3. century B.C.)



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

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.

Algorithms: 3 Levels of Abstractions

1. **Core idea** (abstract)**:**
   the essence of any algorithm (“Eureka moment”)

Algorithms: 3 Levels of Abstractions

1. **Core idea** (abstract)**:**
   the essence of any algorithm (“Eureka moment”)
2. **Pseudo code** (semi-detailed)**:**
   made for humans (education, correctness and efficiency discussions, proofs

Algorithms: 3 Levels of Abstractions

1. **Core idea** (abstract)**:**
   the essence of any algorithm (“Eureka moment”)
2. **Pseudo code** (semi-detailed)**:**
   made for humans (education, correctness and efficiency discussions, proofs
3. **Implementation** (very detailed)**:**
   made for humans & computers (read- & executable, specific programming language, various implementations possible)

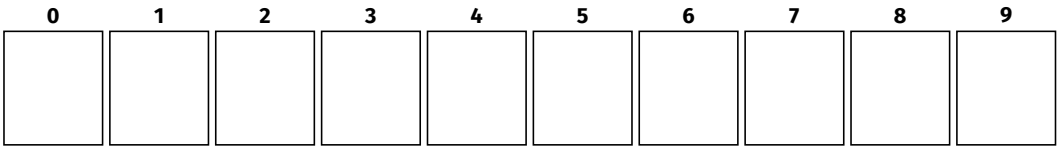
Algorithms: 3 Levels of Abstractions

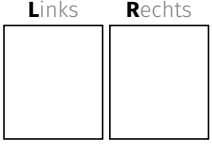
1. **Core idea** (abstract)**:**
   the essence of any algorithm (“Eureka moment”)
2. **Pseudo code** (semi-detailed)**:**
   made for humans (education, correctness and efficiency discussions, proofs
3. **Implementation** (very detailed)**:**
   made for humans & computers (read- & executable, specific programming language, various implementations possible)

