


Eötvös Loránd University
Faculty of Informatics

PROGRAMMING

2019/20 1st semester

Lecture 1

ZS. PLUHÁR, H. TORMA

Content

- The steps of problem solving – the process of creating programs
- The specification
- The algorithm
- Languages for writing algorithms – the structogram
- Concepts related to data
- Basic tasks

Steps of creating a computer program

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

The definition of specification

„Interface” between the developer and the customer

Aim: giving the task in a formalized way

›**Components**:

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

The definition of specification

Attributes:

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

Tools of specification:

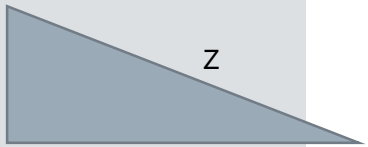
1. Text description
2. Mathematical formulas

The definition of algorithm

Elements of algorithms:

- Sequence (step by step execution)
- Conditional (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)

Example: triangle (specification)



Task: *Could the given 3 numbers be the sides of a right-angled triangle?*

Specification:

Input: $x, y, z \in \mathbb{R}$

$\mathbb{R}$ = set of real numbers

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

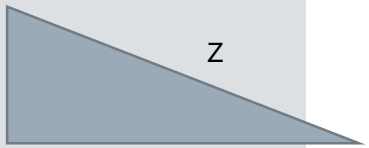
$\mathbb{L}$ = set of logical values

Precondition: $x>0$ **and** $y>0$ **and** $z>0$

Postcondition: $could = (x^2+y^2=z^2)$

Comment: the order of the 3 numbers became fixed **implicite** – z is the length of the hypotenuse!

Example: triangle (specification)



Task: *Could the given 3 numbers be the sides of a right-angled triangle?*

Specification$_2$:

Input: $x, y, z \in \mathbb{R}$

$\mathbb{R}$ = set of real numbers

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

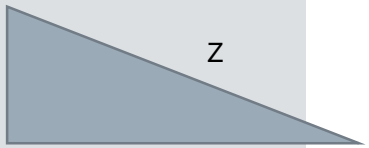
$\mathbb{L}$ = set of logical values

Precondition: $0 < x < y < z$

Postcondition: $could = (x^2 + y^2 = z^2)$

Comment: the order of the 3 numbers became fixed **explicite** – z is the length of the hypotenuse!

Example: triangle (specification)



Specification = function:

Input: $x, y, z \in \mathbb{R}$ — Independent variable

the domain of the function: $\mathbb{R}\times\mathbb{R}\times\mathbb{R}=\mathbb{R}^3$ (the components of which can be refered to in the specification as x, y, z)

Output: $could \in \mathbb{L}$ — Dependent variable

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

Precondition: $0<x<y<z$

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

Postcondition: $could = (x^2+y^2=z^2)$

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

Specification and implementation

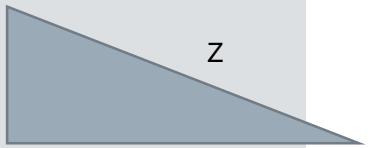
- Specification: „speaks” about objects in the real world – about theirs representation in the real world (e.g. the set of real numbers)
- We need to „give” them to the computer – take, store them into variables (in memory) – implement: only a part of the original set – **type**
- Next steps:
  - Calculate 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 the 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

Example: triangle (algorithm)



Real: real number **type**
Boolean: logical value **type**

Algorithm:

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

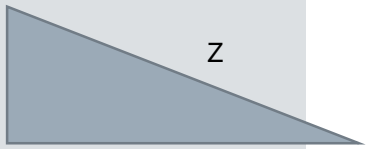
Specification:
Input: $x,y,z \in \mathbb{R}$
Output: $could \in \mathbb{L}$
Precondition: $x>0$ **and** $y>0$ **and** $z>0$
Postcondition: $could = (x^2+y^2=z^2)$

Variable
x,y,z: **Real**
could: **Boolean**

In: x,y,z [x>0 and y>0 and z>0]
could:=$(x^2+y^2=z^2)$
Out: could

We write the declaration and the „basic” instructions each into a „box”. Input and output will not be included in our algorithms later.

Example: triangle (algorithm)



An another algorithm for the essential part:

xx:=x^2
yy:=y^2
zz:=z^2
could:=(xx+yy=zz)

Variable
xx,yy,zz: Real

Extra/helper variables as note

We could/must introduce (inner, own) variables.

