Algorithms and basics of programming

Imre VARGA PhD
University of Debrecen, Faculty of Informatics

version 2.2.0
11 December 2024

Topics

- How to describe and solve a problem?
- What is **algorithmic thinking**?
- What is an algorithm?
- What kind of properties does it have?
- How to represent an algorithm?
- What does ’program writing’ mean?
- Does programming require abstract thinking?
- Examples, examples, examples…
- *And a lot of other things…*

Poet vs Programmer

Good programmer:

language knowledge + algorithmic thinking

(appearance) *(content)*

To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take Arms against a Sea of troubles,
And by opposing end them: to die, to sleep

William Shakespeare

To be, and to be, that is not the question:
The 'tis nobler of outrageous fortune,
Whether slings or arrows in the mind to die
And by opposing Sea against a Arms of troubles,
Or to take end them: to sleep, to suffer

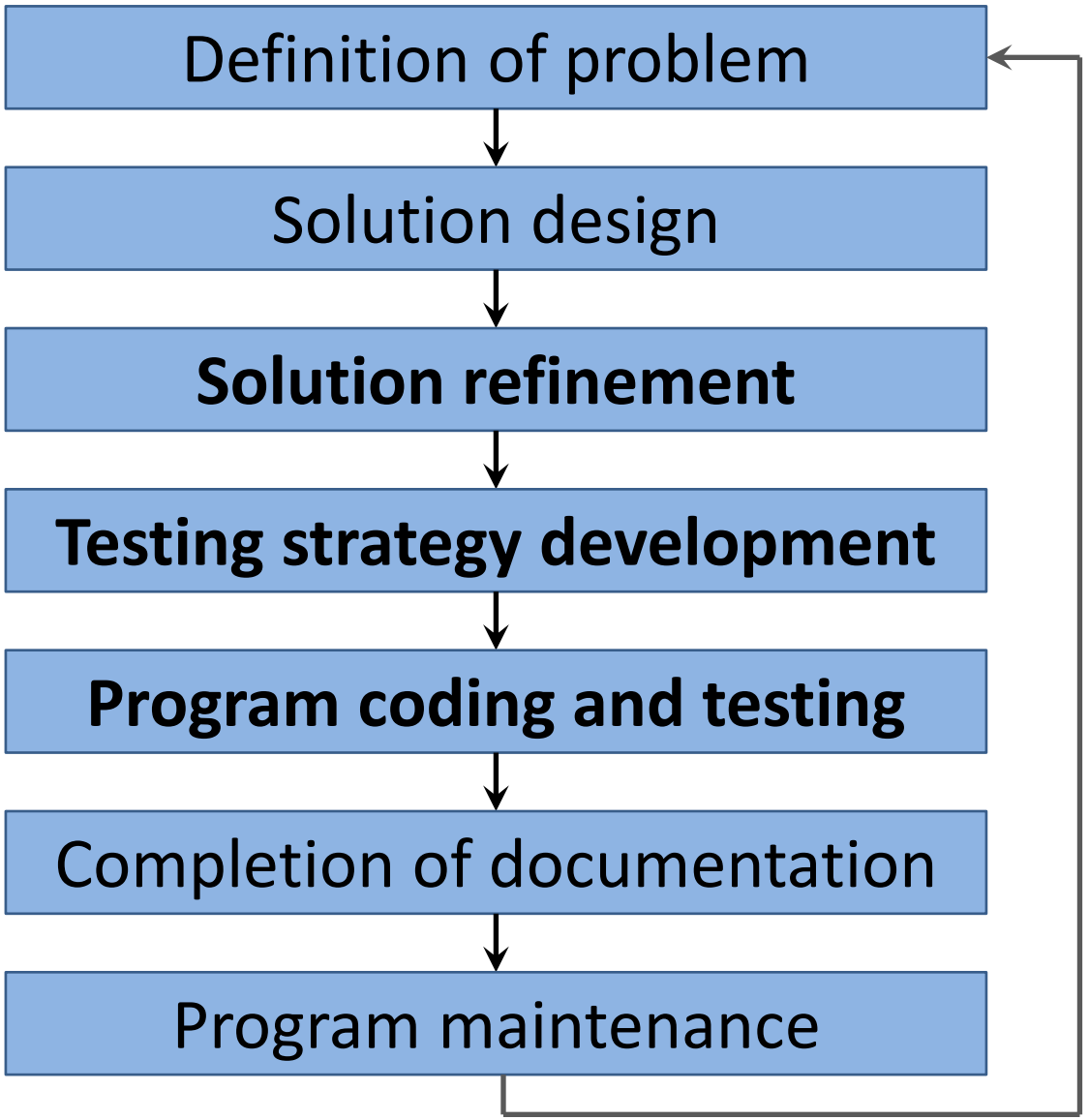
definitely not William Shakespeare

same words
different meaning

Problem solving with computer

Closely related to **software life cycle**

Problem solving with computer



1: Problem definition

- What is the task?
- What is the *unknown* (required result)?
- What is the relationship between the given/known information and the *unknown*?
- Is the given information enough to solve the problem?
- The description of the problem must be precise
- User and programmer must work together
- It leads to complete specifications of the problem, the input data and the desired output

2: Solution design

- Definition of the outline of solution
- Division of the original problem into a number of subproblems
- Subproblems are smaller and easier to solve
- Their solution will be the components of our solution
- „Divide and conquer"
- Finally the problem will be converted to a plan of well-known steps

3: Solution refinement

- Previous step is in very high-level: no indication given how subtasks are to be accomplished
- Refinement is necessary by adding more details
- Avoid any misunderstandings
- A precise method consists of a sequence of well-defined steps called an **algorithm**
- Representation: pseudocode, flowchart, etc.

4: Testing strategy development

- It is necessary to try the algorithm with several different combinations of input data to make sure that it will give correct results in all cases
- These different combinations of input data are called **test case**
- It covers not only normal input values, but also extreme input values to test the limits
- Complete test cases can be used to check the algorithm

5: Program coding and testing

- Description of an algorithm in previous level cannot be executed directly by computer
- Translation needed to a **programming language**
- After coding, the program must be tested using our testing strategy
- If an error has been discovered, appropriate revision must be made, and then the test rerun until the program gives the correct solution under all circumstances
- Process of coding and testing called **implementation**

6: Documentation completion

- Documentation begins with the first step of development and continues throughout the whole lifetime of the program
- It contains:
  - Explanations of all steps
  - Design decisions that were made
  - Occurred problems
  - Program list
  - User instructions
  - etc.

7: Program maintenance

- The program can’t wear out
- Sometimes the program may fail
- The reason of a program fail is that it was never tested for this circumstance
- Elimination of newly detected error is necessary
- Sometimes the users need new features to the program
- Update of documentations is needed

Solution refinement

The algorithm

Algorithm

Plan for performing a sequence of well-understood actions to achieve the result in finite time.

The precise definition of the actions to be performed to accomplish each task of solution design.

Some features:

- precise, unambiguous
- specified for all possible cases
- finite sequence of actions
- achieves the result
- efficiency, elegance, easy to use, …

Representation of algorithms

- **Natural (spoken) human language**
- **Flowchart**
- **Pseudocode**
- Structogram (flowblock)
- Graphs or plots
- Algebraic
- Data-flow diagram
- Hierarchical
- Tabular
- Program language

Example

Function **y=sign(x)**

- What is it?
- What does it mean?
- What is the result?
- How is it work?
- How can we determine its value?
- If x is -4, what is the value of y?
- …

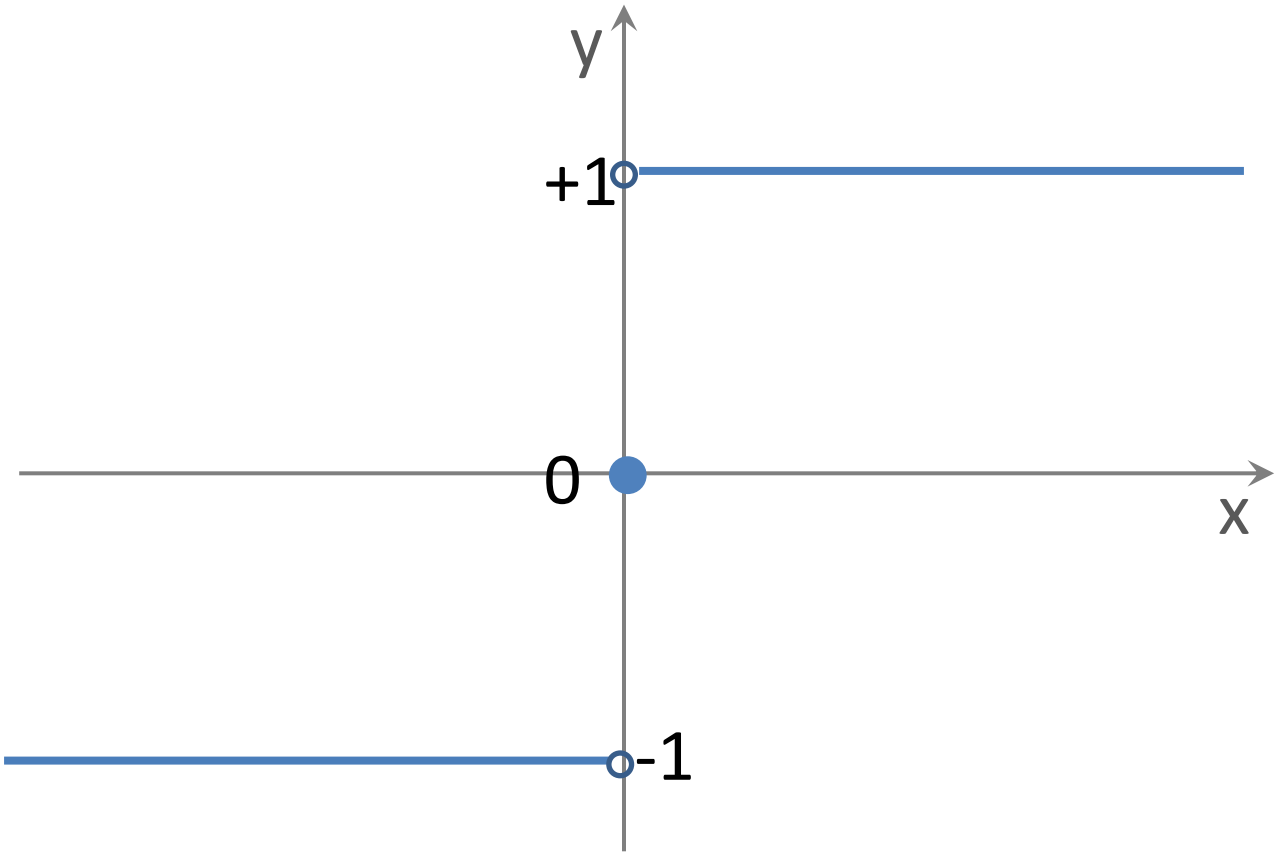
y=sign(x)

Verbal (natural language) representation:

1. If input value x is 0, set the result to y=0.
2. Otherwise, if x>0, let the value of this function +1.
3. Else if x is less than 0, give the function -1.

y=sign(x)

Graph representation:



y=sign(x)

'Algebraic-like' representation:

$x \in \Re$

$y \in \{-1, 0, +1\}$

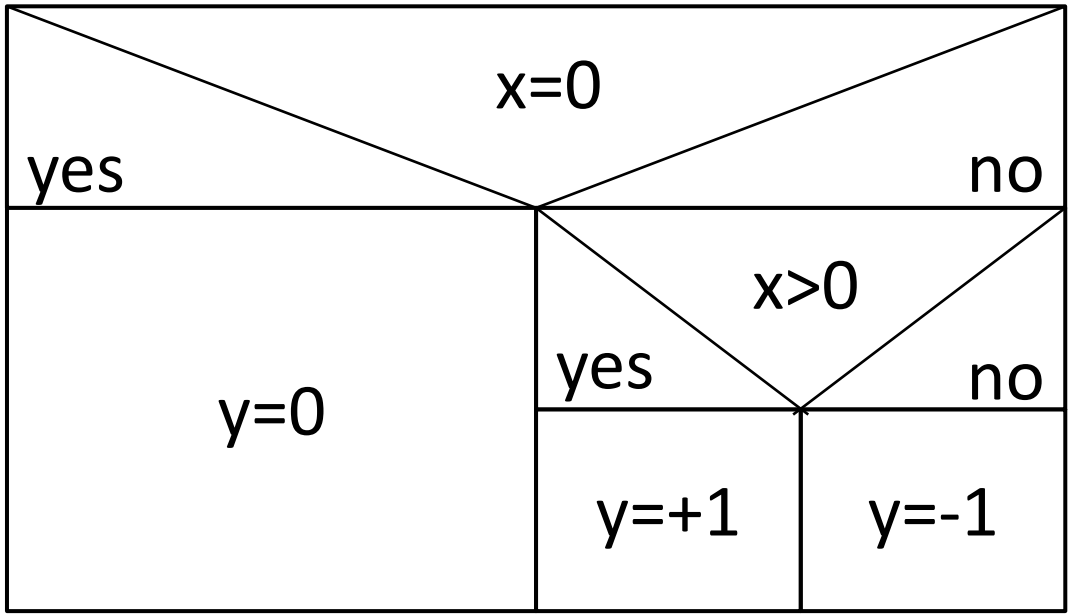
$\forall x,\ x>0 \Rightarrow y=+1$

$\forall x,\ x<0 \Rightarrow y=-1$

$x=0 \Rightarrow y=0$

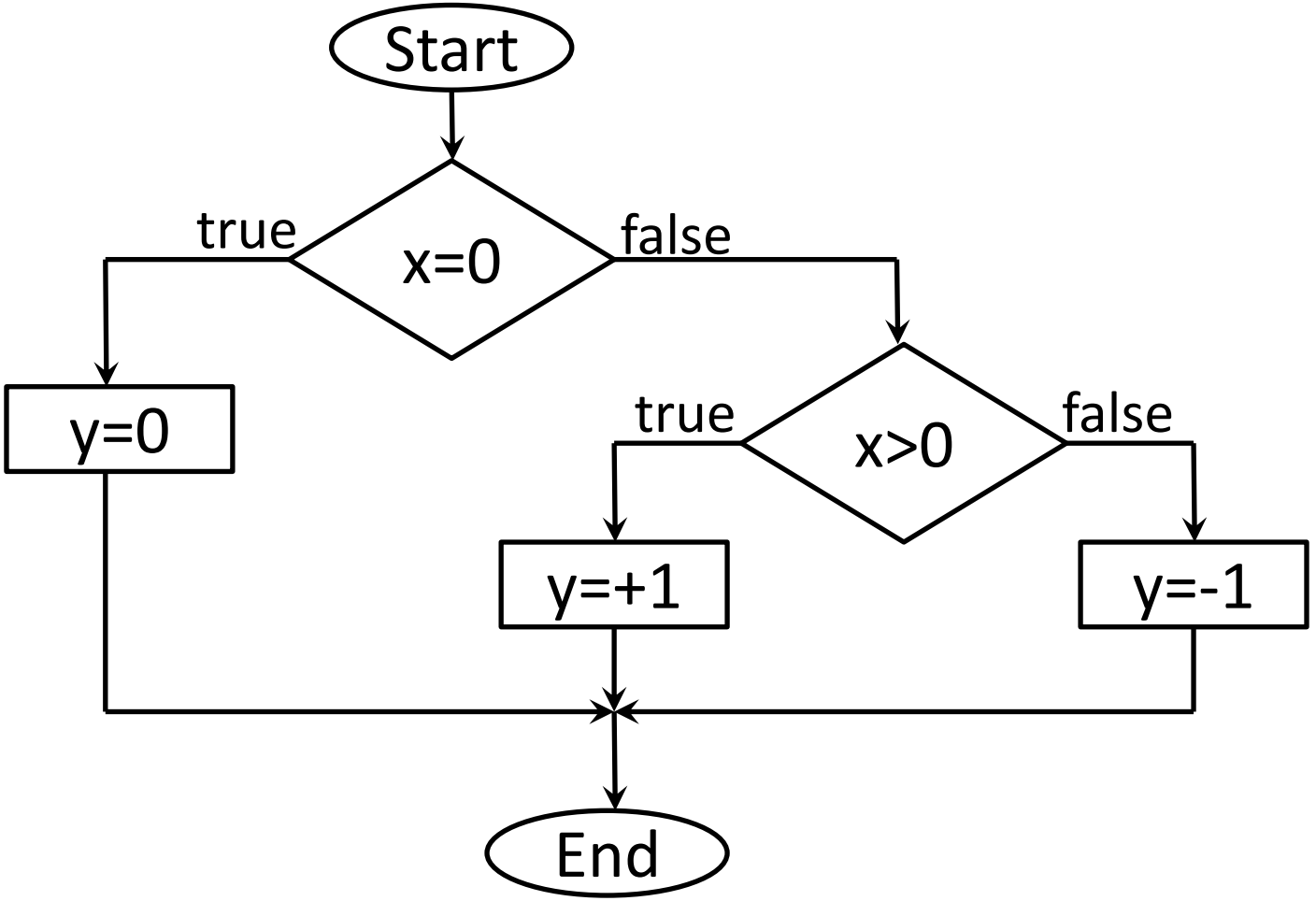
y=sign(x)

Structogram representation:



y=sign(x)

Flowchart representation:



y=sign(x)

Pseudocode representation:

```
if x==0 then
  y=0
else
  if x>0 then
     y=+1
  else
     y=-1
  endif
endif
```

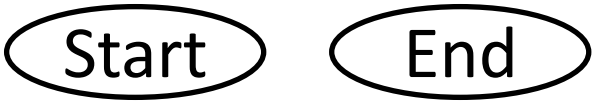
y = sign(x)

Representation by real programming language
(for example, in Python):

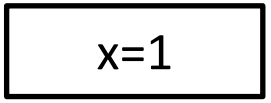
```
if x == 0:
    y = 0
else:
    if x>0:
        y = +1
    else:
        y = -1
```

Components of flowcharts

- Starting/finish point

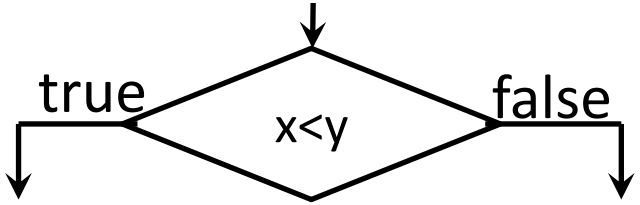


- Atomic instruction

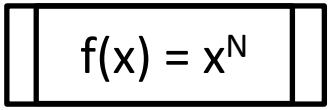


- Input/output

- Condition



- Inserting another algorithm

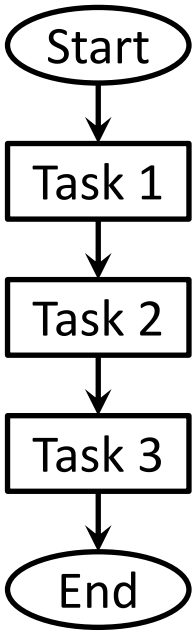


- We can go along arrows.

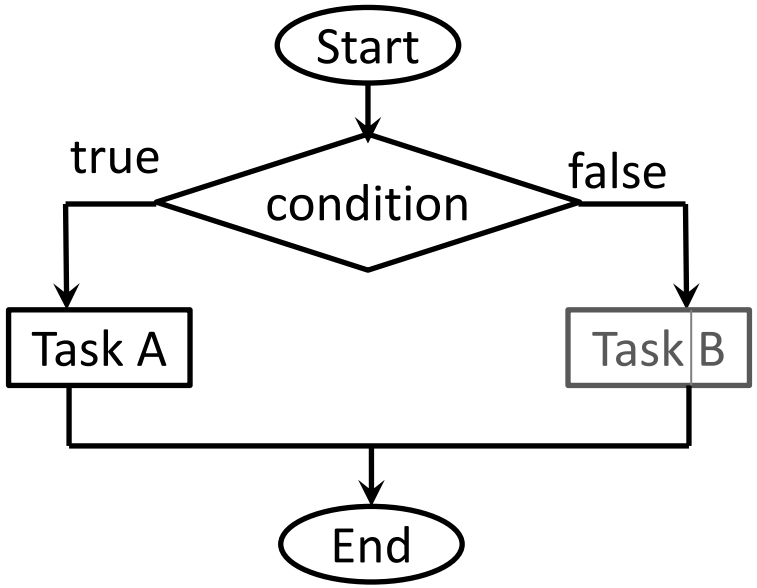
Base structures of algorithms

by flowcharts

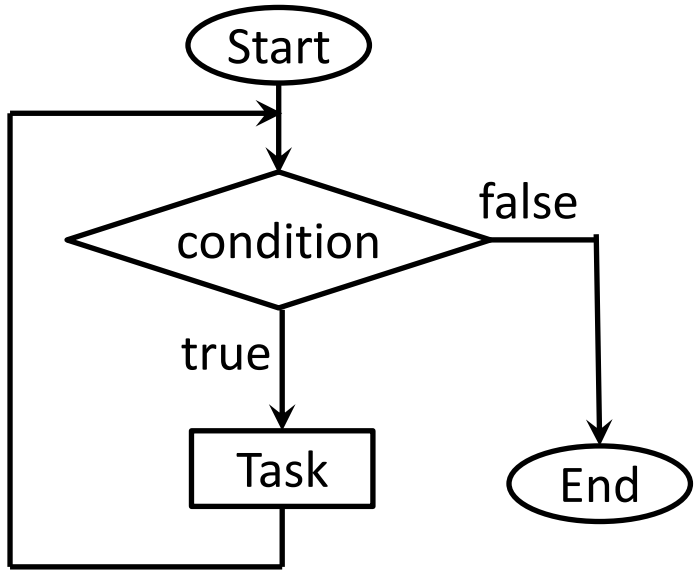
Sequence



Selection



Iteration



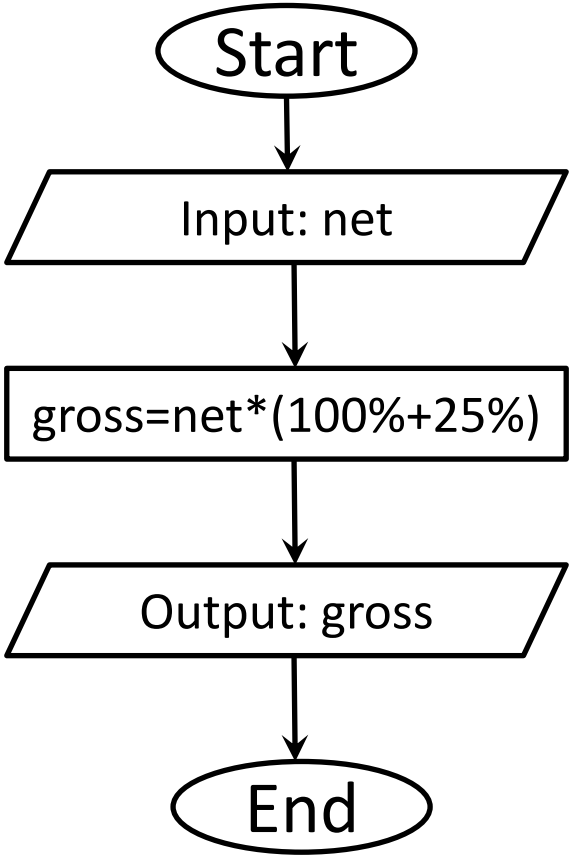
Modifying algorithms

Algorithms often go through many changes to be better.

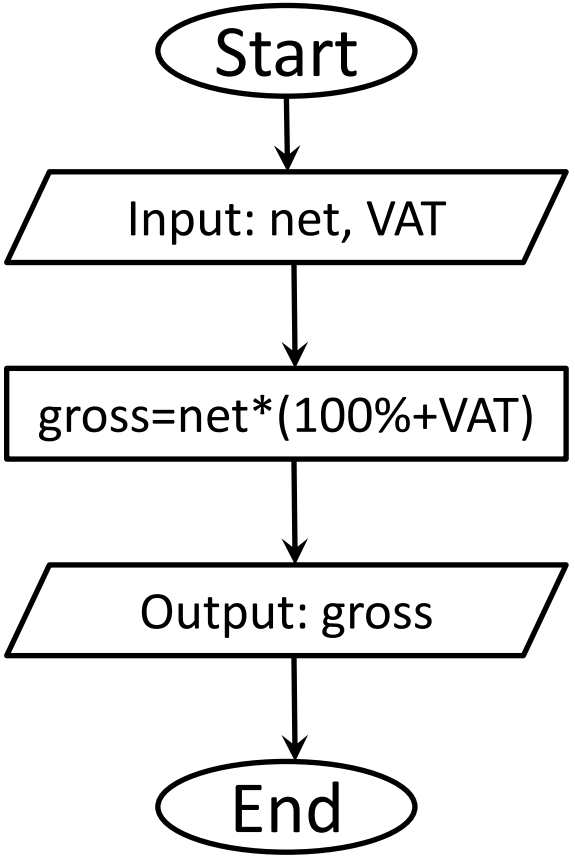
- **Generalizing**:
  making them applicable to more cases
- **Extending**:
  to include new kind of situations
- **Foolproofing**:
  making an algorithm more reliable, failsafe or robust
- **Embedding**:
  re-using that algorithm within another algorithm

Generalizing algorithms

Original:

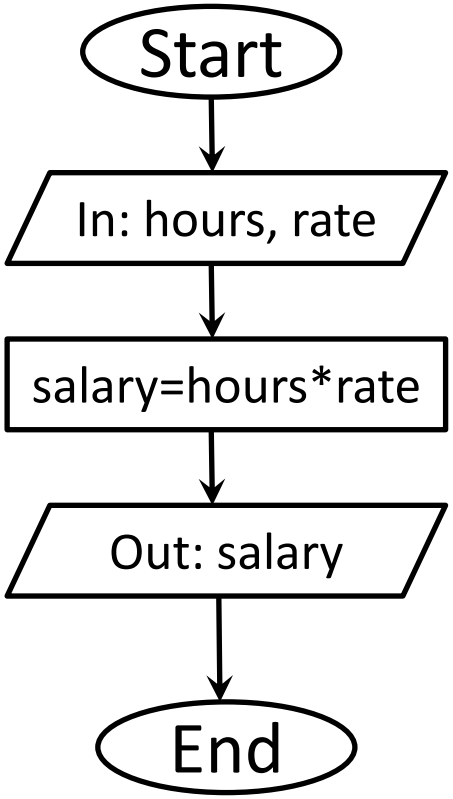


Generalized:

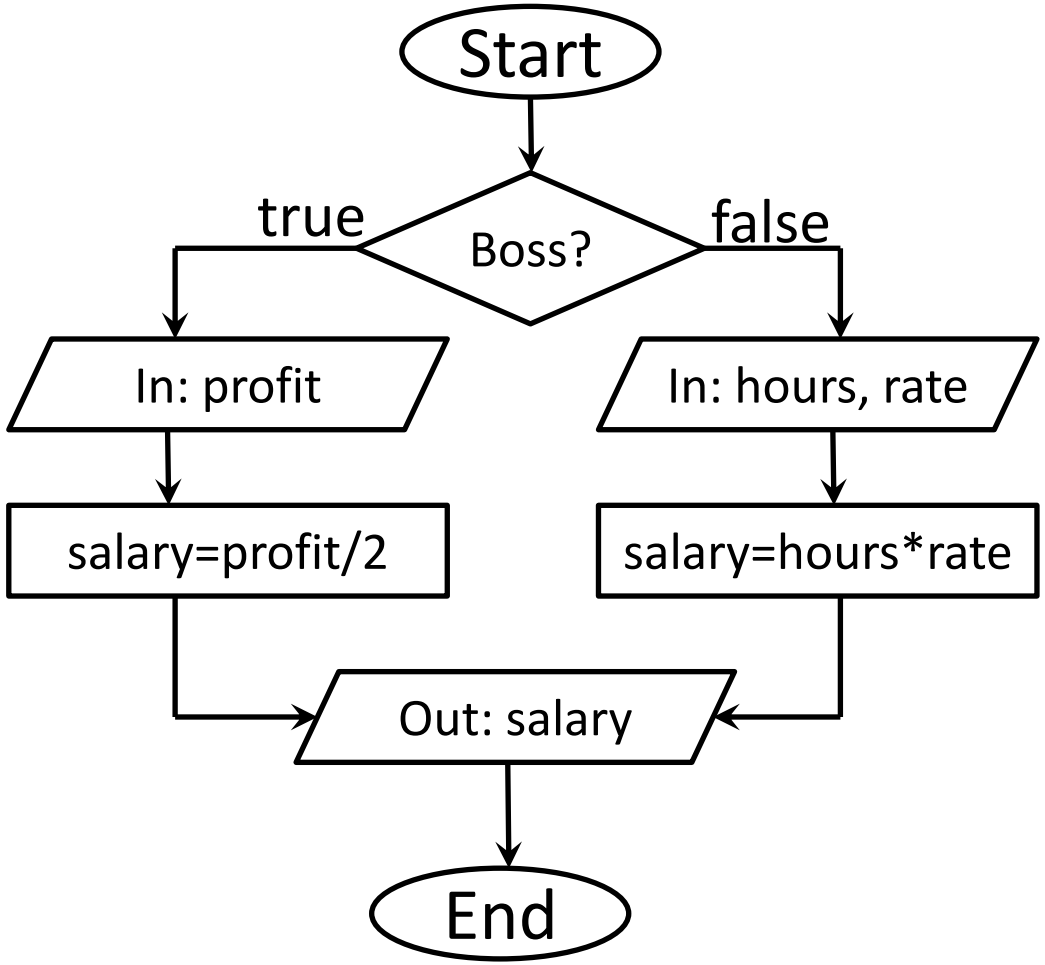


Extending algorithms

Original:

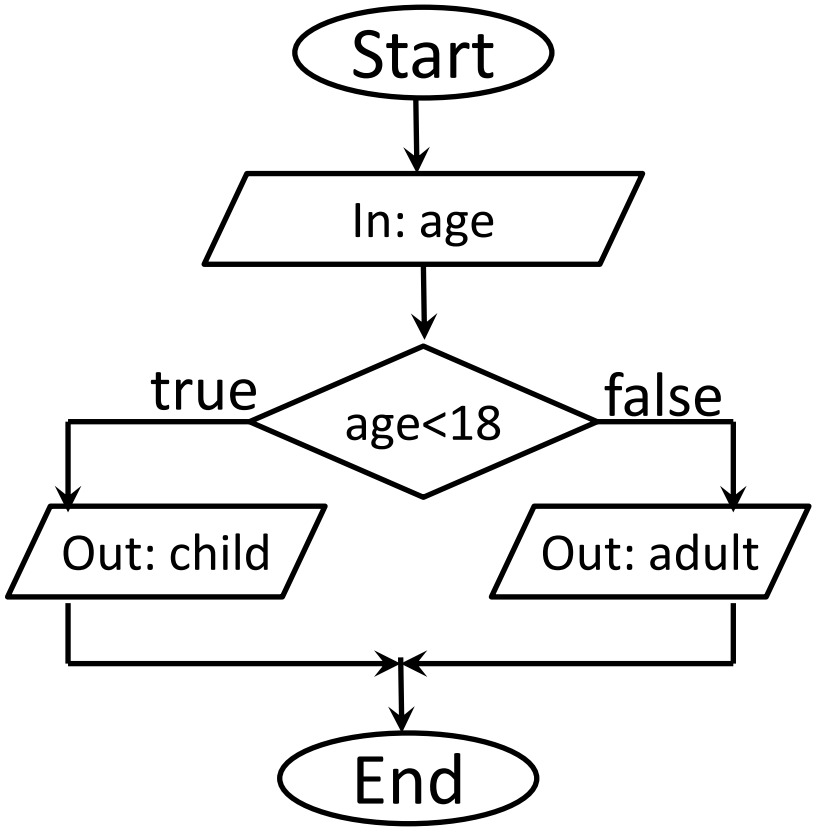


Extended:

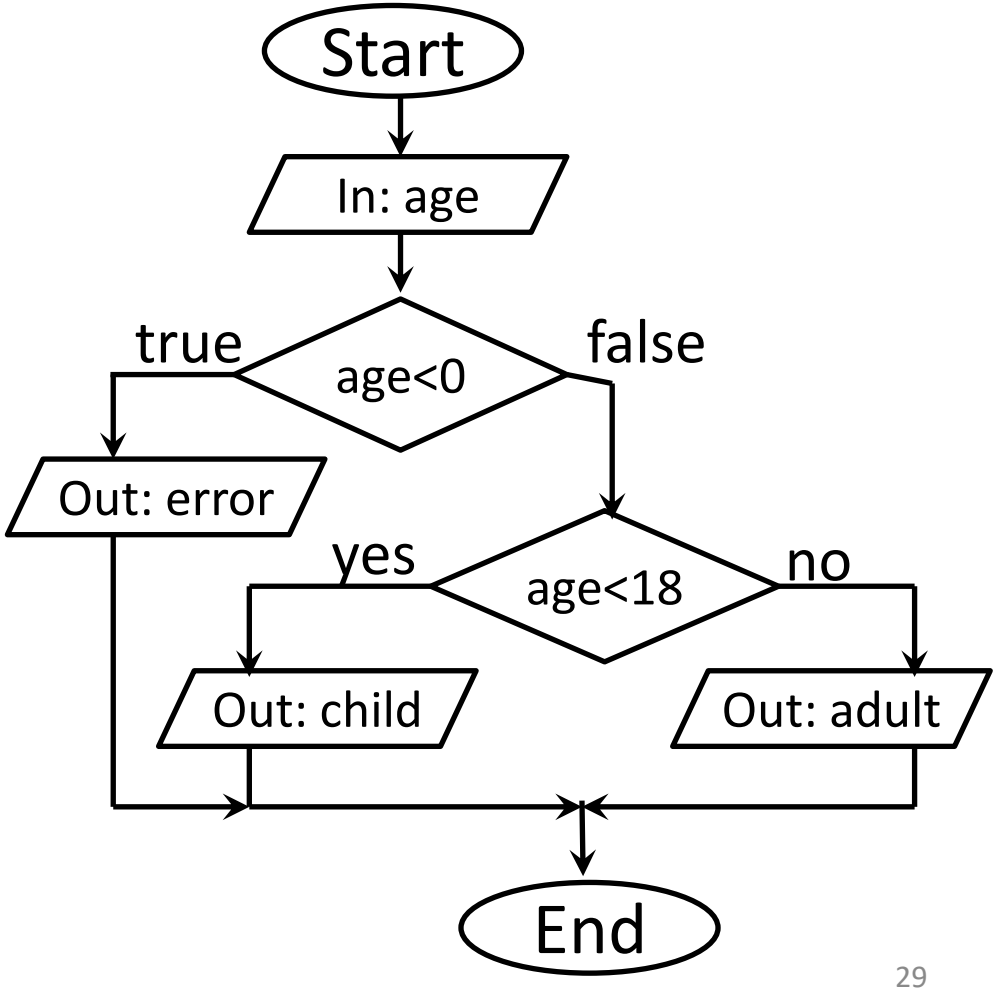


Foolproofing algorithms

Original:

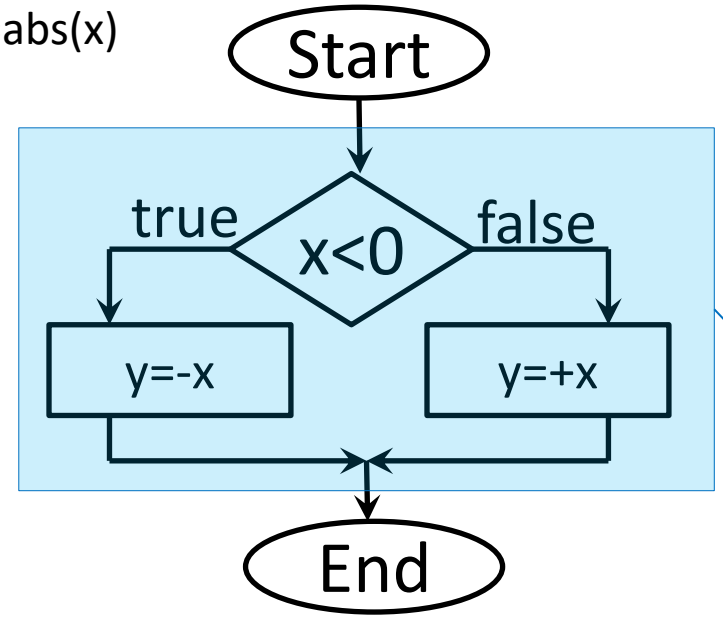


Foolproofed:

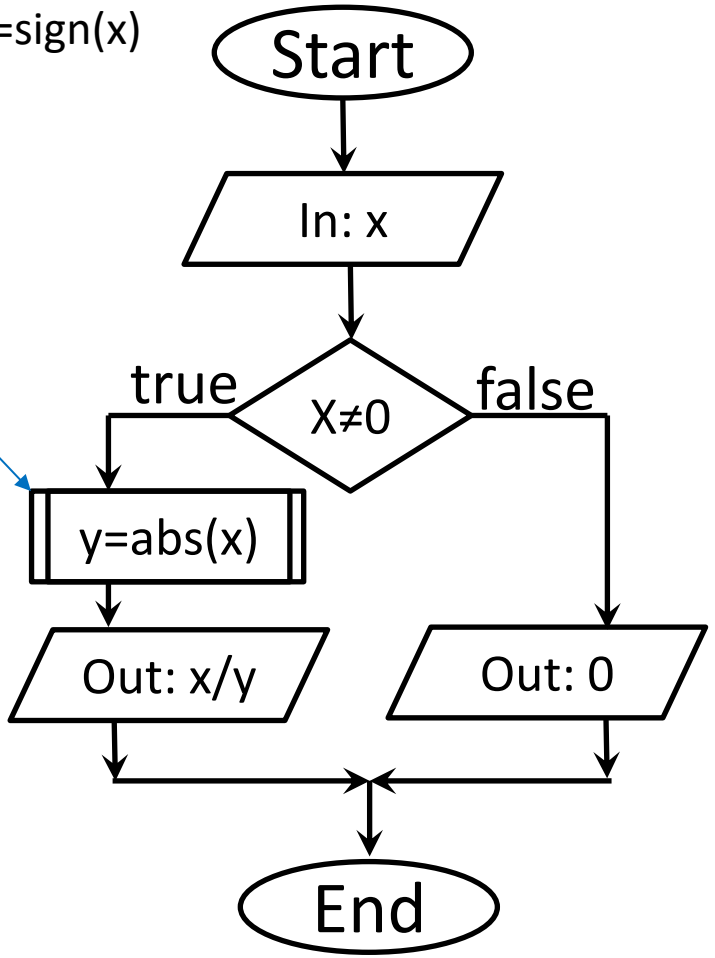


Embedding algorithms

Original:
y=abs(x)



Embedded:
y=sign(x)



Alternative algorithms

There are often many ways to achieve the same thing.

Algorithms can be different in structure, but they can be equivalent in behavior.

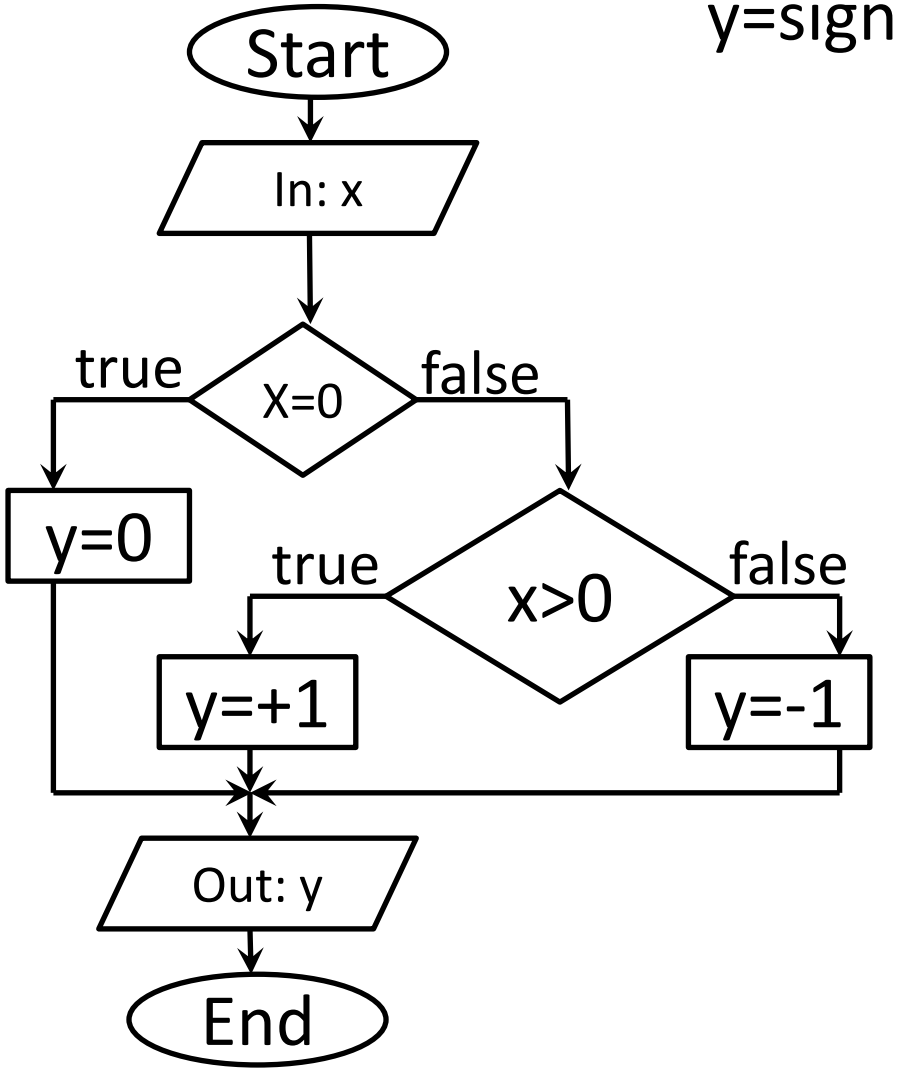
It means for identical input data, they will produce identical results.

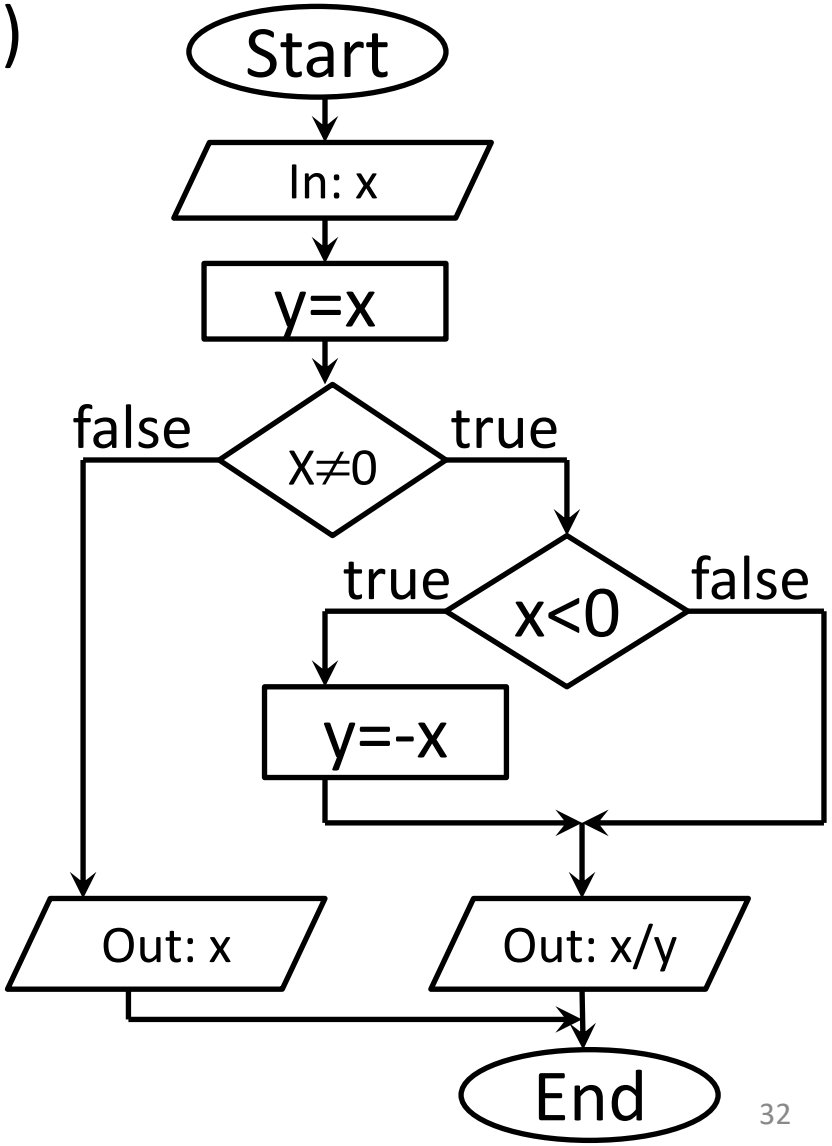
Sometimes there is serious reason to prefer one algorithm over the other, while sometimes there isn't.

In some cases, one algorithm may be considerably smaller, faster, simpler, or more reliable than another.

Alternative algorithms

y=sign(x)





Properties of algorithms

- **Complete**:
  all of actions must be exactly defined
- **Unambiguous**:
  there is only one possible way of interpreting actions
- **Deterministic**:
  if the instructions are followed, it is certain that the desired result will always be achieved
- **Finite**:
  the instructions must terminate after a limited number of steps

Wrong algorithms

How to get to the 5th floor from 2nd by elevator?

1. Call the lift.
2. Get in.
3. Push ‘5’ button.
4. Wait.
5. If the door opens, get out.

Problems (not complete):

- If the lift goes downward…
- If the lift stops on 3rd floor for somebody…

Wrong algorithms

How to make fried chicken?

1. Put the chicken into the oven.
2. Set the temperature.
3. Wait until it is done.
4. Serve it.

Problems (ambiguity):

- What is the optimal temperature (50°C or 200°C)?
- Is the chicken frozen or alive?
- When is it done?

Wrong algorithms

How to be a millionaire?

1. `Buy a lottery.`
2. `Choose numbers.`
3. `Wait for prize or be sad.`

Problems (stochastic, not deterministic):

- In most of the cases we won't be a millionaire.
- Not always works.

Wrong algorithms

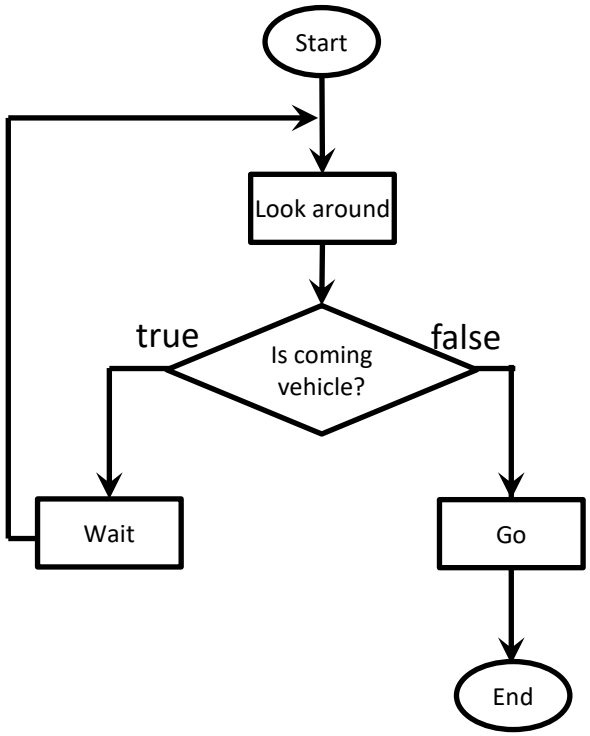
How to use a bus?

1. `Wait for the bus.`
2. `Get on the bus.`
3. `Buy a ticket.`
4. `Sit down.`
5. `Get out of the bus.`

Problems (infinite):

- If we are not in a bus stop, bus won't stop.
- If we are in a building, bus will never arrive.

Crossing straight road on foot



Problems:

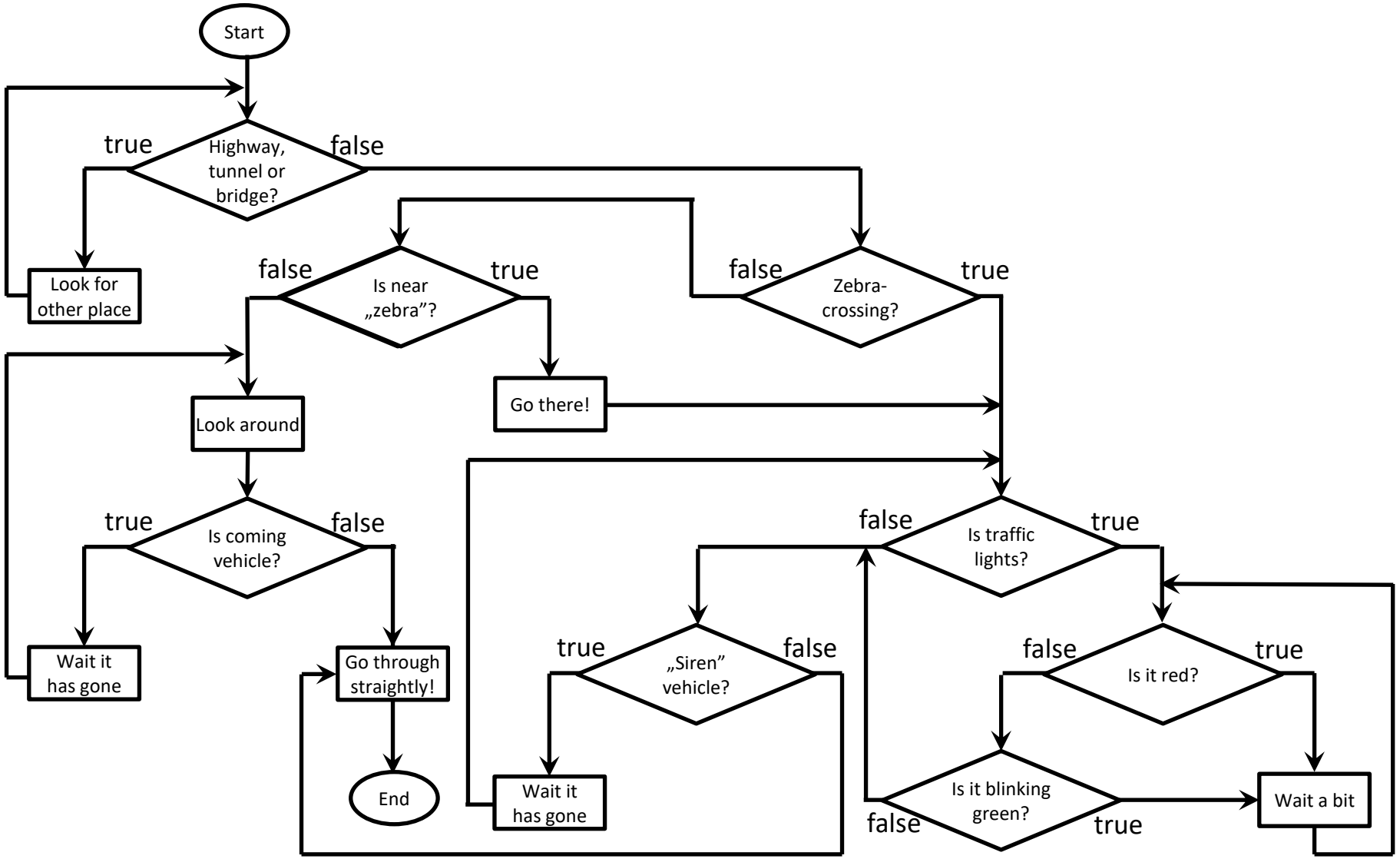
- Not complete
- Ambiguous
- …

Modification:

- Generalizing
- Extending
- Foolproofing
- Completing

Create a more detailed algorithm.

Crossing straight road on foot



Logical operations and expressions

- Logical operations

AND	false	true
false	false	false
true	false	true

OR	false	true
false	false	true
true	true	true

XOR	false	true
false	false	true
true	true	false

- Logical opposite

If this is true,		then it is false
=	==	≠
<	<	≥
>	>	≤
≠	!=	=
≤	<=	>
≥	>=	<

true	=	NOT false
false	=	NOT true
X	=	NOT NOT X

Defined operations

+	Addition, pozitive sign
-	Subtraction, negative sign
*	Multiplication
/	Division
%	Modulo operation (remainder)
(int)	Truncation to integer
[...]	Array indexing
{...}	Array initialization
(...)	Precedence, parameter list
=	Assignment

==	Equality
!=	Not equality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
AND	Logical „and" operation
OR	Logical „or" operation
NOT	Negation (logical „not" operation)
,	Separator of values in lists

The numbers and operations are interpreted in the **decimal number system**.
The `Length` can mean the value of a quantity, while `"Length"` is just a text.

Order of operations

I. Parenthesis
   1. `( )`
II. Unary operations
   1. `sign (+ -)`
   2. `NOT`
   3. `(int)`
III. Arithmetic operations
   1. `* / %`
   2. `+ -`
IV. Relational operations
   1. `< <= > >=`
   2. `== !=`
V. Logic operations
   1. `AND`
   2. `OR`

Order of operations

Frequent issues:

- What is the double of the sum of 3 and 4?
  ~~2*3+4~~ 2*(3+4)
- Is the value of variable `x` between 5 and 10?
  ~~5 < x < 10~~ (5 < x) AND (x < 10)
- Is the value of variable `x` equal to 5 or 10?
  ~~x == 5 OR 10~~ (x == 5) OR (x == 10)

- Write the `size` word!
  ~~output size~~ output "size"

Exercise: order of operations

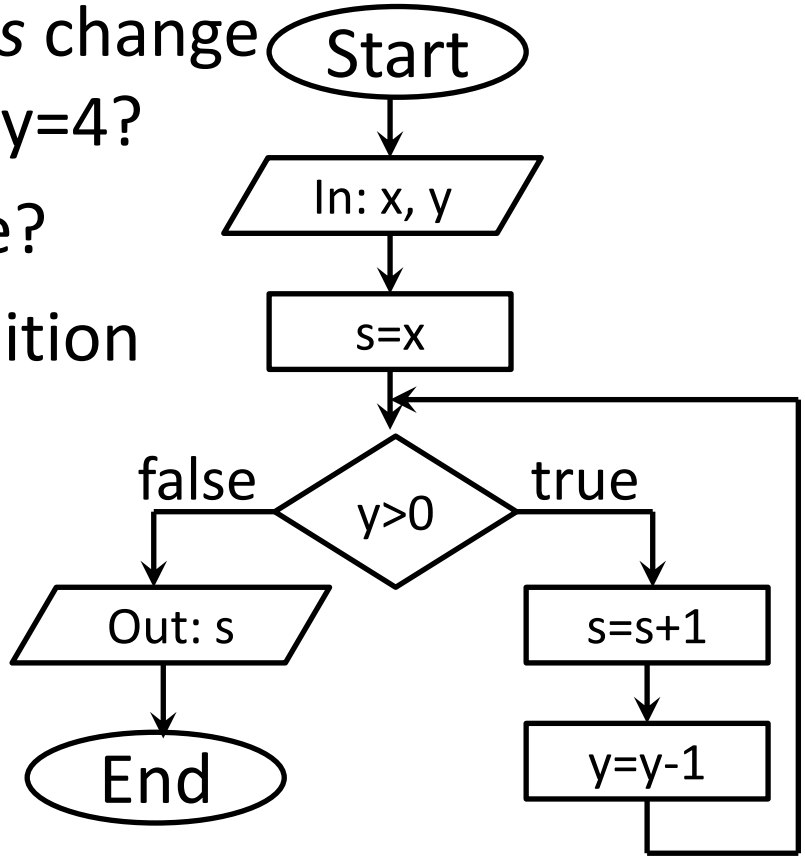
What is the value of the following expressions, if initially `a=10` and `b=20`?

- `a-1/2`
- `2+b / a+1`
- `(int)a/b`
- `-a-b/-2`
- `NOT ( a <= b )`
- `a == 10 OR b > a AND a * b != 200`
- `b+-a*2!=1/2 OR (int)(a/4)==2.5`

Link to the solution

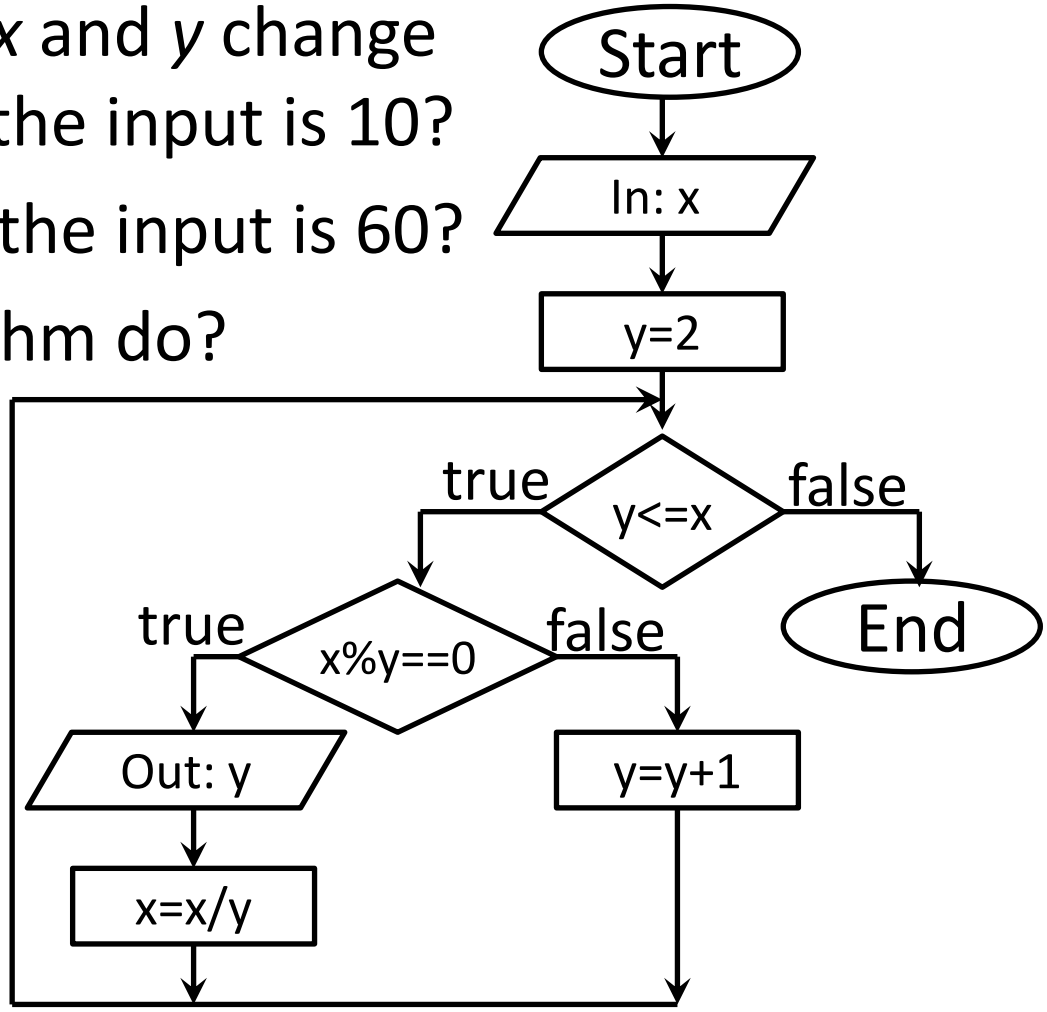
Exercise: flowchart

- How do the values of *x, y* and *s* change during the process, if x=5 and y=4?
- What is the output in this case?
- How many times will the condition evaluated?
- What does this algorithm do?
- How can you modify it to calculate the product of *x* and *y*?