Euclid: Core idea and pseudo code shown, implementation yet missing

Euklid in the Box

Speicher

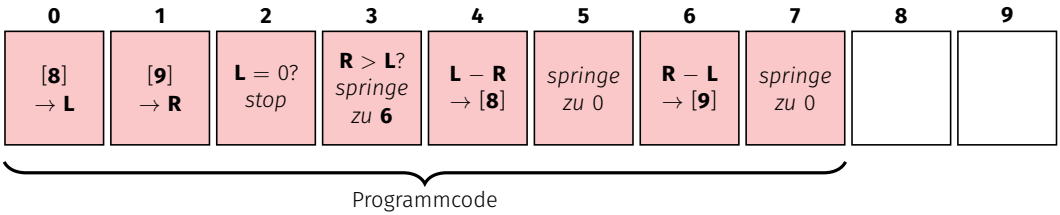


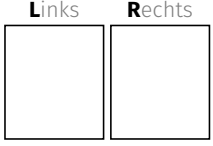


Register

Euklid in the Box

Speicher

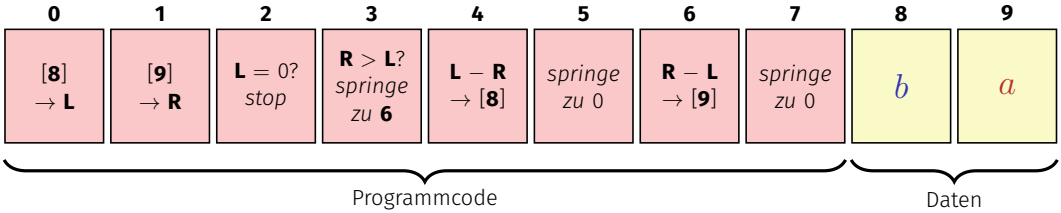


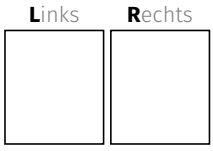


Register

Euklid in the Box

Speicher

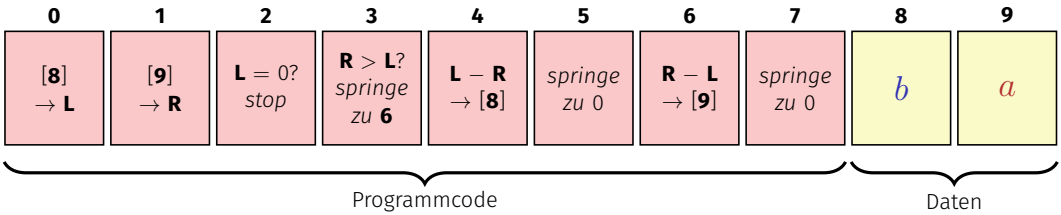


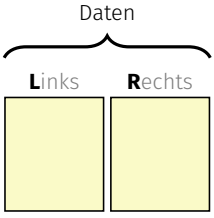


Register

Euklid in the Box

Speicher



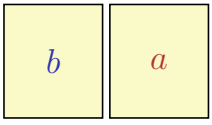


Register

Euklid in the Box

Speicher

0	1	2	3	4	5	6	7	8	9
[**8**] → **L**	[**9**] → **R**	**L** = 0? *stop*	**R** > **L**? *springe zu* **6**	**L** − **R** → [**8**]	*springe zu* 0	**R** − **L** → [**9**]	*springe zu* 0	b	a

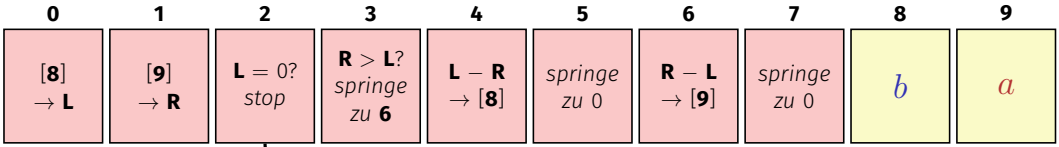


Register

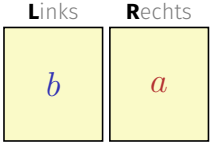
While $b \neq 0$
    If $a > b$ then
        $a \leftarrow a - b$
    else:
        $b \leftarrow b - a$
Ergebnis: a.

Euklid in the Box

Speicher



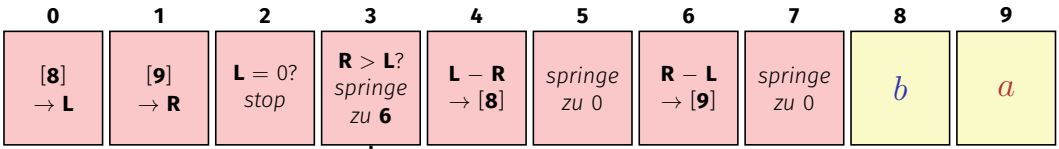
While $b \neq 0$
    If $a > b$ then
        $a \leftarrow a - b$
    else:
        $b \leftarrow b - a$
Ergebnis: a.



Register

Euklid in the Box

Speicher



While $b \neq 0$

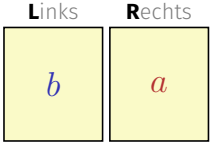
If $a > b$ then

$a \leftarrow a - b$

else:

$b \leftarrow b - a$

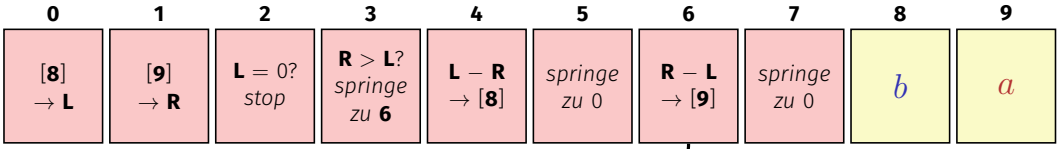
Ergebnis: a.

