Introduction to

Introduction to programming

Imre Varga

University of Debrecen, Faculty of Informatics

For internal use only!

04 May 2014

General informations

Teacher:

Dr. Varga, Imre
University of Debrecen
Faculty of Informatics
Depertment of Informatics Systems and Networks

email: varga.imre@inf.unideb.hu
www: irh.inf.unideb.hu/user/vargai
room: IF13 (building of Faculty of Informatics)

General informations

Requirements, conditions:

Maximum number of absences is 3.
Maximum late from classes is 20 minutes.
More than 20 minutes late means absent from class.

There will be two tests during the semester.
There is only one chance to retake!!!
Activity during class means plus score.

Furter readings:

Adrian Kingsley-Hughes: *Beginning Programming*, Wiley, 2005.

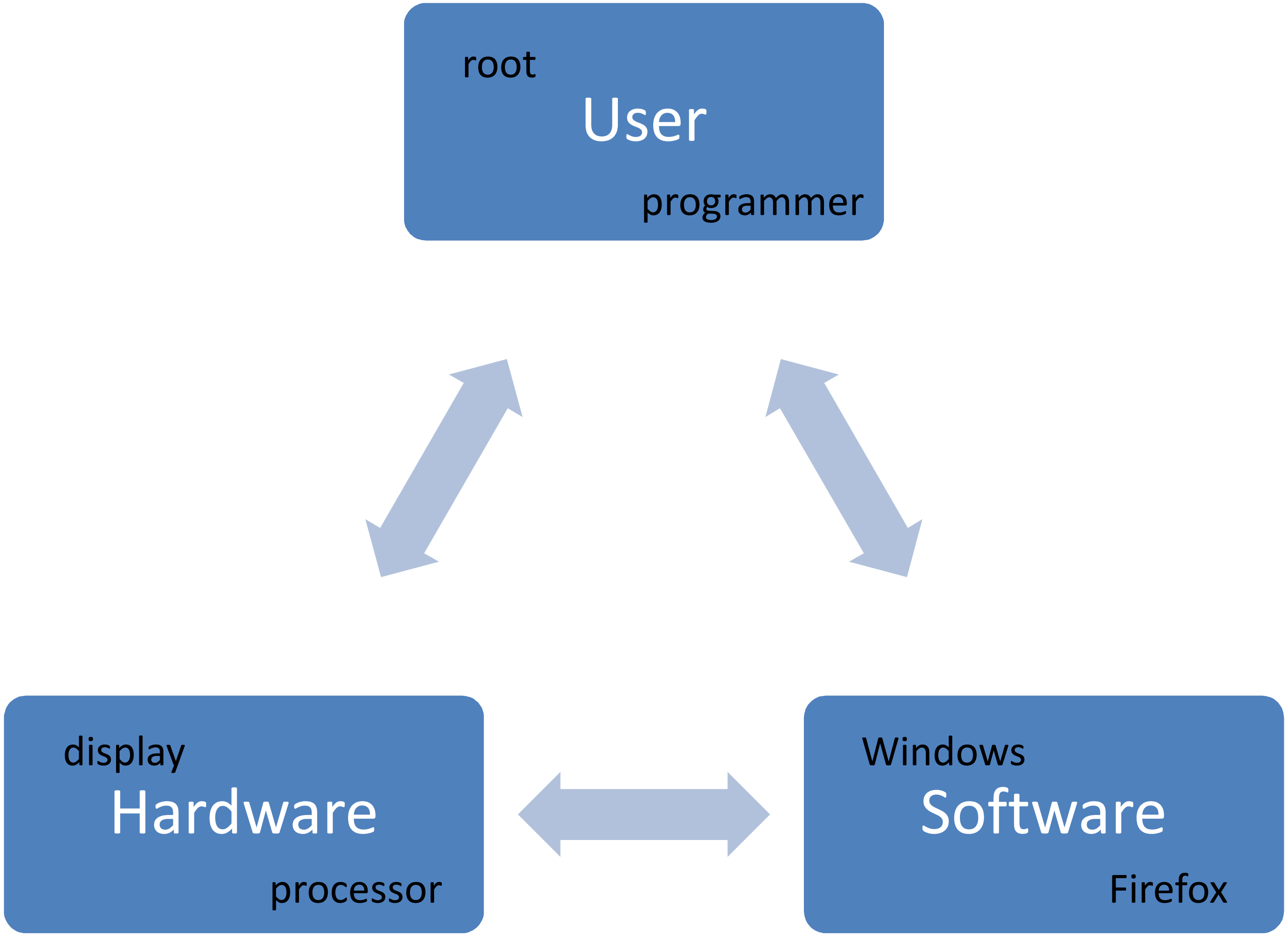
Metrowerks CodeWarrior: *Principles of Programming*

Topics

- What are the basics of Informatics?
- How does a computer built up?
- How does a computer work?
- What is software, application, program?
- How to describe problems and its solution?
- What is algorithmic thinking?
- What does ‘program writing’ mean?
- *Many more things…*

Computer systems

Computer System



root

User

programmer

display

Hardware

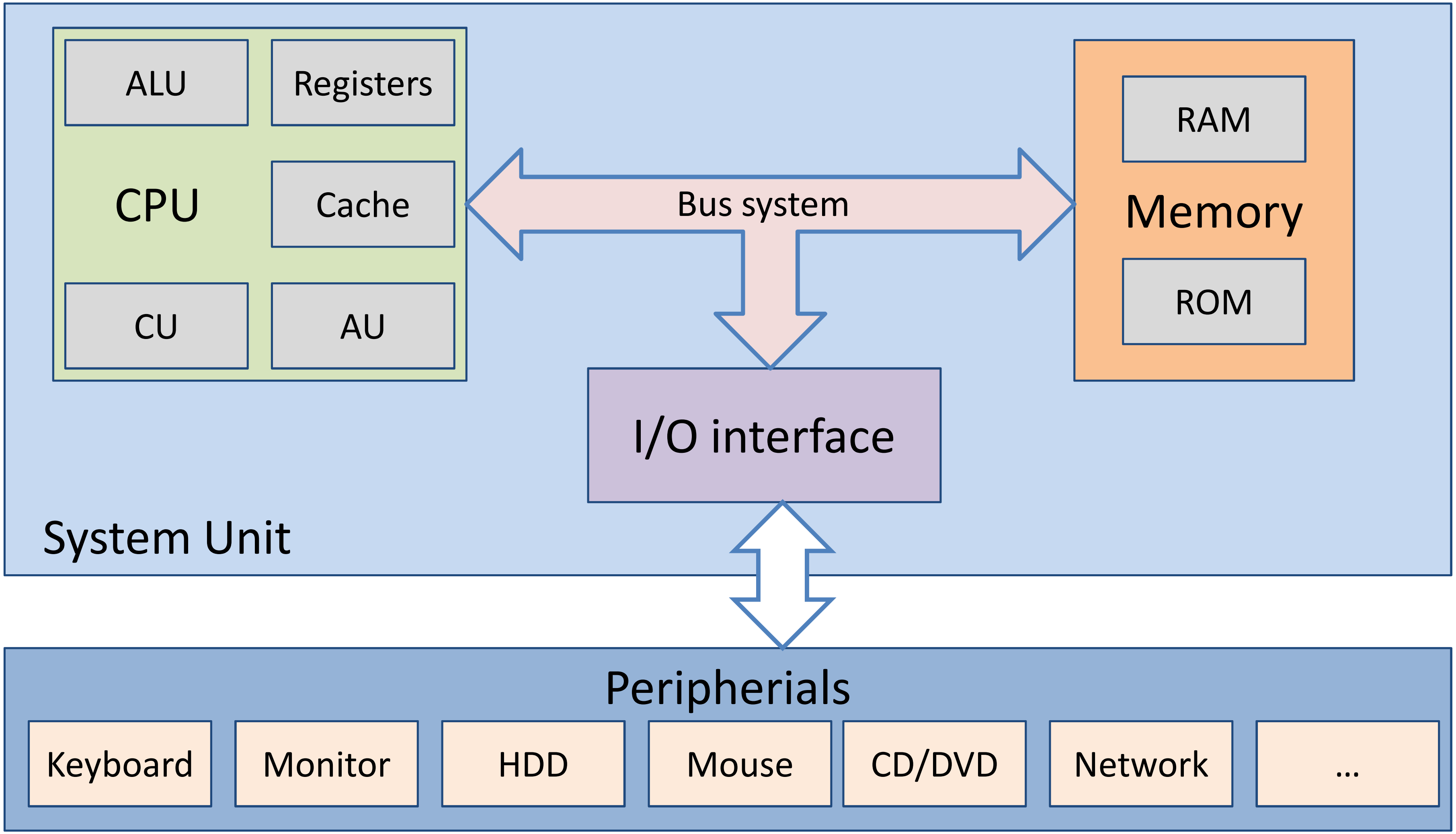
processor

Windows

Software

Firefox

Computer architecture (hardware)

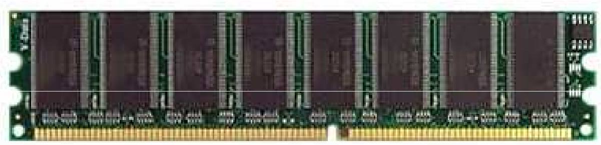


System Unit

Central Processing Unit (CPU):

The brain of computers

Memory:



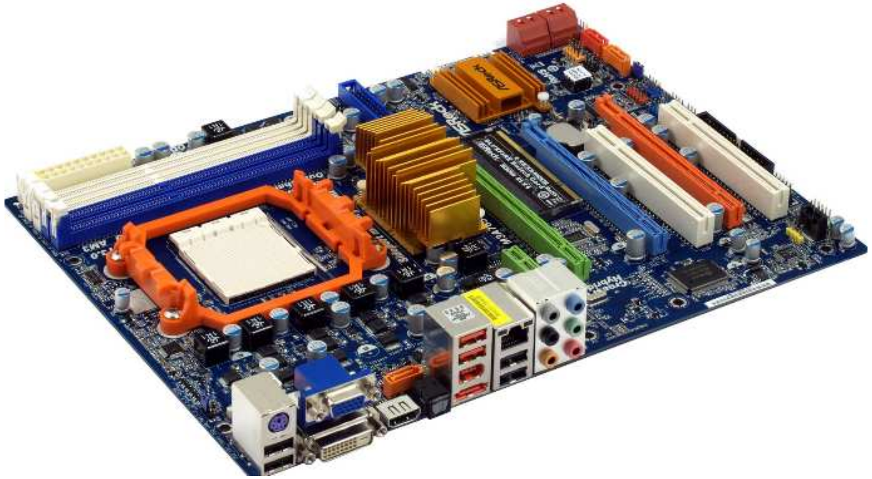
Contains datas and instructions

Input-Output Interface:

Surface between computer and outer world

Bus system:

Connects together



Central Processing Unit

Control Unit (CU): Says what to do, controls the parts of the CPU

Arithmetic Logic Unit (ALU): Performs operations, does calculations

Registers: Some tinny but very fast memory

Cache: small, but fast memory

Addressing Unit (AU): Deals with memory addresses at read/write operation

Memory

Random Access Memory (RAM):

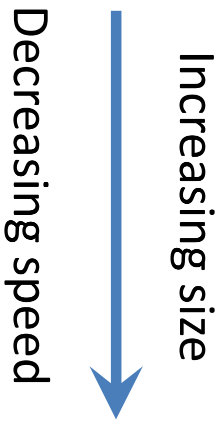
Readable-writeable operative memory

Read Only Memory (ROM)

Not rewritable (eg. BIOS-ROM)

Memory hierarchy:

- Register
- Cache
- Memory
- Hard-disk drive (HDD)



Bus system

Connects the CPU, the Memory and the I/O interfaces

Data bus:

Transports the datas from/to CPU

Address bus:

Contains memory address of reading/writing

Control bus:

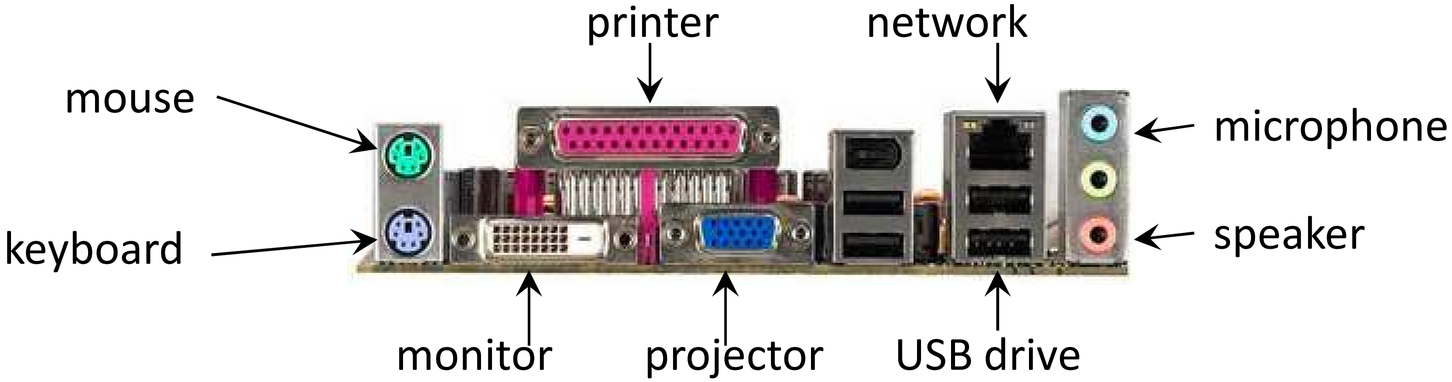
Carries control informations

Input-Output Interfaces

It makes the system accessible to periferials (world)