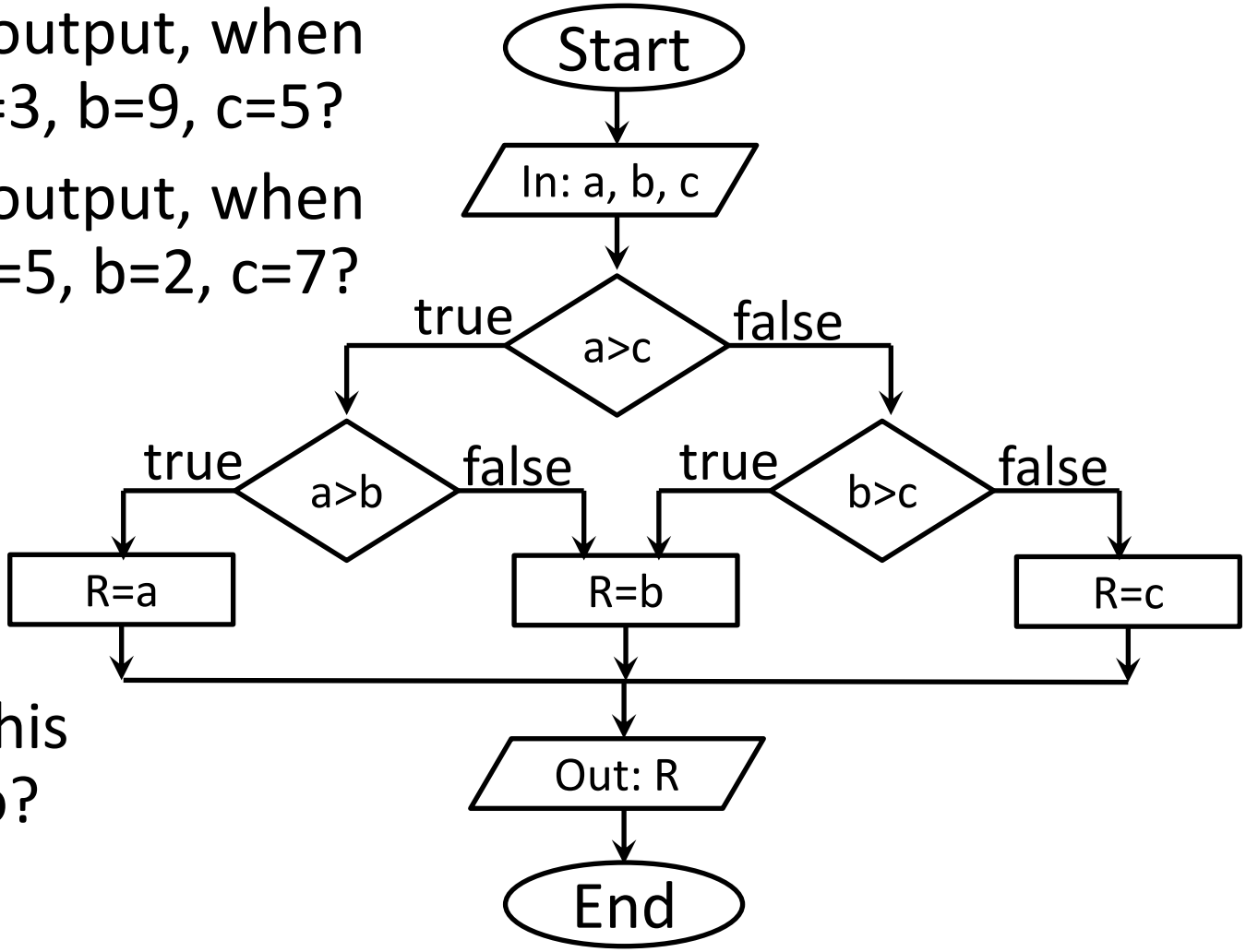
Exercise: flowchart

- How do the values of *x* and *y* change during the process, if the input is 10?
- What is the output, if the input is 60?
- What does this algorithm do?
- Is it work, if x=1?
- If the input is 24, how many iterations will be executed?



Exercise: flowchart

- What is the output, when the input: a=3, b=9, c=5?
- What is the output, when the input : a=5, b=2, c=7?
- What does this algorithm do?



Exercise: flowchart

- Create a flowchart for the solution of a first-degree equation given in **a**x+**b**=0 form.
  For example, a=0.5; b=-6 → x=12
- Create a flowchart to tell whether a given year is a leap year or not.
  For example, 2023 → no; 2024 → yes

- Create a flowchart to print 3 numbers (given by the user) in decreasing order.

- Create a flowchart to calculate the factorial of a positive integer number given by the user.

- Create a flowchart to convert a positive decimal integer to binary notation.

Pseudocode

Sequence:	**Selection (branching)**:	**Iteration (loops)**:

statement1
statement2
statement3
…
	if *condition* then
	  statement$_{(true)}$
	else
	  statement$_{(false)}$
	endif
	…
		while *condition* do
		  statement$_{(true)}$
		enddo
		statement$_{(false)}$
		…

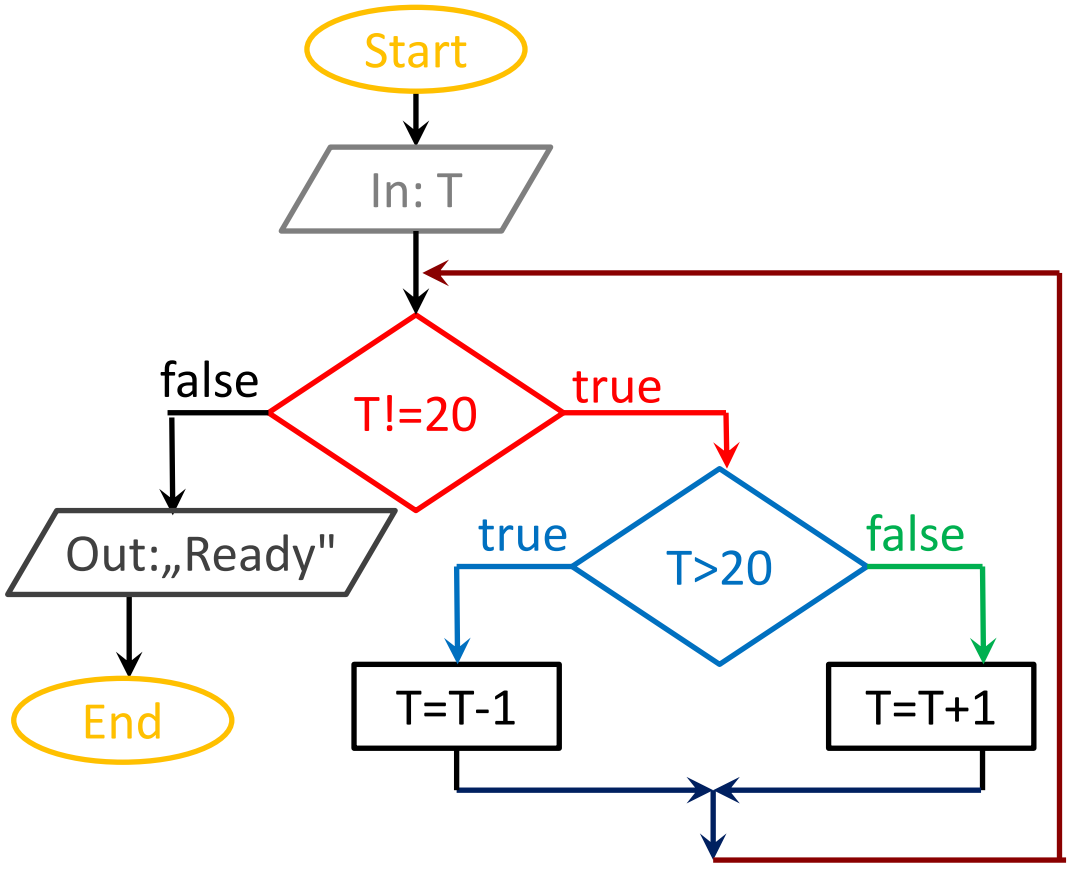
Further keywords: input, output, stop, break, function, endfunction, procedure, endprocedure, return, call

Conversion

flowchart ⇔ pseudocode

- A condition can belong to a selection or an iteration
  - Iteration: if arrow (`enddo`) goes backward
  - Branching: if no way to go back to the condition, and the branches join (`endif`)
- The false branch (`else`) of selection can be skipped
- Repetition happens in case of true condition only
- Branches of a selection can be swapped by the negation of the condition
  - In case of some iteration and selection it can be necessary

Conversion




```
input T
while T!=20 do
   if T>20 then
      T=T-1
   else
      T=T+1
   endif
enddo
output "Ready"
```

Indentation

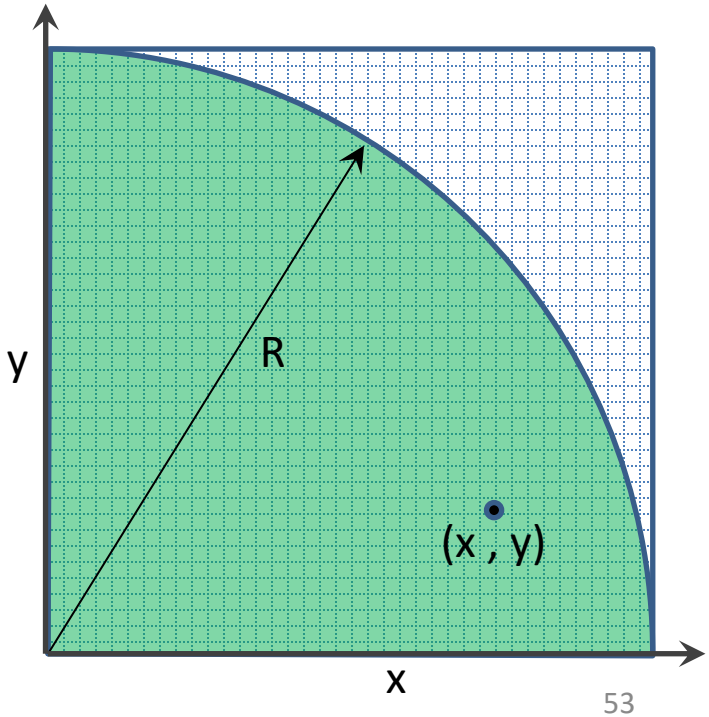
- The lines of a sequence are started in the same position (same level of indentation)
- The body of an iteration indented relative to `while`
  - Between `do` and `enddo`
- The branches of a selection indented relative to `if`
  - Between `then` and `else` as well as between `else` and `endif`
  - Between `then` and `endif` (if no `else` branch)
- Multiple indentation for embedded structures

Pseudocode example

```
input R
i=0
x=0
while x<=R do
   y=0
   while y<=R do
      if x*x+y*y<=R*R then
         i=i+1
      endif
      y=y+1
   enddo
   x=x+1
enddo
output 4*i/((R+1)*(R+1))
```

Approximation of the value of π

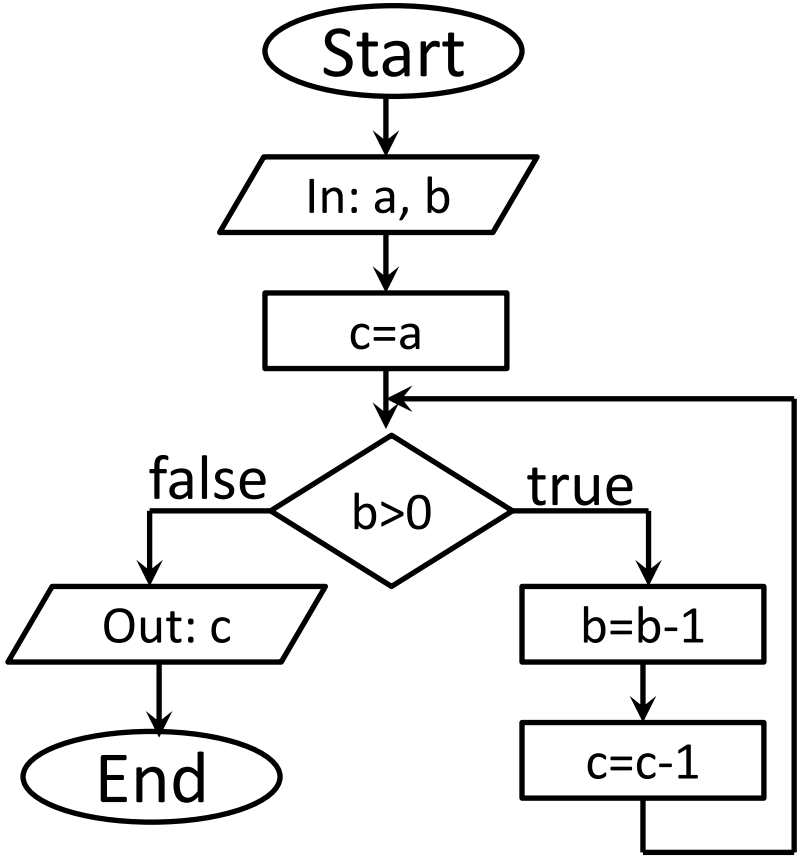
Greater R leads to more precise value



Exercise: Are they the same?

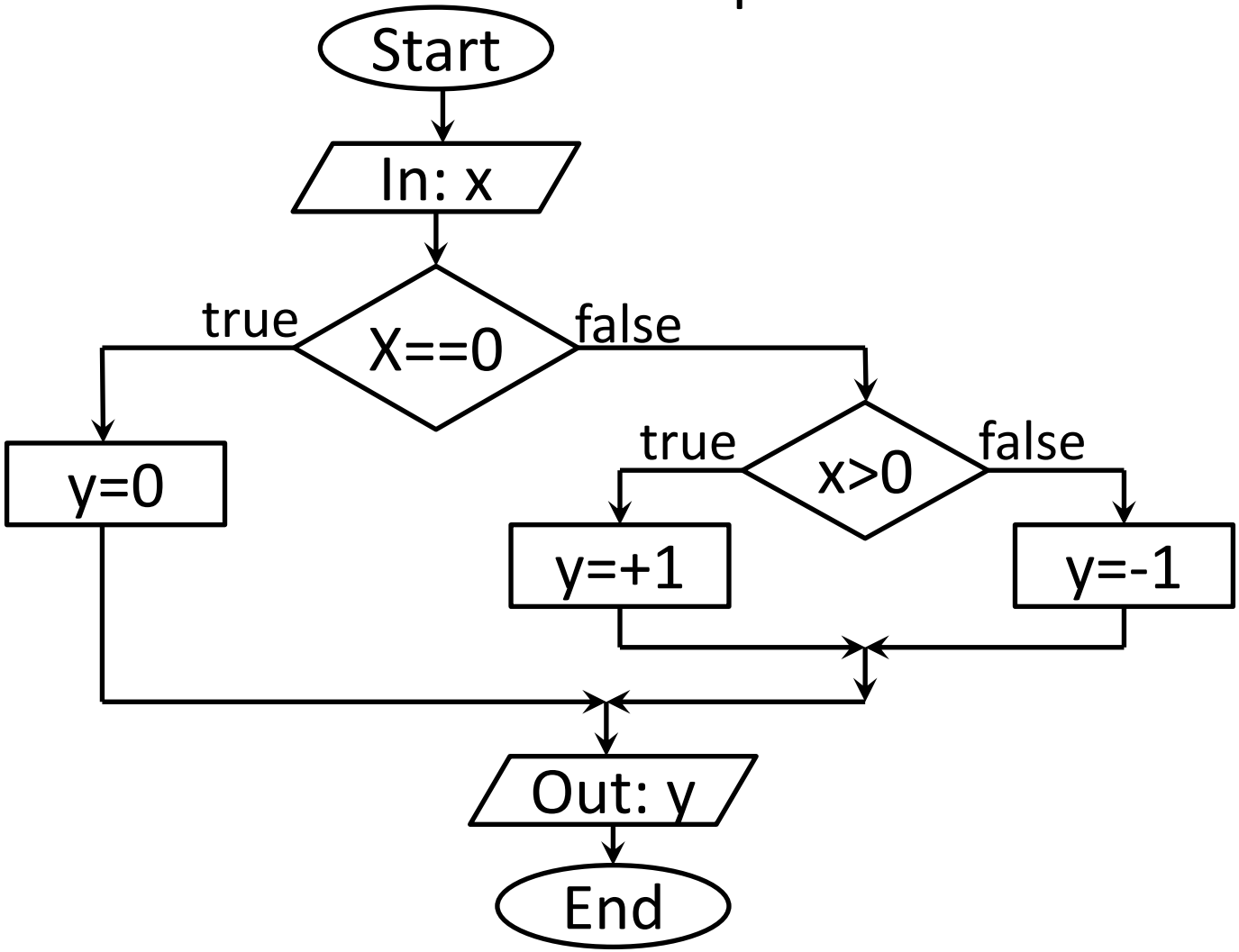
```
input a
input b
c=a
if b>0 then
   b=b-1
   c=c-1
else
  output c
endif
```

- Do the pseudocode and the flowchart describe the same algorithm?



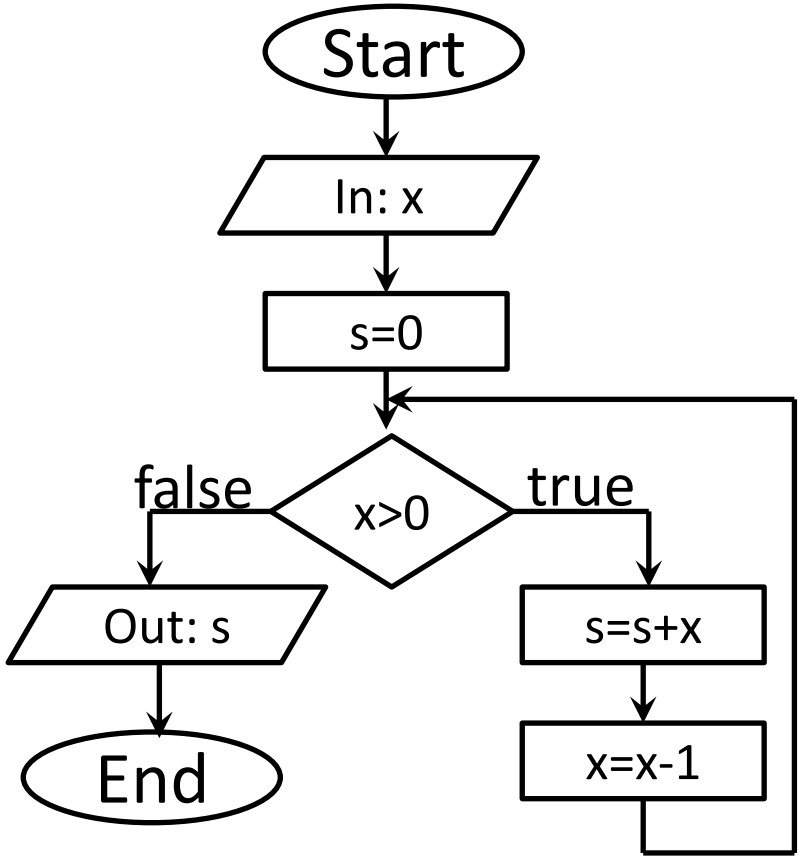
Exercise: conversion 1

- Represent this flowchart with pseudocode.



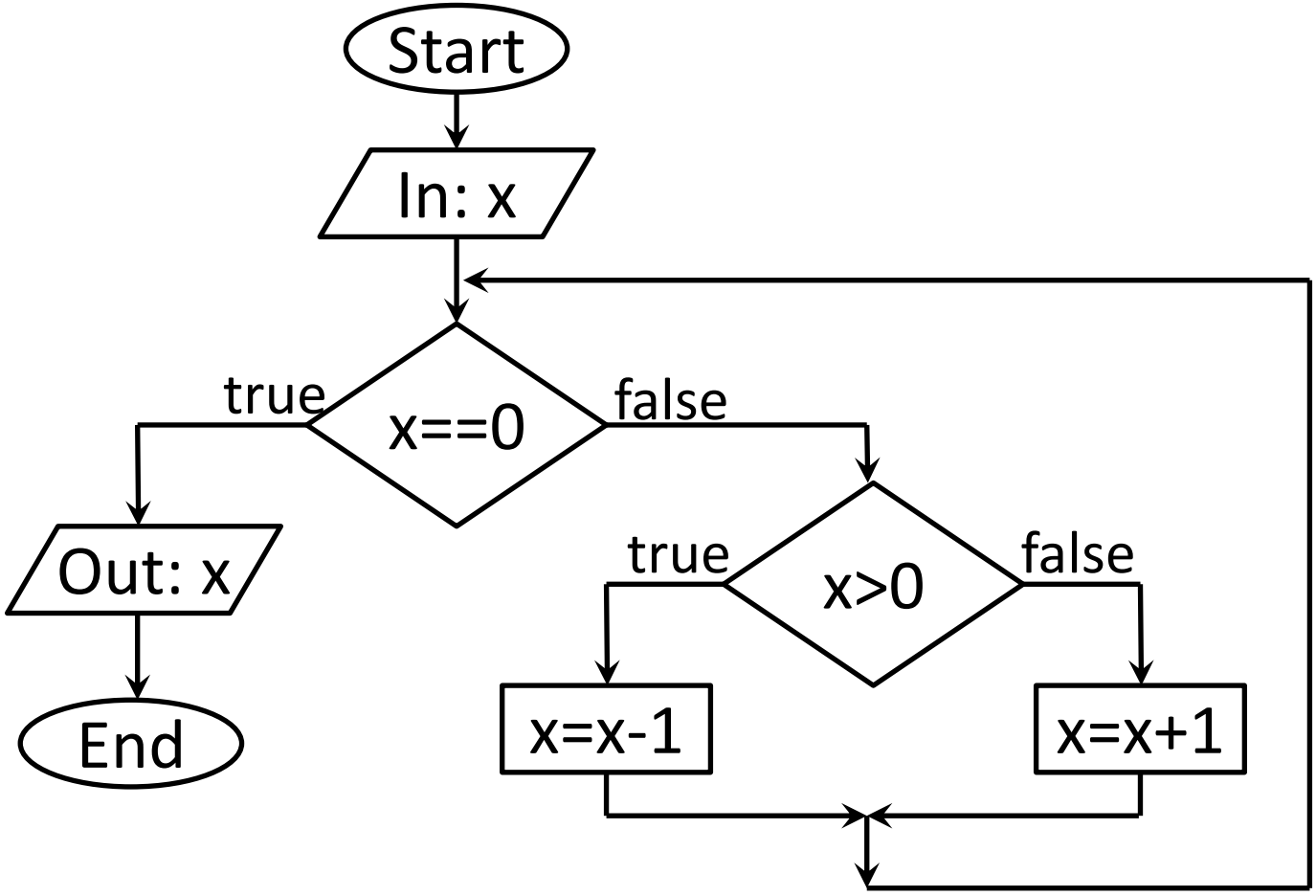
Exercise: conversion 2

- Represent this flowchart with pseudocode.



Exercise: conversion 3

- Represent this flowchart with pseudocode.



Exercise: pseudocode

```
input a
if a<0 then
   b=-1*a
else
   b=a
endif
output b
```

- What is the output if a=10?
- What is the output if a=-4?
- What does the algorithm do?

- What does this algorithm do?

```
input a
if a<0 then
   a=-1*a
endif
output a
```

Exercise: pseudocode

```
input a
input b
c=a
while b>0 do
    b=b-1
    c=c-1
enddo
output c
```

- How do the values of *a, b* and *c* change during the process, if a=7 and b=3?
- What is the output in this case?
- How many times will the condition evaluated?
- What does this algorithm do?
- Convert it to flowchart.

Exercise: pseudocode

```
input N
R=0
while N>0 do
   R=R*10+N%10
   N=(int)(N/10)
enddo
output R
```

- How do the values of *N* and *R* change during the process, if N=73251 initially?
- What is the output in this case?
- What does this algorithm do?

Legend:

%: modulo operation
(remainder after division)

(int): integer part
(ignore fractional part)

Exercise: pseudocode

```
input N
input B
R=0
P=1
while N!=0 do
   R=R+(N%B)*P
   P=P*10
   N=(int)(N/B)
enddo
output R
```

- What is the output, if N=15, B=2?
- What is the output, if N=16, B=2?
- What is the output, if N=10, B=2?
- What is the output, if N=5, B=2?
- What is the output, if N=30, B=3?
- What is the output, if N=20, B=3?
- What is the output, if N=64, B=8?

- What does this algorithm do?

Exercise: pseudocode

```
input A
input B
while B>0 do
   C=B
   B=A%B
   A=C
enddo
output A
```

- How do the values of *A*, B and *C* change during the process, if A=24 and B=18 initially?
- What is the output in this case?
- Try it with A=30 and B=105.
- Try it with A=165 and B=48.
- What does this algorithm do?

(Euclidean algorithm)

Exercise: pseudocode

```
input x,y
z=x*y
while y>=1 do
   w=y
   y=x%y
   x=w
enddo
output z/x
```

- How do the values of *x,* y, z and *w* change during the process, if x=12 and y=30 initially?
- What is the output in this case?
- Try it with x=14 and y=15.
- Try it with x=18 and y=18.
- What does this algorithm do?

Exercise: pseudocode

```
input A
input B
while A!=B do
   if A>B then
      A=A-B
   else
      B=B-A
   endif
enddo
output B
```

- How do the values of *A and* B change during the process, if A=24 and B=18 initially?
- What is the output in this case?
- Try it with A=30 and B=105.
- Try it with A=165 and B=48.
- What does this algorithm do?

- Create a flowchart for this algorithm.

Exercise: congruential generator

```
input m, a, x
x0=x
while 1==1 do
   x=(a*x)%m
   output x, " "
   if x==x0 then
     break
   endif
enddo
```

What is the output of the algorithm in case of the following inputs?

- a=5, m=16, x=1
- a=12, m=7, x=6
- a=12, m=7, x=1
- a=16807, m=2147483647, x=1672552800

Exercise: even or odd

Algorithm represented by natural language:

1. Get a number.
2. Check that it is larger than one or not.
3. If it is larger, subtract two and continue with Step 2.
4. Otherwise check it zero or not.
5. If it is zero, write ‘E’.
6. Else write ‘O’.

Write this algorithm in pseudocode (and with flowchart).

Exercise: selections

- 3 numbers are given. Write the pseudocode algorithm to determine the minimal value of them.

- 3 positive numbers are given. Is it possible to draw a triangle having these 3 side lengths? Write a pseudocode for the triangle inequality problem.

- 3 positive numbers are given. Is it possible to draw a right-angled (90°) triangle having these 3 side lengths?

Exercise: iterations

- Write the pseudocode of the algorithm, which gives the sum of integers within the [10; 20] closed interval.
- Write the pseudocode of the algorithm, which gives the factorial of a positive integer given by the user.
  For example: 6! → 720
- Write the pseudocode of the algorithm, which gives the value of the x^y power. The x and y values are given by the user.

Exercise: square root

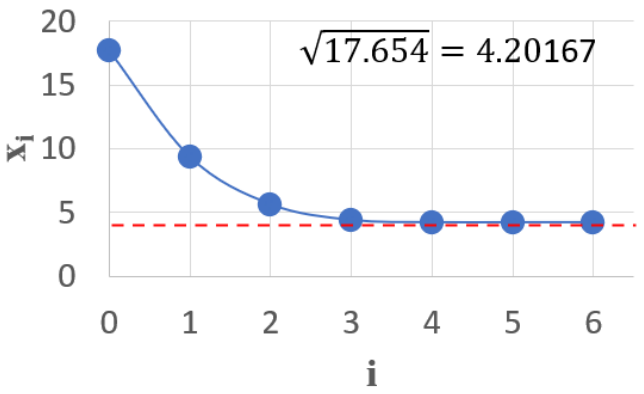
The square root of a number can be approximated by the Newton-Raphson method. The following iterative formula is needed, which converges to the solution.

- Write an algorithm, which does the iteration until the difference between two subsequent values is less than the threshold given by the user.

$$x_0 = N$$

$$x_{i+1} = x_i - \frac{x_i^2 - N}{2x_i}$$

$$\lim_{i \to \infty} x_i = \sqrt{N}$$



Exercise: primes

- Write the pseudocode of the algorithm, which tell whether a number (integer, above 1) given by the user is a prime or not.

- Write the pseudocode of the algorithm, which prints those prime numbers whose product is equal to a positive integer given by the user.
  (Prime factorization)

Exercise: leap days

- The user tells two years (in increasing order) between 1901 and 2099. Write an algorithm to determine how many leap days are between the first days of the two given years.

For example

input:

1979 2023

output:

***11** leap days are between 1 January 1979 and 1 January 2023.*

Exercise: fractions

A traditional fraction can be given by 2 integers.

- Write the pseudocode of the algorithm, which can simplify traditional fractions.
  For example: 36/90 → 2/5
- Write the pseudocode of the algorithm, which can do operations (+ - * / separately) on fractions.
  For example : 4/6+2/8→11/12; (2/3) / (4/5) → 5/6
- Write the pseudocode of the algorithm, which can convert a decimal real number to traditional fraction.
  For example : 0.375 → 3/8

Exercise: sequences

The Fibonacci sequence starts with 0 and 1, then each further element is the sum of the previous 2 values.

- Write the pseudocode of the algorithm, which …
  … print the first 100 elements of Fibonacci sequence.
  … print the elements of the sequence below 1000.

The Collatz conjecture defines a sequence, started by an arbitrary positive integer. Each further element (a_{i+1}) is affected by the prior value (a_i). If a_i is even, then $a_{i+1}=a_i/2$, otherwise $a_{i+1}=3a_i+1$.

- Write the algorithm generating all elements until $a_i=1$, strated by a user-given value.