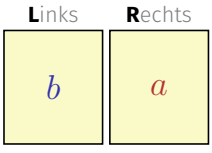


Register

Euklid in the Box

Speicher





Register

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

Euklid in the Box

Speicher

0	1	2	3	4	5	6	7	8	9
[**8**] → **L**	[**9**] → **R**	**L** = 0? *stop*	**R** > **L**? *springe zu* **6**	**L** − **R** → [**8**]	*springe zu* 0	**R** − **L** → [**9**]	*springe zu* 0	b	a

Links **R**echts

b	a

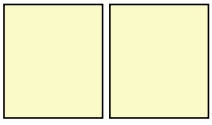
Register

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

Euklid in the Box

Speicher

0	1	2	3	4	5	6	7	8	9
[**8**] → **L**	[**9**] → **R**	**L** = 0? *stop*	**R** > **L**? *springe zu* **6**	**L** − **R** → [**8**]	*springe zu* 0	**R** − **L** → [**9**]	*springe zu* 0	b	a

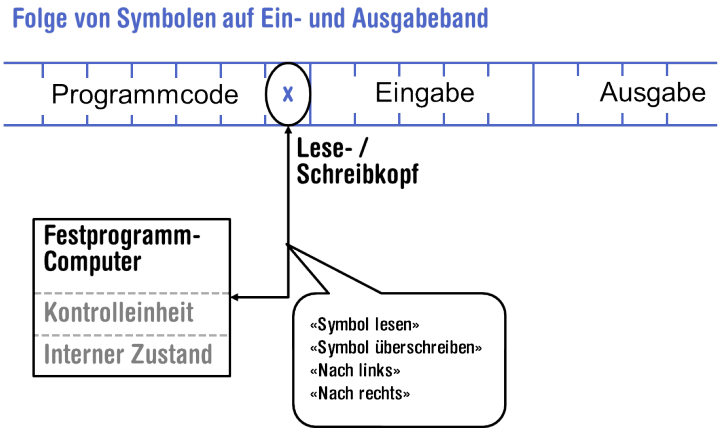


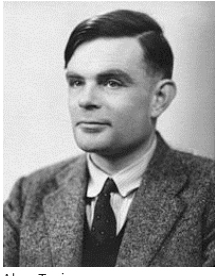
Register

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

Computers – Concept

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

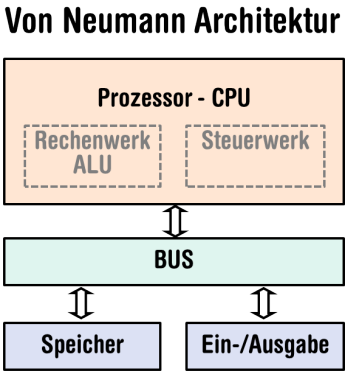




Alan Turing

Computer – Implementation

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



Konrad Zuse

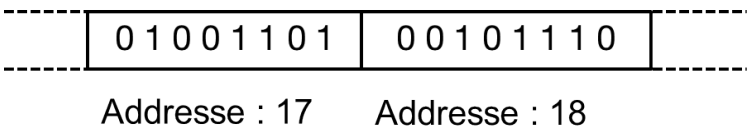
John von Neumann

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)

Memory for data *and* program

- Every memory cell has an address.
- Random access: access time to the memory cell is (nearly) independent of its address.



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**

The Harvard Computers, human computers, ca.1890

Computing speed

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

[1] Uniprocessor computer at 1 GHz.

Computing speed

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

a contemporary desktop PC can process more than 100

[1] Uniprocessor computer at 1 GHz.

Computing speed

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

30 m $\widehat{=}$ more than $100.000.000$ instructions

a contemporary desktop PC can process more than 100 millions instructions [1]

[1] Uniprocessor computer at 1 GHz.

Why programming?

- Do I study computer science or what ...

Why programming?

- Do I study computer science or what ...
- There are programs for everything ...

Why programming?

- Do I study computer science or what ...
- There are programs for everything ...
- I am not interested in programming ...

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...

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 2015-2018, NZZ Online, 1.9.2017

((BTW: Lino Guzzella is not a computer scientist, he is a mechanical engineer and prof. for thermotronics ☺)

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)

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.