Connection to

- Input devices
- Output devices
- Storage devices
- Network devices



Peripherials

Input

- Keyboard
- Mouse
- Scanner

Storage

- Winchester (HDD)
- CD/DVD/Blu-ray drive
- USB drive
- Memory Card

Output

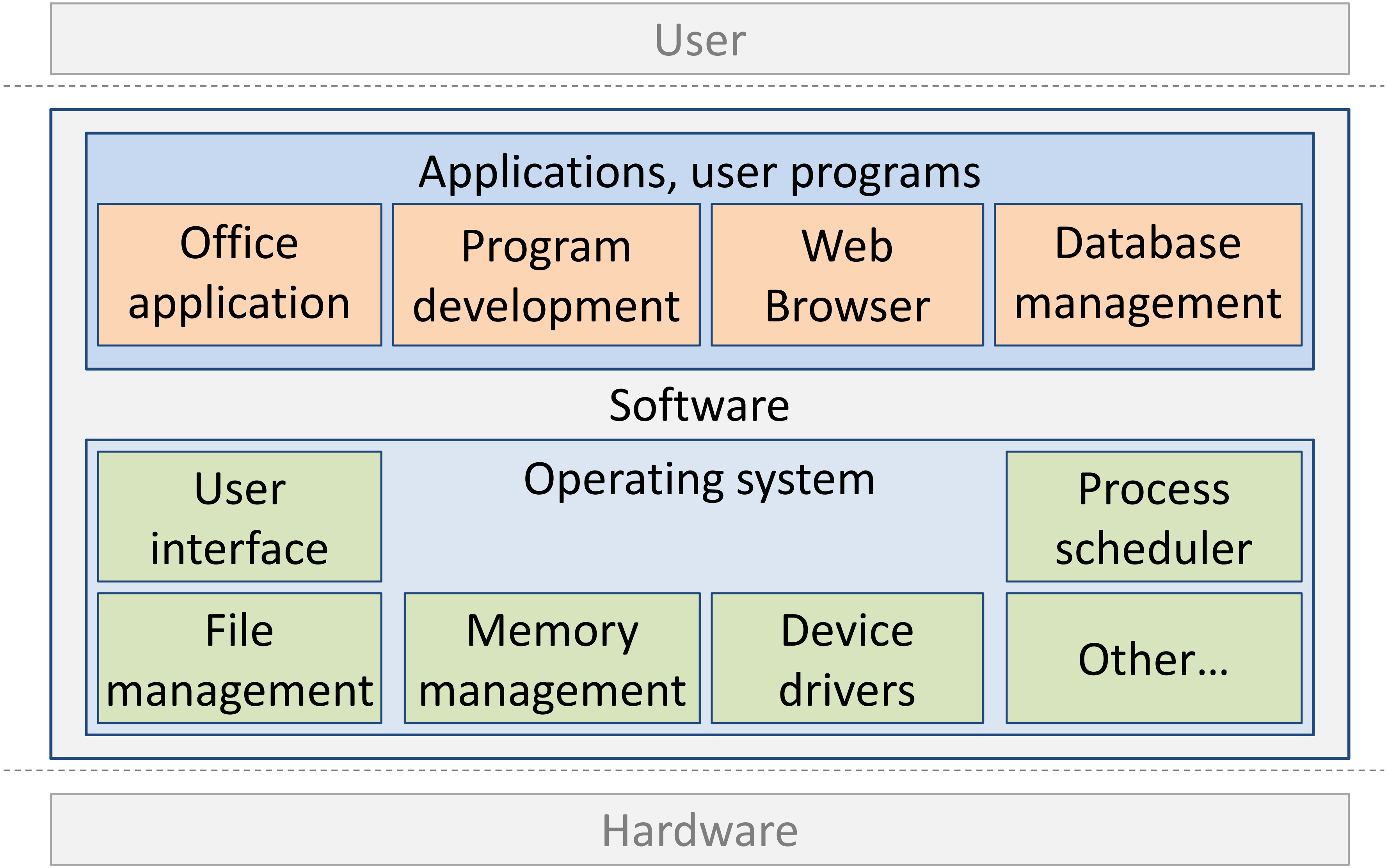
- Monitor
- Printer
- Projektor

Network

- Ethernet
- Wi-Fi

Other

Software



User

Applications, user programs

Office application

Program development

Web Browser

Database management

Software

Operating system

User interface

Process scheduler

File management

Memory management

Device drivers

Other…

Hardware

Operating system

Collection of software that manages hardware resources and provides services for other programs

- **User interface:**
  supports human interaction (shell, GUI)
- **Program scheduler:**
  decides which program can run now, for how long time, which will the next
- **File management:**
  handles the files and directories of volume based on a file system

Operating system

- **Memory management:**
  provides ways to dynamically allocate portions of memory to programs at their request
- **Device drivers:**
  software developed to allow interaction with hardware devices
- **Security:**
  protect against illegal operation and access to datas
- **Others:**
  Networking, Interrupt management, Utilities, …

User applications

- **File manager:**
  Windows Explorer, Midnight commander, …
- **Office application:**
  Microsoft Word/Excel, OpenOffice Write/Calc, …
- **Web browser:**
  Internet Explorer, Firefox, Crome, …
- **Database manager:**
  Microsoft Access, MySQL, DB2, …
- **Graphical program:**
  Microsoft Paint, GIMP, Photoshop, …

User applications

- **Media player:**
  Windows Media Player, Flash Player, QuickTime, …
- **Computer game:**
  Minesweeper, Solitaire, NFS, CoD, FIFA, …
- **Anti-virus program:**
  Virus Buster, NOD32, AVG, …
- **Integrated Development Environment (IDE):**
  BorlandC, Netbeans, CodeBlocks, Dev-C++, …
- **Other**:
  …

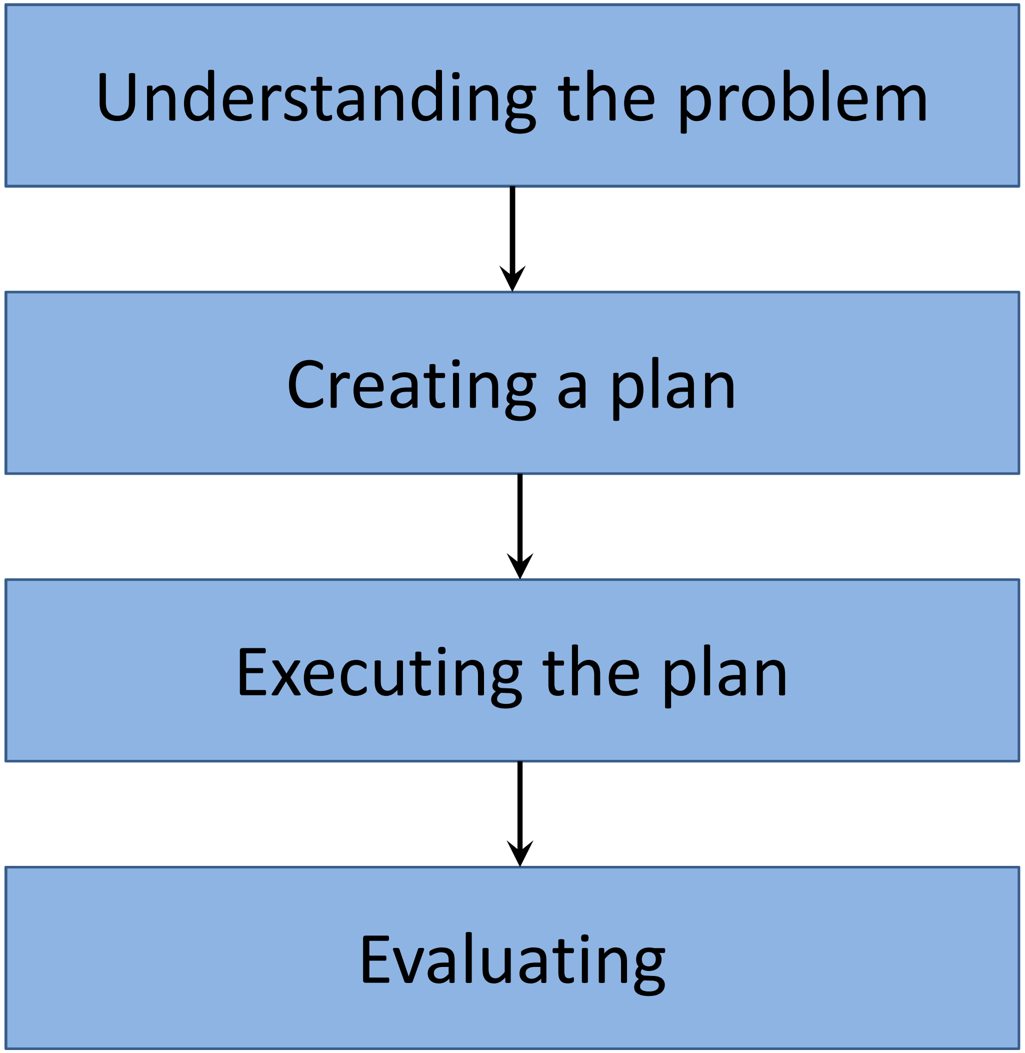
User

Human agent, who uses computer

- **Root:**
  Superuser, system addministrator, has high privilege
- **„Simple” user:**
  computer is just a tool, not the purpose of work
- **Programmer:**
  developes computer applications, writes programs

Problem solving

Pólya’s problem solving steps



Understanding the problem

Creating a plan

Executing the plan

Evaluating

Understanding the problem

- What is the task?
- What is the *unknown* (required result)?
- What is the relationship between the given information and the unknown?
- Is the given information enough to solve the problem?

Creating a plan

General techniques:

- Finding known similar problems (if exists)
- Reshaping the original problem to a similar known problem
- Devide the problem to shorter solvable problems
- Generalizing a restricted problem
- Finding existing work that can help in the search for a solution

Executing the plan

- Follow the steps of the plan
- Each element of the plan should be checked as it is applied
- If a part of the plan is unsatisfactory, the plan should be revised

Evaluating

The result should be examined

- Is it correct?
- Is it full?
- Is it valid?
- Has the problem been solved?

An example

What is the sum of

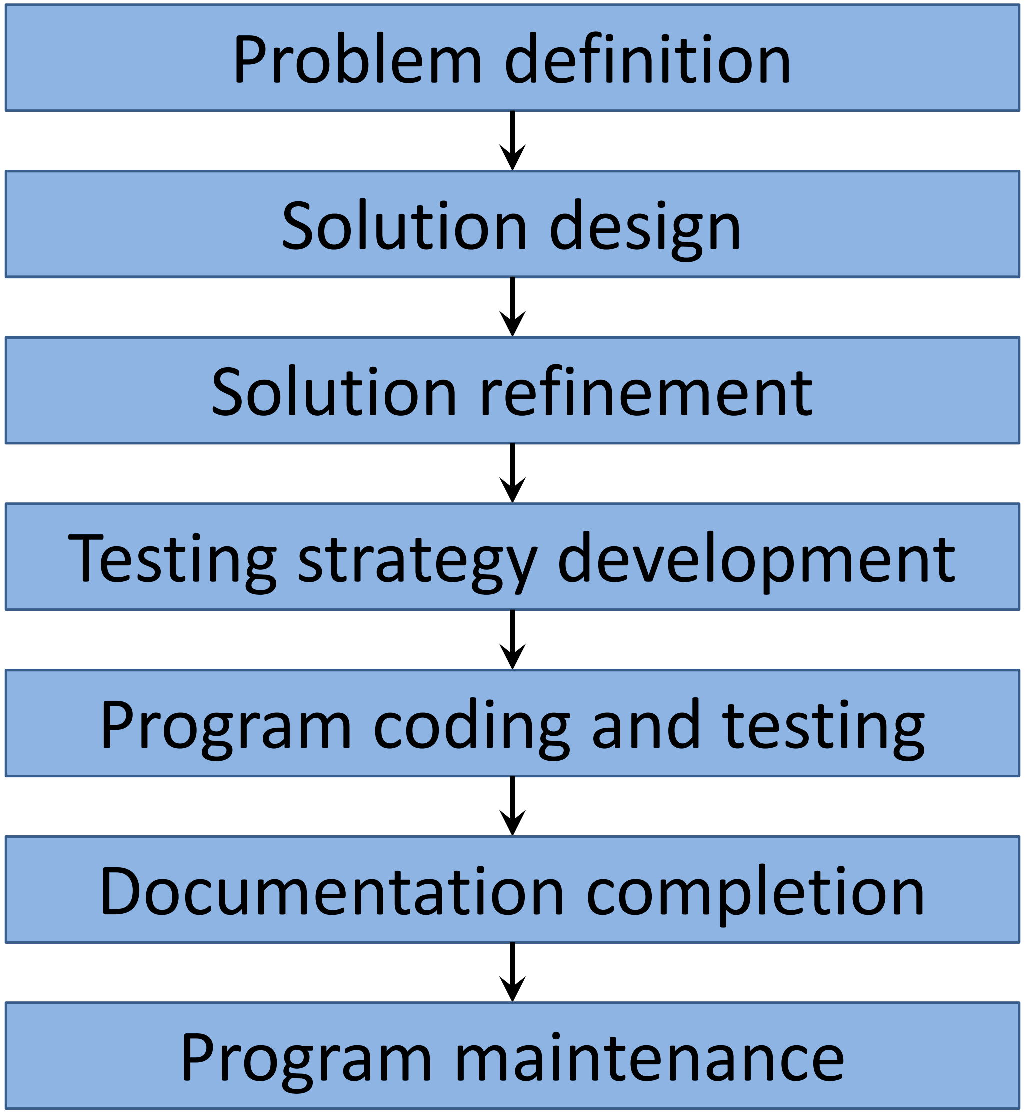
110010110 and 101110101

in binary notation?

See ‘Number systems’ slides!

Software Life Cycle

Software Life Cycle



Problem definition

Solution design

Solution refinement

Testing strategy development

Program coding and testing

Documentation completion

Program maintenance

1: Problem definition

- Similar to Pólya’s first step
- The description of the problem must be precise
- User and programmer must work together
- It leads to complete specitications of the problem, the input datas 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
- „Devide 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 algorithm in previous level cannot be executed directly by computer
- Translation needed to a **programming language**
- After coding program must be tested using our testing strategy
- If an error has been discovered, appropriate revision must be made, and than the test rerun until the program gives 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 dicisions that were made
  - Occured problems
  - Program list
  - User instructions
  - etc.

7: Program maintenance

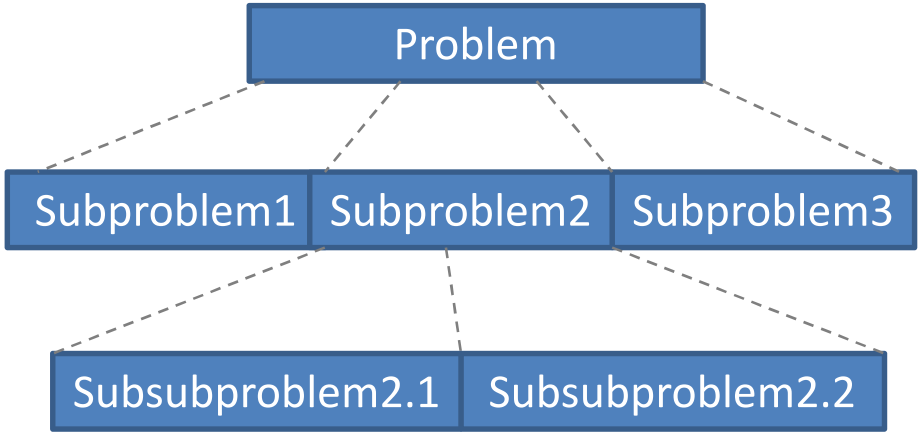
- The program can’t wear out
- Sometimes the program may fail
- The reason of a program fail is that it was never tested for this circumstance
- Elimination of newly deteceted error is necessary
- Sometimes the users need new features to the program
- Update of documentations is needed

