

Computer Science

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

Malte Schwerhoff, Felix Friedrich

AS 2018

Welcome

to the Course Informatik

at the MATH/PHYS department of ETH Zürich.

Place and time:

Tuesday 13:15 - 15:00, ML D28, ML E12.

Pause 14:00 - 14:15, slight shift possible.

Course web page

<http://lec.inf.ethz.ch/ifmp>

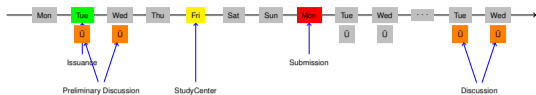
Team

chef assistant	Vytautas Astrauskas	
back office	Inna Grijevitch	
	Martin Clochard	
	Pavol Bielik	
assistants	Eliza Wszola	Moritz Schneider
	Alexander Hedges	Patrik Hadorn
	Viera Klasovita	Philippe Schlattner
	Max Egli	Yannik Ammann
	Christopher Lehner	Adrian Langenbach
	Orhan Saeedi	David Baur
	Maximillian Holst	Corminboeuf Etienne
	Benjamin Rothenberger	Tobias Klenze
	David Sommer	Sefidgar Seyed Reza
lecturers	Dr. Malte Schwerhoff / Dr. Felix Friedrich	

Registration for Exercise Sessions

- Registration via web page
- Registration already open
- 19 groups in total: 9 Tuesday 3-5pm, 10 Wednesday 10-12am
- 16 groups in German, 3 groups in English

Procedure



- Exercises available at lectures
- Preliminary discussion in the following exercise session (on the same/next day)
- StudyCenter (studycenter.ethz.ch)
- Solution must be submitted at latest one day before the next lecture (23:59h)
- Discussion of the exercise in the session one week after the submission. Feedback will be provided in the week after the submission.

Exercises

- The solution of the weekly exercises is thus voluntary but *strongly* recommended.

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 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 second exercise session.

The exam (in examination period 2018) will cover

- Lectures content (lectures, handouts)
- Exercise content (exercise sessions, exercises).

Written exam.

We will test your practical skills (programming skills) and theoretical knowledge (background knowledge, systematics).

- During the semester we offer weekly programming exercises that are graded. Points achieved will be taken as a bonus to the exam.
- The bonus is proportional to the score achieved in specially marked bonus tasks, where a full score equals a bonus of 0.25. The admission to specially marked bonus depends on the successful completion of other exercises. The achieved mark bonus expires as soon as the lecture is given anew.

Offer (Concretely)

- 3 bonus exercises in total; 2/3 of the points suffice for the exam bonus of 0.25 marks
- You can, e.g. fully solve 2 bonus exercises, or solve 3 bonus exercises to 66% each, or ...
- Bonus exercises must be unlocked (\rightarrow experience points) by successfully completing the weekly exercises
- It is again not necessary to solve all weekly exercises completely in order to unlock a bonus exercise
- Details: course website, exercise sessions, online exercise system (Code Expert)

Academic integrity

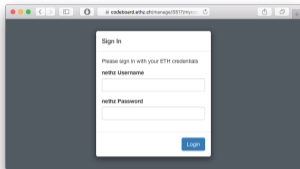
Rule: You submit solutions that you have written yourself and that you have understood.

We check this (partially automatically) and reserve our rights to invite you to interviews.

Should you be invited to an interview: don't panic. Primary we presume your innocence and want to know if you understood what you have submitted.

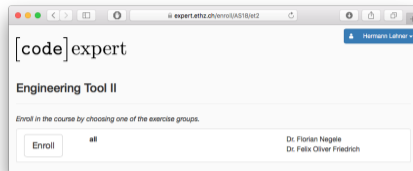
Exercise group registration I

- Visit <http://expert.ethz.ch/enroll/AS18/ifmp>
- Log in with your nethz account.



Exercise group registration II

Register with the subsequent dialog for an exercise group.



Overview

Coding Demo Exercise	Earned XP	Submissions	Handout Date	Due Date
Tasks Solutions	1,000 / 1,000		9. Sep. 2017 00:00	31. Dez. 2027 00:00
Quadratic Equations in C++	1,000 ✓	100%		

Markdown Editor Manual	Submissions	Handout Date	Due Date
Tasks Solutions		1. Aug. 2017 00:00	1. Aug. 2017 00:01