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 (and is useful)!

Programming Languages

- The language that the computer can understand (machine language) is very primitive.
- Simple operations have to be subdivided into (extremely) 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

  $\rightarrow$ Abstraction!

Why C++?

Other popular programming languages: Java, C#, Python, Javascript, Swift, Kotlin, Go, ...

Why C++?

Other popular programming languages: Java, C#, Python, Javascript, Swift, Kotlin, Go, ...

General consensus:

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

Why C++?

Over the years, C++'s greatest strength and its greatest weakness has been its C-Compatibility – B. Stroustrup

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.

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.

Deutsch vs. C++

Deutsch

Alleen sind nicht gefährlich, Rasen ist gefährlich!
(Wikipedia: Mehrdeutigkeit)

C++

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

Syntax and Semantics of C++

Syntax:

- When is a text a *C++ program?*
- I.e. is it *grammatically* correct?
- $\rightarrow$ Can be checked by a computer

Semantics:

- What does a program *mean*?
- Which algorithm does a program *implement*?
- $\rightarrow$ Requires human understanding

Programming Tools

- **Editor:** Program to modify, edit and store C++program texts
- **Compiler:** program to translate a program text into machine language

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.

The first C++ program

```
// 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;
}
```

Most important ingredients... Statements

```
// 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;
}
```

Most important ingredients... Expressions

```
// 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;
}
```

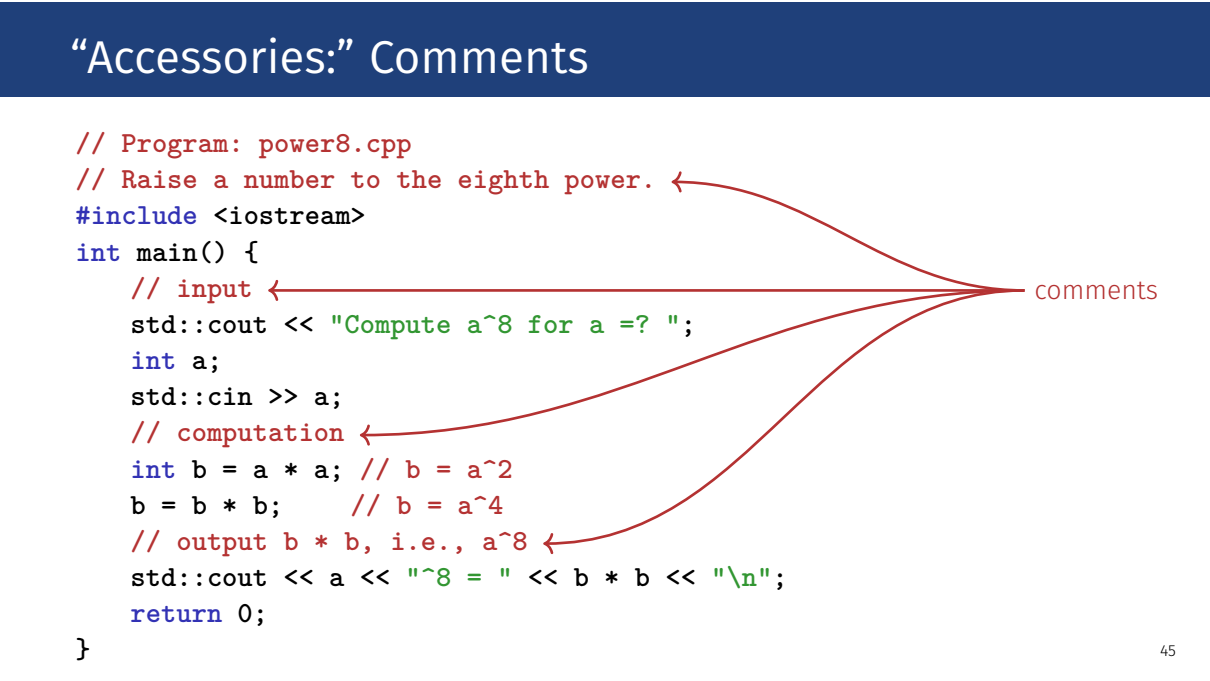
Compute a value (a^2)!