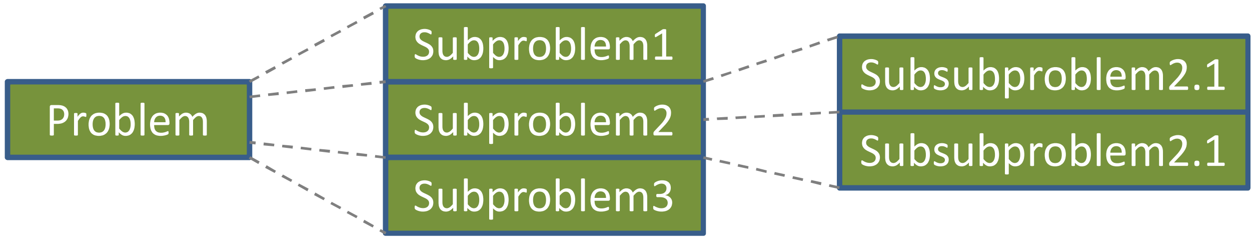
Solution design

by Break-Out Diagrams

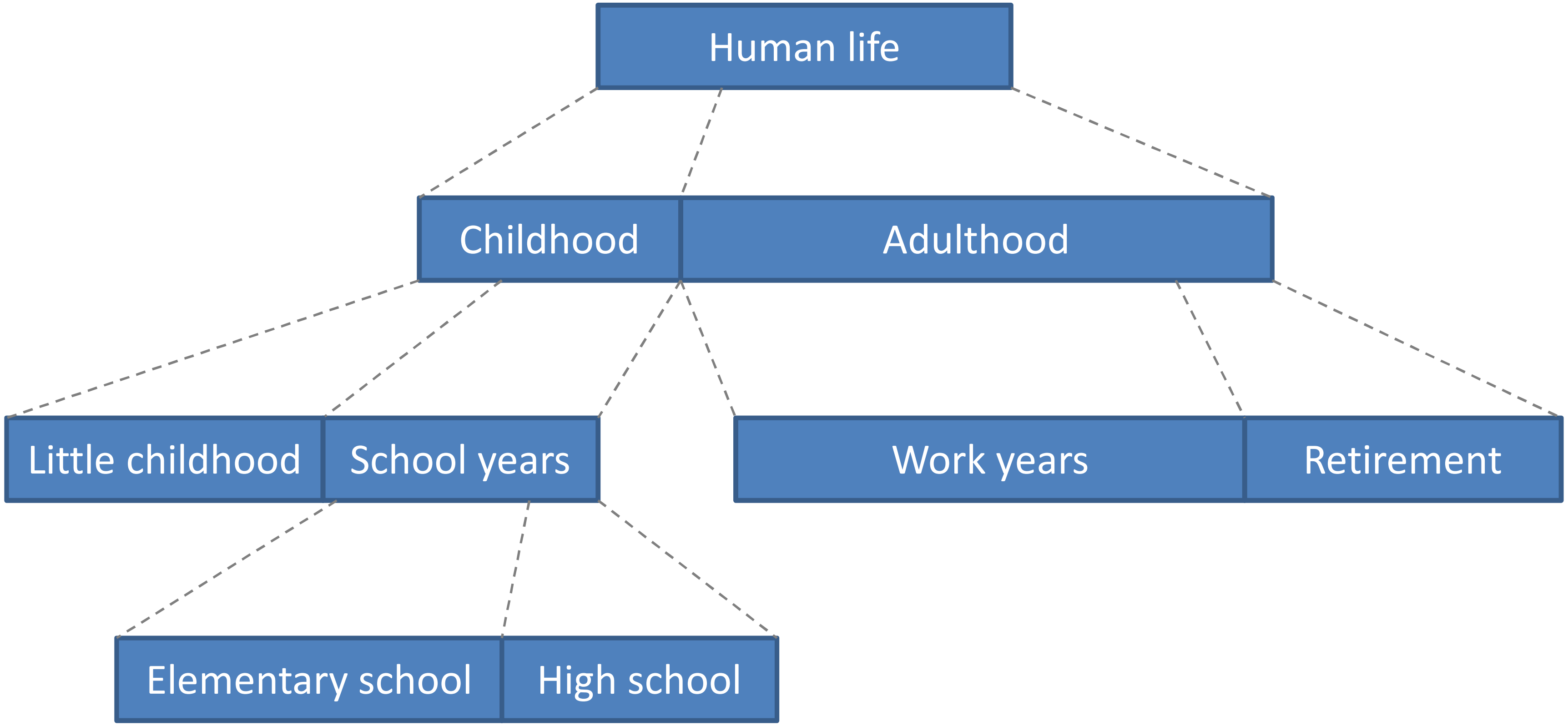
Break-Out Diagrams

- Useful way to make the problem solving manageable
- Tree-like (hierarchical) skeleton of problems
- For viewing problems in levels
- Styles:
  - Vertical
  - Horizontal

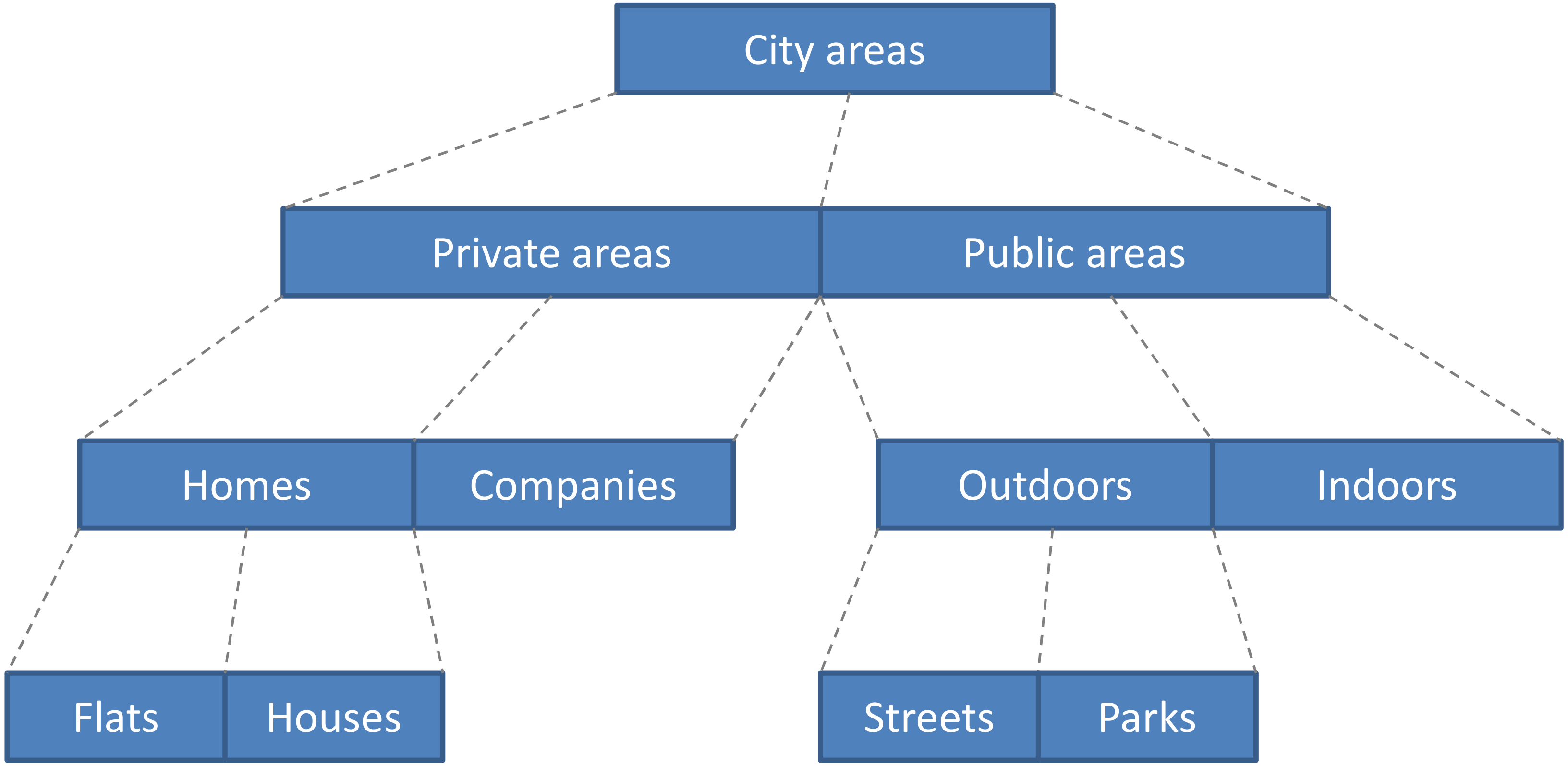




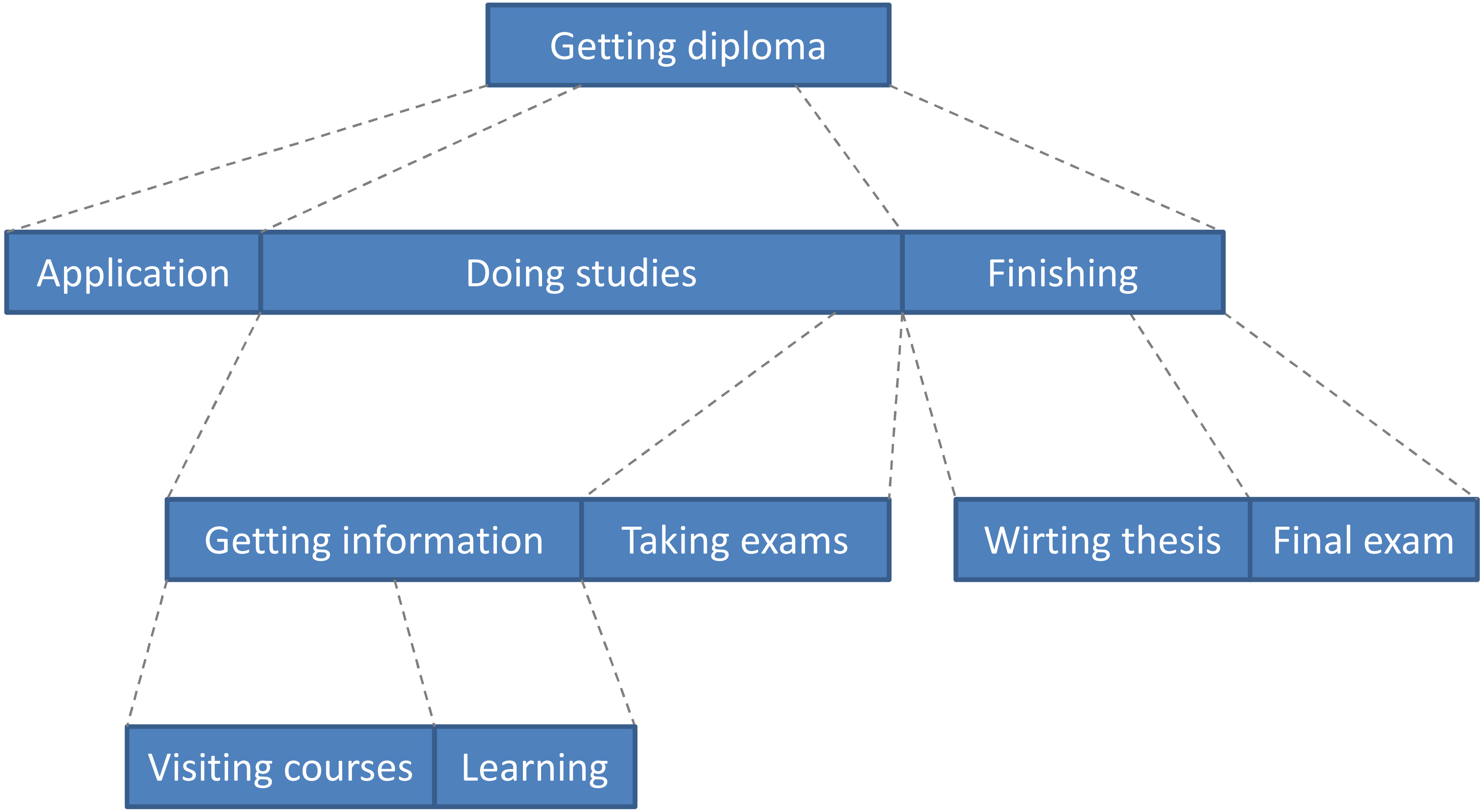
Time BOD


Human life
Childhood
Adulthood
Little childhood
School years
Work years
Retirement
Elementary school
High school

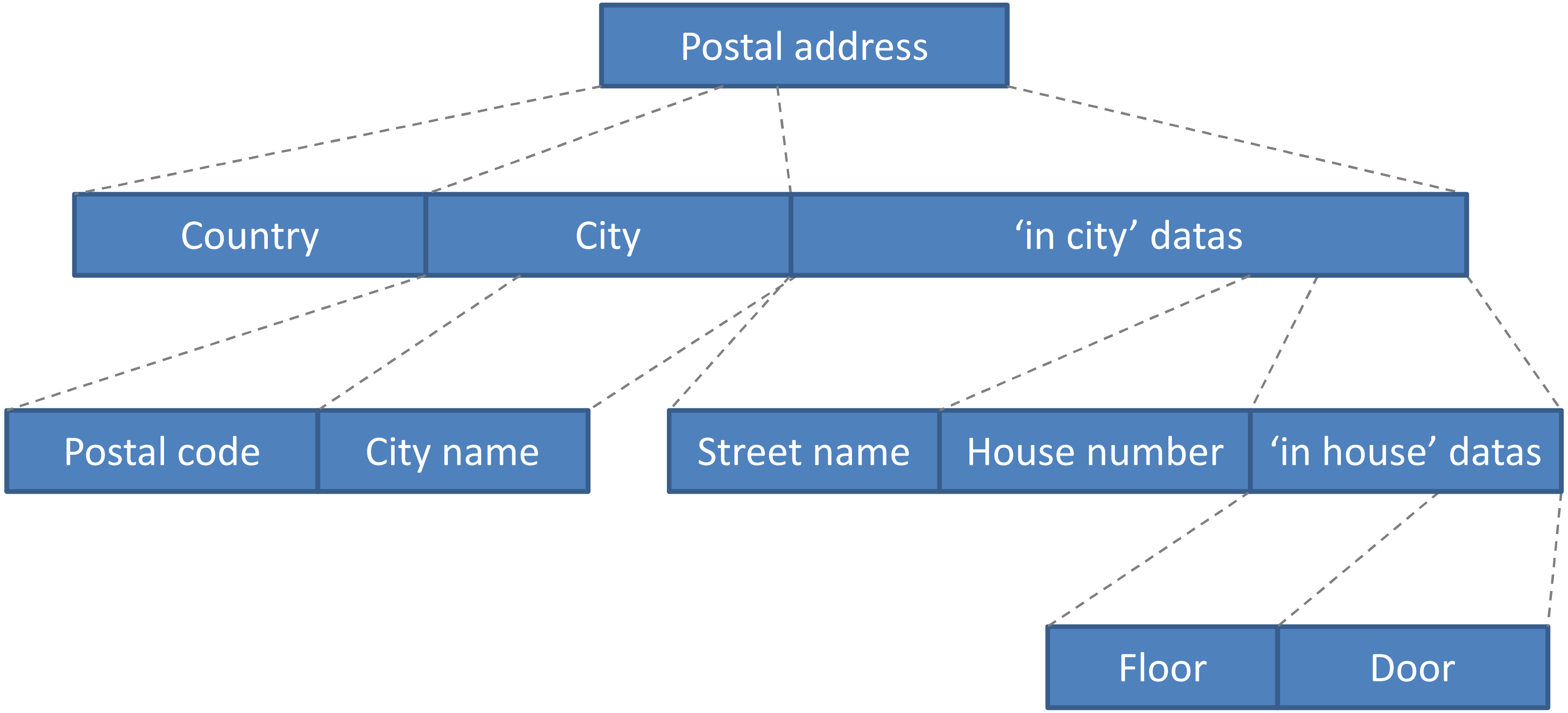
Space BOD


City areas
Private areas
Public areas
Homes
Companies
Outdoors
Indoors
Flats
Houses
Streets
Parks

Action BOD


Getting diploma
Application
Doing studies
Finishing
Getting information
Taking exams
Wirting thesis
Final exam
Visiting courses
Learning

Data BOD

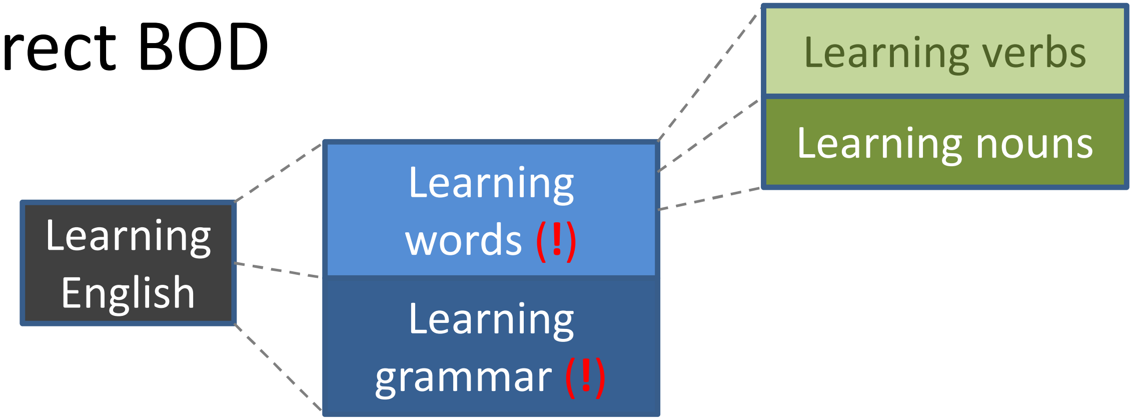

Postal address
Country
City
'in city' datas
Postal code
City name
Street name
House number
'in house' datas
Floor
Door

Properties of BODs

- **Consistent**
  Each break-out must be the same kind.
- **Orderly**
  All blocks at the same level must be separate or independent.
- **Refined**
  Each box of a given level must be break-out of a box at the previous level.
- **Cohesive**
  All of the items within a breakout box must fit together.