Arrays

- Sequence of N elements, where all item is accessible by an index (integer, between 0 and N-1)

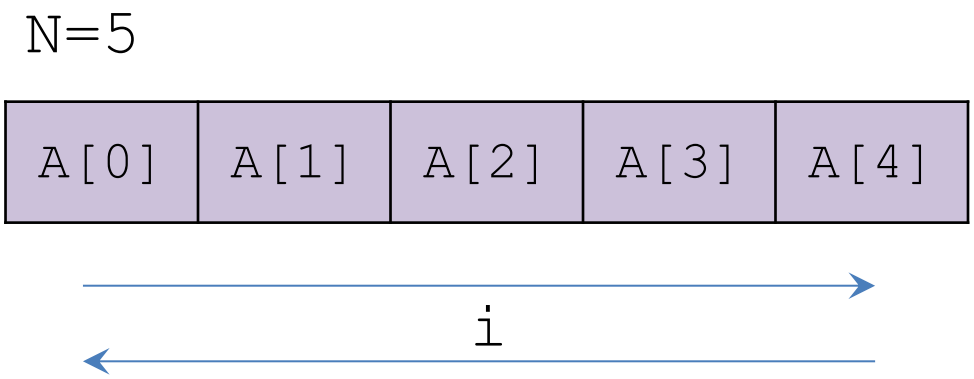
```
N=5
A[]={2,93,4,-1,8}
sum=0
i=0
while i<N do
   sum=sum+A[i]
   i=i+1
enddo
output sum/N
```

initialization:
`A[0]=2, A[1]=93, …, A[4]=8`

Array example

- 5 numbers in array: reading, storing, writing back in reverse order

```
N=5
A[]={0,0,0,0,0}
i=0
while i<N do
   input A[i]
   i=i+1
enddo
while i>0 do
   i=i-1
   output A[i], ” ”
enddo
```



Exercise: signal coding

Given an array of N elements, where all the values are 0s and 1s (bits) in arbitrary order.

- Generate Manchester code for these values, so if a value is 1, then write a zero and a one, otherwise a one and a zero.
- Generate NRZI codes for these values, so first write a zero (independently of the first array element). After then if the given array value is 0 write the previously written value otherwise the opposite of the previous value.

Exercise: number of coins

Create an algorithm with pseudocode, which can tell the minimal number of required coins to pay an amount of money given by the user.

- Possible coins: 1, 2, 5, 10, 20, 50, 100
- Possible coins: 5, 10, 20, 50, 100, 200 (applying rounding rule of shops)

Searching and sorting

- **Searching**
  Is the given value in the array (and if yes, where)?
  - Linear search (with/without sentinel), binary search, etc.
- **Sorting**
  Rearrange elements of array in increasing/decreasing order (in their original location).
  - Selection sort, insertion sort, bubble sort, radix sort, quick sort, heapsort, etc.

Exercise: searching

Suppose that `A` is an initialized array having `N` elements.

- Write a pseudocode to tell whether a value given by the user is stored in the (unsorted/sorted) array. (linear search with/without sentinel, binary search)
- How the number of steps depends on N?
- Write a pseudocode to tell the value of the largest/smallest element of the array.
- Write a pseudocode to tell the index of the first occurrence of the largest element of the array.
- How can we find the second greatest element?

Exercise: sorting

- How to swap the values of two variables?

Suppose that `A` is an initialized array having `N` elements.

- Write a pseudocode to sort the elements of the array in increasing order …
  … according to the selection sort algorithm
  … according to the insertion sort algorithm
  … according to the bubble sort algorithm
  … according to the radix sort algorithm

- How the number of steps depends on N?
- How to make only one sorted array from all the elements of two sorted arrays?

Exercise: bank card number

Bank Card Number (BCN): $a_1, a_2 \ldots, a_{16} \quad 0 \leq a_i \leq 9$

Replace each odd-position digit of a BCN with its double or if this product is greater than 9 reduce it by 9. Then calculate the sum of all digits. The BCN is valid if and only if this sum is dividable by 10.

So, a BCN is valid if

$$\left(\sum_{i=1}^{16} \left((i\%2 + 1) * a_i - \left\{ \frac{a_i}{5} \right\} * (i\%2) * 9 \right) \right) \% \, 10 == 0$$

The 16 digits of a BCN are stored in an array (each digit in a separate array element). Write pseudocode that checks the validity of a BCN.

Exercise: most frequent age

- The number of inhabitants of a town below 18 years is denoted by `N`. The ages of each child are stored in an array called `Age`. Write an algorithm in pseudocode, which tells which age is the most frequent.

For example:

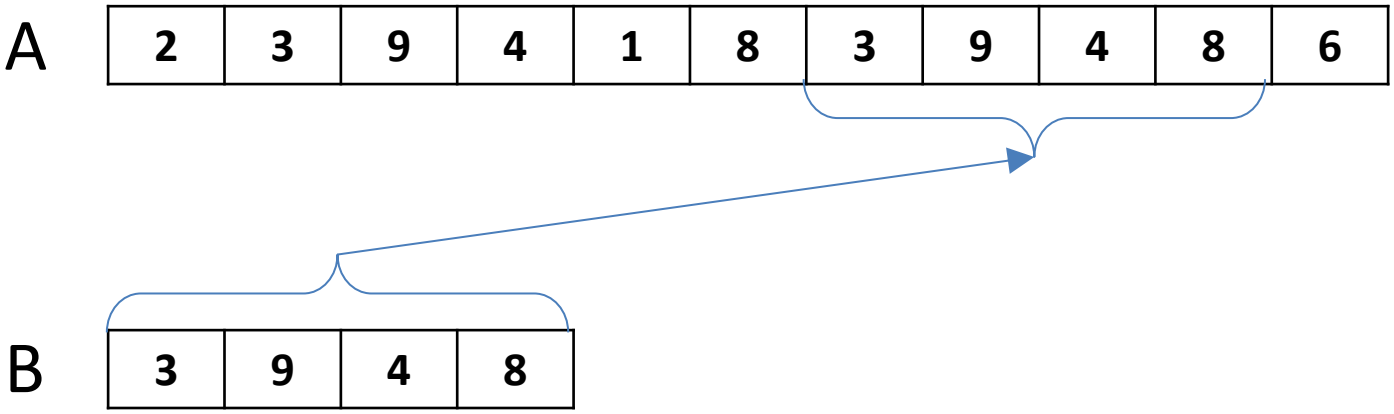
```
N=15
Age[]={1,3,2,0,5,10,17,3,10,0,3,2,15,1,3}
```

The most children is **3** years old.

Exercise: finding a pattern

- Two arrays are given. Array A has N elements and array B has M elements. Write a pseudocode to represent the algorithm determining whether all the elements of array B (continuously, in the same order) can be found in array A or not.

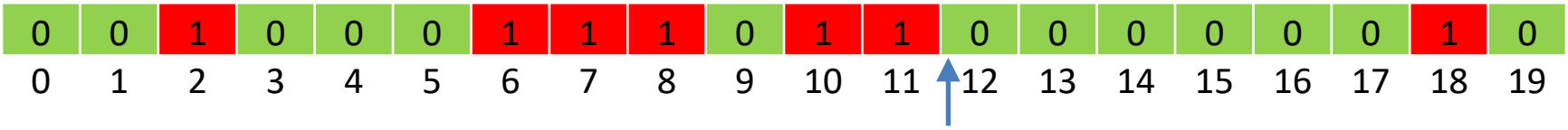
For example:



Exercise: first-fit allocation

The elements of the array (called `M`) illustrate the free and occupied memory cells. Their number is denoted by `N`. `M[i]=1` means that the i^{th} memory cell is occupied and `M[i]=0` means that the i^{th} cell is free.

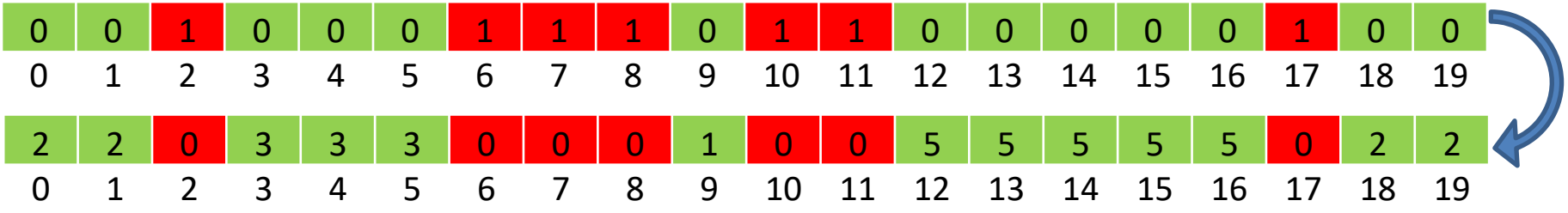
- Write a program, which asks for a positive integer number (`S`) and tells the index of the first cell of the first memory area containing at least `S` free cells continuously.
  For example, in the case of S=4, the answer is 12.



Exercise: memory-map conversion

The elements of the array (called `M`) illustrate the free and occupied memory cells. Their number is denoted by `N`. `M[i]=1` means that the i^{th} memory cell is occupied and `M[i]=0` means that the i^{th} cell is free.

- Write a program to replace the original 1s to 0s and to replace the original 0s to positive integers showing the number of 0s in the original continuous environment of the given cell.

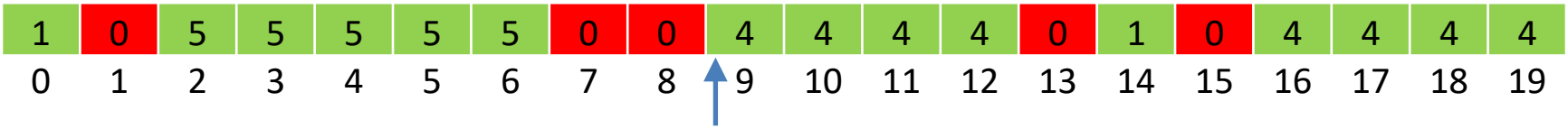


Exercise: best-fit allocation

The elements of the `M` array illustrates memory cell allocations. There are `N` elements. `M[i]=0` mean that the i^{th} memory cell is occupied. `M[i]=x (x>0)` means the i^{th} cell is a part of a continuous memory area of `x` free cells.

- Write a program, which asks a positive integer number (`S`) and tells what is the index of the first cell of the first occurance of the smallest memory area containing at least `S` free cells (if exists)!

  For example, in the case of S=3, the answer is 9.



Subroutines

Separate unit of algorithms

Tool of **recycling** and **abstraction**

- Recycling: not necessary to type the same code-part at different points of the code. We can teach some activity and then use as a basic instruction. An example of *embedding*.
- Abstraction: not just only one activity, but a set of similar activities, specified by parameters. Realization of *generalization*.

Subroutines

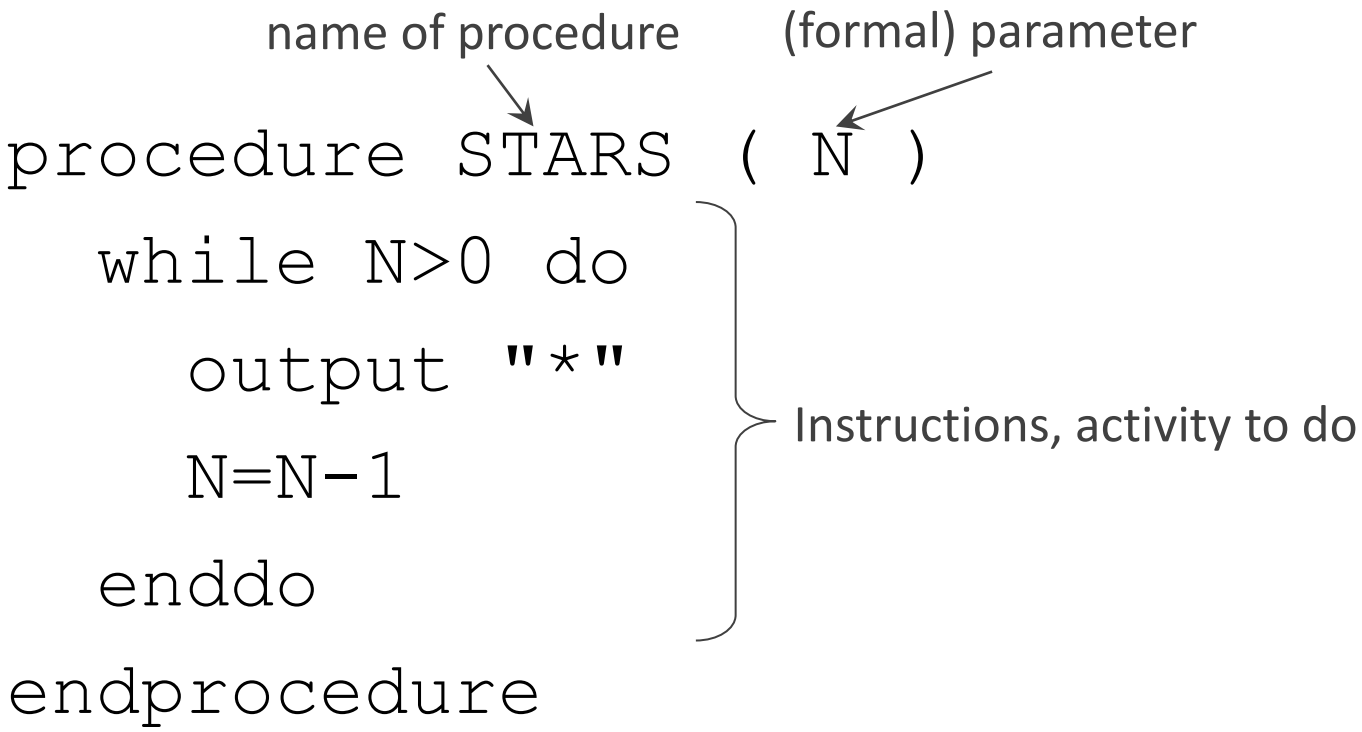
Two sorts of subroutines

- **Function**: a sequence of instructions in order to determine a value somewhere in the code
  E.g., What is the cosine of 45°?
  ```
  x=cos(45)
  ```

- **Procedure**: a sequence of activities to do something at a given point of the code (no return value)
  E.g., Sort the N elements of the A array increasingly.
  ```
  call SortArray(A,N)
  ```

Procedure example

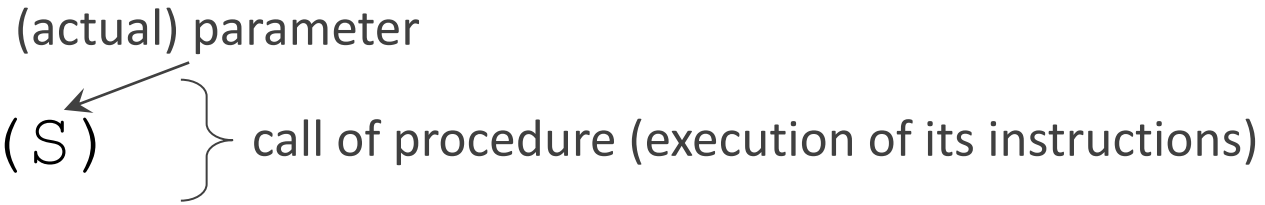
Definition of procedure to print the * character N-times



Procedure example

Calling the previous procedure within a code

```
output "How many stars you need?"
input S
call STARS (S)
output "I hope, you like it."
```

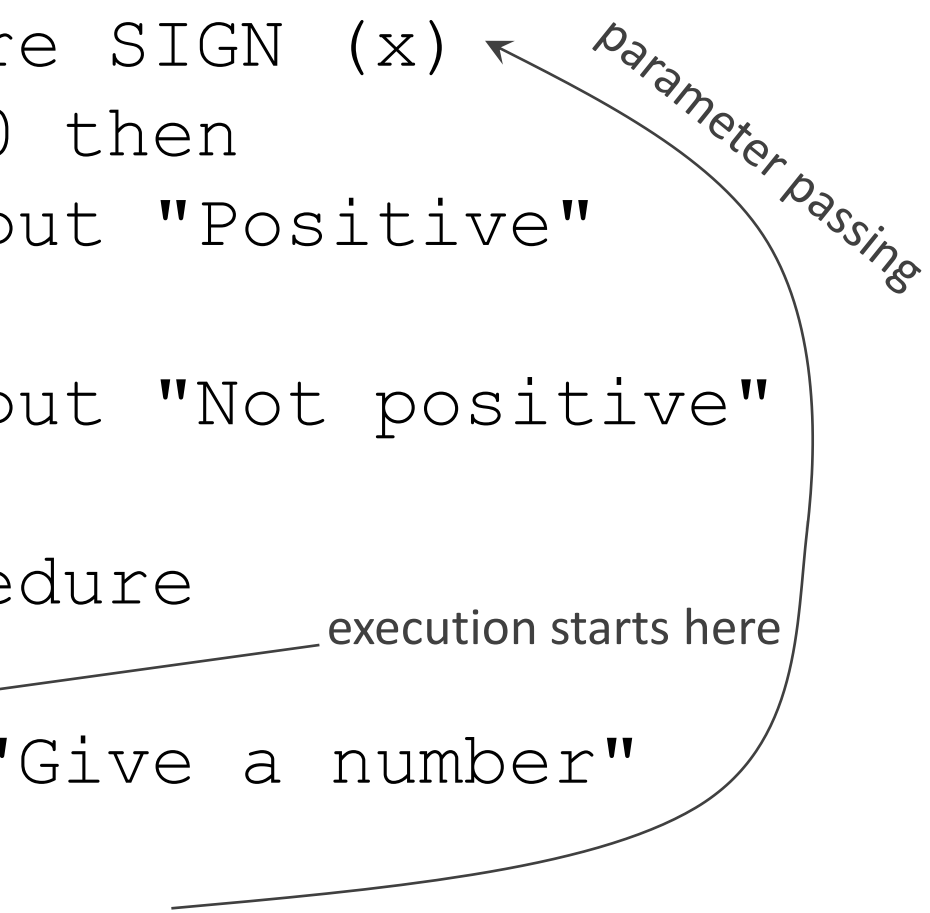


Procedure example

Another procedure to tell the sign of a number

```
procedure SIGN (x)
   if x>0 then
     output "Positive"
   else
     output "Not positive"
   endif
endprocedure

output "Give a number"
input N
call SIGN (N)
```

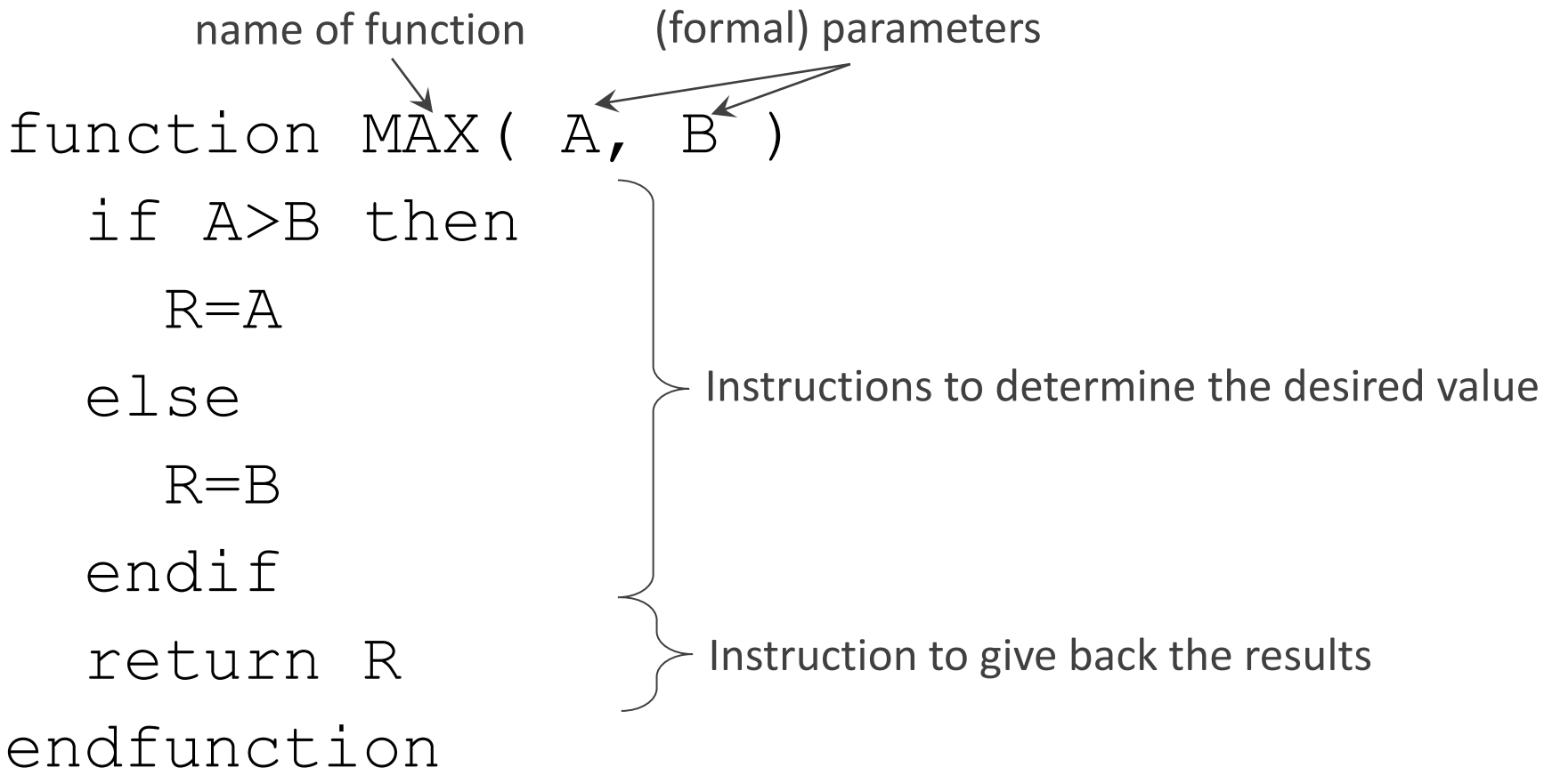


definition of procedure

main unit

Function example

Definition of a function to determine the maximum of two values



Function example

Calling the previous function within a code

```
output „Give two numbers"
input a, b
c = MAX (a,b)
output „The maximum: ", c
```

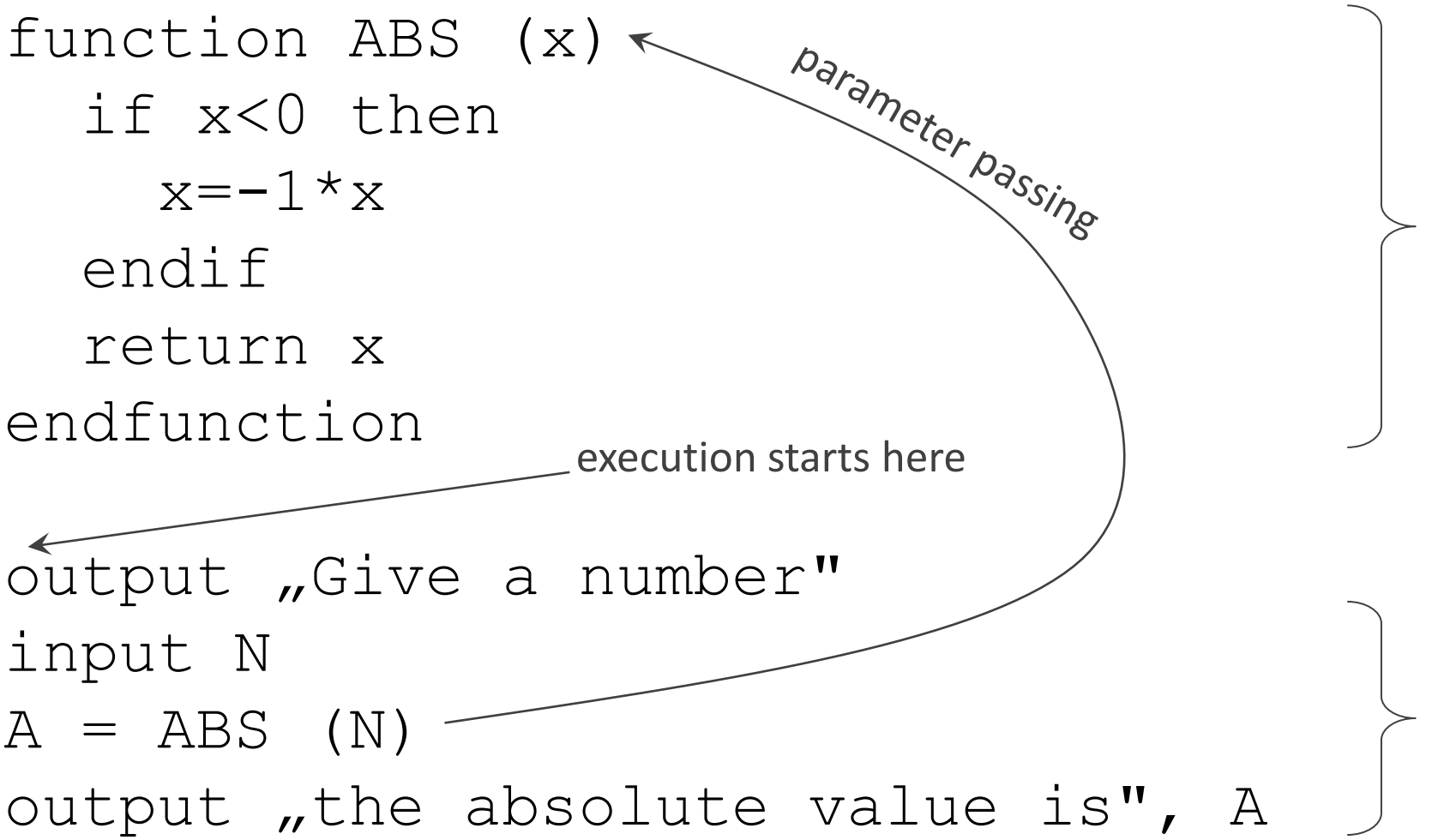
(actual) parameters

call of function (determination of a value)

The returned value is stored in variable c.

Function example

Another function to calculate the absolute value



Procedure vs function

```
procedure Avg(x,y,z)
  a=(x+y+z)/3
  output a
endprocedure
call Avg(2,5,4)
```

≠

```
function Avg(x,y,z)
  a=(x+y+z)/3
  return a
endfunction
if Avg(2,5,4)<4 then
  output "Week!"
else
  output "Good job!"
endif
```

The average is on the screen.

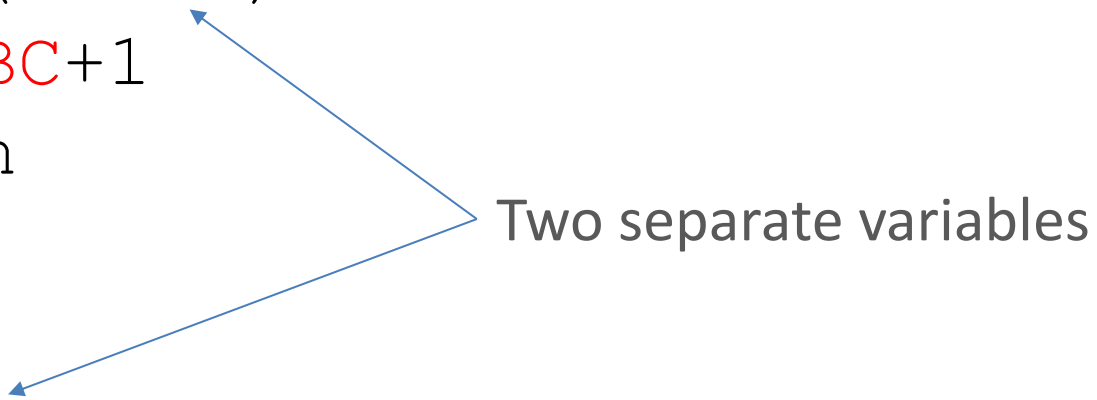
The average is **not** on the screen.

Scope

- Program units (functions, procedures, main program) have own variables
- The names of a variables in two separate program units can refer to different variables
  - Even if the names are the same

```
function F( ABC )
   return ABC+1
endfunction

ABC = 3
ABC = F(7)
ABC = F(ABC)*2
```



Exercise: subroutine

```
function PP( a )
  b=0
  while b<a do
    a = a-1
    b = b+1
  enddo
  if a==b then
    return 1
  else
    return 0
  endif
endfunction

input a, b
a = a*2
b = PP(a+b)+1
output b
```

- How the values of variables are changing during the execution, if the user gives 1 and 4 as inputs?
- What is the output in this case?
- What does the function do when the parameter is a positive integer?

Exercise: subroutine

```
function CHANGE( a )
   return 1-a
endfunction
input Max
i=0
j=0
while j<Max do
   i = CHANGE(i)
   j = j+i
   output j, " "
enddo
output j
```

- What is the output if input is 5?
- What does the program do?
- What is the role of the function?

Exercise: subroutine

```
function min(x,y)
  if x<y then
    return x
  endif
  return y
endfunction
function even(N)
  return 1-N%2
endfunction
C[]={2,3}
input x,y
N=(min(min(x,C[1]),x-3)+C[even(y)]*5)%6
output N
```

- What is on the screen, when the user enters 4 and 5?

Exercise: multplication table

- Write a pseudocode, which contains a procedure to to display the NxN multiplication table.
  For example, if N=4:

Make a new line:

```
output NEWLINE
```

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Exercise: chessboard

- Write a pseudocode, which contains a procedure to create an NxN chessboard pattern of ’0’s and ’1’s. For example

if N=3:

0	1	0
1	0	1
0	1	0

if N=4:

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

Exercise: time

Write an algorithm with pseudocode as follows:

- The code has to contain a function, which has 3 parameters (`hour, minute, second`) and based on this it tells which second of the day is the time.
- The code has to contain a procedure which has an integer parameter (it is the number of seconds in a day) and the procedure prints out the time in `hour:minute:second` format.
- Call these subroutines to check they work well.