“Accessories:” Comments

```
// 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;
}
```

“Accessories:” Comments

```
// 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;
}
```



Comments and Layout

The compiler does not care...

```
#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;}
```

Comments and Layout

The compiler does not care...

```
#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

```
// 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;
}
```

“Accessories:” Include and Main Function

```
// 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 directive

“Accessories:” Include and Main Function

```
// 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;
}
```

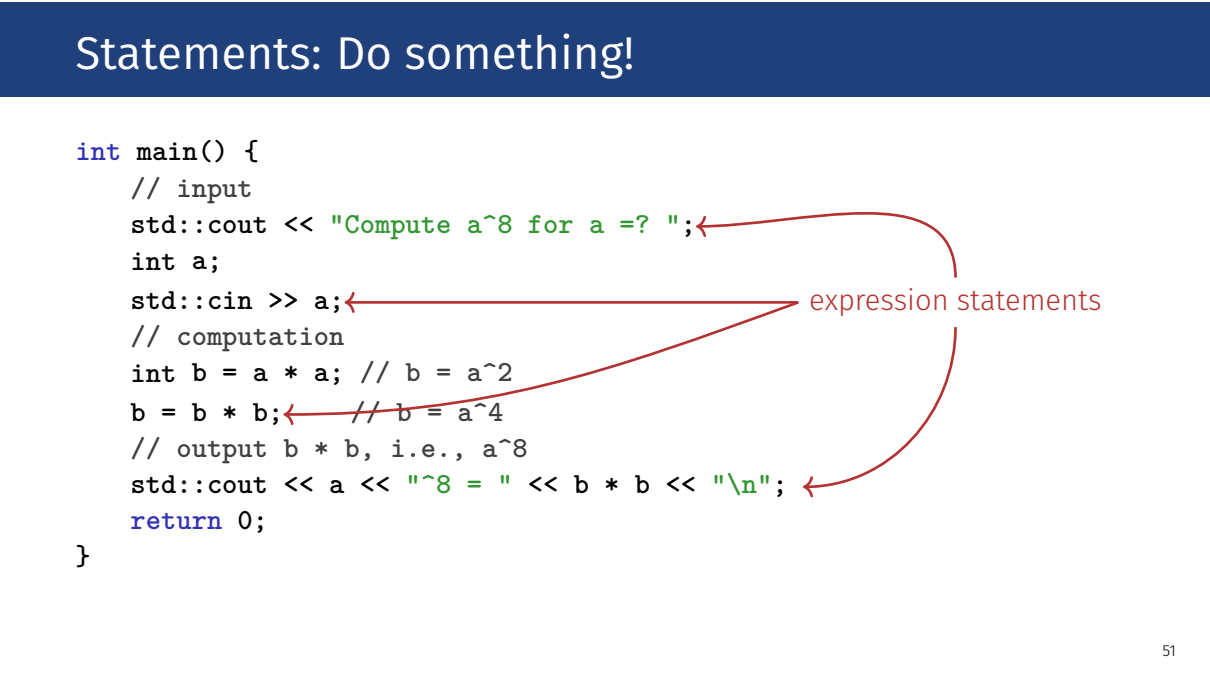
declaration of the main function

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: 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: 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;
}
```

return statement

Statements – Effects

```
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;
}
```

effect: output of the string `Compute ...`

Effect: input of a number stored in a

Effect: saving the computed value of $a \cdot a$ into b

Effect: saving the computed value of $b \cdot b$ into b

Effect: return the value 0

Effect: output of the value of a and the compu

Statements – Variable Definitions

```
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 statement

type names

Variables

- represent (varying) values
- have
  - **name**
  - **type**
  - **value**
  - **address**

Variables

- represent (varying) values
- have
  - **name**
  - **type**
  - **value**
  - **address**

`int a;` defines a variable with

- name: `a`
- type: `int`
- value: (initially) undefined
- Address: determined by compiler

Expressions: compute a value!

