ELTE | FACULTY OF INFORMATICS

PROGRAMMING

Lecture 1

Zsuzsa Pluhár

pluharzs@inf.elte.hu

Steps of creating a computer program

1. **Specification** (from what?, what?) $\rightarrow$ specification
2. **Design** (with what?, how?) $\rightarrow$ data+ algorithmic description
3. **Coding** (how by computer?) $\rightarrow$ code (representation + implementation)
4. **Testing** (any errors/bugs?) $\rightarrow$ list of errors (diagnosis)
5. **Searching for bugs** (where is the bug?) $\rightarrow$ location and reason of bug
6. **Correction** (how would it be correct?) $\rightarrow$ correct program
7. **Quality assurance, efficiency** (could we make it better? how?) $\rightarrow$ good program
8. **Documentation** (how does it work?) $\rightarrow$ usable program
9. **Usage, maintenance** (is it still ok?) $\rightarrow$ durable program

Specification

Aim: give the task in a formalized way

Components:

- **Input data** (identifier, set of values [unit of measure])
  - Information about the input (**precondition**)
- **Output,** results (identifier, set of values)
- The statement used to get the result (**postcondition**)
- Definitions of the used concepts
- Requirements against the solution
- Restricting conditions

state-based concept

Specification

Attributes:

- „Unambiguous”, succinct, precise, complete
- Short, compact, formalized (well-structured)
- Expressive, understandable (concepts)

Tools of specification:

- Text description
- Mathematical formulas

Algorithm

Elements of algorithms:

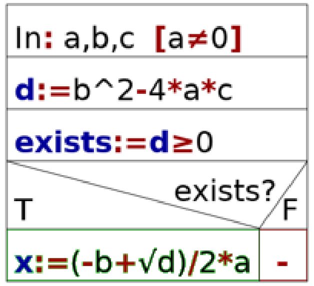
- Sequence (step by step execution)
- Conditional branch (choice from 2 or more options based on a condition)
- Loop (repeat certain amount of times or until a certain condition is met)
- Subprogram (a complex activity, with a unique name – abstraction)

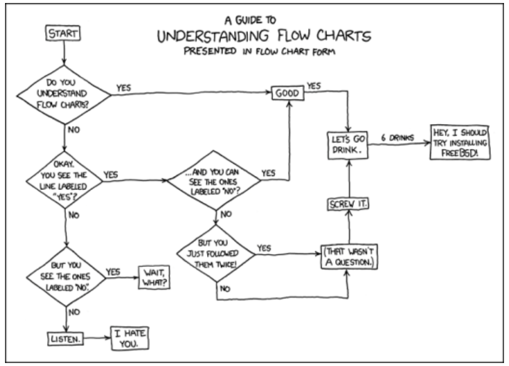
Algorithm – the language

- Textual description
  - Writing with sentences
  - Sentence-like elements – pseudocode

```
Program QuadraticEquation:
      Variables
      a,b,c,x:Real
      exists:Boolean
      d:Real
      In:a,b,c [a≠0]
      d:=b²-4*a*c
      exists:=(d≥0)
      If exists then
      …
End of Program
```

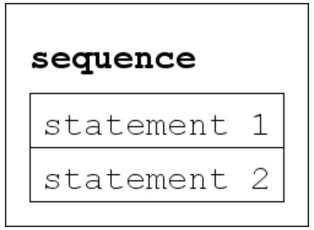
- Drawings
  - Flow chart
  - **Structogram** (Nassi–Shneiderman diagram)





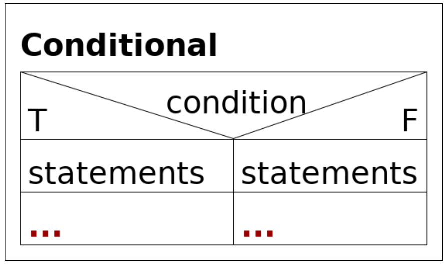
Algorithmic elements – the language

Sequence

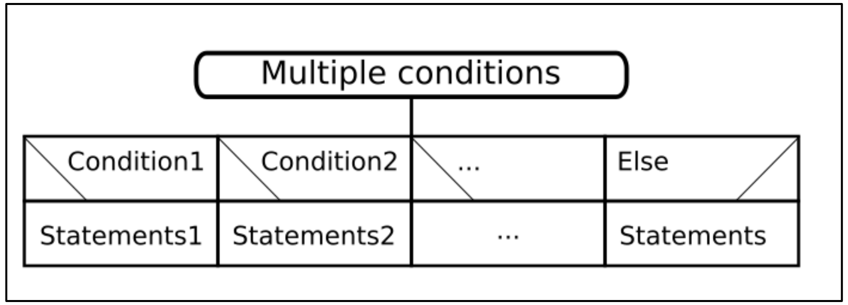



```
Statement1
Statement2
```

Conditional branch




```
If Conditions then
  Statements of true branch
else
  Statements of false branch
End If
```