Exercise: formula

- Write a pseudocode function, which has only one parameter (N) and returns with the factorial of the value x given by the following formula. Call the function in the main program unit for a positive integer number given by the user and print out the return value of the function.

$$x = \left(\frac{\sum_{i=1}^{N} (2i - 1)}{N^2} \right)$$

Exercise: DCB arithmetics

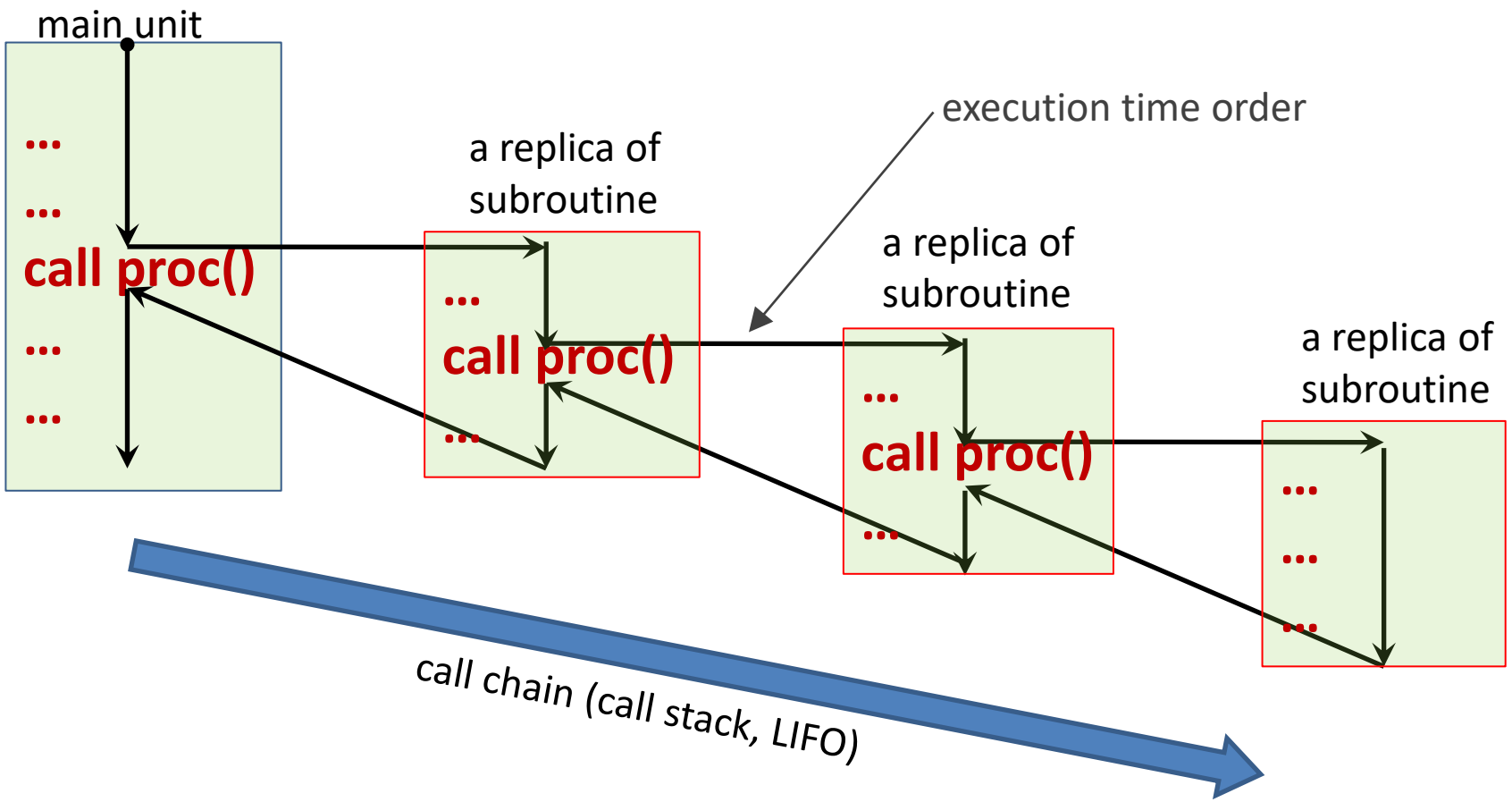
We would like to make mathematical operations with binary numbers, but the pseudocode uses only decimal numbers. Let’s apply a "trick". The operations should use decimal values containing only 0 and 1 digits, similar to binary numbers. Use subroutines.

- Write a pseudocode, which is able to calculate the value of the following formula, where the values are non-negative binary values given by the user.

$$Q = \frac{X^Y}{Z}$$

Recursion

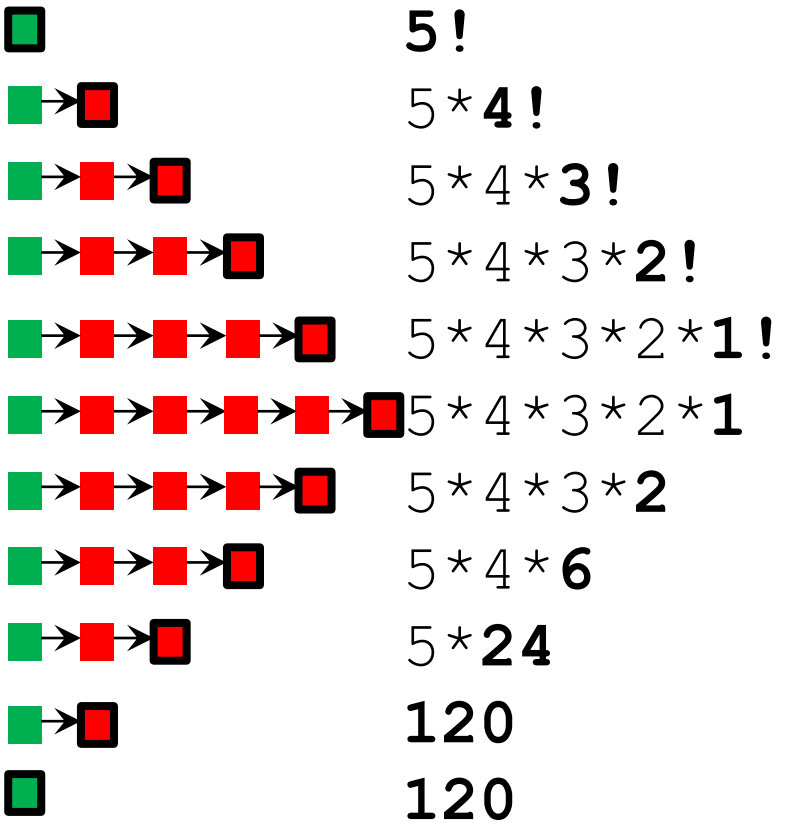
- When a subroutine calls itself



Recursion example

- Factorial (N!):
  - If the number (N) is 1, then the factorial is 1.
  - Otherwise, N! = N * (N-1)!

```
function fact(N)
   if N==1 then
      return 1
   else
      return N*fact(N-1)
   endif
endfunction
output fact(5)
```



Exercise: recursion 1

```
procedure CONV ( N , B )
   if N!=0 then
      call CONV ( (int)(N/B) , B)
      output N%B
   endif
endprocedure

call CONV (16 , 8)
output NEWLINE
call CONV (19 , 2)
```

integer part of the quotient

- Try to execute the algorithm.
- What is the output?
- How long is the call chain?
- Where is the reverse order of digits coded?

Exercise: recursion 2

- What is the output of the following recursive algorithm?

```
procedure something(B,E,P)
   if E>0 then
      call something(B,E-1,P*B)
   else
      output P
   endif
endprocedure
call something(5,3,1)
```

Exercise: recursion 3

- Write an algorithm, which asks for numbers from the user until 0 is given and after then it prints all the values (except the 0) in reverse order. The cardinality of numbers is unknown. Don't use any array in the algorithm.

For example:

input: 1, 4, 65, -9, 2, 9, 2, 0

output: 2, 9, 2, -9, 65, 4, 1

Algorithmic thinking

Expectations:

- Knowledge (*learning*)
- Experiences (*practice*)
- Creativity, intelligence (*capability, talent*)

No guide (algorithm) on how to write algorithms.

The ability of algorithmic thinking cannot be acquired through conning or pure memorizing.

No programmer without algorithmic thinking.

Thought-provoking question

- What will the user enter? (input)
- What must the user see? (output)
- Does anything need to be stored, or data remembered? (variable)
- Has a variable already been assigned a value before we use it? (initial value)
- How should a value be calculated? In what order and which operations are required? (expression evaluation)
- Are there any steps that are only necessary sometimes in certain cases? (branching)

Thought-provoking question

- Do I need to repeat certain steps? (cycle)
- When should I perform something? (condition)
- Is it necessary to have all situations at the same time or is one sufficient? (logical operator)
- Do I need to handle a series of connected values? (array)
- Is there a group of activities that can be required in multiple places of the code? (procedure, function)
- What are the values that influence the operation of a group of activities? (parameter)

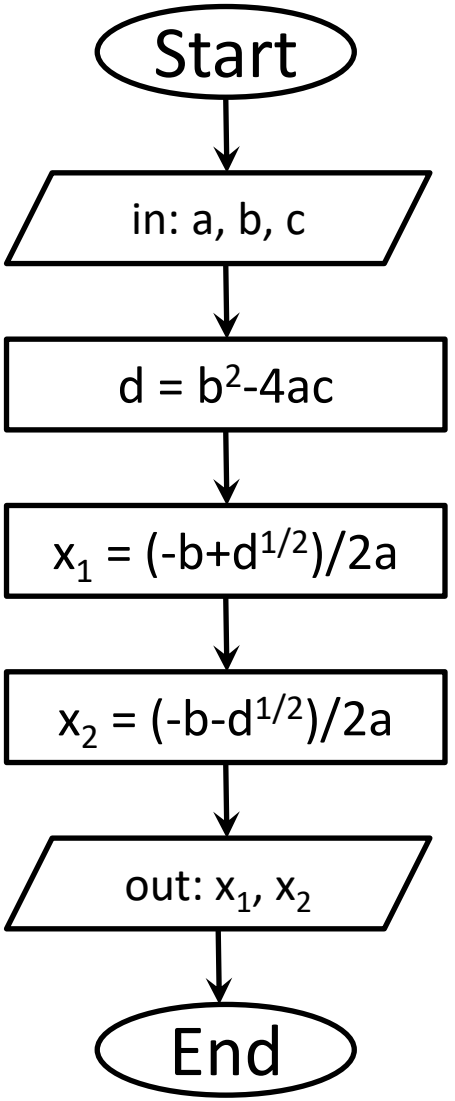
Testing strategy development

Example of testing strategy

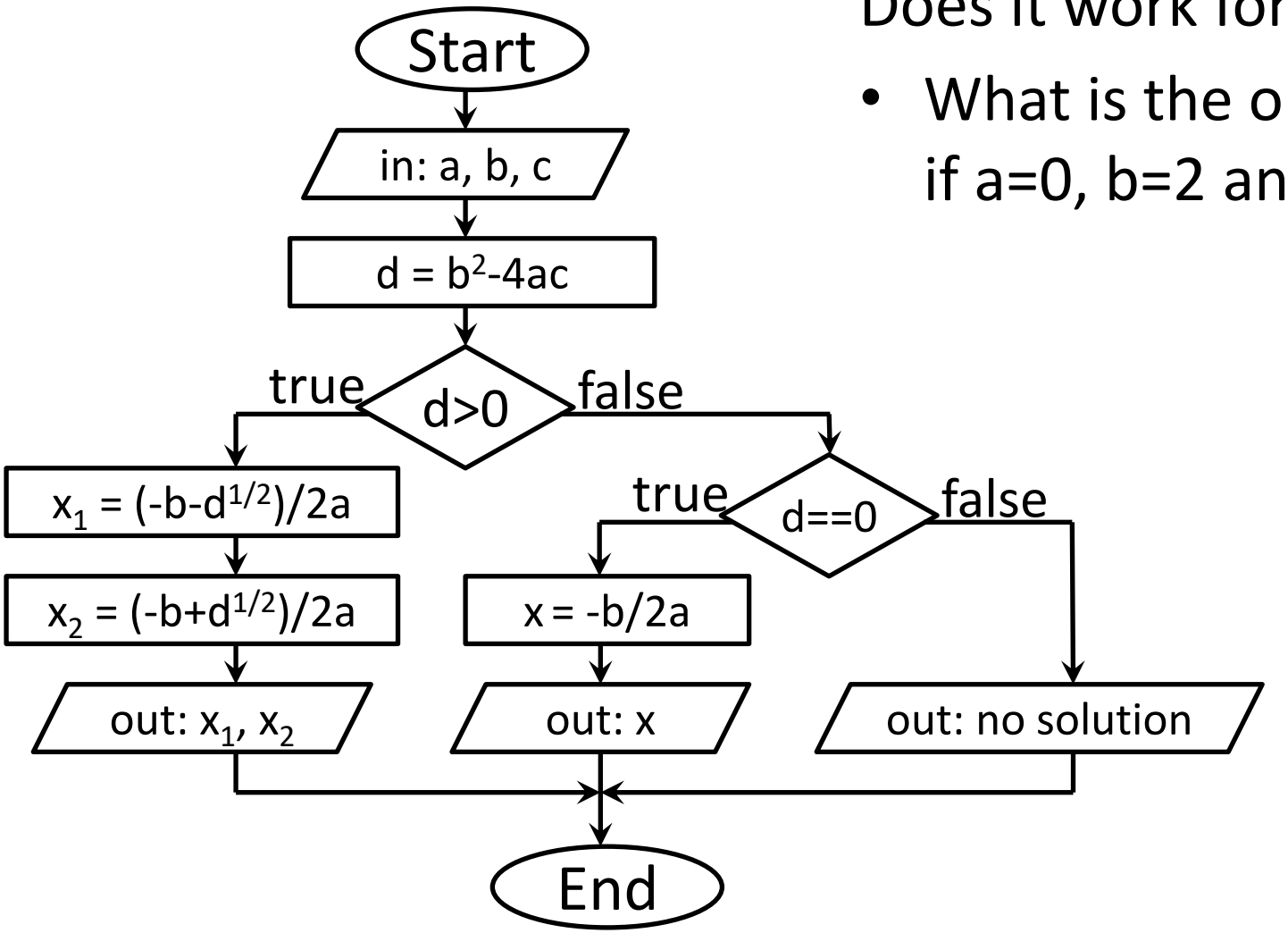
- Solving second degree equation
- General form: $\mathbf{a}x^2 + \mathbf{b}x + \mathbf{c} = 0$
- Input parameters: a, b, c
- Solution: $\mathbf{x_{1,2}} = \frac{-\mathbf{b} \pm \sqrt{\mathbf{b^2 - 4ac}}}{\mathbf{2a}}$

Does it work for all input?

- What is the output if a=1, b=2 and c=1?
- What is the output if a=1, b=2 and c=2?



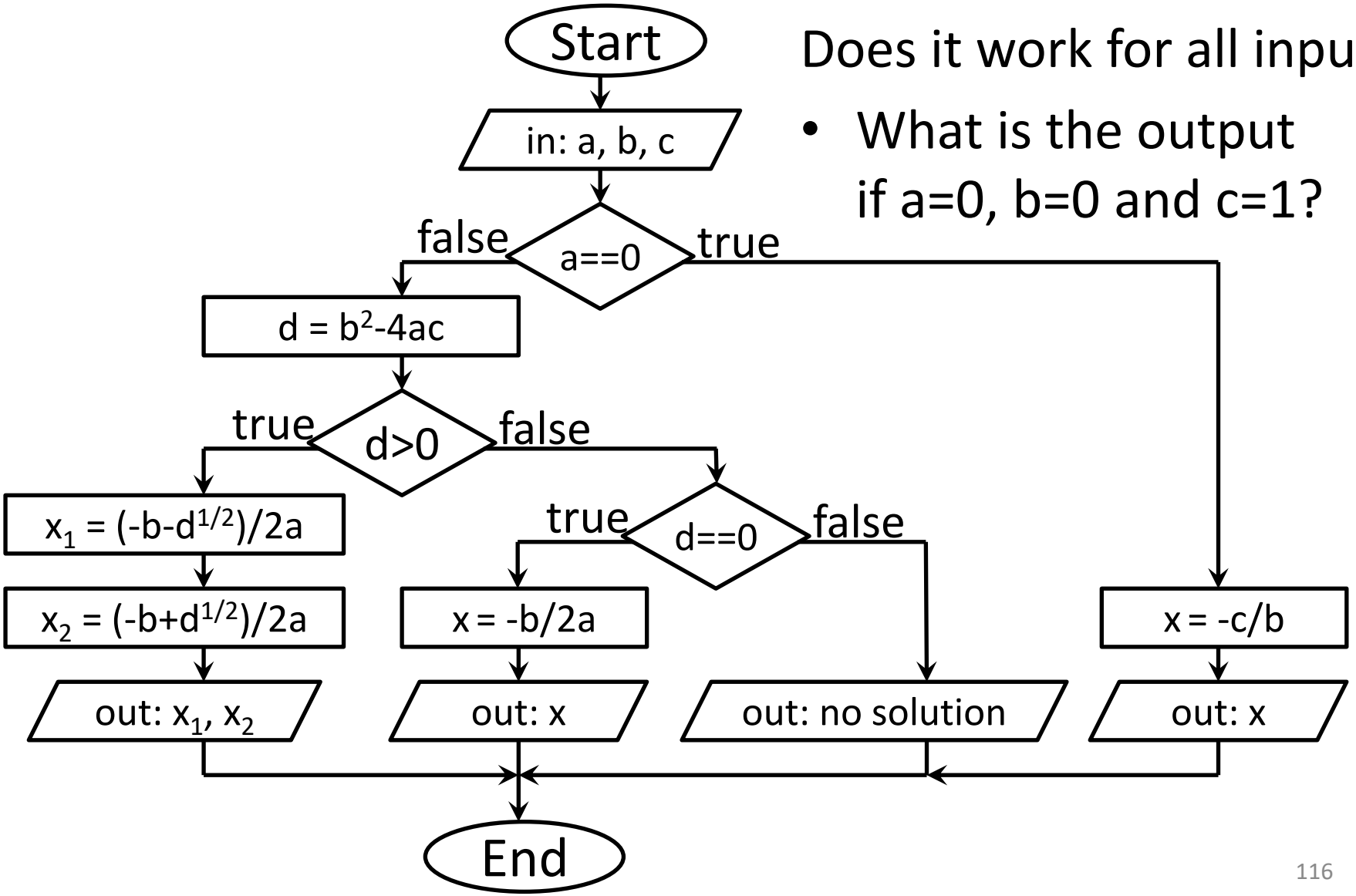
Example of testing strategy



Does it work for all input?

- What is the output if a=0, b=2 and c=6?

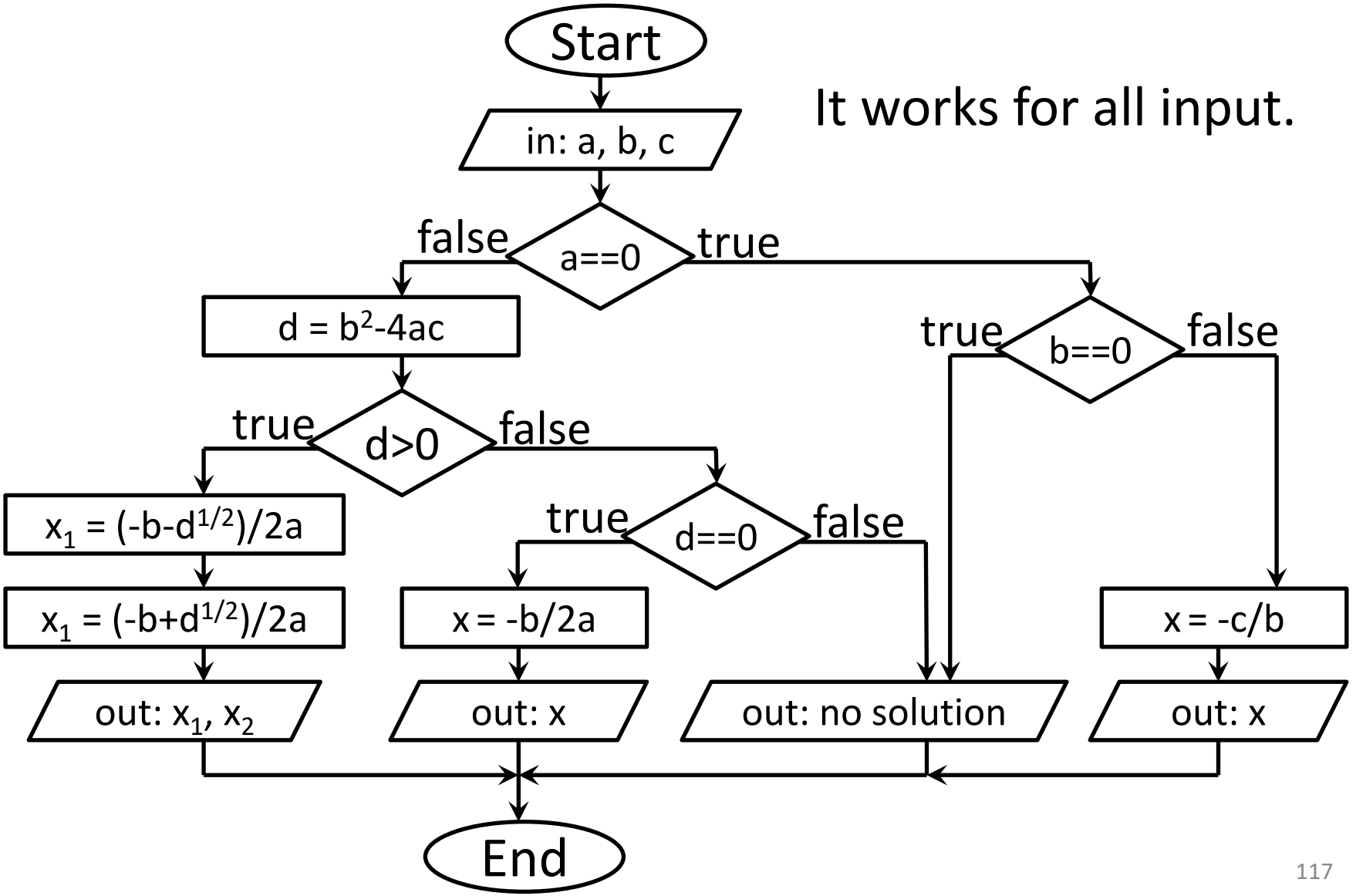
Example of testing strategy



Does it work for all input?

- What is the output if a=0, b=0 and c=1?

Example of testing strategy



It works for all input.

Example of testing strategy

Good solution in pseudocode:

```
input a, b, c
if a==0 then
  if b==0 then
    output "error"
  else
    x=-c/b
    output x
  endif
else
  d=b*b-4*a*c
  if d>0 then
    x1=(-b+sqrt(d))/(2*a)
    x2=(-b-sqrt(d))/(2*a)
    output x1, x2
  else
    if d==0 then
      x=-b/(2*a)
      output x
    else
      output "error"
    endif
  endif
endif
```

It works for all input.

To reach this state we have had to test the algorithm with more different input combinations the so-called **test cases** and then we have had to modify the algorithm.

We have used testing strategy.

The used testing strategy

a	b	c	output	Comment	OK
3	7	2	-1/3 ; -2	general case (not zero, d>0)	✓
0	2	6	-3	**a** is zero (first degree)	✓
2	0	-8	2 ; -2	**b** is zero (x^2=-c/a)	✓
1	2	0	0 ; -2	**c** is zero (x[ax+b]=0)	✓
0	0	1	"error"	more zeros (not equation)	✓
1	2	2	"error"	**d**<0 (no solution)	✓
1	2	1	-1	**d**=0 (only one solution)	✓
-2	-10	-1	-4.9 ; -0.1	negative inputs	✓
2.3	-4.2	0.83	1.6 ; -0.23	not integer values	✓
10^{-5}	10^5	1	-10^{-5} ; 10^{10}	extreme small/large values	✓

test cases

Exercise: testing strategy

Conversion of number notation

- Create a testing strategy for this algorithm.
- Which values of N and B are acceptable?
  (When does the algorithm give expected result?)

```
input N
input B
R=0
P=1
while N!=0 do
   R=R+(N%B)*P
   P=P*10
   N=(int)(N/B)
enddo
output R
```

Exercise: testing strategy

- You have a pseudocode, which has two finite input values (`x` and `y`) and prints out the value of x^y. The algorithm must work for any integer `y` values and all real numbers `x`. If the theoretical/mathematical result is an undefined value or not a finite value, an error message is required. If `y` is a number which is not an integer, an error message is also required.

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) = \boldsymbol{x}^{\boldsymbol{y}}$$
$$x \in \mathbb{R} \setminus \{\infty\}$$
$$y \in \mathbb{Z} \setminus \{\infty\}$$
$$f(x, y) \in \mathbb{R} \setminus \{\infty\}$$

Program coding

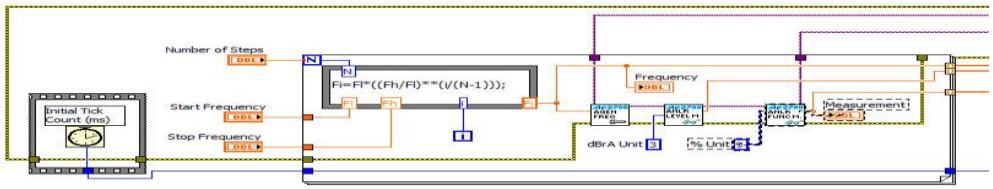
Creating source code
in real programming language

Programming levels

High-level programming languages

Low-level programming languages

…



4GL SELECT name FROM people WHERE age=20

3GL if (x==0) printf("zero\n");

2GL mov eax, DWORD PTR [rbp-4]

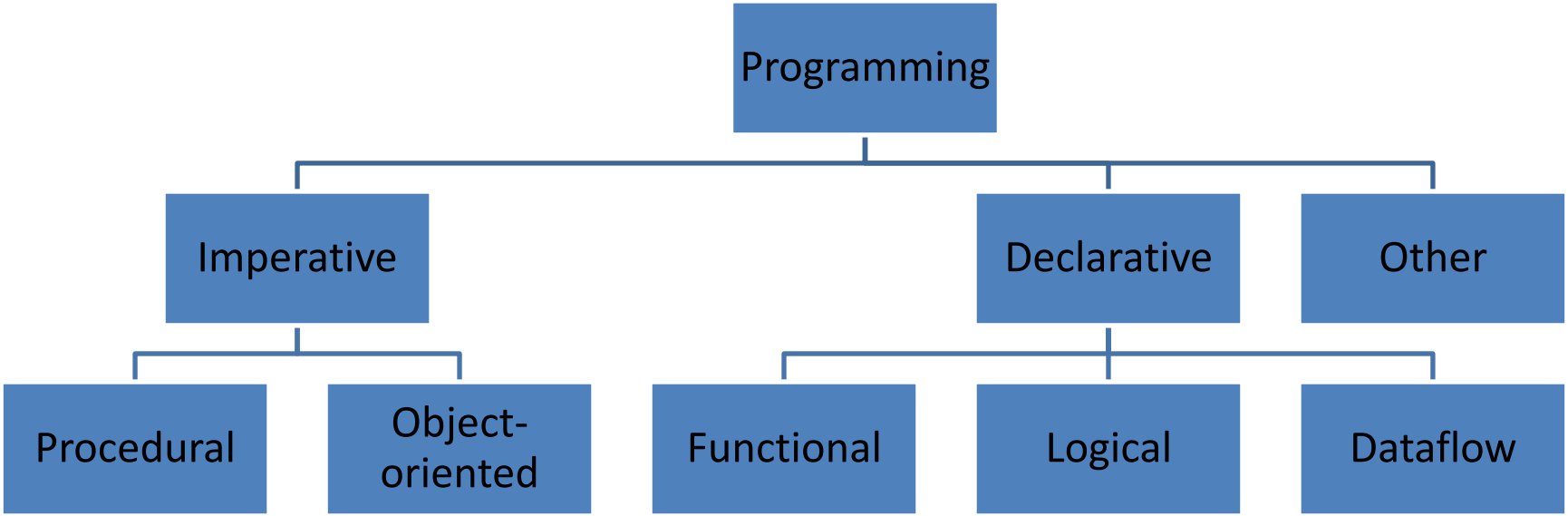
1GL 10001011 01000101 11111100

time

assembly

machine code

Language paradigms



- C
- Pascal
- Fortran

- Java
- C++
- Smalltalk

- R
- LISP
- Haskell

- Prolog

- Labview
- Verilog

Syntax and semantics

Syntax: Formal rules of the program text.

Semantics: Does it describe the desired algorithm?