Expressions

- represent *Computations*

Expressions: compute a value!

Expressions

- represent *Computations*
- are either **primary** (`b`)

Expressions: compute a value!

Expressions

- represent *Computations*
- are either **primary** (`b`)
- or **composed** (`b*b`)…

Expressions: compute a value!

Expressions

- represent *Computations*
- are either **primary** (`b`)
- or **composed** (`b*b`)…
- …from different expressions, using **operators**

Expressions: compute a value!

Expressions

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

Expressions: compute a value!

Expressions

- 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

```
// 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;
```

Expressions

```
// 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;
```

variable name, primary expression (+ name and address)

variable name, primary expression (+ name and address)

literal, primary expression

composite expression

```
// 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;
```

composite expression

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

return
```

Four times composed expression

Literals

- represent constant values
- have a fixed **type** and **value**
- are "syntactical values"

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

L-Values and R-Values

```
// 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;
```

L-Values and R-Values

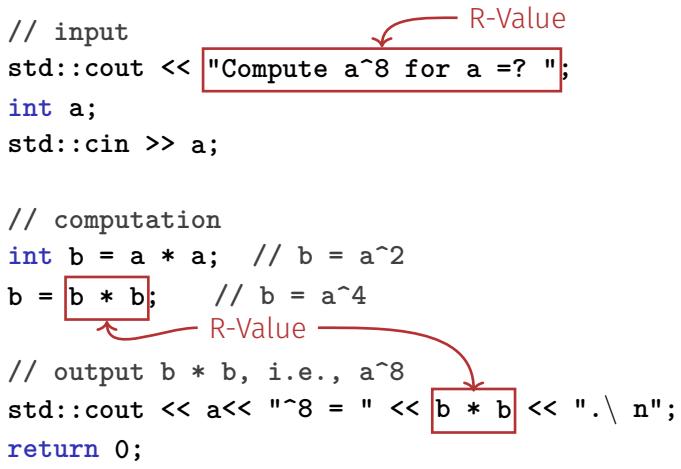
```
// input
std::cout << "Compute a^8 for a =? ";
int a;
std::cin >> a;

// 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;
```

L-value (expression + address)

L-value (expression + address)

R-Value (expression that is not an L-value)

L-Values and R-Values



L-Values and R-Values

L-Wert (“**L**eft of the assignment operator”)

- Expression with **address**
- **Value** is the content at the memory location according to the type of the expression.

L-Values and R-Values

L-Wert (“**L**eft 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

L-Values and R-Values

R-Wert (“**R**ight of the assignment operator”)

- Expression that is no L-value

Example: literal `0`

L-Values and R-Values

R-Wert (“**R**ight of the assignment operator”)

- Expression that is no L-value
- Any L-Value can be used as R-Value (but not the other way round)

Example: literal `0`

L-Values and R-Values

R-Wert (“**R**ight of the assignment operator”)

- Expression that is no L-value
- Any L-Value can be used as R-Value (but not the other way round)
  Every E-Bike can be used as normal bike, but not the other way round

Example: literal `0`

L-Values and R-Values

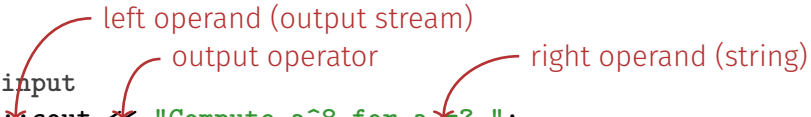
R-Wert (“**R**ight of the assignment operator”)

- Expression that is no L-value
- Any L-Value can be used as R-Value (but not the other way round)
- An R-Value *cannot change* its value

Example: literal `0`

```
// 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;
```

Operators and Operands




```
// 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;
```

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

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

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

right operand (variable name)

input operator

left operand (input stream)

```
// 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 a^8
std::cout << a << "^8 = " << b * b << "\n";
return 0;
```

assignment operator

multiplication operator