Mistakes and corrections

Incorrect BOD



Learning English

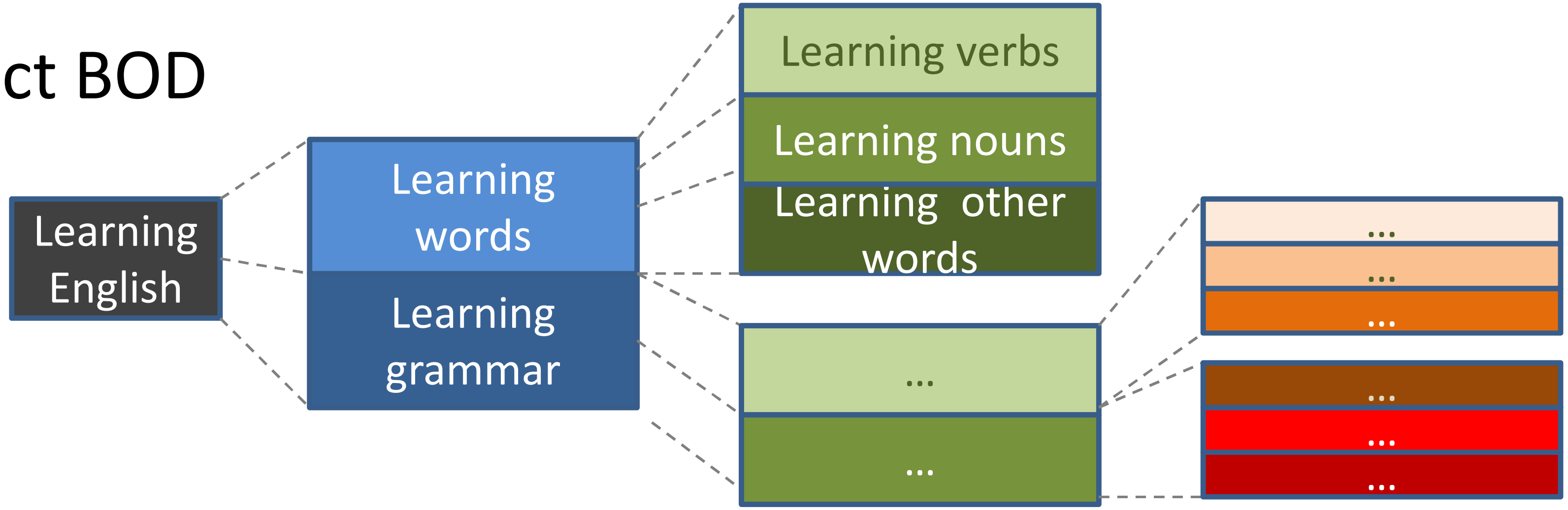
Learning words (!)

Learning grammar (!)

Learning verbs

Learning nouns

Correct BOD



Learning English

Learning words

Learning grammar

Learning verbs

Learning nouns

Learning other words

…

…

…

…

…

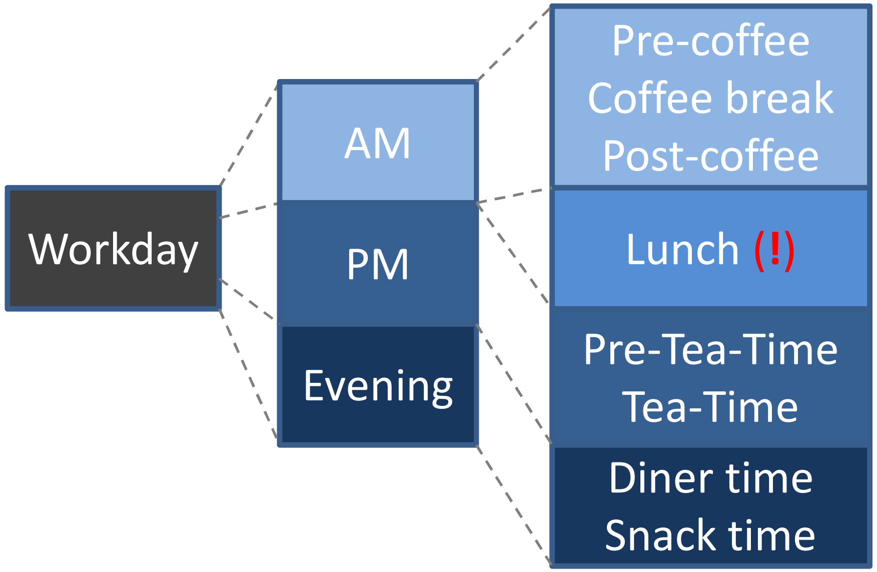
…

…

…

Mistakes and corrections

Incorrect BOD



Workday

AM

PM

Evening

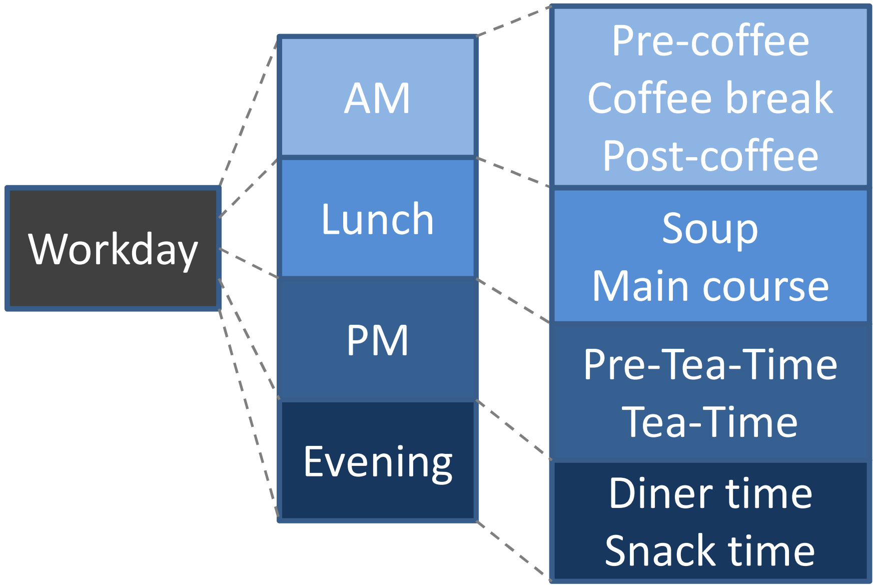
Pre-coffee
Coffee break
Post-coffee

Lunch (!)

Pre-Tea-Time
Tea-Time

Diner time
Snack time

Correct BOD



Workday

AM

Lunch

PM

Evening

Pre-coffee
Coffee break
Post-coffee

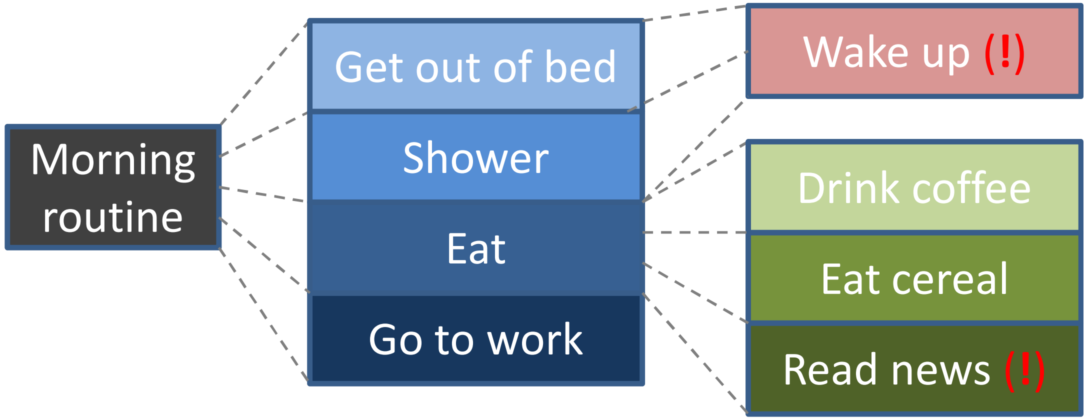
Soup
Main course

Pre-Tea-Time
Tea-Time

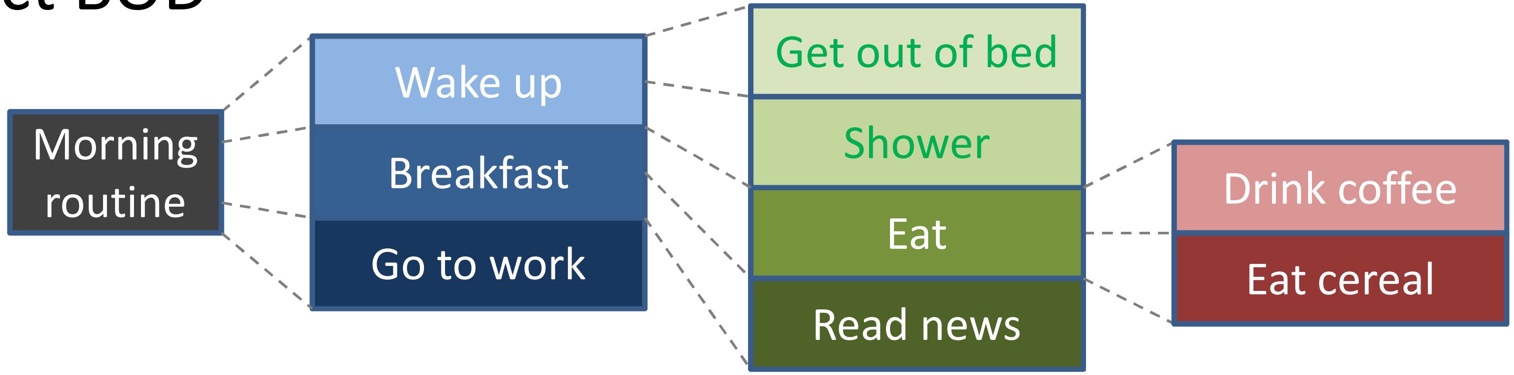
Diner time
Snack time

Mistakes and corrections

Incorrect BOD



Correct BOD



Exercises and examples

Draw BODs of the following things.

- Computer architecture
- Human body
- Surface of Earth
- Phone number
- Starting a car
- Set up a TV set
- Using computer
- Your plan for tomorrow
- …

Solution refinement

Algorithms

Algorithm

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

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

Some properties:

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

Representation of algorithms

- Algebraic
- Data-flow diagram
- Flowblocks
- **Flowchart**
- Graphs or plots
- Hierarchical
- **Pseudocode**
- Tabular
- **Verbal**

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?
- …

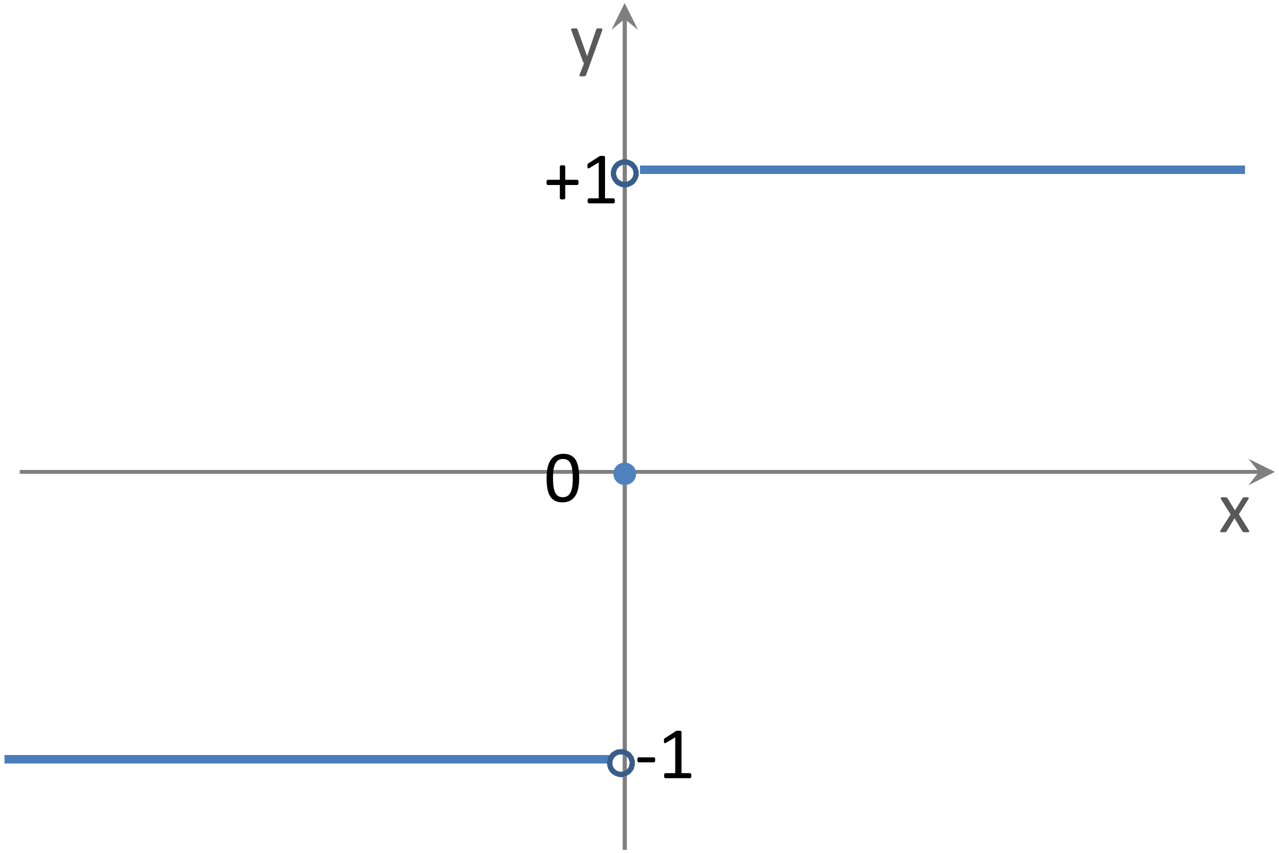
y=sign(x)