Example (sign of a value):

```
function sign (a)
    b=a
    if a>0 then
       b=-1*a
    enidf
    return a/b
endfunction
```

Semantic error (`a<0`)

Syntax error (`endif`)

Runtime error (if `/0`)

Syntax of programing languages

Fortran:

```
      REAL FUNCTION FACT(N)
      FACT=1
      IF (N .EQ. 0 .OR. N .EQ. 1) RETURN
      DO 20 K=2,N
20    FACT=FACT*N
      RETURN
      END
```

Python:

```
def Fact(N):
    f=1
    for i in range(1,N+1):
        f*=i
    return f
```

Pascal:

```
FUNCTION FACT(N:INTEGER):REAL;
BEGIN
  IF N=0 THEN FACT:=1
  ELSE FACT:=FACT(N-1)*N;
END;
```

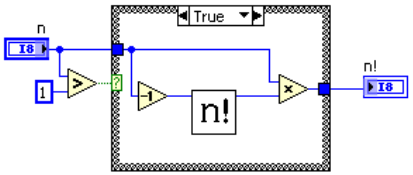
C:

```
int Fact(unsigned N){
  return N<=1?1:N*Fact(N-1);
  }
```

APL:

```
FACT←{×/⍳⍵}
```

LabView:



Exercise: Syntax and semantics

- Find syntactic and semantic errors of the following algorithm written in pseudocode to determine the not negative integer (E) power of the base (B).

```
input B
R=0
wihle E<=0
  R=R*B
  E-1=E
endo
output R
```

Interpreter and Compiler

- Processors understand only machine codes
  - High-level source codes need some conversion
- Language implementations use different techniques
  - Compiler
    E.g.: Pascal, C, C++, Labview
  - Interpreter
    E.g.: PHP, JavaScript, Python
  - Combined (bytecode to virtual machine)
    E.g.: Java, C#

Compiler

- A software that creates a so-called **object-code** from the **source code**
- Compiler makes lexical-, syntactic- and semantic analysis, code generation
  - Source codes have to be syntactically correct
- A so-called **linker** creates executable from object codes, and the **loader** load it to RAM to run
- Compilation once, execution later several times
  - Compilation and execution is separate
- Execution is fast

Interpreter

- Direct execution
  - Analysis and generation at run time
- No object code
- Interpretation of instructions one by one
  - Single instruction as input
- Syntactically incorrect code can be executed
  - Errors may be hidden
- Interpreter is needed for any execution
  - Interpretation and execution belong together
- Execution is often slow

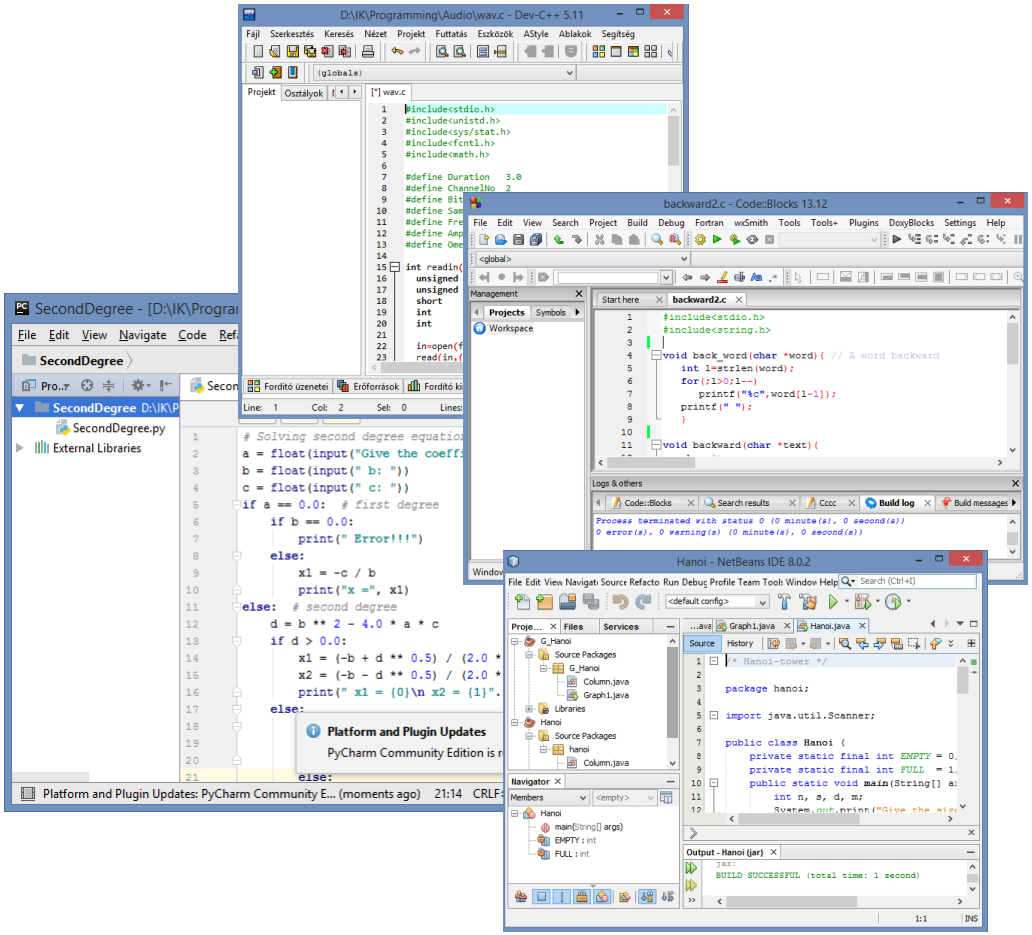
Integrated Development Environment

- A (graphical) program to make the software development easy and quick, provides tools/help to the programmer
- **IDE** contains:
  - Language-sensitive editor
  - Compiler/interpreter, linker, loader
  - Debugger
  - Version control tool
  - Project management
  - Simulator

Integrated Development Environment

Most often used IDEs:

- Code::Blocks
- Dev-C++
- NetBeans
- Eclipse
- PyCharm
- MS Visual Studio
- Jbuilder
- MPLAB



Data representation, Datatypes

- Every data is stored in the memory in binary
- Different datatypes are used
  - with different representation
  - with different data-domain
  - with different operations
- Most often applied data types:
  - integer (5)
    00000000000000000000000000000101
  - real (5.)
    01000000001010000000000000000000000000000000000000000000000000000
  - string ("5")
    0011010100000000
  - char ('5')
    00110101

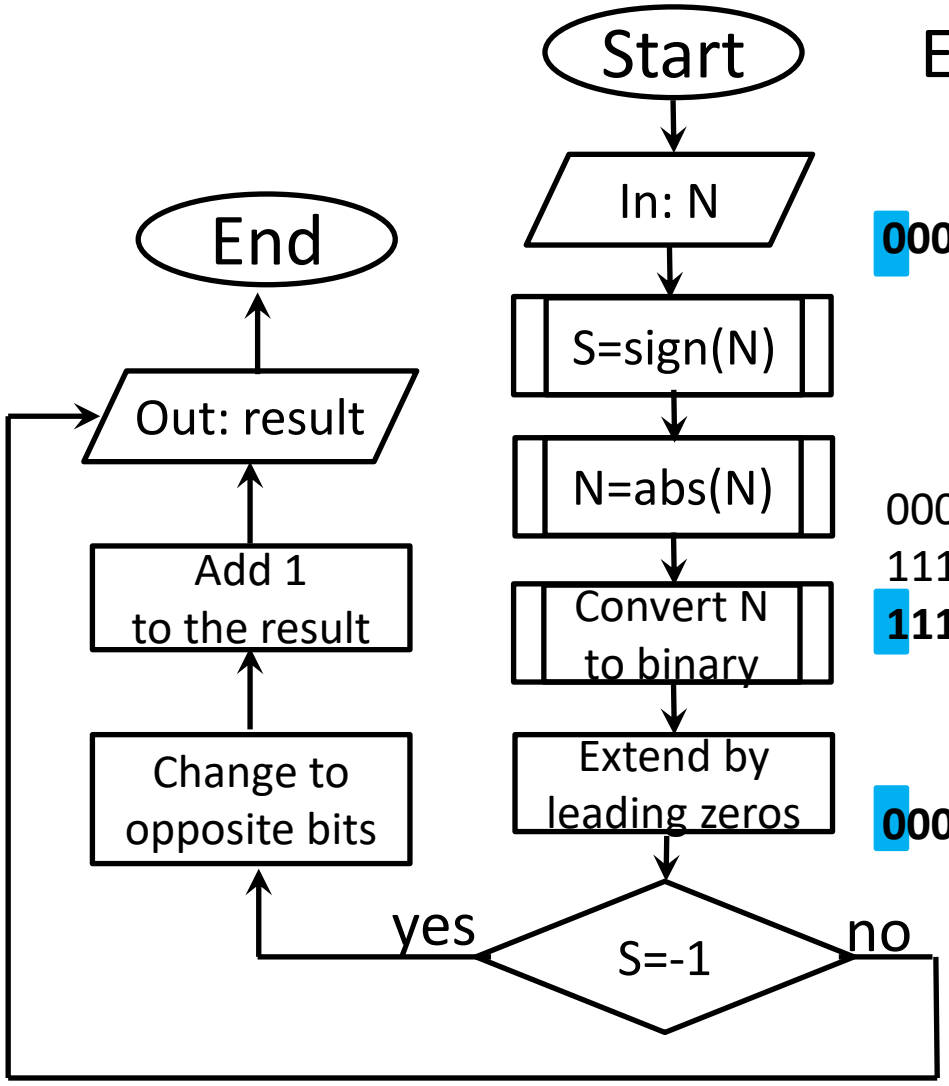
Fixed-point representation

How the computer stores (signed) **integer** numbers?

Steps:

- Remember the sign of the given value and convert the absolute value of the integer into binary
- Add leading zeros (if needed) to reach given number of digits (bits)
- If the sign is -1 (so if the value was negative) then
  - Change every bit to the opposite
  - Add 1 to the result in binary
- The fixed-point representation of the integer is ready

Fixed-point representation



Examples

+195
11000011
00000000 00000000 00000000 11000011

-195
+195
11000011
00000000 00000000 00000000 11000011
11111111 11111111 11111111 00111100
11111111 11111111 11111111 00111101

0
0
00000000 00000000 00000000 00000000

MSB = sign bit

Fixed-point representation

	Representation length	Minimum value	Maximum value
Unsigned	1 byte	0	255
	2 byte	0	65 535
	4 byte	0	4 294 967 295
Signed	1 byte	-128	127
	2 byte	-32 768	32 767
	4 byte	-2 147 483 648	2 147 483 647

Exercise: data representation

- Give the approximate population of the Earth with 32-bit fixed-point representation.
- Write the value of -1 with 32-bit fixed-point representation.
- Which 4 bytes long fixed-point representation bit sequence means: 15908?
- Which 4 bytes long fixed-point representation bit sequence means: -666?
- What is the meaning of the following 4 bytes long fixed-point representation bit sequence?

```
10000000 00000000 00000010 01001001
```

Units/elements of the source code

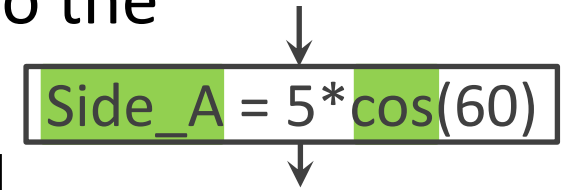
- Character set
- Lexical units
- Syntactic units
- Instructions
- Program units
- Compiling units
- Program

We use different characters, symbols, special keywords, expressions, and rules in each language.

Keywords, identifier, comments

Examples in C language.

- Keyword
  Sequence of characters with special meaning
  E.g.: `if, else, while, do, for, return`
- Identifier
  Sequence of characters to give name to the programmer's own tools/objects
  E.g.: `i, Count, var2, abs_val`



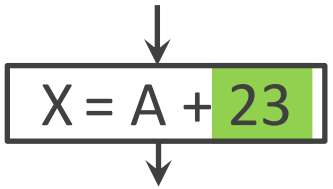
- Comment
  Text in the code not for the compiler/interpreter but for the programmer (reader human) as remark

Constants

- Constants (literals) means fix value in the source code that cannot be altered by the program at runtime
- It has type and value
  - The value is defined by itself
  - The type is defined by the form
- Special: Named constant is a fix value with identifier
- Examples

```
-12.34, 5, .5, 5., 0x1F, 'F', "F", "apple"
```

```
while x>100 do
```



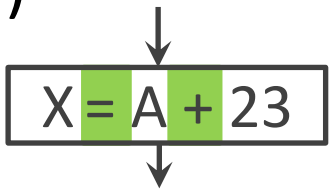
Variables

- A memory location with identifier to store a value
- Most important tool in procedural languages
- Its components
  - Name (identifier)
  - Value (bit series in the RAM)
  - Attributes (type)
  - Address (RAM location)
- Example (in C language)

```
int A = 10;
float Jump1 = 11.5;
```

Operators

- Represents simple operations on data
- Can be unary, binary or ternary
- General groups of operators
  - Arithmetic (E.g.: `+`, `-`, `*`, `/`, `%`)
  - Comparison (E.g.: `>`, `<`, `==`, `>=`, `<=`, `!=`)
  - Logical (E.g.: `&&`, `||`, `!`)
  - Bitwise (E.g.: `&`, `|`, `^`, `~`, `<<`, `>>`)
  - Assignment (E.g.: `=`, `+=`, `*=`)
  - Other (E.g.: `*`, `&`, `? :`, `.`, `->`)

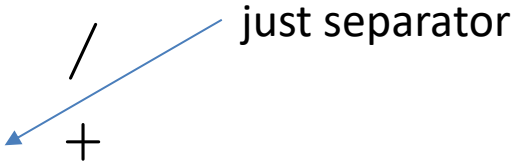


Expressions

- Operators & Operands & Parentheses
- Operand can be: constant, variable, function call
- An expression has type and value (evaluation)
- Form can be
  - Infix (preference/strength is necessary)
    E.g.: `4 + 3 * 2`
  - Prefix
    E.g.: `+ * 3 2 4`
  - Postfix
    E.g.: `4 3 2 * +`

Exercise: expression

- What is the value of this infix expression?
  `9+2*6/3>8-7`
- What is the value of this expression (in C language)?
  `2>3&&3*5-6/2>=11%2`
- What is the value of this prefix expression?
  `*; +; 1; 2; -; 9; 6`
  `+; 1; -; *; 2; 13; /; 25; 5`
- What is the value of this postfix expression? Convert them into infix form.
  `30; 2; 15; 4; 6; +; -; *; /`
  `1; 2; 13; *; 25; 5; /; -; +`



Instructions

Unit of programs, that can be grouped as

- Declaration
- Assignment
- Conditional statement
  - 2 branch
  - More branch
- Iteration
  - Conditional loop
  - Counted loop
- Other

not executable

executable

Declaration, Assignment

Declaration

- Associate identifier and type
- RAM allocation, (sometimes) initialization of variable
- `int i = 0;`
- `float Weight;`

Assignment

- Giving value to a variable
- `i = 6;`
- `Weight = 80.3 * i;`

Conditional statement

Choosing from 2 execution branch

- Two separate instruction block
- Skip or execute an instruction block
- ```
  if(N<0.0)   S=1;
  else        S=0;
  ```

Selecting from several execution branch

- ```
  switch (i){
     case 1:  X=1;  break;
     case 2:  X=10; break;
     default: X=100;
  }
  ```

Iteration

Repetition of instructions, activities several times

From operational point of view limiting cases

- Empty loop (the body/core never executed)
- Infinite loop (never stops, semantic error)

Types of iterations

- Conditional loop
  - Pre-condition
  - Post-condition
- Counted loop
- Other (Infinite, Combined)

Pre-conditional loop

The **head** contains a **condition**

Semantics

1. Evaluation of condition
2. If it is true, **body** is executed and evaluate again (go 1.)
   Else loop ends, go to next instruction behind the loop

It can be empty loop
if condition is false initially

```
while (x<2) {
    i=i+1;
    x=x-2;
}
```

Body must change the condition →

Post-conditional loop

The **end** contains the **condition**

Semantics

1. Execute the **body** once
2. Evaluation of condition
3. If it is true (false), execute again the body (go Step 1.)
   Else loop ends, go to next instruction behind the loop

It cannot be empty loop
body is executed at least once

```
do{
   i=i+1;
   x=x-2;
   }
while (x<2);
```

Pseudocode to Python

Pseudocode:

```
input a
if a>0 then
  b=a
else
  b=-1*a
endif
while b!=0 do
  b=b-1
enddo
output b
```

Python:

```
a = int(input())
if a > 0:
    b = a
else:
    b = -1*a

while b != 0:
    b = b-1

print(b)
```

Python programming language

```
# Solving second degree equations
a = float(input("Give the coefficients\n a: "))
b = float(input(" b: "))
c = float(input(" c: "))
if a == 0.0:  # first degree
    if b == 0.0:
        print(" Error!!!")
    else:
        x1 = -c / b
        print("x =", x1)
else:  # second degree
    d = b ** 2 - 4.0 * a * c
    if d > 0.0:
        x1 = (-b + d ** 0.5) / (2.0 * a)
        x2 = (-b - d ** 0.5) / (2.0 * a)
        print(" x1 = {0}\n x2 = {1}".format(x1, x2))
    else:
        if d == 0.0:
            x1 = -b / (2.0 * a)
            print(" x =", x1)
        else:
            print(" Error!!!")
```

Exercise: Python

Find the occurrence of the following concepts in this Python code.

- keyword
- comment
- identifier
- constant
- variable
- operator
- expression
- statement

```
# Some calculation
Sum=0
for i in range(N):
    Sum+=i
if(Sum==0):
    print("Total"+Sum)
else:
    z=10%2+N/N+cos(90)
#return z
```

Further readings

- Simon Harris, James Ross: *Beginning algorithms*, (Wiley Publishing, 2006)
- Narasimha Karumanchi: *Data Structures and Algorithmic Thinking with Python*, (CareerMonk, 2017)
- Peter Wentworth, Jeffrey Elkner, Allen B. Downey and Chris Meyers: *How to Think Like a Computer Scientist: Learning with Python 3,* (online, 2012)
- Metrowerks CodeWarrior: *Principles of Programing* (online, 1995)

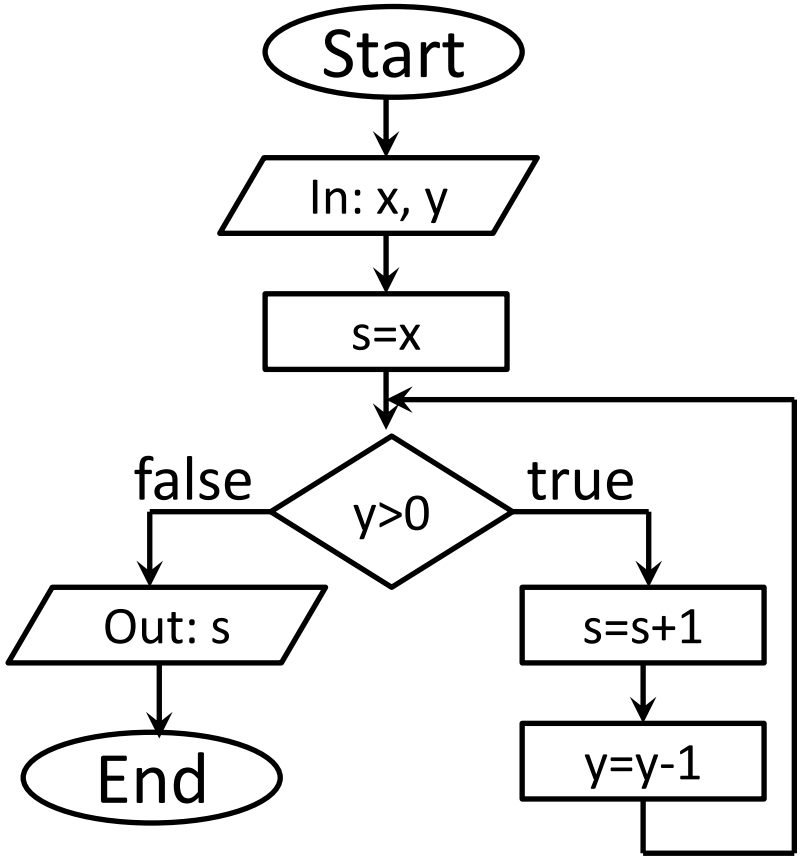
Solutions

Back to the Exercise

Solution: flowchart

- Debugging:

x	y	s
5	4	?
5	4	5
5	3	6
5	2	7
5	1	8
5	0	**9**

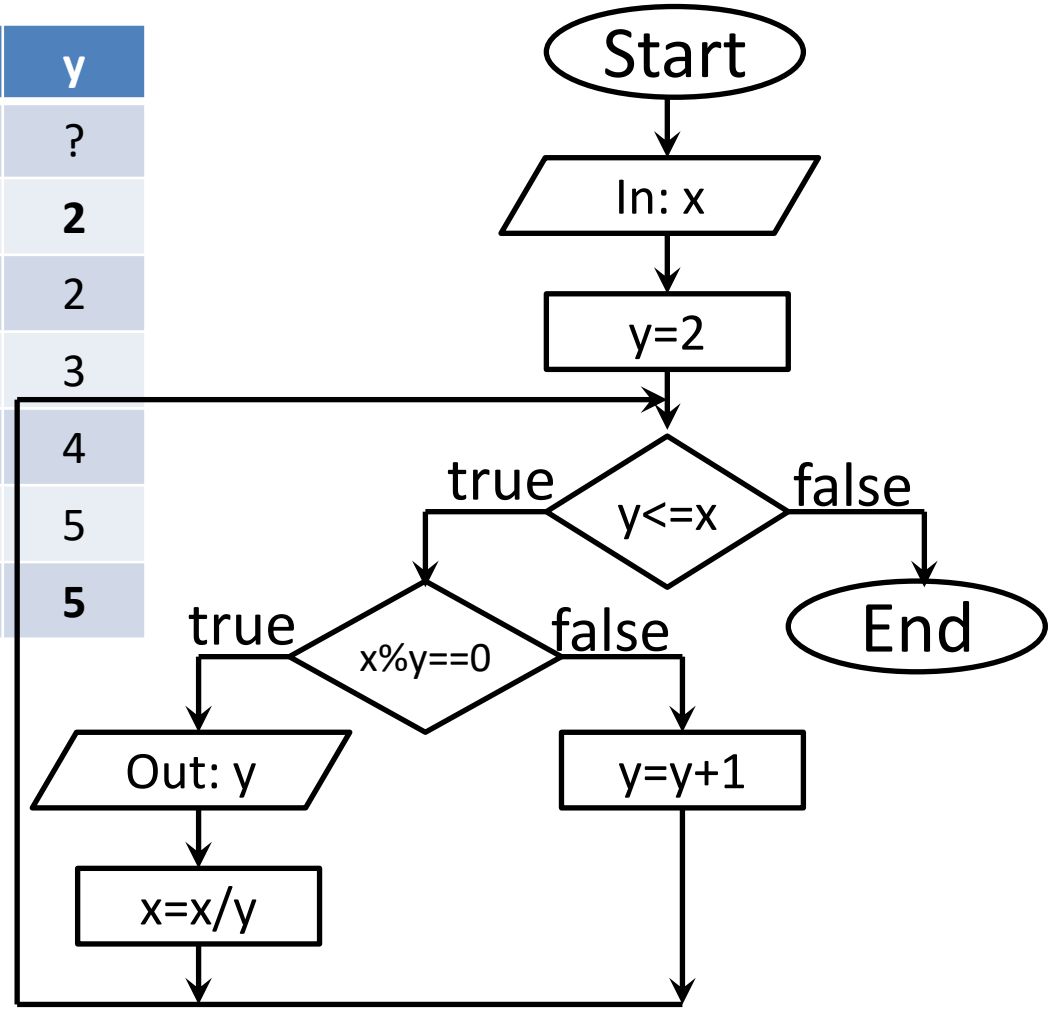


- Output: 9
- 5 expression evaluations
- Addition: s=x+y
- Modifications (3 alternatives):
  s=s+1 → s=s+x s=x → s=0
  s=s+1 → s=s+x y>0 → y>1
  s=s+1 → s=s+x Out: s → Out: s-x

Solution: flowchart

- Debugging:

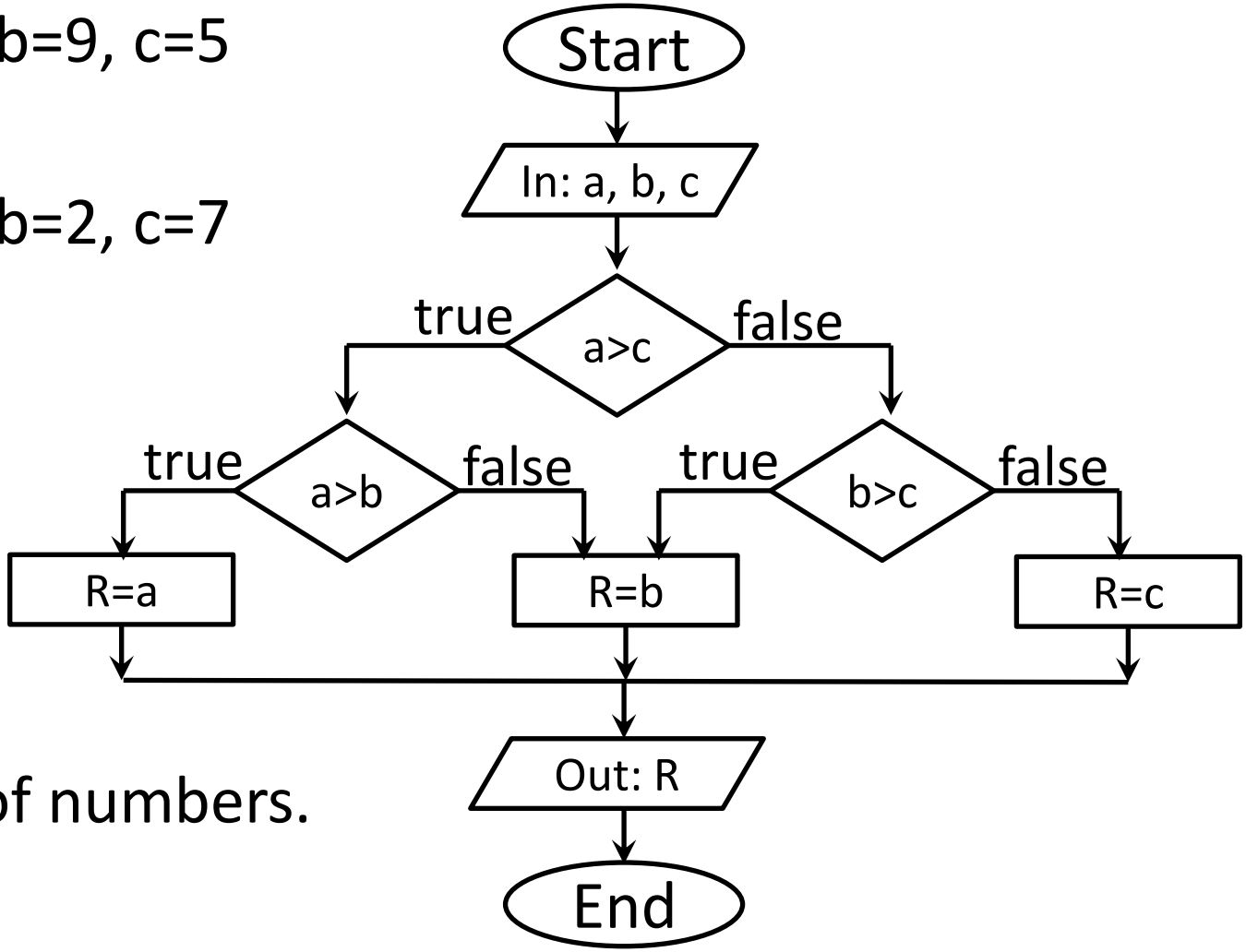
x	y
10	?
10	**2**
5	2
5	3
5	4
5	5
1	**5**



- Output in case of 60: 2 2 3 5
- Prime factorization
- No output
- Number of repetitions: 5

Solution: flowchart

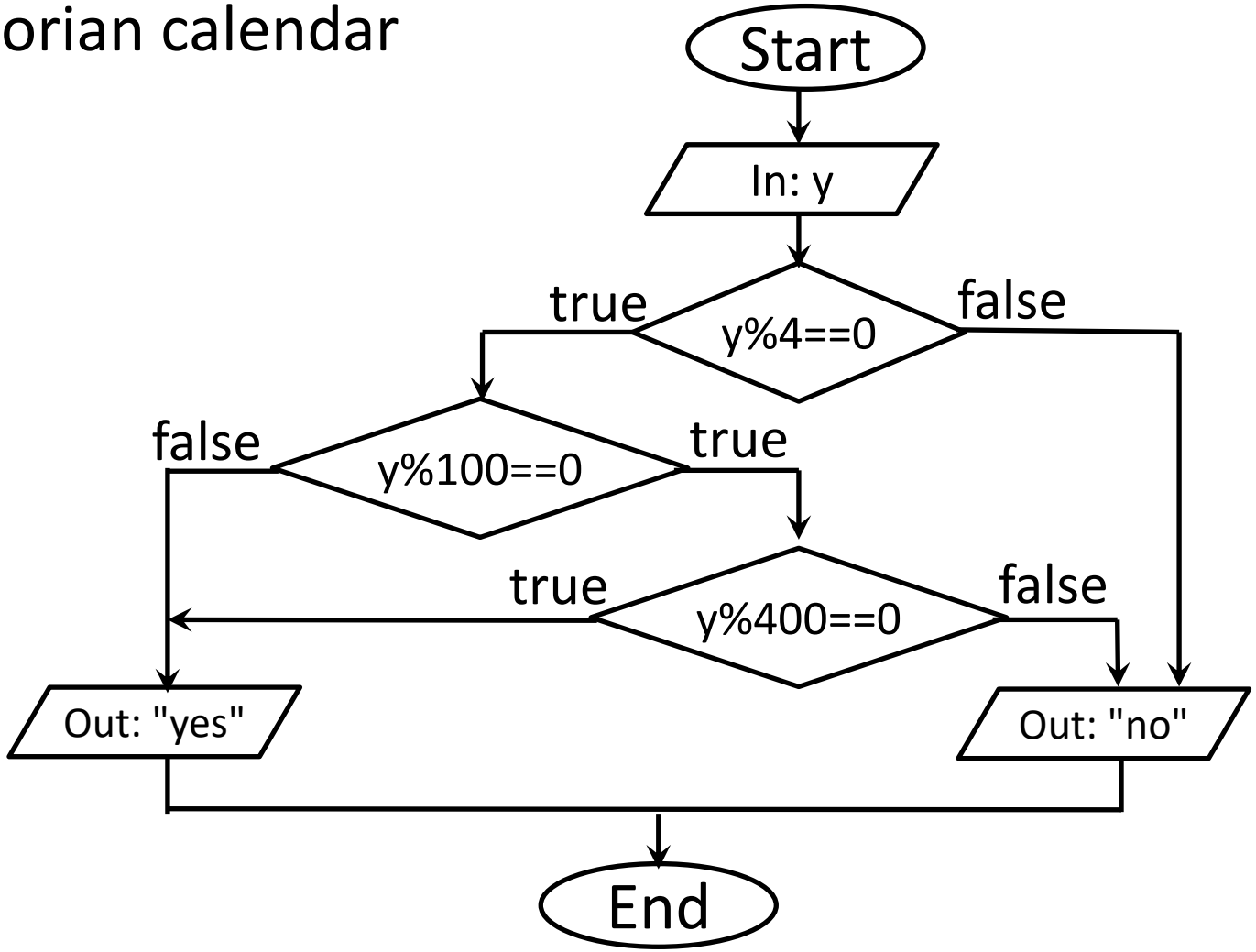
- Input: a=3, b=9, c=5
  Output: 9
- Input: a=5, b=2, c=7
  Output: 7



- Maximum of numbers.