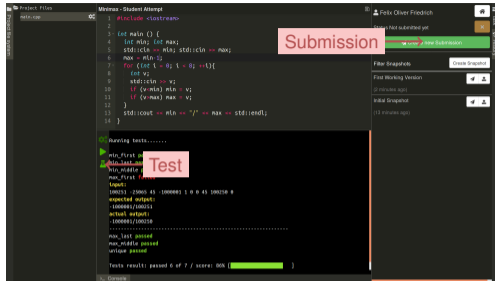
Programming Exercise

```
1 #include <iostream>
2
3 int main () {
4     int n; int max;
5     std::cin >> n;
6     max = min;
7     for (int i = 0; i < n; ++i) { // there is
8         int v;
9         std::cin >> v;
10        if (v > min) min = v;
11        if (v > max) max = v;
12    }
13    std::cout << "min == " << min << " " << max << std::endl;
14 }
```

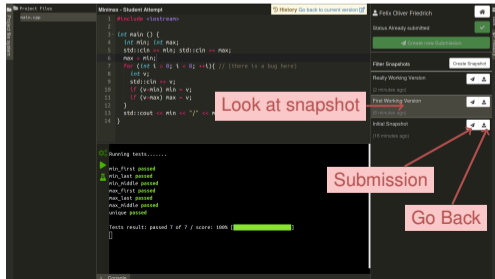
D: description
E: History

A: compile
B: run
C: test

maximum of a series of integers.
Input format: 10 consecutive integers
example:
2 30000 45 9 0 1 000000 45 70000 0
Expected output format: minmax: int
" " maxmin: int ; example:
-100000,100001



- The file system is transaction based and is saved permanently (“autosave”). When opening a project it is found in the most recent observed state.
- The current state can be saved as (named) *snapshot*. It is always possible to return to saved snapshot.
- The current state can be submitted (as snapshot). Additionally, each saved named snapshot can be submitted.



- The course is designed to be self explanatory.
- Skript together with the course Informatik at the D-MATH/D-PHYS department.
- Recommended Literature
 - B. Stroustrup. *Einführung in die Programmierung mit C++*, Pearson Studium, 2010.
 - B. Stroustrup, *The C++ Programming Language* (4th Edition) Addison-Wesley, 2013.
 - A. Koenig, B.E. Moo, *Accelerated C++*, Addison Wesley, 2000.
 - B. Stroustrup, *The design and evolution of C++*, Addison-Wesley, 1994.

- Lecture:
 - Original version by Prof. B. Gärtner and Dr. F. Friedrich
 - With changes from Dr. F. Friedrich, Dr. H. Lehner, Dr. M. Schwerhoff
- Script: Prof. B. Gärtner
- Code Expert: Dr. H. Lehner, David Avanthay and others

Computer Science

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

Malte Schwerhoff, Felix Friedrich

AS 2018

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”)

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 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 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?

25

ETH: pioneer of modern computer science

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



Neue Zürcher Zeitung, 30. August 1950

27

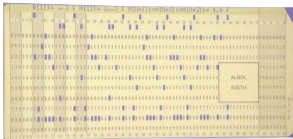
ETH: pioneer of modern computer science

1956:



ETH: pioneer of modern computer science

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).

Vortrag Walter Gander, 50 Jahre Programmieren, ETH Zürich, 2014

29

ETH: pioneer of modern computer science



Die Klasse 1964 im Jahr 2015 (mit einigen Gästen)

<http://www.inf.ethz.ch/personal/walter-gander/50years-ethz-1964-2014>

30

ETH: pioneer of modern computer science

1968–1990: Niklaus Wirth entwickelt an der ETH die Programmiersprachen Pascal, Modula-2 und Oberon und 1980 die *Lilith*, einen der ersten Computer mit grafischer Benutzeroberfläche.



Back from the past: This course

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

31

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)

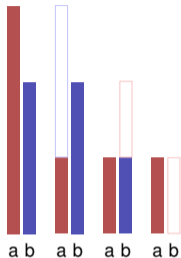


"Dixit algorizmi..." (Latin translation)

<http://de.wikipedia.org/wiki/Algorismus>

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 .

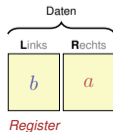
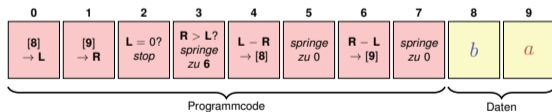
Algorithms: 3 Levels of Abstractions

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

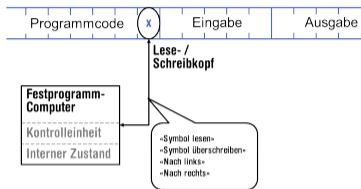


While $b \neq 0$
 If $a > b$ then
 $a \leftarrow a - b$
 else:
 $b \leftarrow b - a$
 Ergebnis: a .

Computers – Concept

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

Folge von Symbolen auf Ein- und Ausgabeband



Alan Turing

http://en.wikipedia.org/wiki/Alan_Turing

37

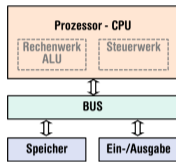
Computer – Implementation

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



Konrad Zuse

Von Neumann Architektur



John von Neumann

<http://www.hi.hawaii.edu/psd/g77/ha/biog-77zuse.htm>
<http://computerhistory.org/wiki/File:JohnVonNeumann.jpg>

38

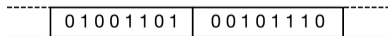
Computer

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

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.