Verbal representation:

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

y=sign(x)

Graph representation:



y=sign(x)

'Algebraic-like' representation:

$x \in \Re$

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

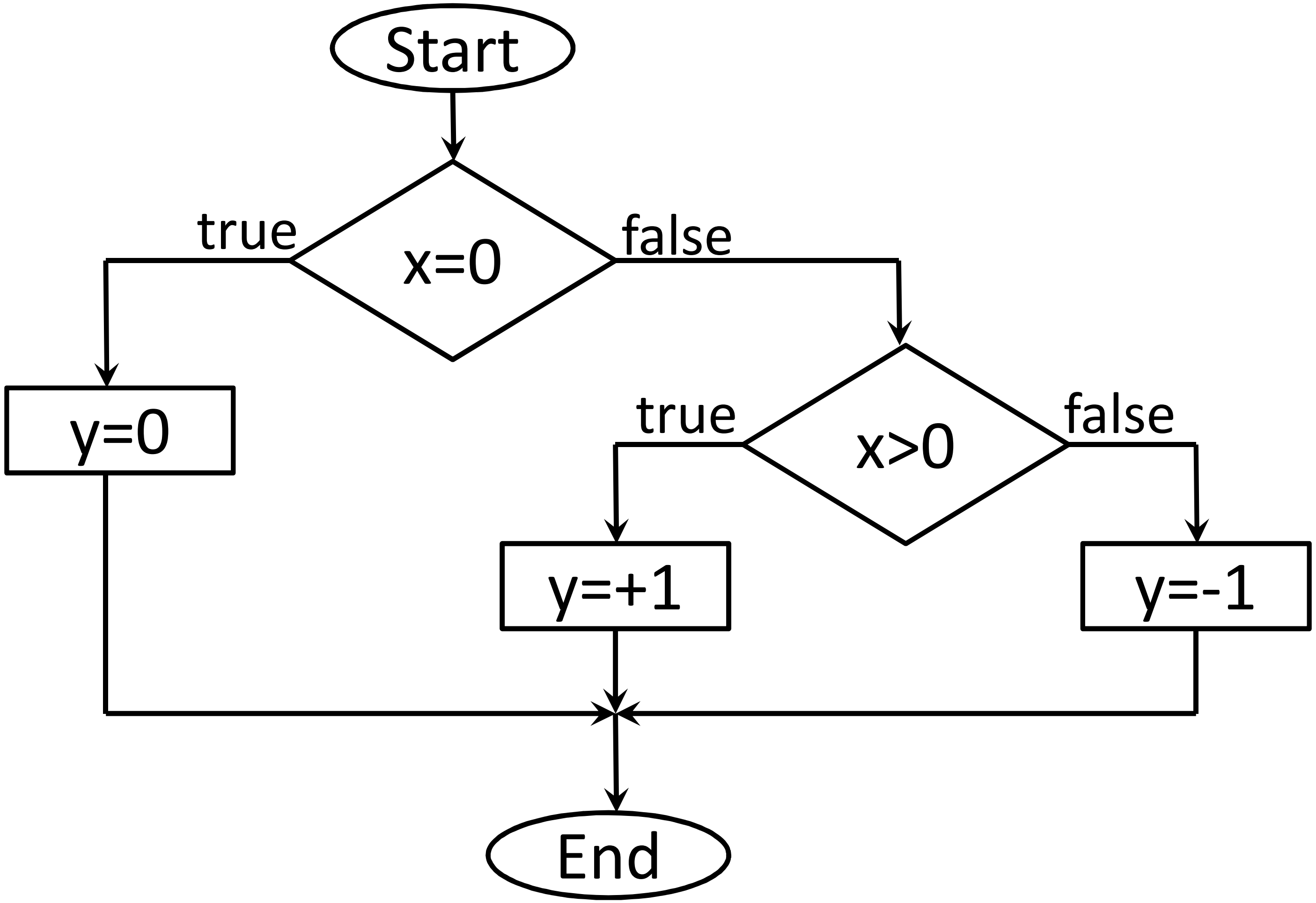
$\forall x, x>0 \Rightarrow y=+1$

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

$x=0 \Rightarrow y=0$

y=sign(x)

Flowchart representation:



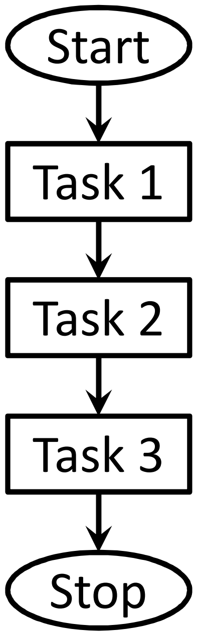
y=sign(x)

Pseudocode representation:

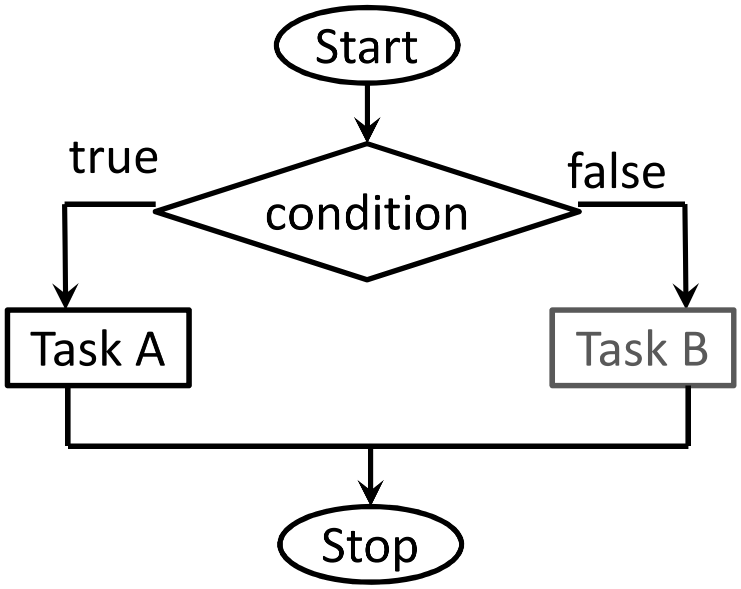
```
if x=0 then
  y=0
else
  if y>0 then
     y=+1
  else
     y=-1
  endif
endif
```

Base structures of algorithms

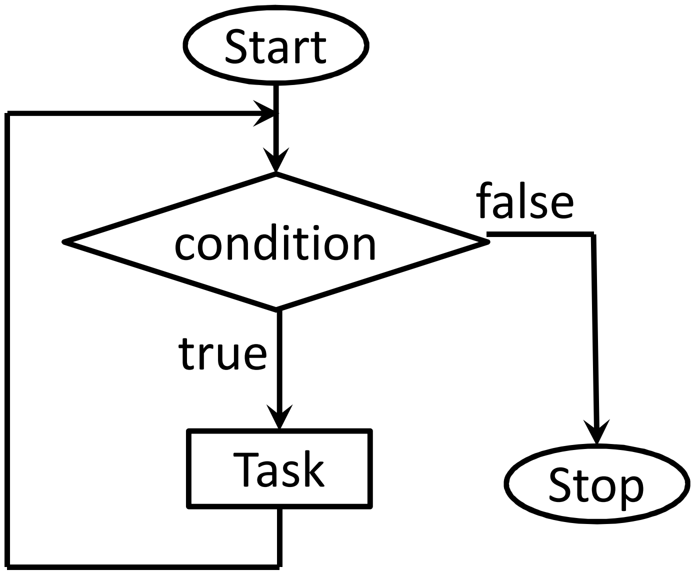
Sequence



Selection



Iteration



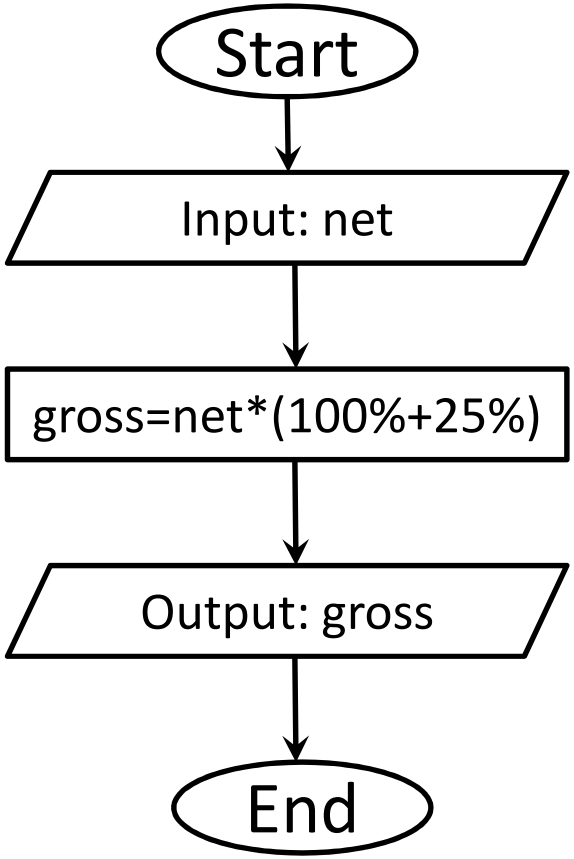
Modifying algorithms

Algorithms often go through many changes to be better.

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

Generalizing algoritms

Original:



Start

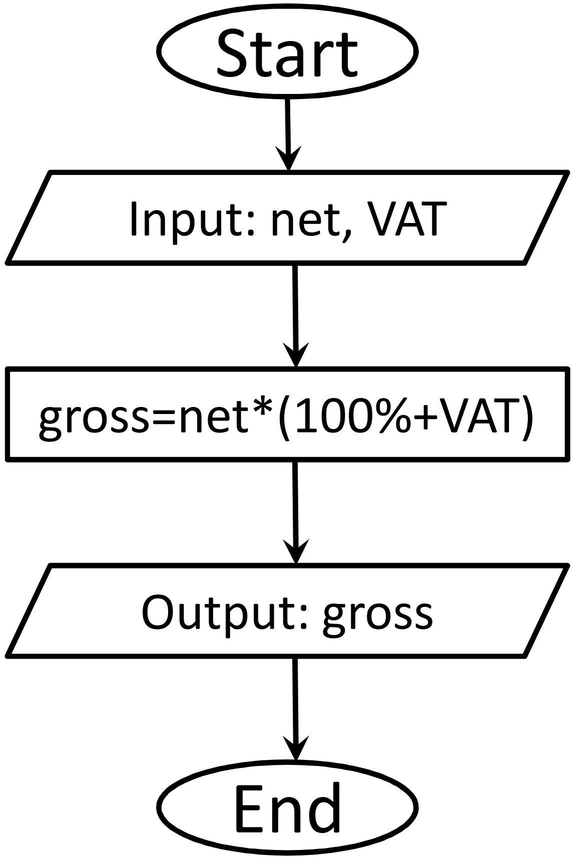
Input: net

gross=net*(100%+25%)

Output: gross

End

Generalized:



Start

Input: net, VAT

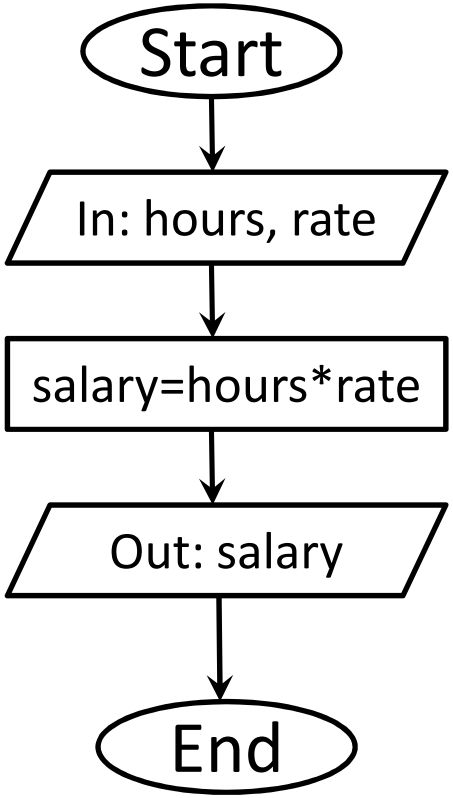
gross=net*(100%+VAT)

Output: gross

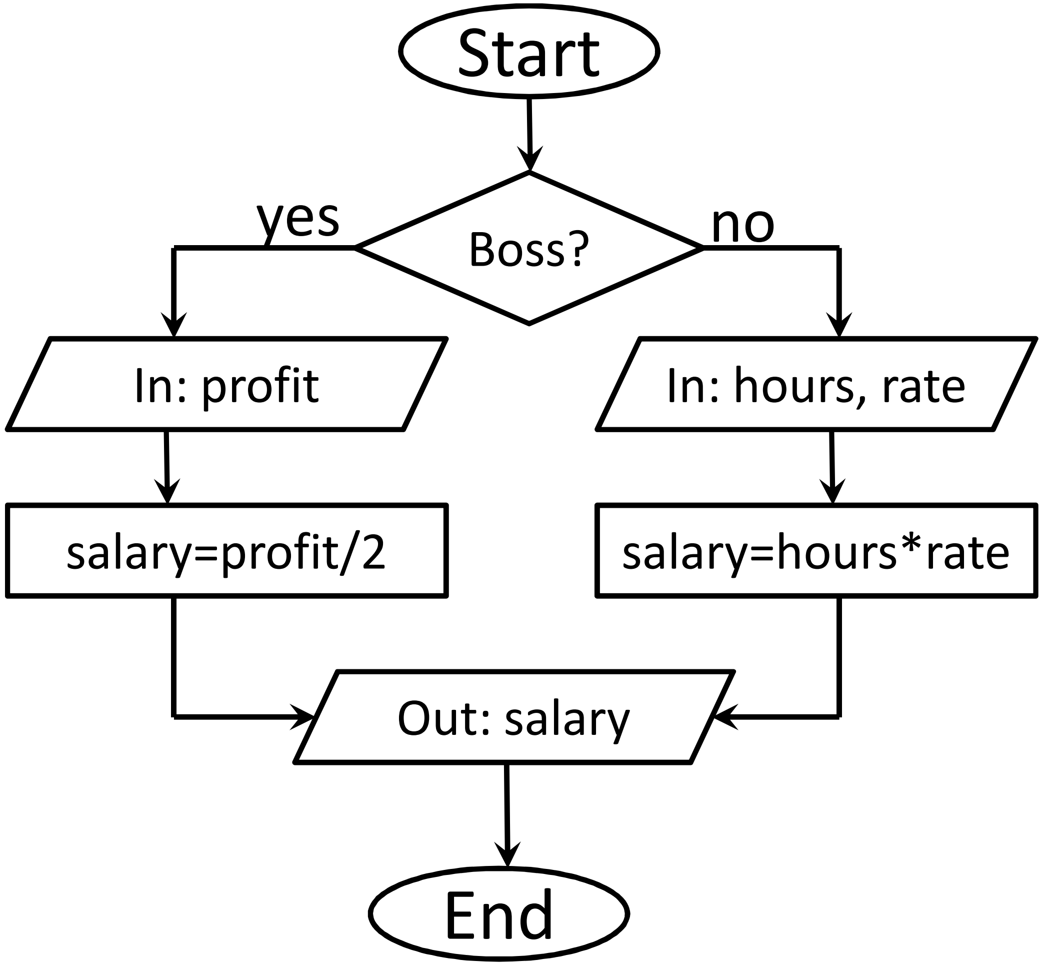
End

Extending algorithms

Original:

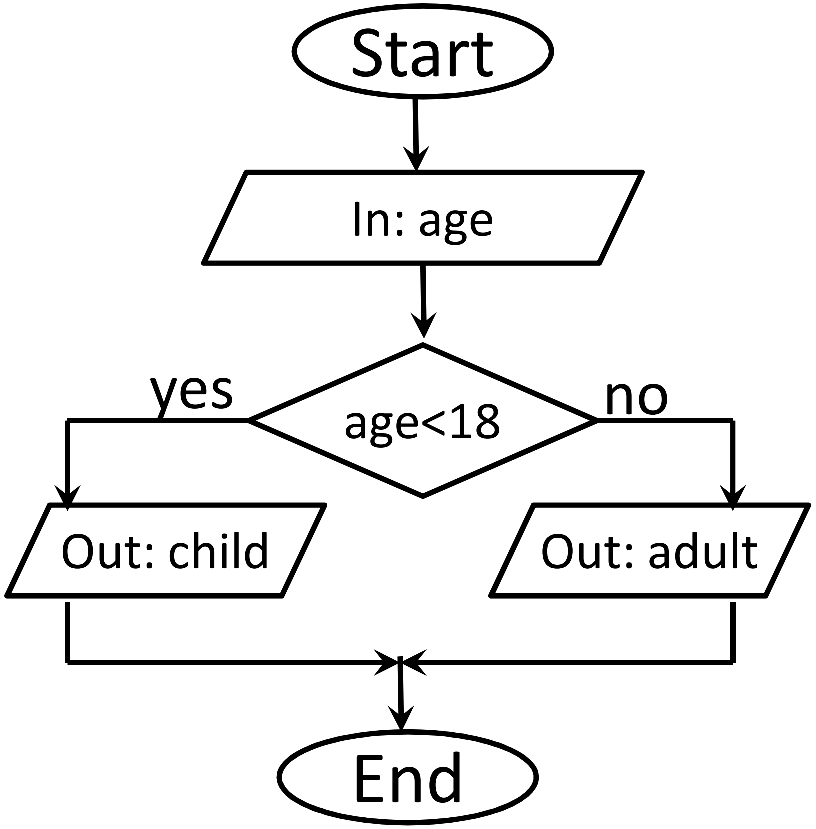


Extended:

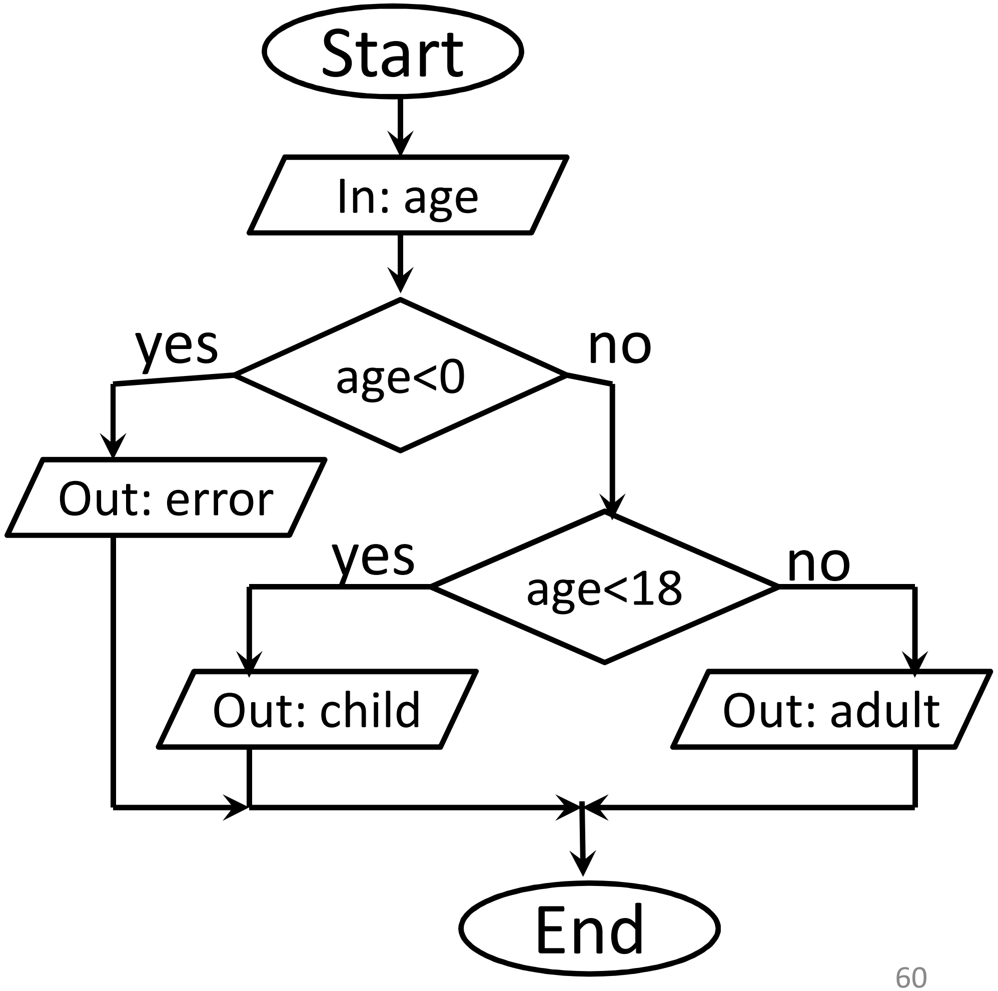


Foolproofing algorithms

Original:

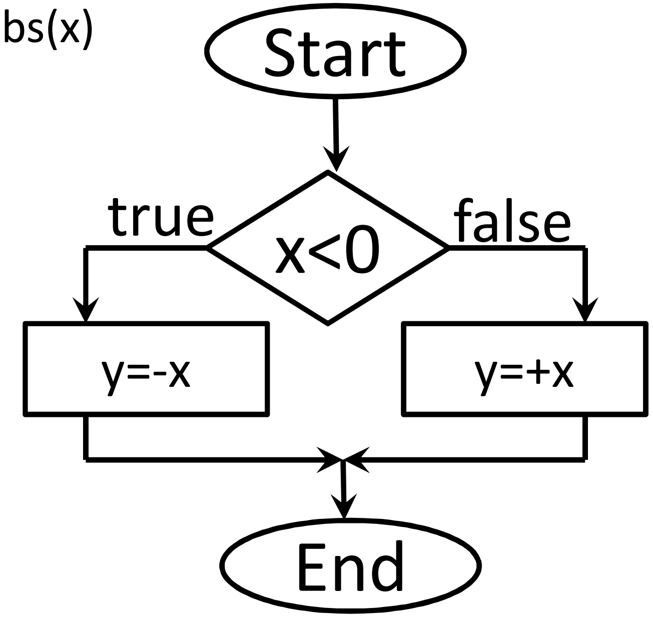


Foolproofed:

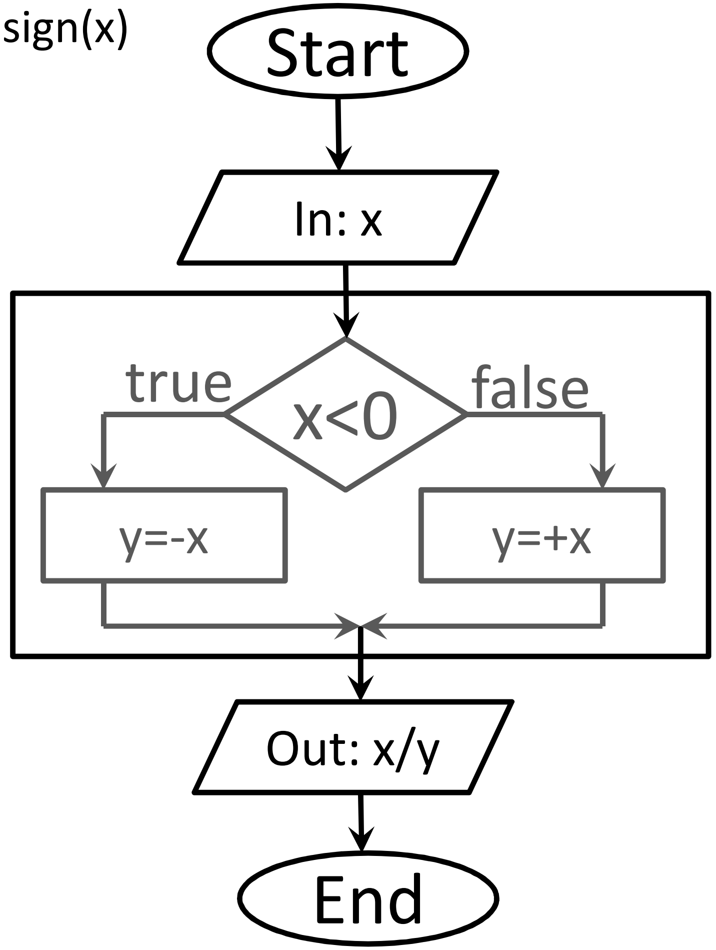


Embeddig algorithms

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.

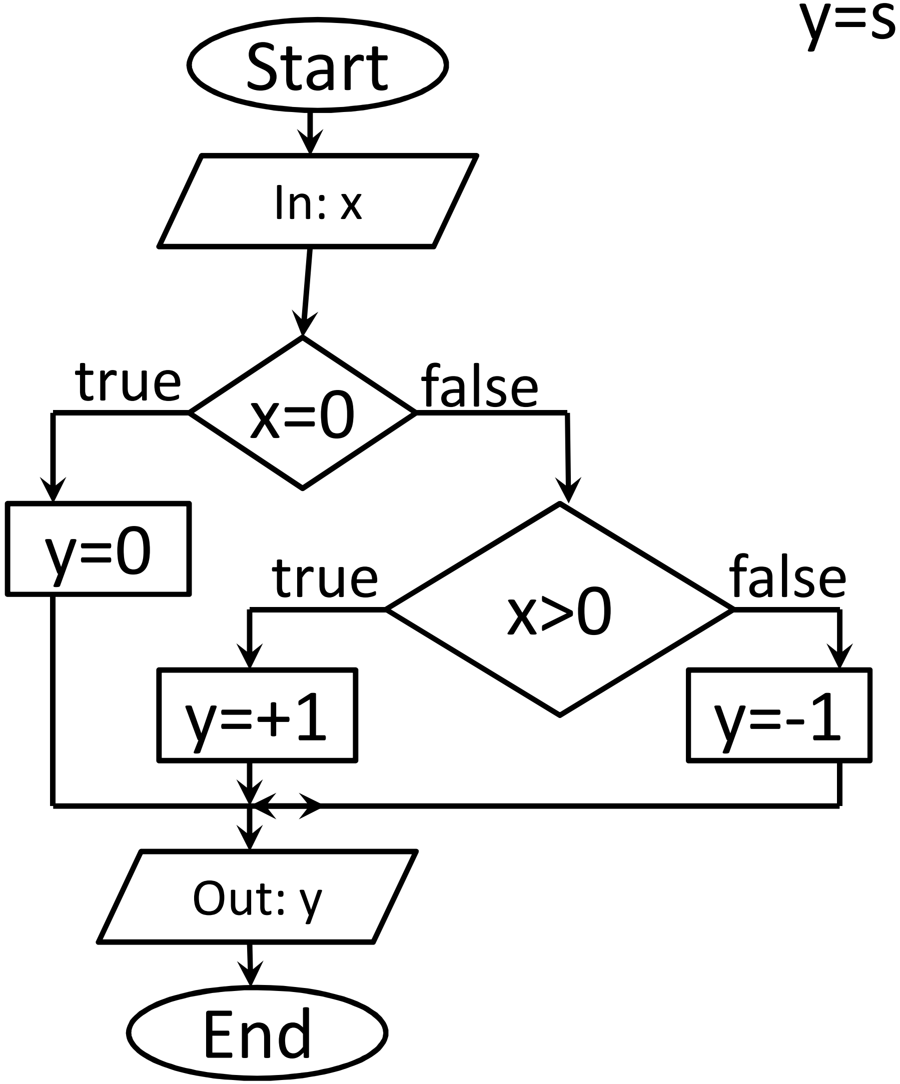
It means: for identical input data, they will produce identical results.

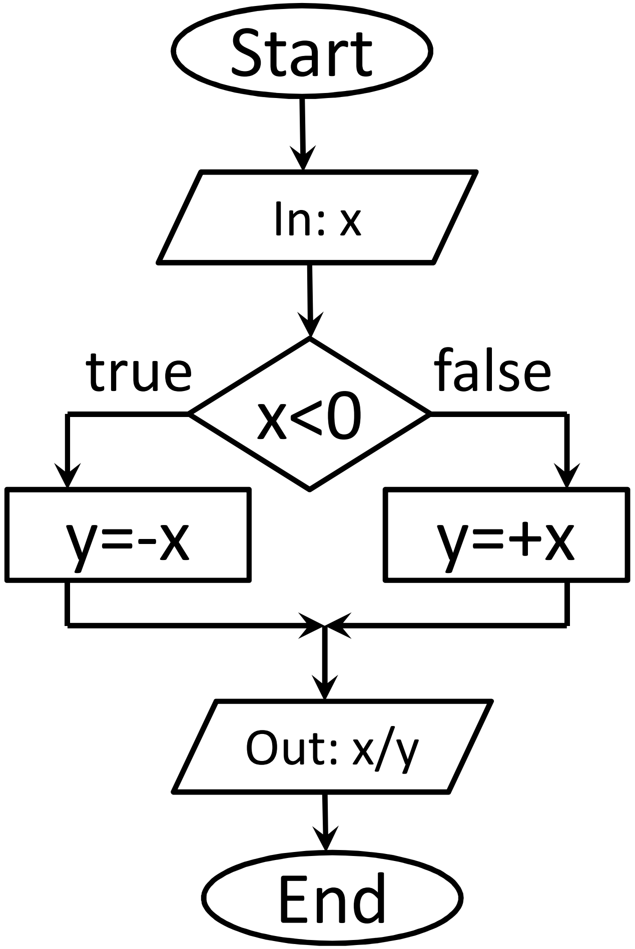
Sometimes there is serious reason to prefer one algorithm over the other, while sometimes there isn’t.