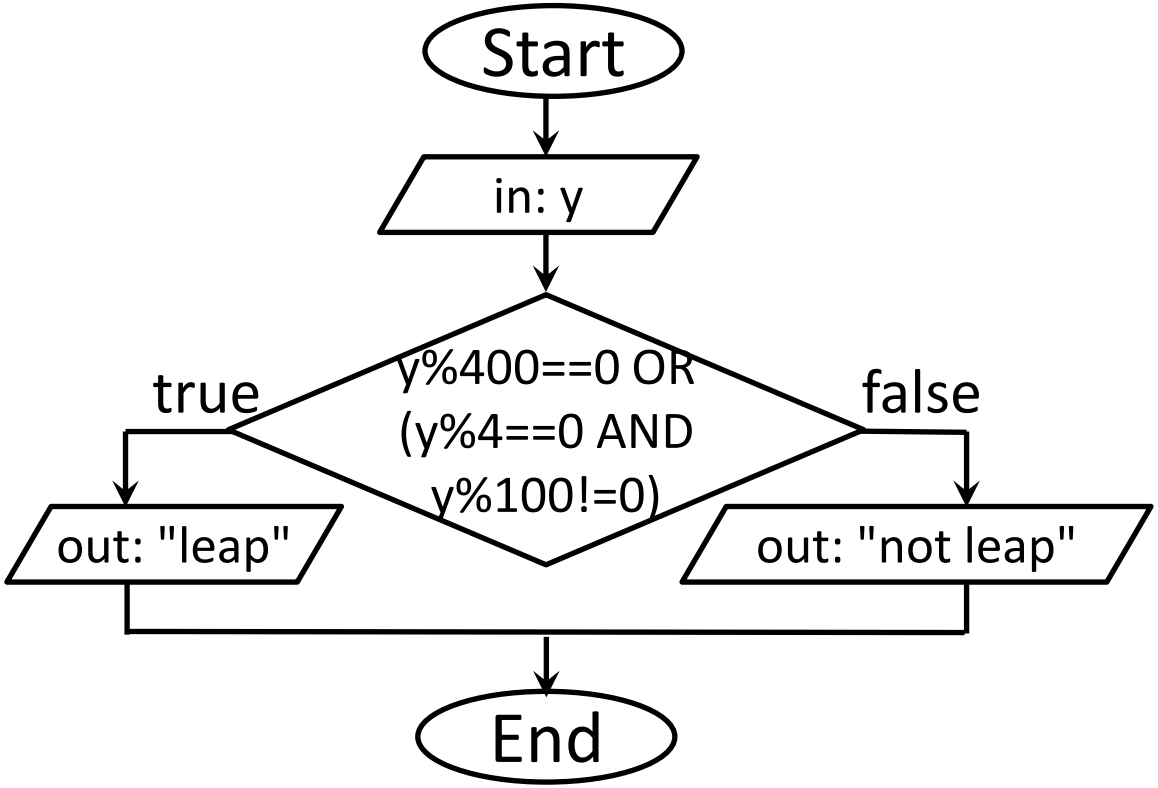
Solution: leap year 1

- Gregorian calendar



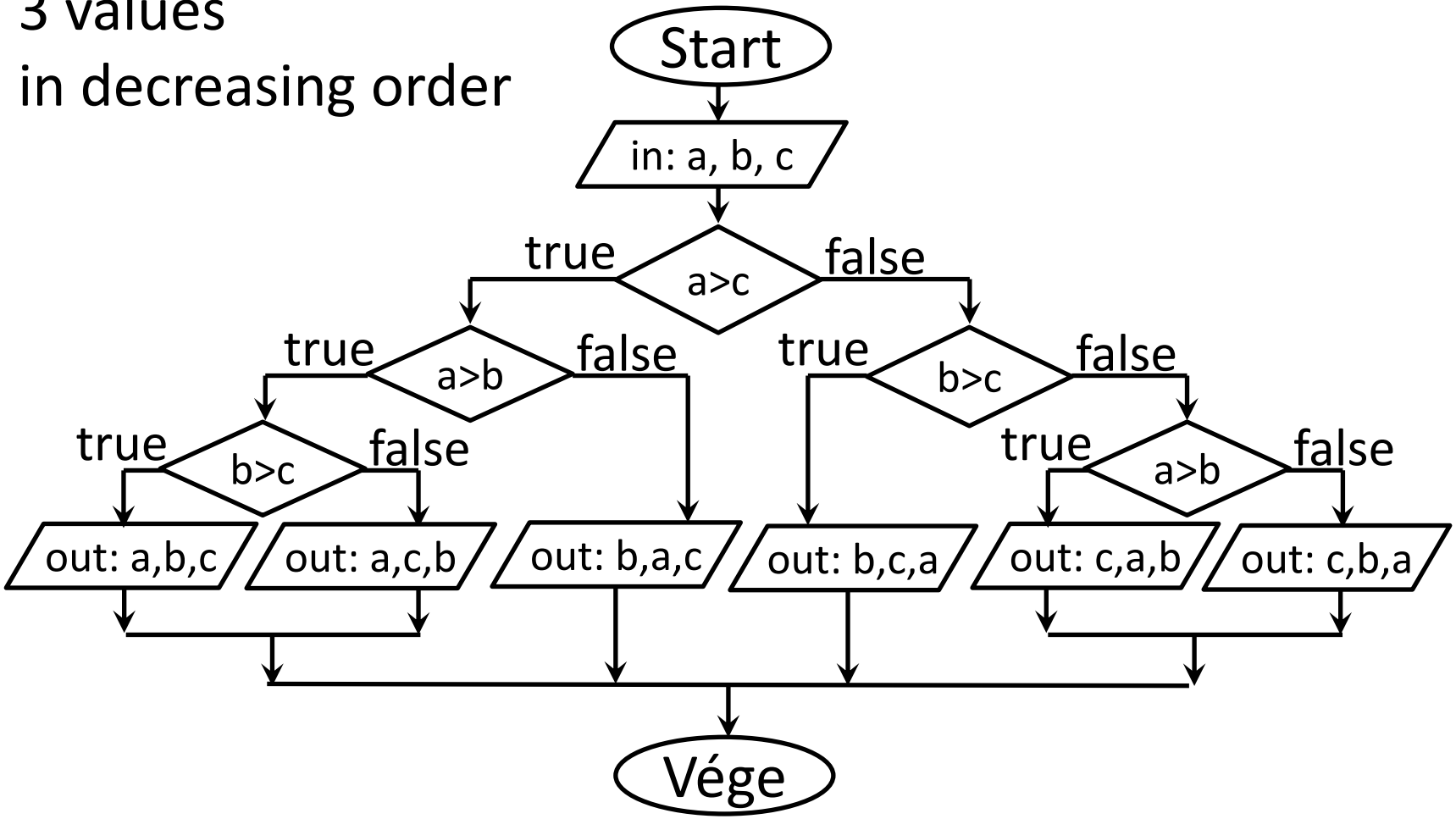
Solution: leap year 2

- Gregorian calendar



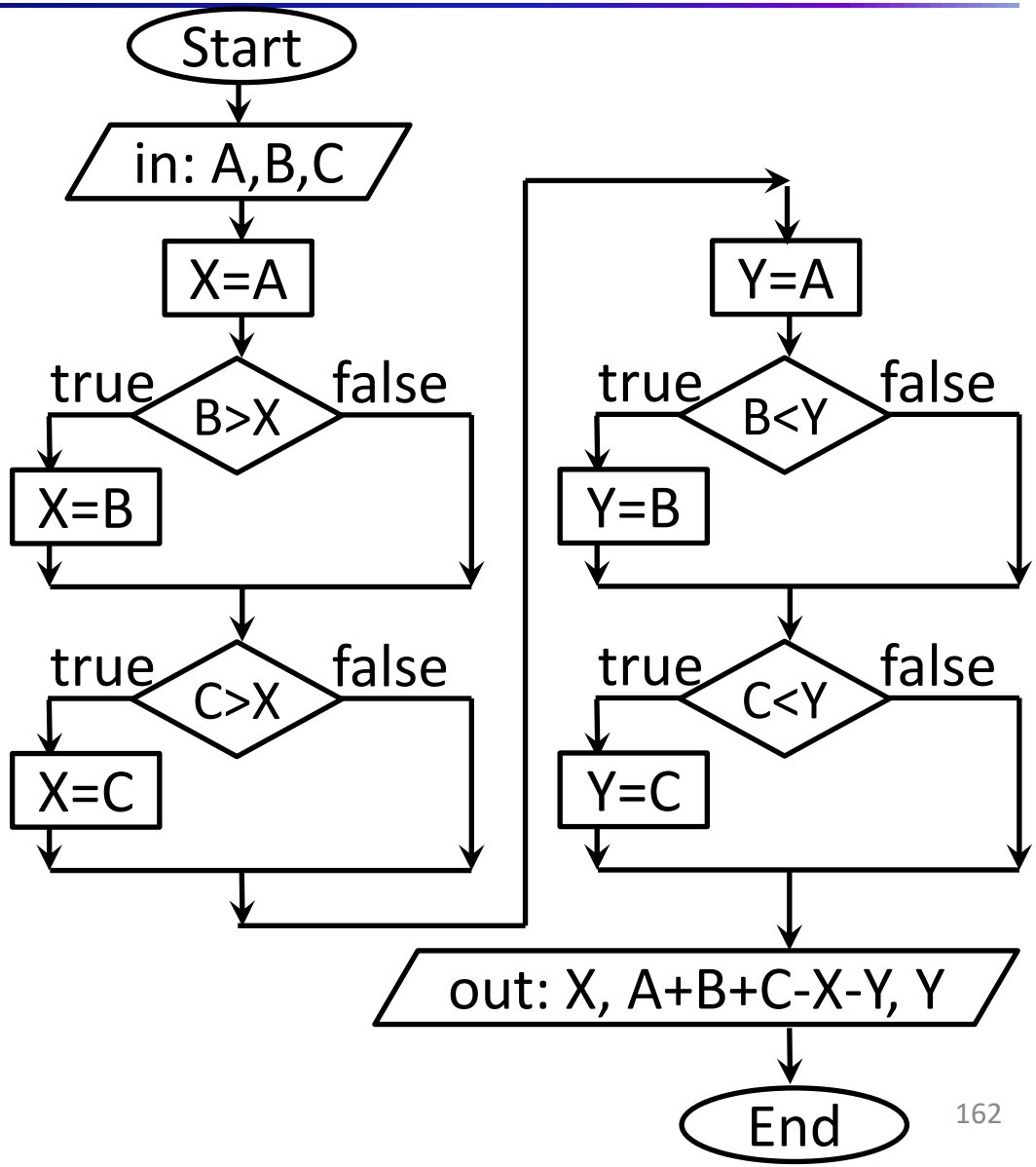
Solution: three values 1

- 3 values
  in decreasing order

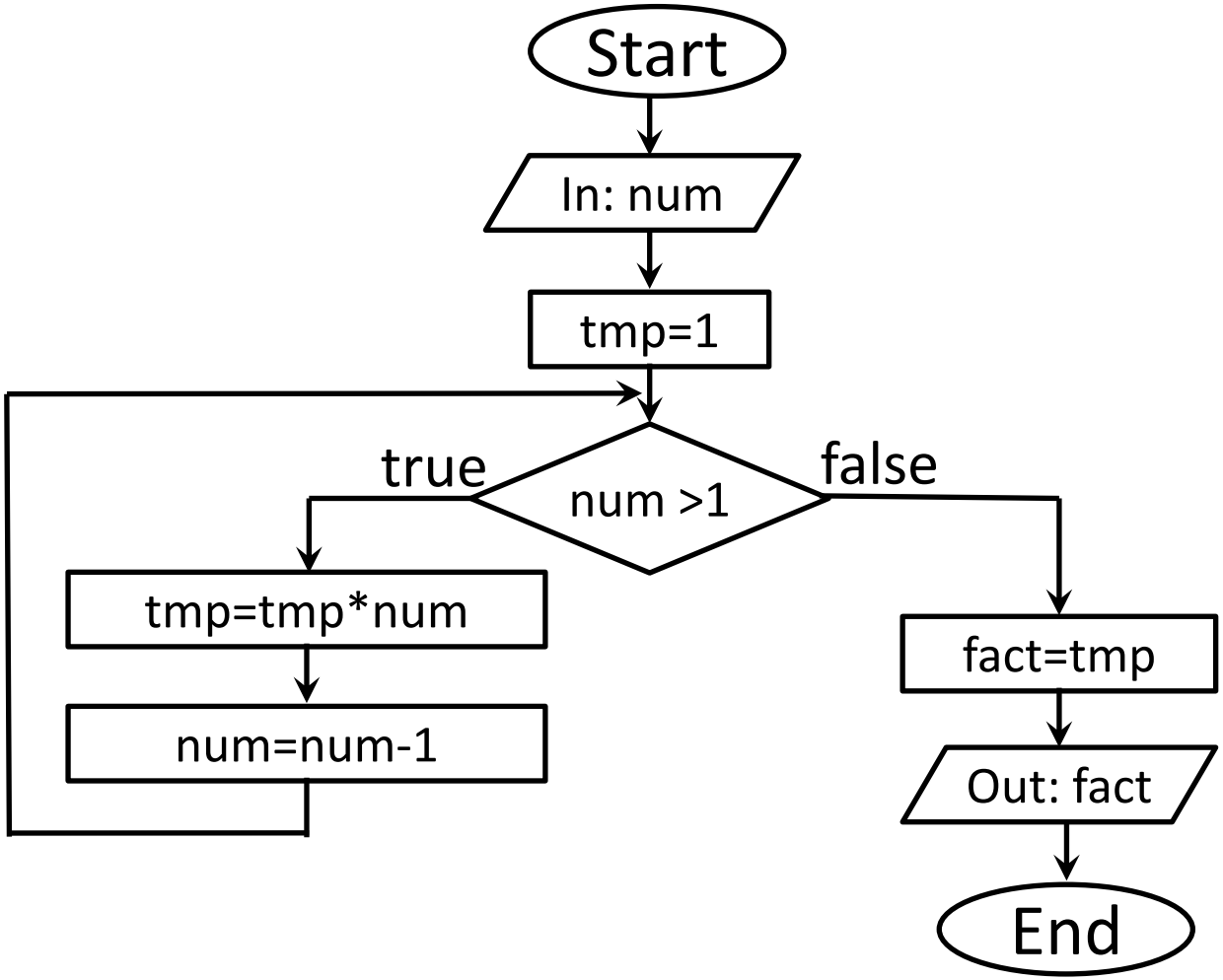


Solution: three values 2

- 3 values
  in decreasing order

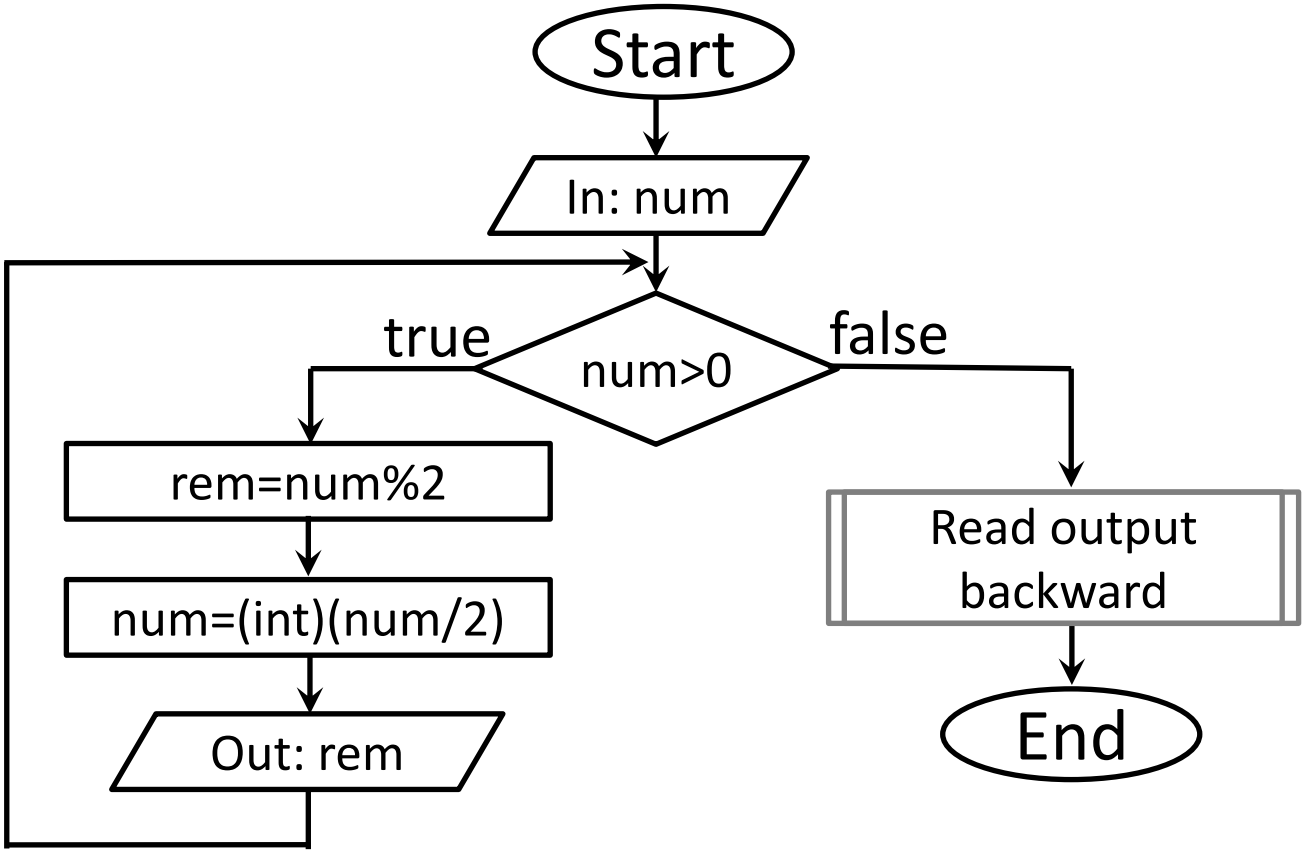


Solution: factorial



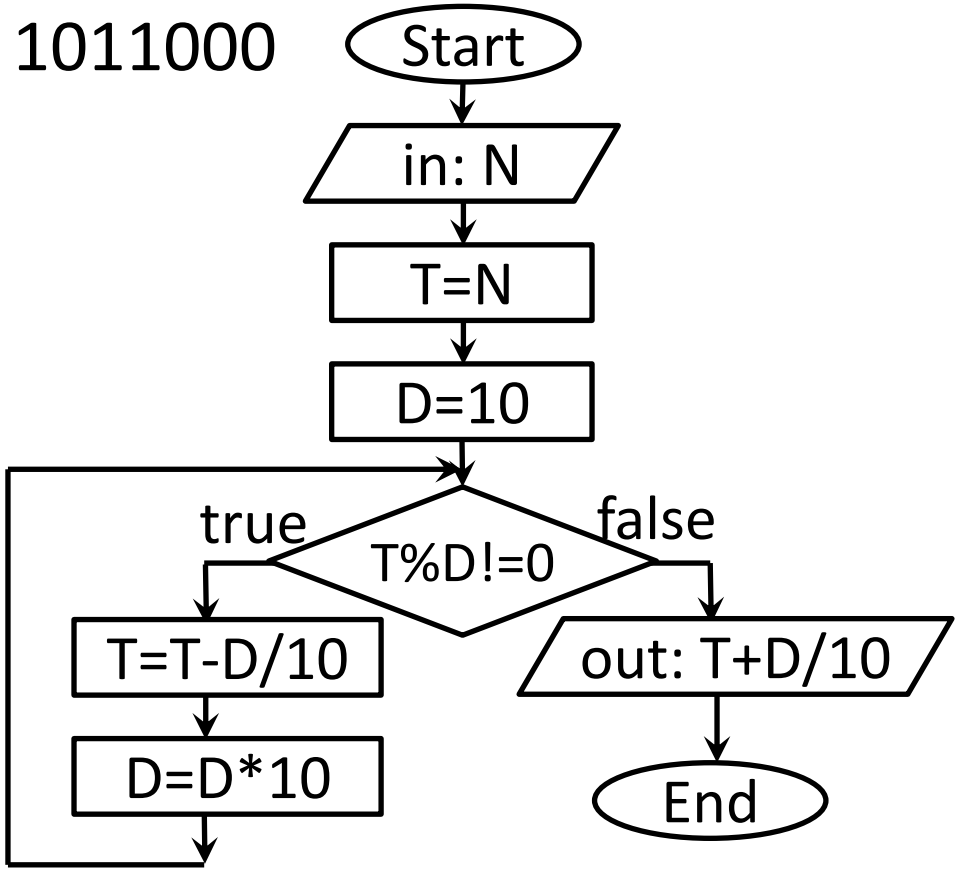
Solution: binary numbers

- Conversion from decimal to binary (10 → 2)
  E.g.: $21_{10} = 10101_2$



Solution: flowchart

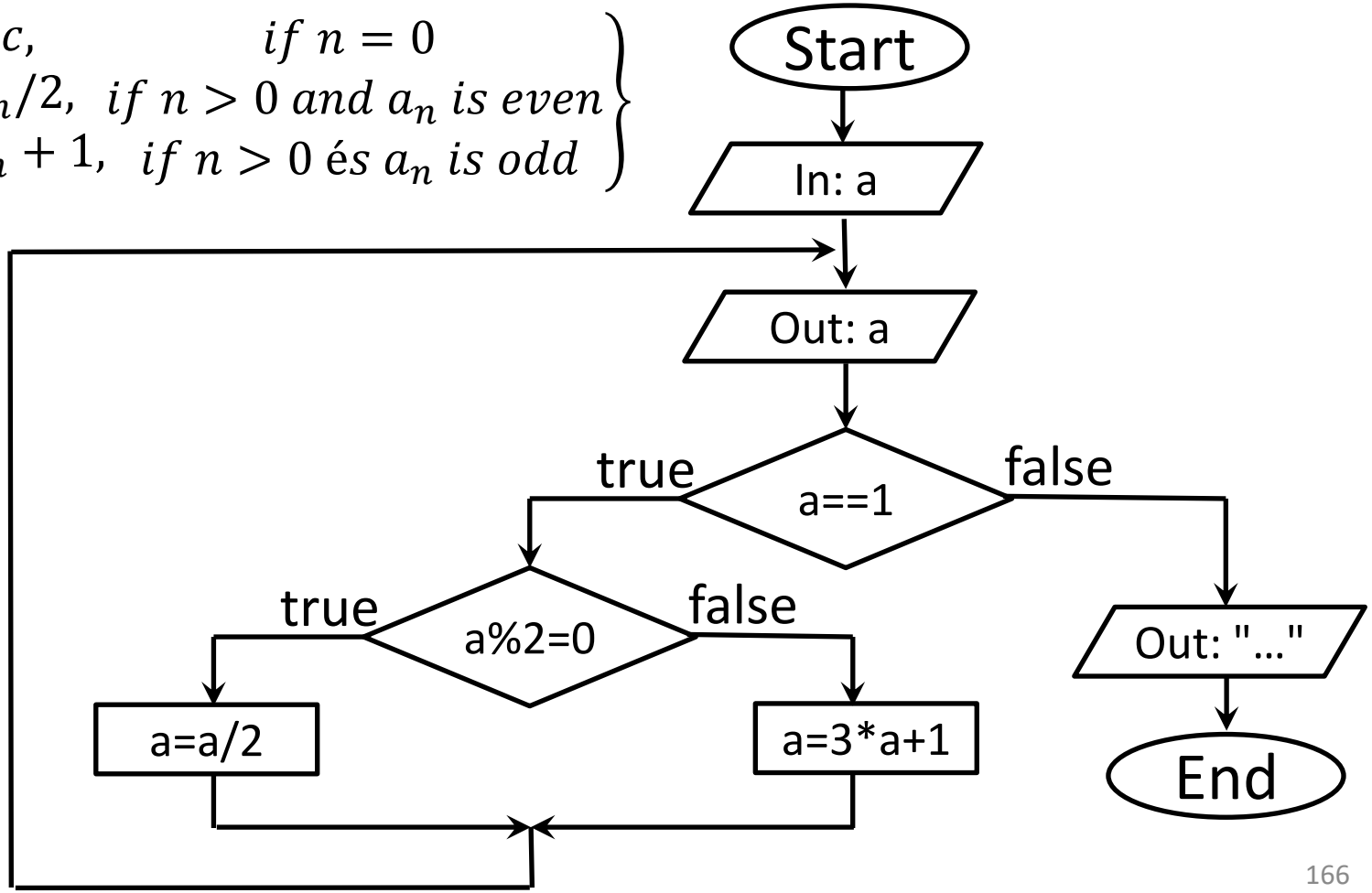
- Binary incrementation
  E.g.: 1010111 → 1011000



Solution: flowchart

- Collatz-conjecture

$$a_{n+1} = \begin{cases} c, & if\ n = 0 \\ a_n/2, & if\ n > 0\ and\ a_n\ is\ even \\ 3a_n + 1, & if\ n > 0\ és\ a_n\ is\ odd \end{cases}$$



Solution: pseudocode

```
input a
if a<0 then
   b=-1*a
else
   b=a
endif
output b
```

- Input: 10; Output: *10*
- Input : -4; Output : *4*
- Absolut value of a number

- Absolut value of a number

```
input a
if a<0 then
   a=-1*a
endif
output a
```

Alternative algorithms!

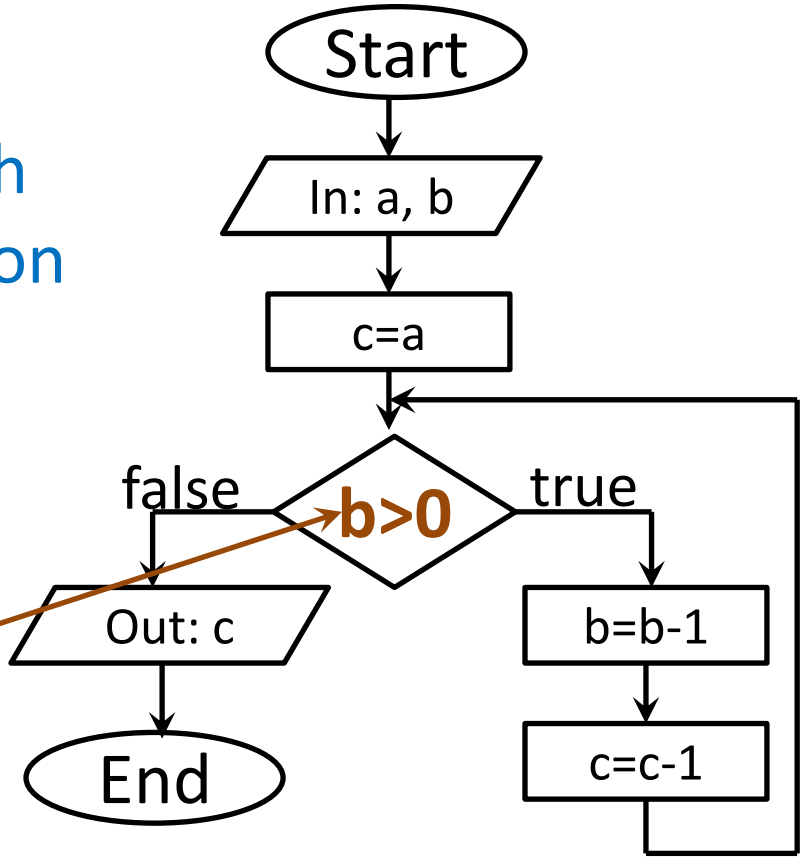
Solution: Are they the same?

```
input a
input b
c=a
if b>0 then
    b=b-1
    c=c-1
else
   output c
endif
```

- Different meaning
  (not alternative algorithms)

Branch condition

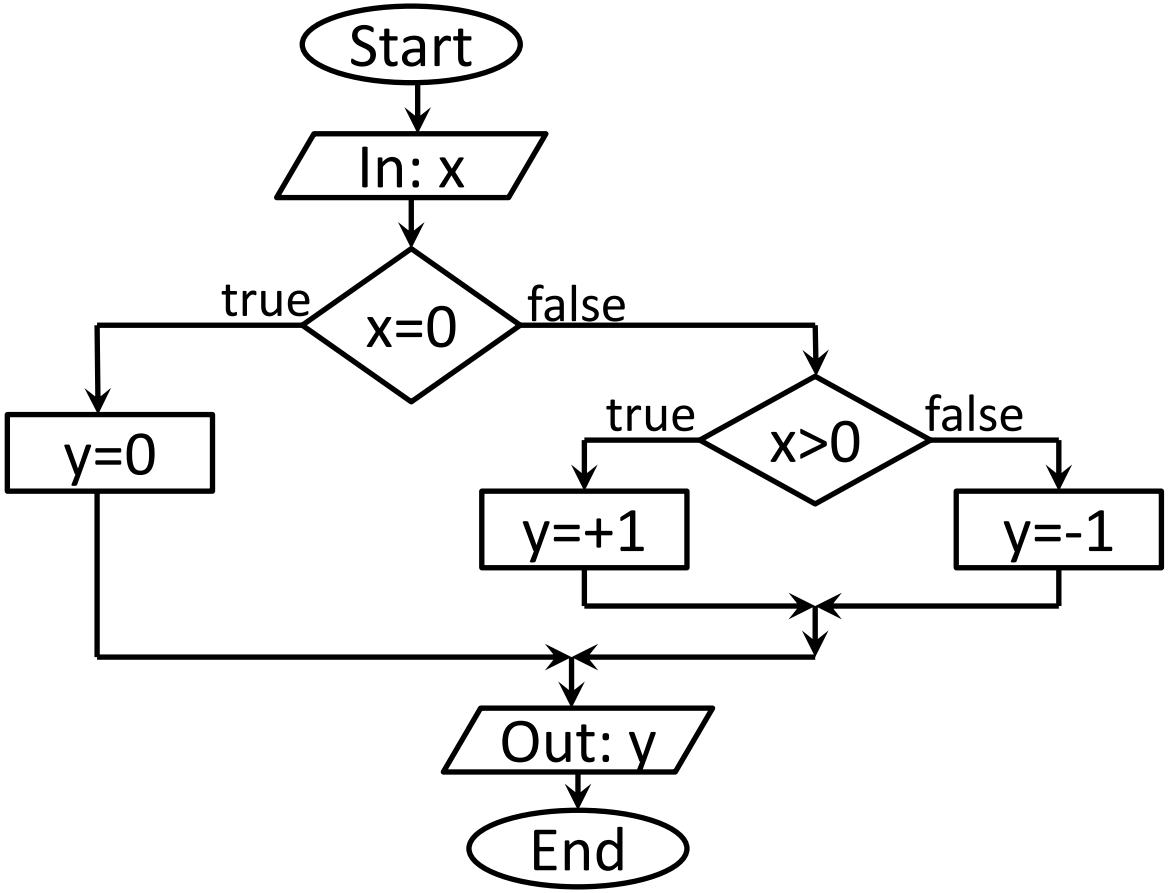
Loop condition



Solution: conversion 1

Sign functions:

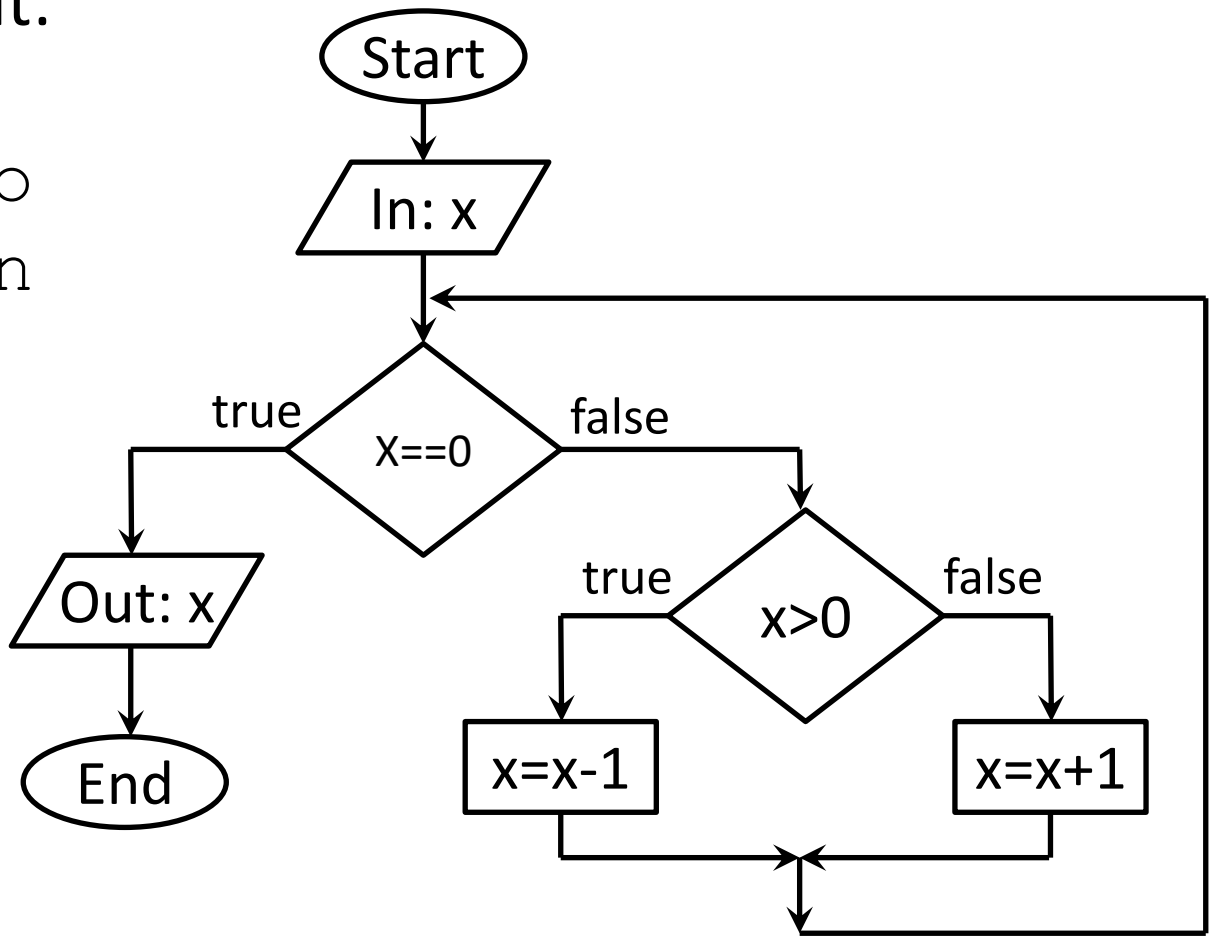
```
input x
if x==0 then
   y=0
else
   if x>0 the
      y=+1
   else
      y=-1
   endif
endif
output y
```



Solution: conversion 3

Always zero output:

```
input x
while x!=0 do
   if x>0 then
      x=x-1
   else
      x=x+1
   endif
enddo
output x
```



Solution: pseudocode

```
input a
input b
c=a
while b>0 do
    b=b-1
    c=c-1
enddo
output c
```

- Debugging:

a	b	c
7	3	?
7	3	7
7	2	6
7	1	5
7	0	**4**

- Input: a=7, b=3; Output: 4
- 3 iteration steps
- 4 expression evaluations
- Calculation of difference c=a-b

Solution: pseudocode

```
input N
R=0
while N>0 do
   R=R*10+N%10
   N=(int)(N/10)
enddo
output R
```

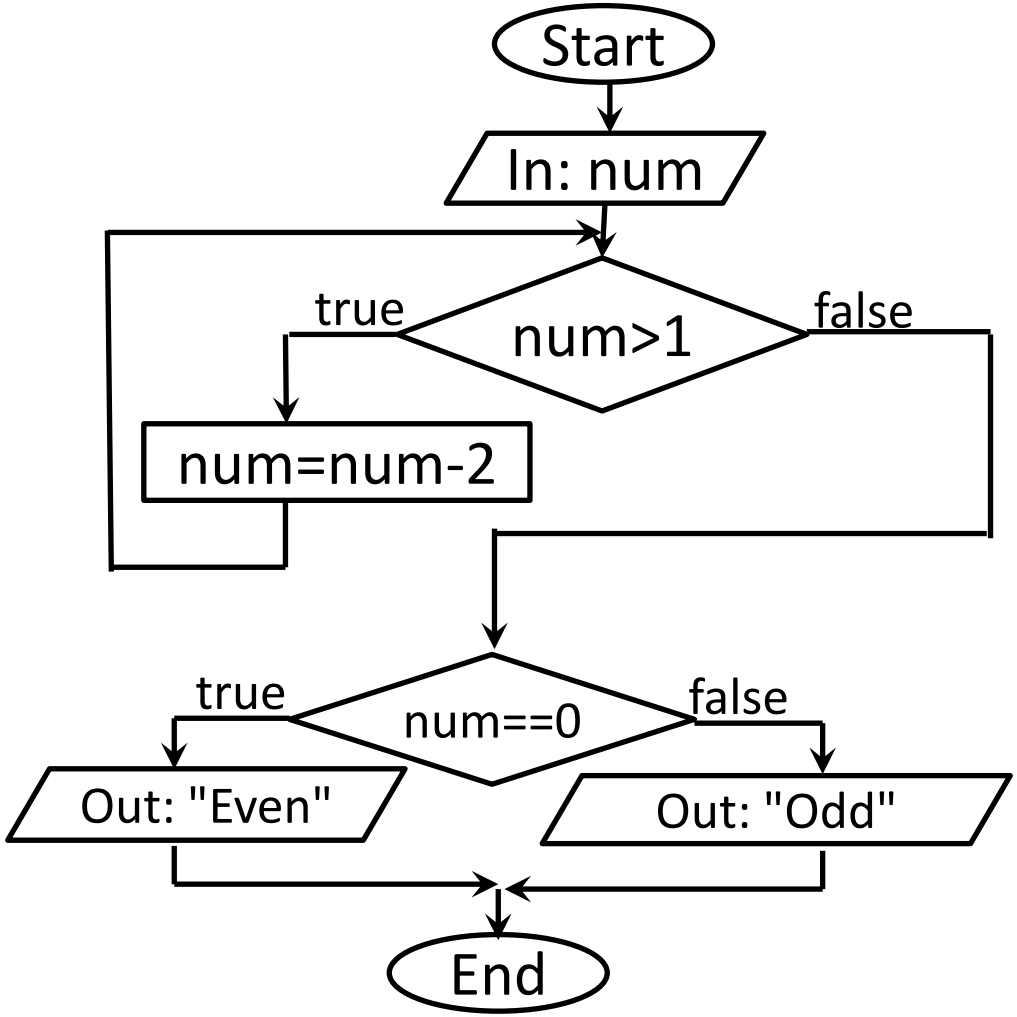
- Debugging:

N	R
73251	0
7325	1
732	15
73	152
7	1523
0	**15237**

- Output: 15237
- Change the order of digits of an integer.

Solution: even or odd

```
input num
while num>1 do
   num=num-2
enddo
if num==0 then
   output "Even"
else
   output "Odd"
endif
```



Solution: selections

```
input a,b,c
if a<b then
  if a<c then
    output a
  else
    output c
  endif
else
  if b<c then
    output b
  else
    output c
  endif
endif
```

```
input a,b,c
min=a
if b<min then
  min=b
endif
if c<min then
  min=c
endif
output min
```

```
input a
input b
input c
if a<b AND a<c then
  output a
else
  if b<c then
    output b
  else
    output c
  endif
endif
```

```
input a,b,c
if a<b then
  d=a
else
  d=b
endif
if c<d then
  output c
else
  output d
endif
```

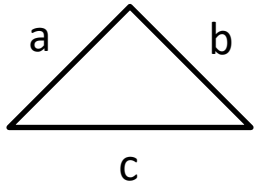
```
input a,b,c
if a<=b AND a<=c then
  output a
  stop
endif
if b<=a AND b<=c then
  output b
  stop
endif
if c<=b AND c<=a then
  output c
endif
```

```
input a,b,c
if c<a AND c<b then
  output c
  stop
endif
if b<a then
  output b
  stop
endif
output a
```

Minimum of 3 numbers: 6 different solutions

Solution: selections

- Triangle inequality




```
input a, b, c
if a<b+c AND b<c+a AND c<a+b then
 output "Drawable triangle"
else
 output "Not drawable triangle"
endif
```

Solution: iterations

```
input B
input E
if E==0 AND B==0 then
  output "Undefined!"
  stop
endif
if E<0 then
  if B!=0 then
    B=1/B
  else
    output "Infinity!"
    stop
  endif
  E=E*(-1)
endif
P=1
while E>0 do
  P=P*B
  E=E-1
enddo
output P
```

Raising to power ($P=B^E$)

General solution:

$$f(x,y,z) = x^{y/z}$$

Solution: iterations

```
output "Base"
input B
output "Exponent numerator"
input EN
if EN!=0 then
  output "Exponent
denominator"
  input ED
else
  ED=1
endif

if B==0 AND EN==0 then
  output "Undefined!"
  stop
endif

if ED<0 then
  ED=-1*ED
  EN=-1*EN
endif

if EN<0 then
  if B!=0 then
    EN=-1*EN
    B=1/B
  else
    output "Intinity!"
    stop
  endif
endif
```

```
T1=EN
T2=ED
while T2>0 do
  T3=T2
  T2=T1%T2
  T1=T3
enddo
EN=EN/T1
ED=ED/T1

P=1
TE=EN
while TE>0 do
  P=P*B
  TE=TE-1
enddo

if ED==1 do
  output P
  stop
endif

if P<0 AND ED%2==0 then
  output "No real root!"
  stop
endif

if P<0 AND ED%2==0 then
  output "No real root!"
  stop
endif
```

```
R=0
D=1
if P<0 then
  S=-1*P
else
  S=P
endif
Th=0.0000001
while S>Th do
  A=1
  TE=ED
  while TE>0 do
    A=A*R
    TE=TE-1
  enddo
  F=0
  if D==1 AND A>P then
    F=1
  endif
  if D==-1 AND A<P then
    F=1
  endif
  if F==1 do
    S=S/2.0
    D=-1*D
  endif
  R=R+D*S
enddo
output R
```

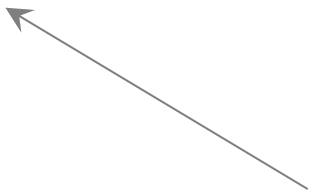
Solution: square root

```
output „Which number?"
input Num
output „What is the threshold?"
input Threshold
Old=Num
Diff=Num
while Diff>Threshold do
  New=Old-(Old*Old-Num)/(2*Old)
  Diff=Old-New
  Old=New
enddo
output „Square root:" Old
```

Solution: primes

- Is prime?

```
input N
S=2
while S<N do
   if N%S==0 then
      output "no"
      stop
   endif
   S=S+1
enddo
output "yes"
```



Solution: sequences

- First 100 elements of Fibonacci-sequence

```
F1=0
F2=1
output F1, F2
N=2
while N<=100 do
   F3=F1+F2
   output F3, " "
   F1=F2
   F2=F3
   N=N+1
enddo
```

```
F1=0
F2=1
output F1, F2
N=2
while N<=100 do
   F1=F1+F2
   F2=F1+F2
   output F1, " "
   output F2, " "
   N=N+2
enddo
```

Solution: sequences

- Elements of Fibonacci-sequence below 1000

```
F1=0
F2=1
output F1, F2
F3=F1+F2
while F3<1000 do
   output F3, " "
   F1=F2
   F2=F3
   F3=F1+F2
enddo
```

$$f_i = \begin{cases} 0, & if\ i = 1, \\ 1, & if\ i = 2, \\ f_{i-1} + f_{i-2} & otherwise \end{cases}$$

Solution: leap days

```
input y1, y2
n=0
while y1<y2 do
   if y1%4==0 then
      n=n+1
   endif
   y1=y1+1
enddo
output n
```

```
input y1, y2
if y1%4!=0 then
   y1=y1+4-y1%4
endif
if y2%4==0 then
   y2=y2-4
else
   y2=y2-y2%4
endif
output (y2-y1)/4+1
```

Performace?

Solution: searching

- Finding a value in an array

```
N=10
A[]={3,5,2,-7,0,34,5,3,567,9}
input Demanded
i=0
while i<N AND A[i]!=Demanded do
  i=i+1
enddo
if i<N then
  output "Found"
else
  output "Not found"
endif
```

Solution: searching

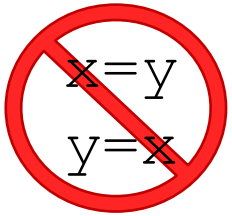
- Maximal value and its index in an array

```
N=10
A[]={3,5,2,-7,0,34,5,3,567,9}
max_i=0
i=1
while i<N do
   if A[i]>A[max_i] then
     max_i=i
   endif
   i=i+1
enddo
output max_i, A[max_i]
```

Solution: sorting

- Exchange variable contents (x=5; y=8 → x=8; y=5)

```
z=x
x=y
y=z
```



- Alternative solution without temporary variable

```
x=x+y
y=x-y
x=x-y
```

Solution: sorting

- Bublesort

```
N=10
A[]={3,5,2,-7,0,34,5,3,567,9}
end=N-1
while end>0 do
  i=0
  while i<end do
    if A[i]>A[i+1] then
      x=A[i]
      A[i]=A[i+1]
      A[i+1]=x
    endif
    i=i+1
  enddo
  end=end-1
enddo
```

Visit this as well:
https://hu.wikipedia.org/wiki/Buborékrendezés

Solution: bank card number

```
a[]={1,3,4,2,6,5,7,8,9,7,6,5,1,0,5,7}
i = 0
while i<=15 do
 a[i] = 2*a[i]
 if a[i]>=10 then
  a[i] = a[i]-9
 endif
 i = i+2
enddo
i = 0
s = 0
while i<=15 do
 s = s+a[i]
 i = i+1
enddo
if s%10==0 then
 output "valid"
else
 output "invalid"
endif
```

```
i = 0
s = 0
while i<=15 do
 s=s+(2-i%2)*a[i]
 s=s-{a[i]/5}*(1-i%2)*9
enddo
if s%10!=0 then
 output "not "
endif
output "valid"
```

Luhn's algorithm

Solution: most frequent age

```
N=15
Age[]={1,3,2,0,5,10,17,3,10,0,3,2,15,1,3}
Distr[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
i=0
while i<N do
  Distr[Age[i]]=Distr[Age[i]]+1
  i=i+1
enddo
freq=0
i=1
while i<18 do
  if Distr[i]>Distr[freq] then
    freq=i
  endif
  i=i+1
enddo
output "The age ", freq, " is the most frequent."
```

Solution: best-fit allocation

```
N=20
M[]={1,0,5,5,5,5,5,0,0,4,4,4,4,0,1,0,4,4,4,4}
input S
i=0
pos=0
while i<N do
  if M[i]==S then
    pos=i
    break
  endif
  if M[i]>S AND M[i]<M[pos] then
    pos=i
  endif
  i=i+1
enddo
if pos>0 OR M[pos]>=S then
  output pos
endif
```

Solution: finding a pattern

```
N=11
A[]={2,3,9,4,1,8,3,9,4,8,6}
M=4
B[]={3,9,4,8}
iA=0
iB=0
while iA<N AND iB<M do
  if A[iA]==B[iB] then
    iB=iB+1
    iA=iA+1
  else
    iA=iA-iB+1
    iB=0
  endif
enddo
if iB==M then
  output "Pattern found."
else
  output "Not found."
endif
```

Solution: subroutine

```
function PP( a )
   b=0
   while b<a do
      a = a-1
      b = b+1
   enddo
   if a=b then
      return 1
   else
      return 0
   endif
endfunction
```

```
input a, b
a = a*2
b = PP(a+b)+1
output b
```

a	b	a	b
1	4	-	-
2	4	-	-
2	4	6	
2	4	6	0
2	4	5	1
2	4	4	2
2	4	3	3
2	2	-	-

- The output: 2
- In case of even parameter it gives 1, in odd case it gives 0.

Solution: subroutine

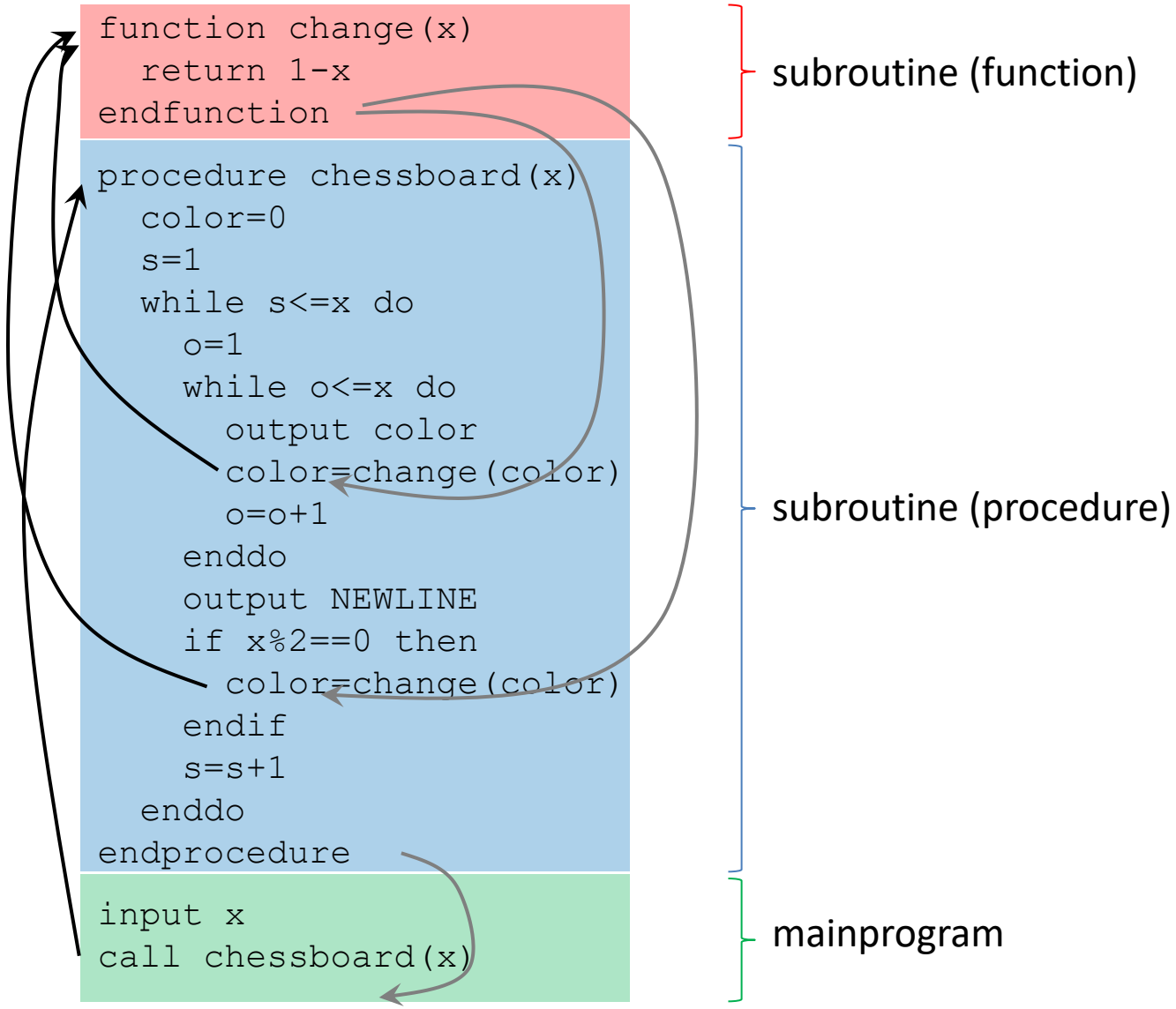
```
function CHANGE( a )
   return 1-a
endfunction
input Max
i=0
j=0
while j<Max do
   i = CHANGE(i)
   j=j+i
   output j
enddo
output j
```

- Max=5 → 1 1 2 2 3 3 4 4 5 5
- Writes natural numbers twice.
- 0→1 and 1→0 change

Solution: multiplication table

```
procedure MT(N)
   i=1
   while i<=N do
      j=1
      while j<=N do
         output i*j, " "
         j=j+1
      enddo
      output NEWLINE
      i=i+1
   enddo
endprocedure
call MT(8)
```

Solution: chessboard



Solution: time

```
function secs(hour, min, sec)
   return (hour*60+min)*60+sec
endfunction
procedure time(t)
   s=t%60
   m=((int)(t/60))%60
   h=(int)(t/3600)
   output h, ":", m, ":", s
endprocedure
call time(secs(12,15,30))
```

Solution: DCB arithmetics

```
function get_digit(Num,Pos)
  return (int)(Num/Pos)%10
endfunction

function set_digit(Num,Pos,Bit)
  big=(int)(Num/(Pos*10))
  little=Num%Pos
  return big*Pos*10+Bit*Pos+little
endfunction

function magnitude(Num)
  Pos=10
  while Num>=Pos do
    Pos=Pos*10
  enddo
  return Pos/10
endfunction

function max(A,B)
  if A>B then
    return A
  else
    return B
  endif
endfunction

function add(A,B)
  bigpos=max(magnitude(A),magnitude(B))
  sum=0
  carry=0
  pos=1
  while pos<=bigpos do
    dA=get_digit(A,pos)
    dB=get_digit(B,pos)
    digit=dA+(dB+carry)
    if digit>1 then
      digit=digit-2
      carry=1
    else
      carry=0
    endif
    sum=set_digit(sum,pos,digit)
    pos=pos*10
  enddo
  if carry==1 then
    sum=set_digit(sum,pos,1)
  endif
  return sum
endfunction
```

```
function sub(A,B)
  bigpos=max(magnitude(A),magnitude(B))
  diff=0
  carry=0
  pos=1
  while pos<=bigpos do
    dA=get_digit(A,pos)
    dB=get_digit(B,pos)
    digit=dA-(dB+carry)
    if digit<0 then
      digit=digit+2
      carry=1
    else
      carry=0
    endif
    diff=set_digit(diff,pos,digit)
    pos=pos*10
  enddo
  if carry==1 then
    output "Negative difference"
  endif
  return diff
endfunction

function mul(A,B)
  pos=magnitude(B)
  prod=0
  while pos>=1 do
    prod=add(prod,pos*A*get_digit(B,pos))
    pos=pos/10
  enddo
  return prod
endfunction

function div(A,B)
  pos=magnitude(A)
  quotient=0
  D=(int)(A/pos)
  while pos>=1 do
    if D>=B then
      digit=1
      D=sub(D,B)
    else
      digit=0
    endif
    quotient=quotient*10+digit
    pos=pos/10
    D=D*10+get_digit(A,pos)
  enddo
  return quotient
endfunction
```

```
function pow(A,B)
  p=1
  while B>0 do
    p=mul(p,A)
    B=sub(B,1)
  enddo
  return p
endfunction

output "Enter 3 binary values: "
input X,Y,Z
output div(pow(X,Y),Z)
output "Oh, it was so easy, isn't it?"
```

Solution: testing strategy

N	B	Result	OK?
11	2	1011	✓
2	5	2	✓
0	9	0	✓
43	0	-	×
-10	4	22	×
9	-2	-999	×
10	1.5	-	×
3.5	4	-	×
31	16	25	×
64	1	-	×
1024	2	1410065408	×

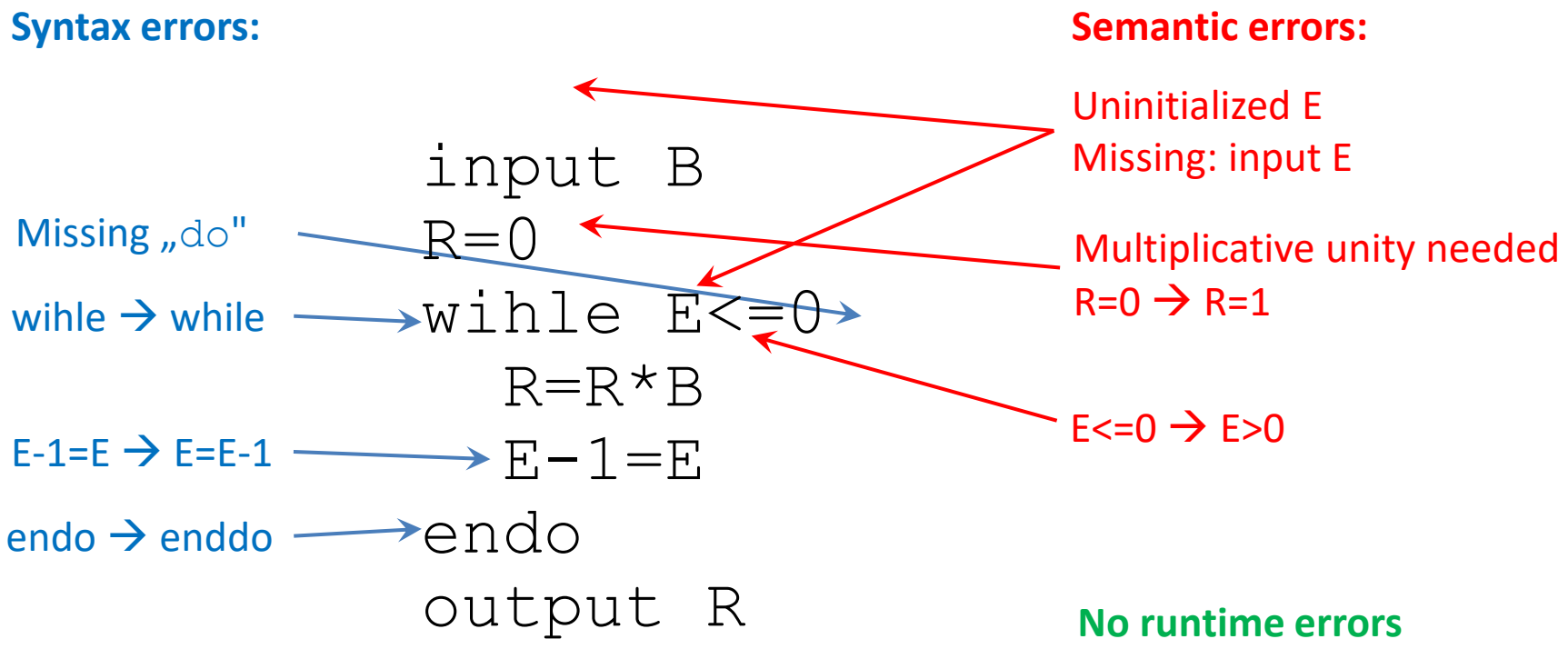
B ∈{2,3,4,5,6,7,8,9,10}
N not negative integer number (N<Limit(B))

```
input N
input B
R=0
P=1
while N!=0 do
   R=R+(N%B)*P
   P=P*10
   N=(int)(N/B)
enddo
output R
```

Solution: testing strategy

Input: x	Input: y	Expected output
2	3	8
0	3	0
2	0	1
0	0	Undefined value.
-2	3	-8
-2	4	16
2	-3	0.125
-2	-3	-0.125
0	-3	Infinite
0.2	3	0.008
-0.2	3	-0.008
0.2	-3	125
-0.2	-3	-125
2	0.3	Exponent is not an integer.
100	13	11 991 163 848 716 906 297 072 721

Solution: syntax and semantics



Solution: data representation

- Population is more than 7 000 000 000.
  Unsigned fixed point maximum is $2^{32}-1 = 4294967295$.
  Not representable! (33 bits)
- `-1:`
  `11111111 11111111 11111111 11111111`
- `15908:`
  `00000000 00000000 00111110 00100100`
- `-666:`
  `11111111 11111111 11111101 01100110`
- `10000000 00000000 00000010 01001001:`
  `-2147483063` (signed), `2147484233` (unsigned)

Solution: expression

- `((9+((2*6)/3))>(8-7))`
  value: true
- `((2>3)&&(((3*5)-(6/2))>=(11%2)))`
  value: false (C language: 0)
- `* + 1 2 - 9 6`
  value: 9, i.e. ((1+2)*(9-6))
  `+ 1 - * 2 13 / 25 5`
  value: 22, i.e. (1+((2*13)-(25/5)))
- `30 2 15 4 6 + - * /`
  value: 3, i.e. (30/(2*(15-(4+6))))
  `1 2 13 * 25 5 / - +`
  value: 22, i.e. (1+((2*13)-(25/5)))

Solution: Python

- Keyword: for, in, if, else
- Comment: # Some calculation, #return z
- Identifier: Sum, N, print, z, cos
- Constant: 0, 10, 2, 90
- Variable: Sum, N, z
- Operator: =, +=, ==, %, +, /
- Expression: Sum=0, Sum+=i, Sum==0, "Total"+Sum, z=10%2+N/N+cos(90)
- Function call: cos(90)
- statement: Sum=0, for …, if …else…, Sum+=1, print(…), z=10%2+N/N+cos(90)

```
# Some calculation
Sum=0
for i in range(N):
    Sum+=i
if(Sum==0):
    print("Total"+Sum)
else:
    z=10%2+N/N+cos(90)
#return z
```