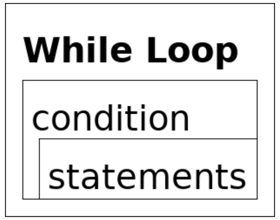



```
Conditional
 Condition1 then Statements1
 Condition2 then Statements2
 …              …
 else   Statements
End of Conditional
```

Algorithmic elements – the language

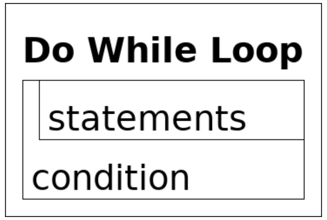
Conditional loop

pretesting the condition



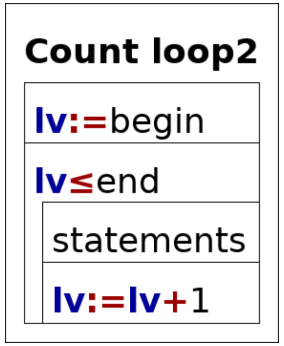

```
Loop while Condition
  loop cycle statements
Loop end
```

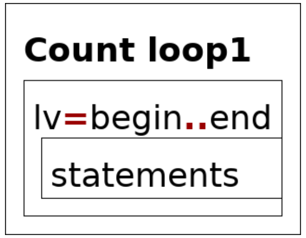
posttesting the condition




```
Loop
  loop cycle statements
while Condition
Loop end
```

Counter loop






```
Loop from lv=begin to end
  loop cycle statements
Loop end
```

Examples: task $\rightarrow$ specification $\rightarrow$ algorithm

1. Let's calculate the linear movement, the uniform motion in a straight line if the path and the time is given.
2. Let's determine whether a number is even or not.
3. Let's change the value of 2 variables.
4. Could the given 3 numbers be the sides of a right-angled triangle?

Example1: linear movement

The term contains the numbers with the operators

Specification

Input: `s,t`$\in\mathbb{R}$
Output: `v`$\in\mathbb{R}$
Precondition: `s,t≥0 and t≠0 (or: s≥0 and t>0)`
Postcondition: `v=s/t`

Algorithm

`In: s, t        [s≥0 and t>0]`
`v:=s/t`
`Out: v`

→

`v:=s/t`

Example2: even number

Specification

ℕ: natural numbers, including the 0

The term contains the numbers with the operators

Input: $a \in \mathbb{N}$

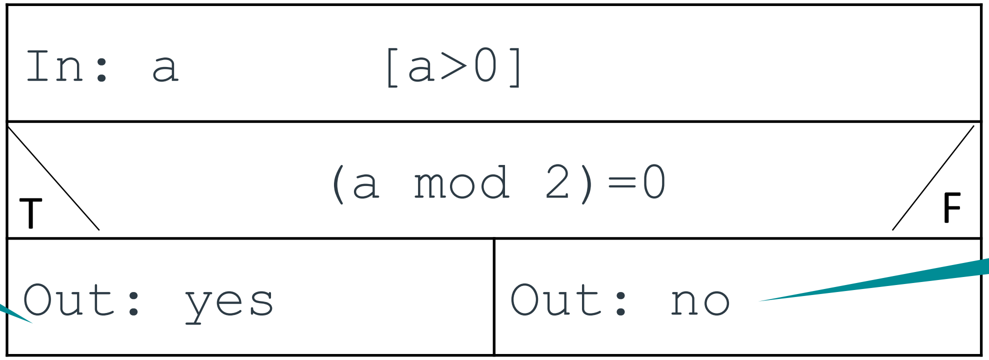
Output: $even \in \mathbb{L}$

Precondition: $a>0$

Postcondition: $even \boxed{=} ((a \bmod 2) \boxed{=} 0)$

Algorithm

even:=true



even:=false

Example3: change the values

Specification

Input: `a,b∈ℕ`

Output: `a',b'∈ℕ`

Precondition: –

Postcondition: `a'=b and b'=a`

The same variables are used. Only the values will be changed

Algorithm

```
a,b:=b,a
```

→

```
hv:=a
a:=b
b:=hv
```

Helper (auxiliary) variable
- we don't write in the specification

Example4: triangle

Specification

Input: a,b,c∈ℝ

Output: could∈𝕃

Precondition: a>0 and b>0 and c>0

Postcondition: could=($a^2+b^2=c^2$)

Without the order of the sites:

could=($a^2+b^2=c^2$) or
could=($a^2+c^2=b^2$) or
could=($c^2+b^2=a^2$)

Algorithm

could:=($a^2+b^2=c^2$)

In the algorithm we need to check the order of the sites then!