Example: quadratic equation (specification)

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*

Example: quadratic equation (specification)

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*

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:

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

Example: quadratic equation (specification)

Specification$_2$:

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

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

Precondition: **a≠0**

Postcondition$_2$: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Open questions:

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

Is there one solution?

Example: quadratic equation (specification)

Extending the output:

The range of the „task function” $\mathbb{R}\times\mathbb{L}$

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

Postcondition: **exists=(b²−4ac ≥ 0) and**

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

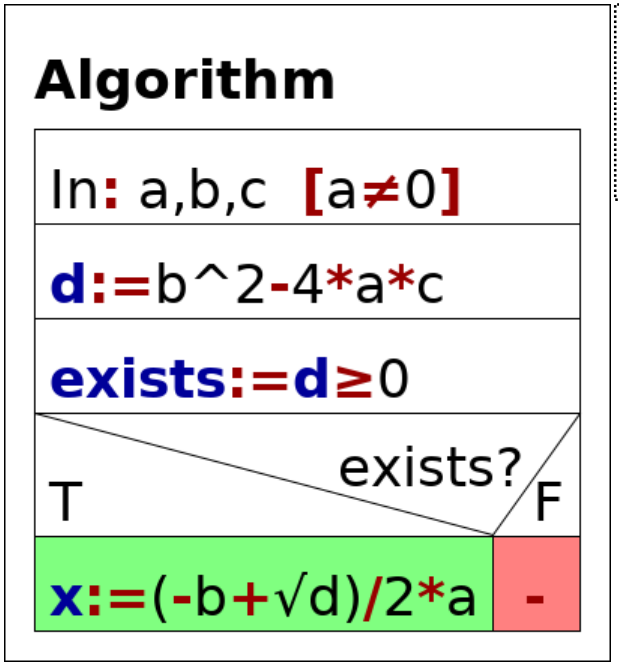
Open questions:

Is there only one solution? – homework

Example: quadratic equation (algorithm)

Specification$_2$:
Input: a,b,c∈ℝ
Output: x∈ℝ, **exists**∈𝕃
Precondition: a≠0
Postcondition: **exists=(b²−4ac ≥ 0) and**

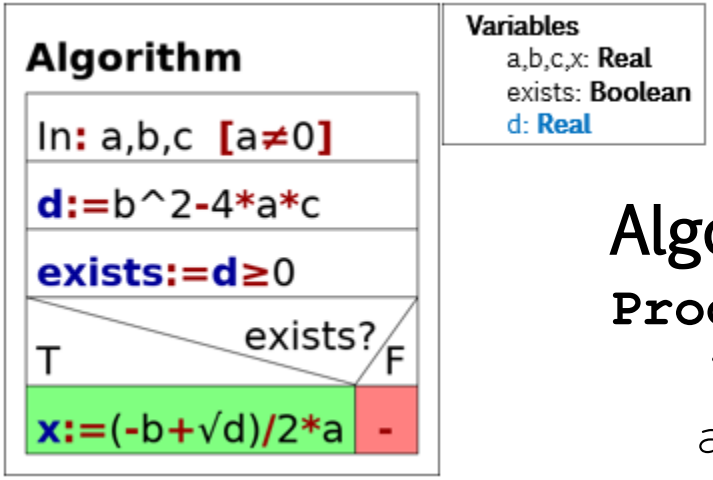
exists → $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$



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

The conditional statement is a „3-box" structure.

Example: quadratic equation (algorithm)



Algorithm in another way

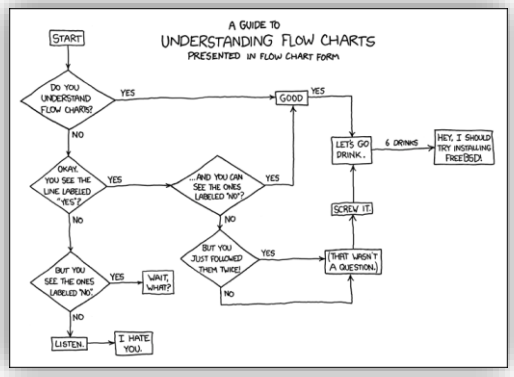
```
Program QuadraticEquation:
    Variables
    a,b,c,x:Real
    exists:Boolean
    d:Real
     In:a,b,c [a≠0]
     d:=b²-4*a*c
     exists:=(d≥0)
     If exists then
     …
End of Program
```