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 complet):

- If the list 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 frosen 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 allways works.

Wrong algorithms

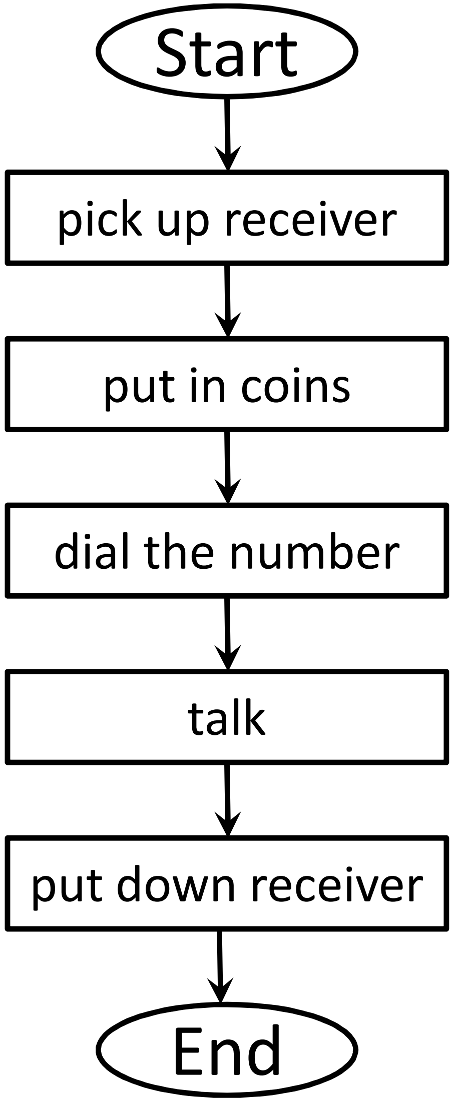
How to use a bus?

```
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.

Using public coin phone



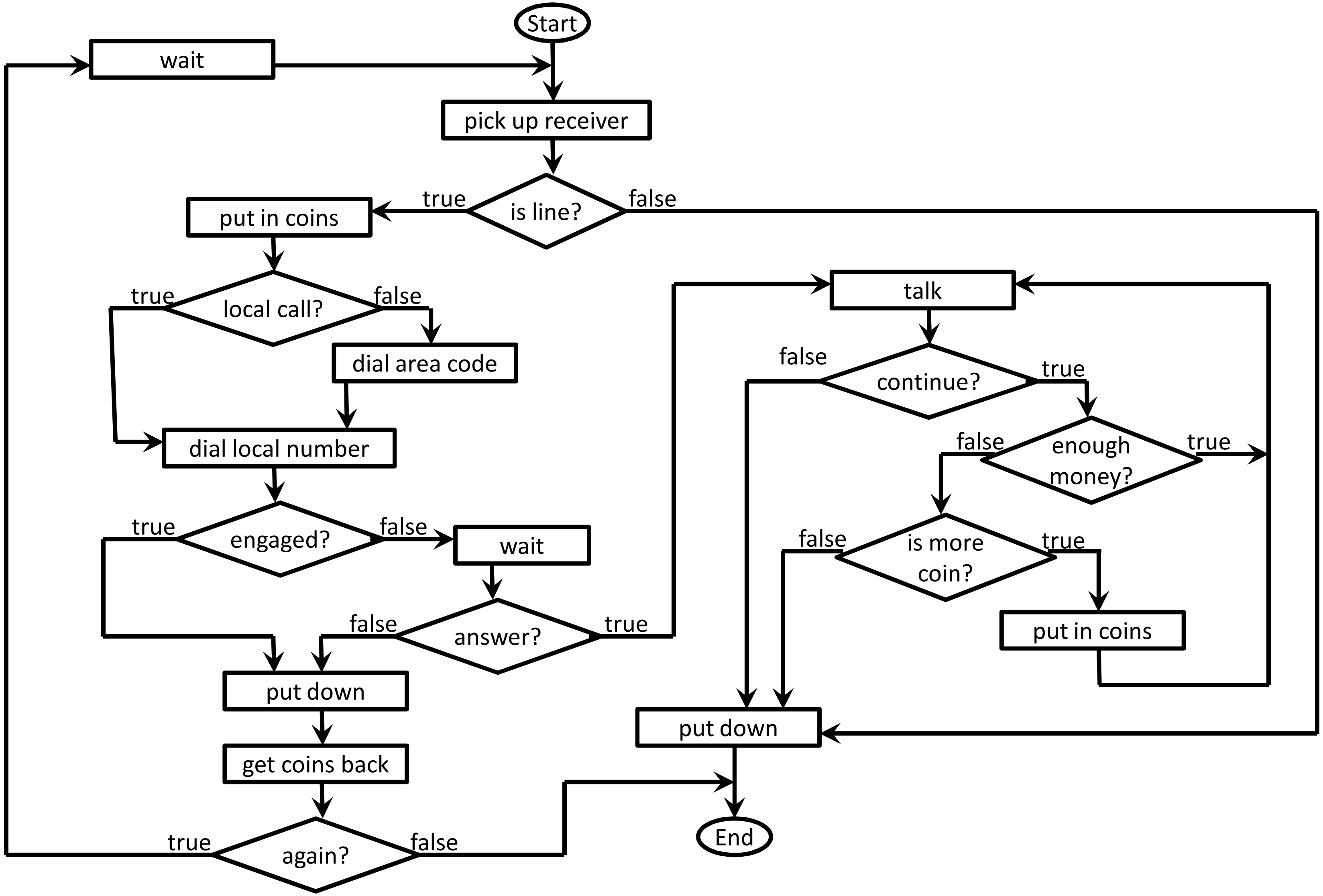
Problems:

- Not complete
- Ambiguous

Modification:

- Generalizing
- Extending
- Foolproofing
- Completing
- Avoiding misunderstans

Using public coin phone

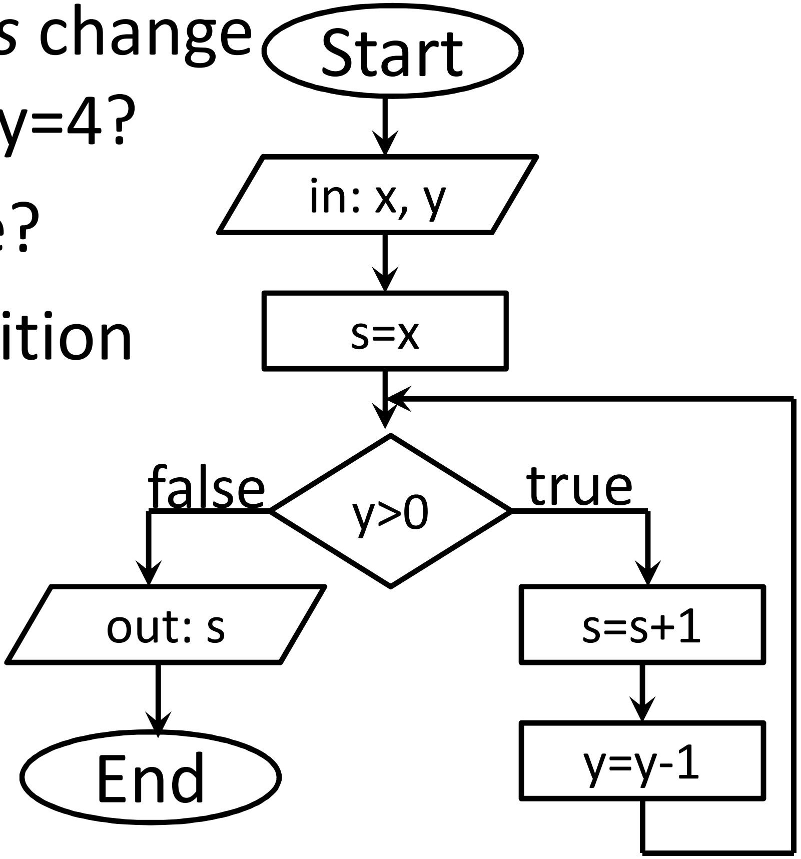


Exercises and examples

- Buying shoes
- Watching TV
- Using microwave oven
- Paying at cash-desk
- Making a call with mobile phone
- Going trough a road on foot
- Driving through a crossroads
- Leap year
- …

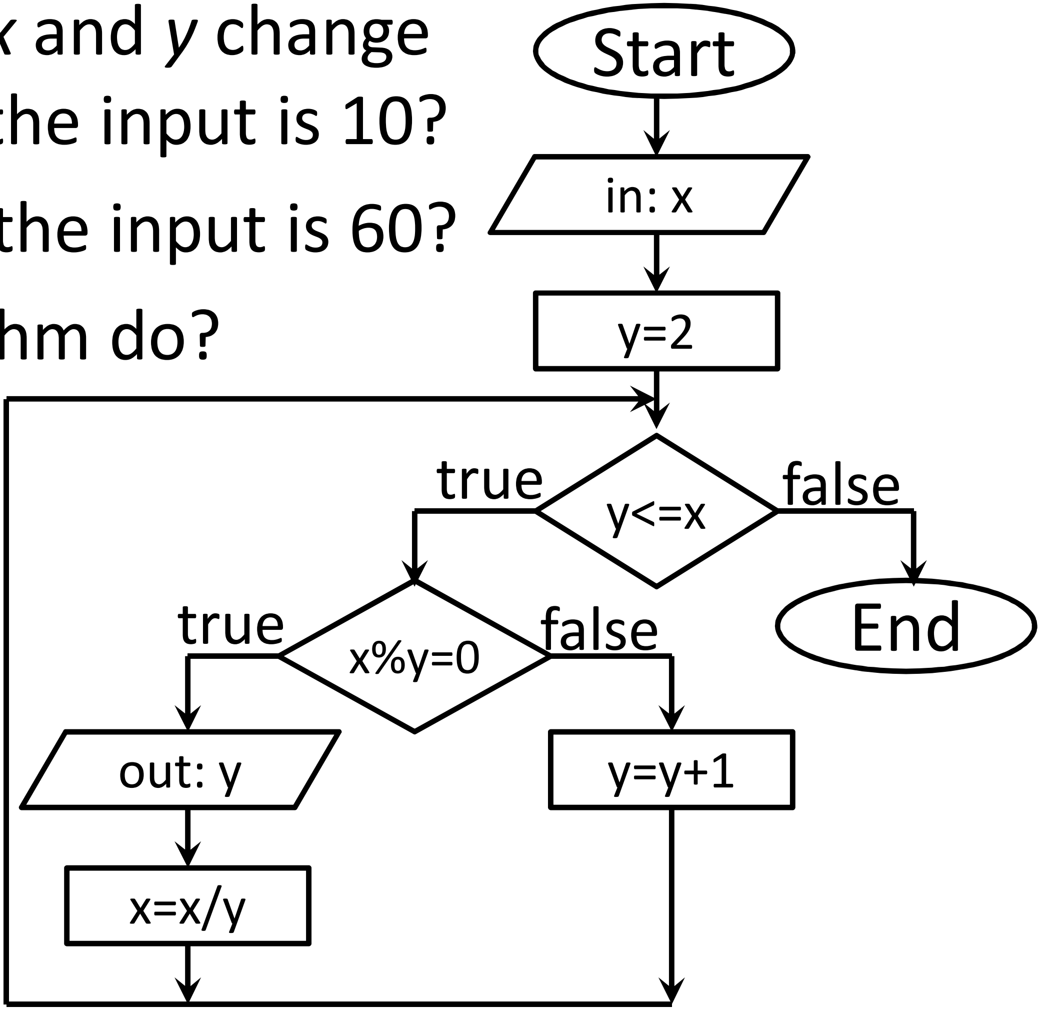
Exercises and examples

- 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*?



Exercises and examples

- 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?
- How can it faster?



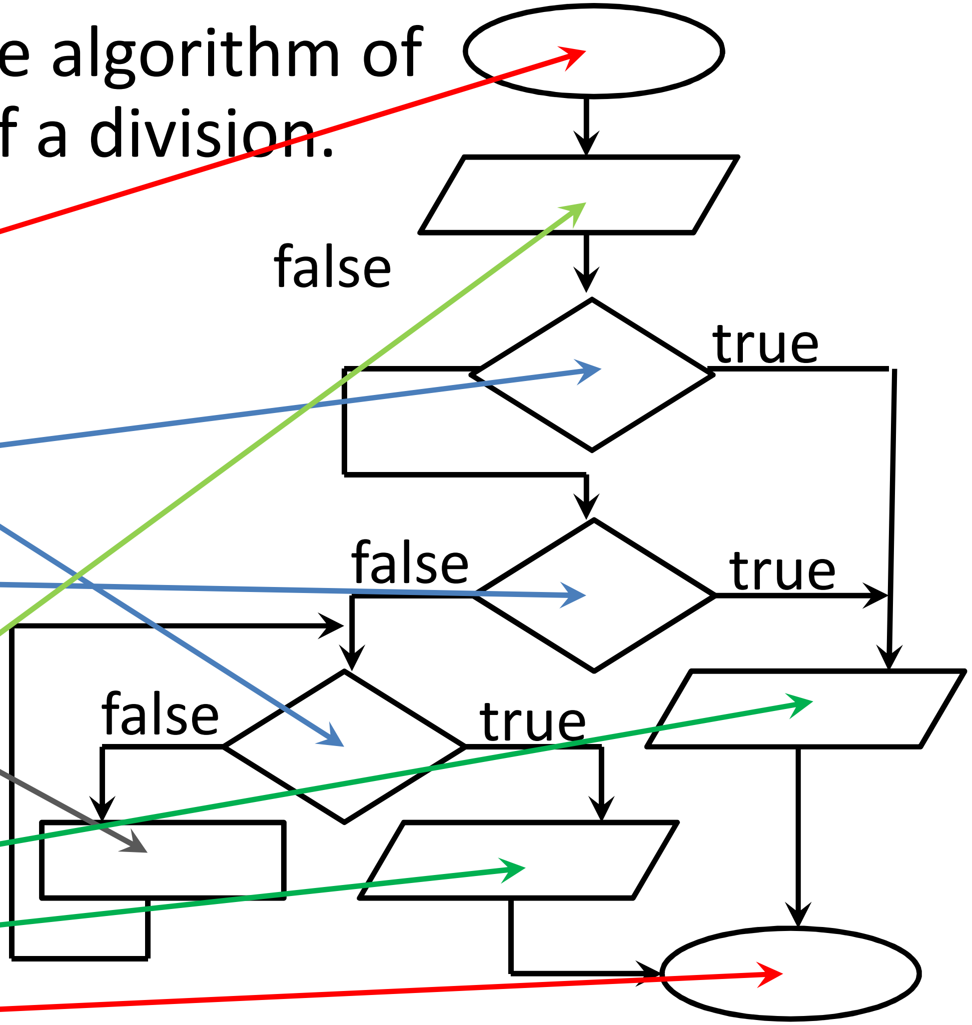
Exercises and examples

- Day of year
- Searching in ordered binary tree
- Raising to power
- Conversion of decimal number to binary
- Addition of binary numbers
- Solving second degree equation
- Calculating factorial
- …

Exercises and examples

This flowchart describes the algorithm of calculation of the remain of a division. Complete it.

- Start
- a<=b
- a<0
- b<=0
- a=a-b
- in: a, b
- out: error
- out: a
- End



Pseudocode

Sequence:

```
statement1
statement2
statement3
…
```

Selection:

```
if condition then
   statement1
else
   statement2
endif
…
```

Iteration:

```
while condition do
   statement1
enddo
…
```

Exercises and examples

```
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?