Example4: triangle

Specification

Input: $a, b, c \in \mathbb{R}$

Output: $could \in \mathbb{L}$

Precondition: $\boxed{0<a_0<b_0<c_0}$

Postcondition: $could=(a^2+b^2=c^2)$

Algorithm

$$\boxed{could:=(a^2+b^2=c^2)}$$

Conclusions – 1.

Specification = function:

Input: $a,b,c \in \mathbb{R}$

the domain of the function: $\mathbb{R}\times\mathbb{R}\times\mathbb{R}=\mathbb{R}^3$ (the components of which can be referred to in the specification as a,b,c)

Output: $could \in \mathbb{L}$

the range of the function: $\mathbb{L}$ (which we can refer to in the specification with „could”)

Precondition: $0<a<b<c$

limiting the domain of the function ($\mathbb{R}^3$) to positive numbers ($\mathbb{R}_+^3$)

Postcondition: $could=(a^2+b^2=c^2)$

what is true of the final result: the „rule” of getting the solution

Specification and implementation

Specification:

→ objects in the real world – their representation in the real world (e.g. the set of real numbers)

→ „give” them to the computer – store them into variables (in memory) – implement only a part of the original set – **type**

Next steps:

- Calculate the result with the help of functions and store it into memory
- Give back the result into the real world

Specification and implementation

Some variables are used only in calculating.

- A specification of a method can talk about the parameters and return value of the method, but it should **never talk about local variables of the method** or private fields of the method’s class.
- You should consider the implementation invisible to the reader of the specification.

Example4: triangle

Specification

Input: $a, b, c \in \mathbb{R}$

Output: $could \in \mathbb{L}$

Precondition: $0<a<b<c$

Postcondition: $could=(a^2+b^2=c^2)$

Variable:
a,b,c: **Real**
could: **Boolean**

In: a,b,c [c>b>a>0]
$could:=(a^2+b^2=c^2)$
Out: could

The program is composed of 4 parts: **declaration** and input of data, **computing** the result, **writing out** the result

Example4: triangle – 2nd solution

Specification

Input: $a,b,c \in \mathbb{R}$

Output: $could \in \mathbb{L}$

Precondition: $0<a<b<c$

Postcondition: $could=(a^2+b^2=c^2)$

Variable:
a,b,c: Real
could: Boolean

aa,bb,cc : Real

$aa:=a^2$
$bb:=b^2$
$cc:=c^2$
$could:=(aa+bb=cc)$

extra/helper variable

Don’t written in the specification!

Example5: quadratic equation

Task: Let’s specify one root of a quadratic equation! The equation is: $ax^2+bx+c=0$

Questions:

- What does the solution depend on? – *input*
- What is the solution? – *output*
- What does „being a solution mean”? – *postcondition*
- Does a solution always exist? – *precondition*
- Are we sure there is only one solution? – *output/postcondition*

Example4: quadratic equation

Specification$_1$

Input: $a, b, c \in \mathbb{R}$

Output: $x \in \mathbb{R}$

Precondition: –

Postcondition$_1$: $ax^2+bx+c=0$

Comment: the postcondition does not provide us with information how to create the algorithm. No problem, it's usually the case, but let's try again…
Formula of solution:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Example4: quadratic equation

Specification$_2$

Input: a,b,c$\in\mathbb{R}$

Output: x$\in\mathbb{R}$

Precondition: $\boxed{\mathbf{a\neq 0}}$

Postcondition$_2$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2\boxed{a}}$$

Is there always a solution?/When is there a solution?

Are we sure there is only one solution?

Example4: quadratic equation

Specification$_3$

Input: $a,b,c \in \mathbb{R}$

Output: $x \in \mathbb{R}$, $\boxed{\texttt{exists} \in \mathbb{L}}$

Precondition: $a \neq 0$

Postcondition: $\texttt{exists} = (b^2 - 4ac \geq 0)$ **and**

$$\texttt{exists} \rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2\,a}$$

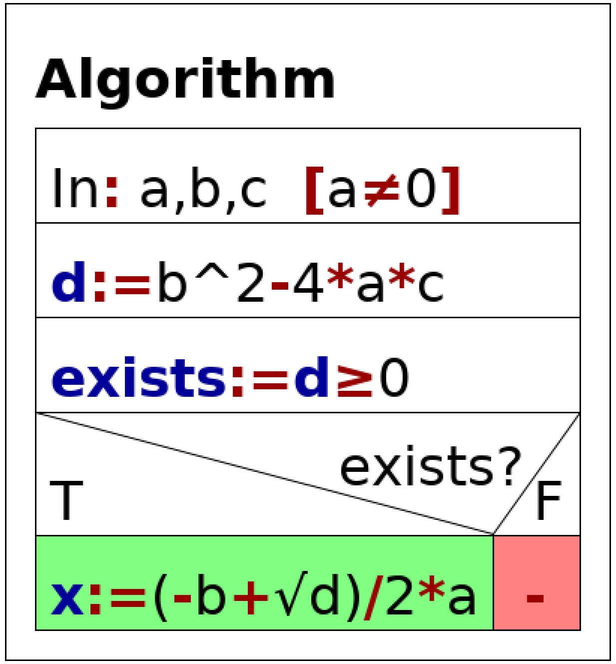
Complete

the postcondition

Are we sure there is only one solution?