Languages for writing algorithms

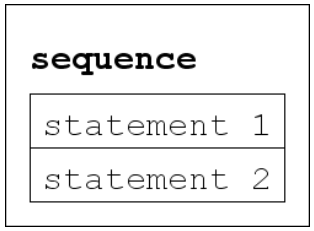
```
Program QuadraticEquation:
      Variables
       a,b,c,x:Real
       exists:Boolean
       d:Real
       In:a,b,c [a≠0]
       d:=b^2-4*a*c
       exists:=(d≥0)
       If exists then
       …
End of Program
```

- Textual description
  - Writing with sentences
  - Sentence-like elements – pseudocode
- Drawings
  - Flow chart
  - Structogram

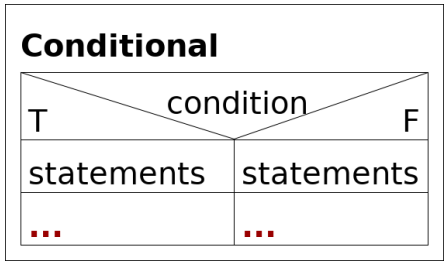
In: a,b,c [a≠0]	
d:=b^2-4*a*c	
exists:=d≥0	
exists? T	F
x:=(-b+√d)/2*a	-



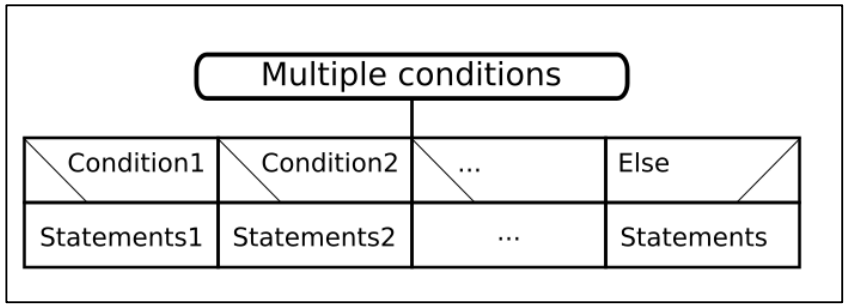
Structogram (and pseudocode)




```
Statement1
Statement2
```



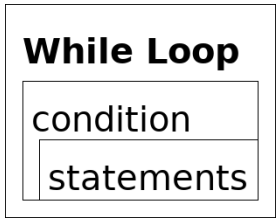

```
If Conditions then
  Statements of true branch
else
  Statements of false branch
End If
```



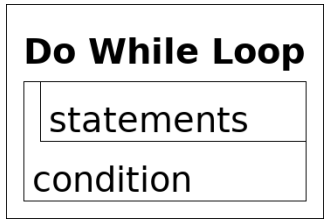

```
Conditional
 Condition1 then Statements1
 Condition2 then Statements2
 …                  …
 else  Statements
End of Conditional
```

Structogram (and pseudocode)

Conditional loops:

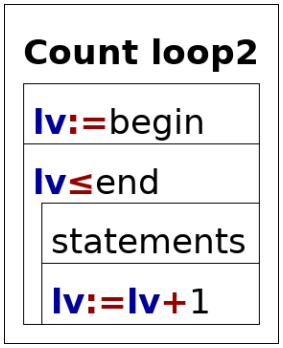


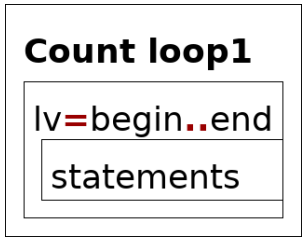

```
Loop while Condition
  loop cycle statements
Loop end
```




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

Count loops:






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

Creating structograms:

- Spreadsheet software / word processor
- Software for this purpose

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

Types of variables according to their aims

- input: gets a value at input
- result: a computation belongs to it
- intermediate result: a computation belongs to it, more calculations are derived from it
- … *(there will be some more)*

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.

Summary of data attributes

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.

The type

From the perspective of **complexity** (structuredness) we differentiate between

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

Elementary types

As an example, assuming 4-byte representation

Integer

- Value set: $-2^{31}..+2^{31}-1$ (Min'Integer..Max'Integer)
- Operators: `+`, `-`, `*`, `Div` (integer division), `Mod` (remainder), `-` (unary minus), `^` (power to positive exponent)

Beyond reading in, writing out, and value assignment

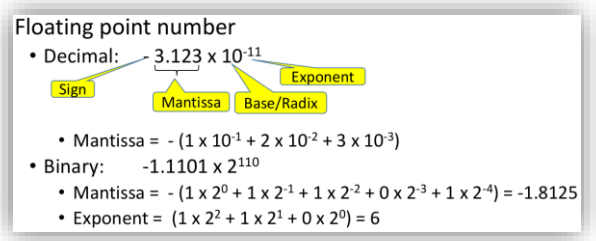
- Relational operators: `=`, `<`, `≠`, `≤`, `≥`, `>`
- Representation: two's complement
- Variations: depends on sign and size

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: `+, −, *, /, ^, −` (unary minus)
- Relational operators: `=, <, ≠, ≤, ≥, >`
- 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 signs` – ASCII (Min'Character..Max'Character: the 0 char, and the 255 char)
- 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: `=, <, ≠, ≤, ≥, >` (based on their internal representation → not alphabetical order!)