Adresse : 17 Adresse : 18

39

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)

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 ...

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

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

¹Uniprocessor computer at 1 GHz.

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!

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

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

- 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 (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
→ Abstraction!

Programming languages – classification

Differentiation into

- Compiled vs. interpreted languages
 - *C++*, C#, Java, Go, Pascal, Modula vs. Python, Javascript, 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#, Python, Javascript, Swift, Kotlin, Go,

- C++ is practically relevant (widespread) and “runs everywhere”.
- For the computational computing (as required in math and physics), C++ offers a lot of useful concepts.
- C++ is standardized i.e. there is an “official” C++.
- C++ is one of the “fastest” programming languages
- C++ well-suited for systems programming since it enables/requires careful resource management (memory, ...)

Why 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.

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

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

C++

```
// computation
int b = a * a; // b = a2
b = b * b;    // b = a4
```

- Das Auto fuhr zu schnell.
- DasAuto fuh r zu sxhnell.
- 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]

53

Syntax and Semantics of C++

Syntax:

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

Semantics:

- What does a program *mean*?
- Which algorithm does a program *implement*?
- → Requires human understanding

55

Syntax and semantics of C++

The ISO/IEC Standard 14822 (1998, 2011, 2014, ...)

- is the “law” of C++
- defines the grammar and meaning of C++ programs
- since 2011, continuously extended with features for *advanced programming*

54

- **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.

- 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...

```
// 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:

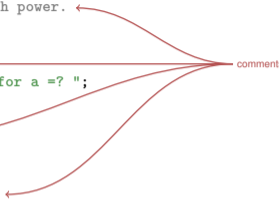
- program accepted by the compiler (syntactically correct)
- Compiler error

During runtime:

- correct result
- incorrect result
- program crashes
- program does not terminate (endless loop)

“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

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,
- Indentations that should reflect the program logic

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;
}
```



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;  
}
```

expression statements

return statement

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;`

where *expr* is an expression

- specify the return value of a function

Example: `return 0;`

Statements – Effects

```
int main() {  
    // input  
    std::cout << "Compute a^8 for a =? ";  
    int a;  
    std::cin >> a;  
    // computation  
    int b = a * a;  
    b = b * b;  
    // 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*a into b

// b = a^2

// b = a^4

Effect: saving the computed value of b*b into b

Effect: return the value 0

Effect: output of the value of a and the computed value c

Values and Effects

- determine what a program does,
- are purely semantical concepts:
 - Symbol 0 means Value $0 \in \mathbb{Z}$
 - `std::cin >> a;` means effect "read in a number"
- depend on the program state (memory content, inputs)

Statements – Variable Definitions

```
int main() {  
    // input  
    std::cout << "Compute a^8 for a=? ";  
    int a; ← declaration statement  
    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;  
}
```

type names

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`)
- ...

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 $1.2 \cdot 10^5$.

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 (→ Computer Science 1).

Variables

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

Example

`int a;` defines a variable with

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

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)

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

Analogy: building blocks

```

// 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 0;

```

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

a * a
 composed of
 variable name, operator symbol, variable name
 variable name: primary expression

- can be put into parentheses

a * a is equivalent to (a * a)

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

```
// 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;
```

Annotations in the code:

- `a` in `std::cin >> a;` is labeled as **L-value (expression + address)**.
- `a` in `"Compute a^8 for a =? "` is labeled as **R-Value**.
- `b` in `int b = a * a;` is labeled as **L-value (expression + address)**.
- `b * b` in `b = b * b;` is labeled as **R-Value**.
- `b * b` in `std::cout << a << "^8 = " << b * b << ".\ n";` is labeled as **R-Value**.
- `0` in `return 0;` is labeled as **R-Value (expression that is not an L-value)**.

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

L-Values and R-Values

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

L-Value and R-Value



```

// input
std::cout << "Compute a^8 for a=? ";
int a;
std::cin >> a;
// compute a^8
int b = 1;
b = b * b;    // b = a^4
// output
std::cout << a << "^8 = " << b * b << "\n";
return 0;

```

Annotations in the code:

- left operand (output stream) points to `std::cout`
- output operator points to `<<`
- right operand (string) points to `"Compute a^8 for a=? "`
- right operand (variable name) points to `a`
- input operator points to `>>`
- left operand (input stream) points to `std::cin`
- assignment operator points to `=`
- multiplication operator points to `*`

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`

Operators

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

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.

Output Operator <<

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

- 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. `ifm::integer`

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;
}
```