Example4: quadratic equation

Algorithm



Variables
a,b,c,x: **Real**
exists: **Boolean**
d: Real

Concepts related to data

Constant

Data that **cannot change** its **value** during statement execution. It preserves its *state*.

Variable

As the name indicates, its actual value may change, so statements can assign a new value to the variable. In a more scientific phrasing: the state set of the variable has more elements.

Concepts related to data

Identifier

A sequence of symbols that allows us to refer to the data content, as well as to modify the data content.

Initial value

The value assigned at the “birth” of the identifier.

In the case of constants, this is an explicit value, in the case of variables this depends on how the actual programming environment handles the initial values.

Concepts related to data

Value assignment (:=)

A statement that changes the value of the identifier. Thus the identifier gets from its actual state to a new state. (We cannot assign new values to a constant.)

Type

This is a “contract” for a variable (or constant): we define what kind of data the variable may represent, so that we can have a fixed set of states, and applicable operations.

The type

From the perspective of complexity (structuredness) we differentiate between

- **unstructured** (scalar, elementary) data type: it cannot be broken into pieces (at least on this observation level)
- **structured** (compound) data type: formed using elementary types

Elementary types: integer

- **Value set:** $-2^{31}..+2^{31}-1$
  (Min'Integer..Max'Integer)
- **Operators:** $+$, $-$, $*$, `div` (integer division) , `mod` (remainder), $-$ (unary minus), ^ (power to positive exponent)
- **Relational operators**: $=, <, \neq, \leq, \geq, >$
- **Representation**: two's complement
- **Variations**: depends on sign and size

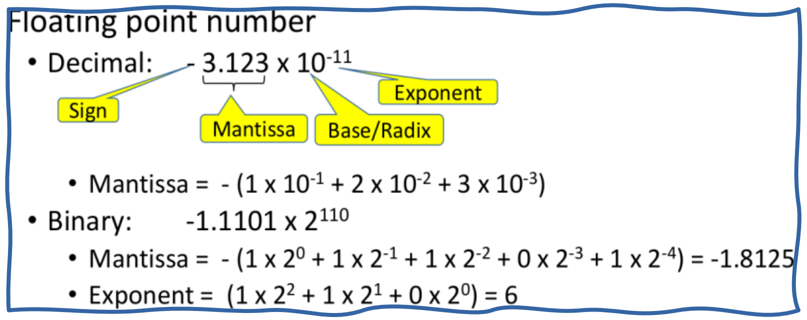
4-byte representation

Beyond reading in, writing out and value assignment

Eg. 3-bit two's complement integers:
$+0=0|00_2$, $+1=0|01_2$, $+2=0|10_2$, $+3=1|11_2$,
$-1=1|11_2$, $-2=1|10_2$, $-3=1|01_2$, $-4=1|00_2$
Can you figure out the rule?

Elementary types: real

- **Value set:** ??? … ??? (Min'Real..Max'Real are not defined, or depend on implementation)
- **Operators:** $+ , - , * , / , \wedge , -$ (unary minus)
- **Relational operators**: $=, <, \neq, \leq, \geq, >$
- **Representation**: floating point (could be more precisely called „rational number” as it can only represent those numbers)



Elementary types: character type

- **Value set:** `0..255` – coded sign, ASCII
  (Min'Character..Max'Character: the 0 and the 255)
- **Operators:** character-specific (only int)
  (maybe function `Code:Character→Integer`, and its inverse, the function `Character:Integer→Character`, but we use these only in relation to internal representation)
- **Relational operators**: $=, <, \neq, \leq, \geq, >$ (based on their internal representation → not alphabetical order!)

Elementary types: Boolean (logical) type

- **Value set:** `False..True` – coded sign, ASCII
  (Min'Boolean..Max'Boolean: the False and the True)
- **Operators:** `not, and, or` (usual logical operators)
- **Relational operators:** $=, <, \neq, \leq, \geq, >$
- **Representation:** `0=False, 1=True` (or any non-0 value=True)
- **Comment:** sorting does not really have relevance

Thank you for your attention!