Elementary types

Boolean (logical type)

- Value set: `False..True`
  (Min'Boolean..Max'Boolean: False, and True)
- Operators: `not, and, or` (the usual logical operators)
- Relational operators: `=, <, ≠, ≤, ≥, >`
- Representation: `0` = False, `-1`=True (or any non-0 value = True)
- Comment: **sorting** does not really have relevance

Algorithm examples


Sample task for conditional statements – blood type 1

Task:

The blood type of a person (Rh negative or positive) is determined by a pair of genes. Both of them can be a type of „+” or „–”. The „++” and the „+–” are „Rh positive”, whilst „– –” is „Rh negative”.

Write a computer program that determines the blood type of a person by knowing his/her pair of genes!

Sample task for conditional statements – blood type 1

Specification:

Input: $x, y \in \mathbb{C}h$

Output: $a \in \mathbb{T}$

$\mathbb{C}h$ = set of characters

$\mathbb{T}$ = set of character strings, texts

Precondition: $x, y \in \{'+', '-'\}$

Postcondition: $(x='+' \text{ or } y='+') \text{ and } a=\text{"Rh+"} \text{ or } (x='-' \text{ and } y='-') \text{ and } a=\text{"Rh-"}$

From now on, we do not include declaration of program parameters, input and output

Algorithm

x='+' or y='+' : T	x='+' or y='+' : F
a:="Rh+"	a:="Rh-"

Sample task for conditional statements – blood type 1

Specification:

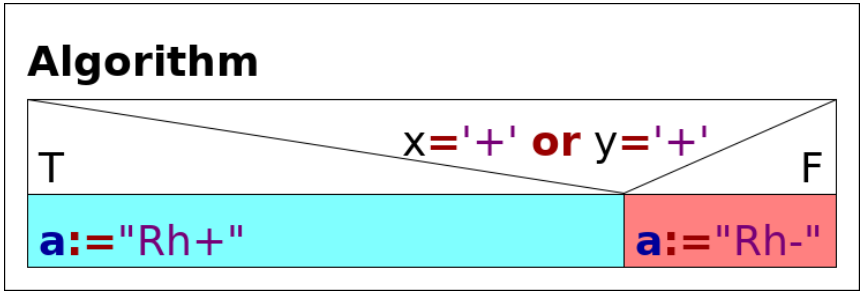
Input: $x, y \in \mathbb{C}$

Output: $a \in \mathbb{T}$

Precondition: $x, y \in \{'+', '-'\}$

Postcondition: $(x='+' \text{ or } y='+') \rightarrow a=\text{"Rh+"} \text{ and } (x='-' \text{ and } y='-') \rightarrow a=\text{"Rh-"}$

$(A \rightarrow B) \wedge (\neg A \rightarrow C) \equiv (A \wedge B) \vee (\neg A \wedge C)$



Sample task for conditional statements – blood type 2

Task:

The blood type of a person (A, B, AB or 0) is determined by a pair of genes. Both of them can be a type of a, b or 0.

The way to determine blood type: A={aa, a0}; B={bb,b0}; AB={ab}; 0={00}.

Write a computer program that determines the blood type of a person by knowing his/her pair of genes!

Sample task for conditional statements – blood type 2

Specification:

Input: $x, y \in \mathbb{C}h$

Output: $bt \in \mathbb{T}$

Precondition: $x, y \in \{'a', 'b', '0'\}$

Postcondition:

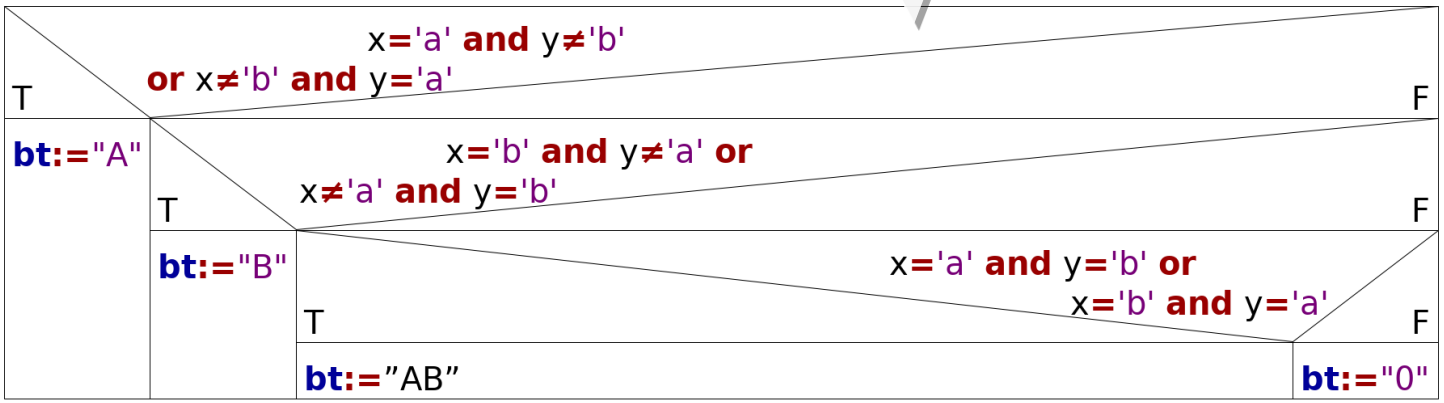
```
(x='a' and y≠'b' and y='a')
 and bt="A" or
(x='b' and y≠'a' or x≠'a' and y='b')
 and bt="B" or
(x='a' and y='b' or x='b' and y='a')
 and bt="AB" or
(x='0' and y='0' and bt='0')
```

Sample task for conditional statements – blood type 2

Specification:
Input: x,y∈ℂh
Output: bt∈𝕋
Precondition: x,y∈{'a', 'b', '0'}
Postcondition: (x='a' and y≠'b' and y='a')
and bt="A" **or**
(x='b' and y≠'a' or x≠'a' and y='b')
and bt="B" **or**
(x='a' and y='b' or x='b' and y='a')
and bt="AB" **or**
(x='0' and y='0' and bt='0')

By nesting two-way conditionals

Blood type algorithm 1



Sample task for conditional statements – blood type 2

Specification:
Input: x,y∈ℂh
Output: bt∈𝕋
Precondition: x,y∈{'a', 'b', '0'}
Postcondition: (x='a' and y≠'b' and y='a')
and bt="A" **or**
(x='b' and y≠'a' or x≠'a' and y='b')
and bt="B" **or**
(x='a' and y='b' or x='b' and y='a')
and bt="AB" **or**
(x='0' and y='0' and bt='0')

By multi-way conditionals

x='a' and y≠'b' or x≠'b' and y='a'	x='b' and y≠'a' or x≠'a' and y='b'	x='a' and y='b' or x='b' and y='a'	x='0' and y='0'
bt:="A"	bt:="B"	bt:="AB"	bt:="0"

Sample task for conditional statements – blood type 2

Specification:
Input: x,y∈ℂh
Output: bt∈𝕋
Precondition: x,y∈{'a', 'b', '0'}
Postcondition: (x='a' and y≠'b' and y='a')
and bt="A" **or**
(x='b' and y≠'a' or x≠'a' and y='b')
and bt="B" **or**
(x='a' and y='b' or x='b' and y='a')
and bt="AB" **or**
(x='0' and y='0' and bt='0')

By using auxiliary variables

Blood type algorithm 2

existsa:=x='a' **or** y='a'

existsb:= x='b' **or** y='b'

existsa

T F

existsb **existsb**

T F T F

bt:="AB"	**bt:=**"A"	**bt:=**"B"	**bt:=**"0"

Coding (C++)

Code:

two-way and multi-way conditionals

```
if (Cond)
{
   Statement_T
}
else
{
   Statement_F
}
```

Can be skipped

```
if (Cond_1) {
   Statement_1
}
else if (…) {
   …
}
else if (Cond_N) {
   Statement_N
}
else {
   Statement
}
```

2 variations of coding styles (ANSI/K&R)

Coding (C++)

Code:

two-way *and* conditionals *multi-way* (special)

```
if (Cond) {
   Statement_T
}
else {
   Statement_F
}
```

```
switch (expression)
{
   case value_1: St_1;
break;
   case …      : …  ;
break;
   case valuek_N: St_N;
break;
   default     : St ;
}
```

2 variations of coding styles (ANSI/K&R