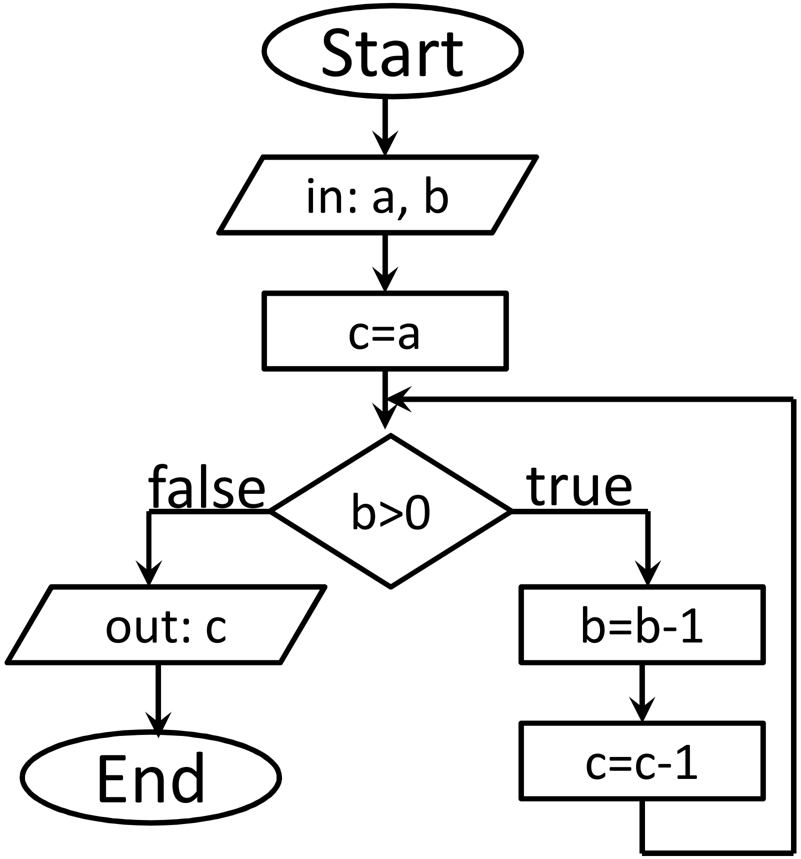
- What does this algorithm do?

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

Exercises and examples

```
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?



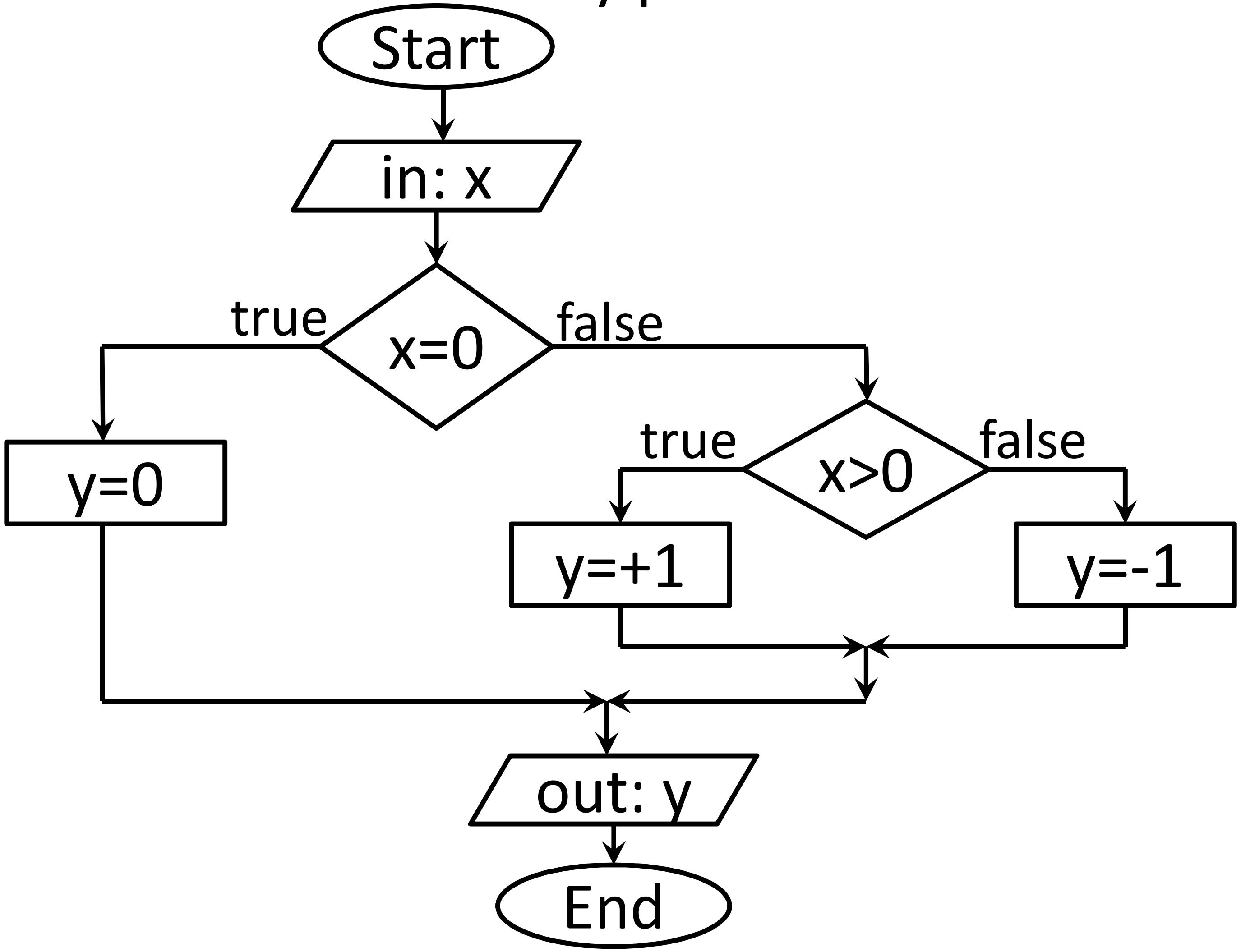
Exercises and examples

```
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?

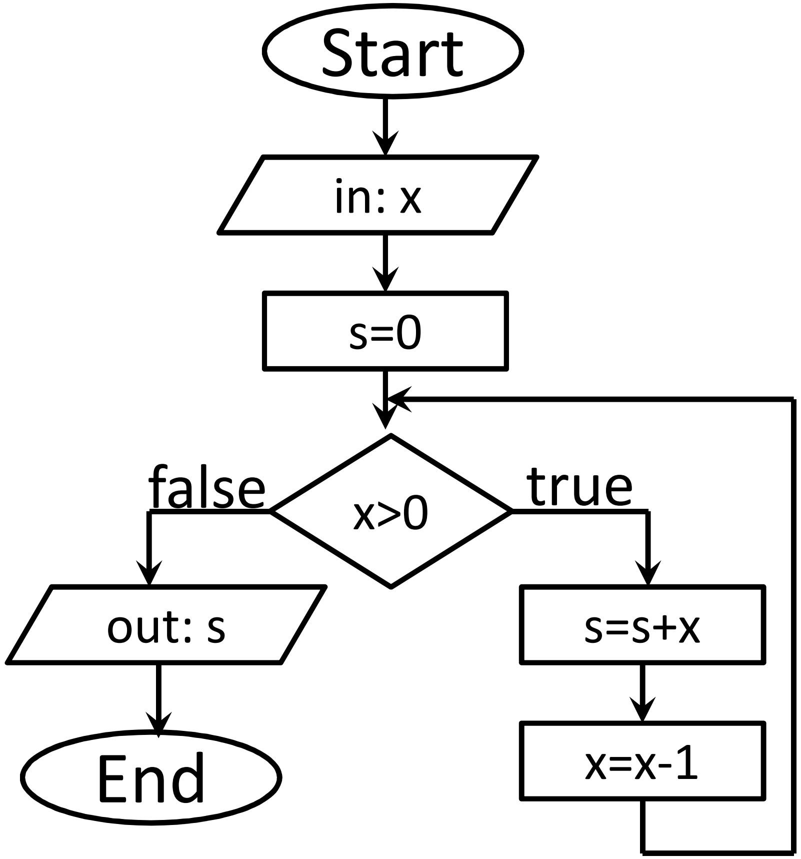
Exercises and examples

- Describe this flowchart by pseudocode.



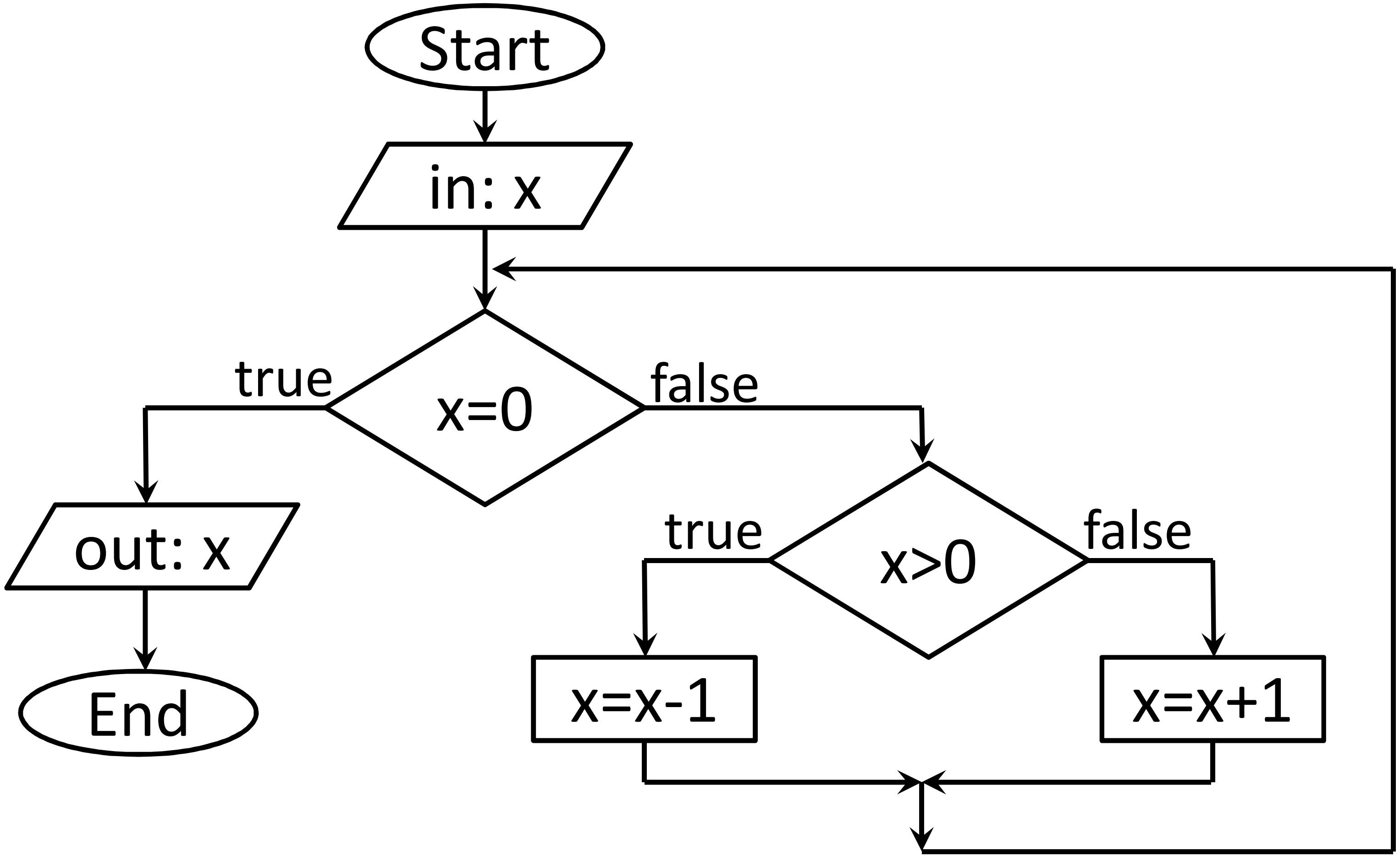
Exercises and examples

- Describe this flowchart by pseudocode.



Exercises and examples

- Describe this flowchart by pseudocode!



Exercises and examples

Verbal represented algorithm:

1. Give a number.
2. Check that it is larger then one or not.
3. If it is larger, substract two and continue with Step 2.
4. Otherwise check it zero or not.
5. If it is zero, write ‘even’.
6. Else write ‘odd’.

Write this algorithm in pseudocode.

Exercises and examples

- Absolute value
- Raising to power
- Calculating factorial
- Prime or not
- Prime factorization
- Solution of first degree equation
- Sum of numbers from 10 to 20

Exercises and examples

- Average of an array
- Minimum/maximum search
- Finding a value in (ordered) list
- Replacement of two values
- Selection sort
- Insertion sort
- Bubble sort

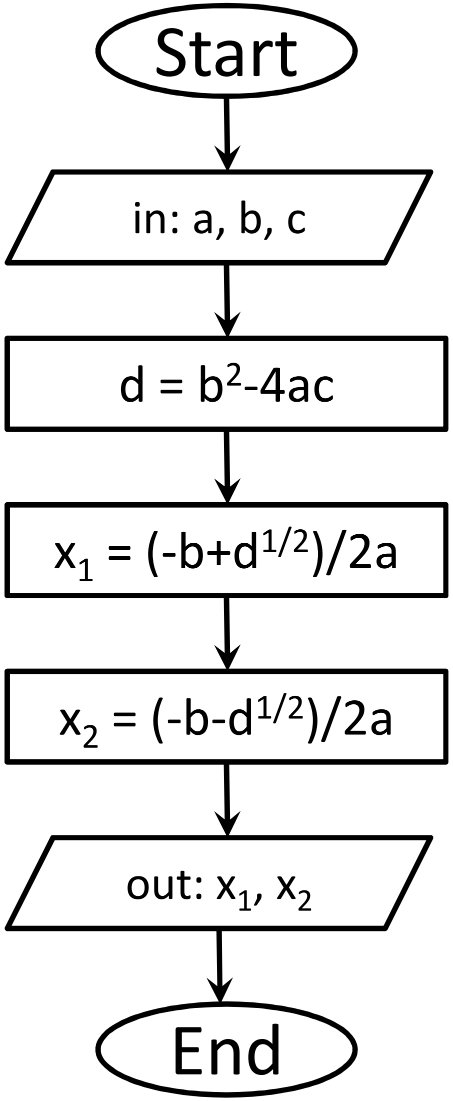
Testing strategy development

Example of testing strategy

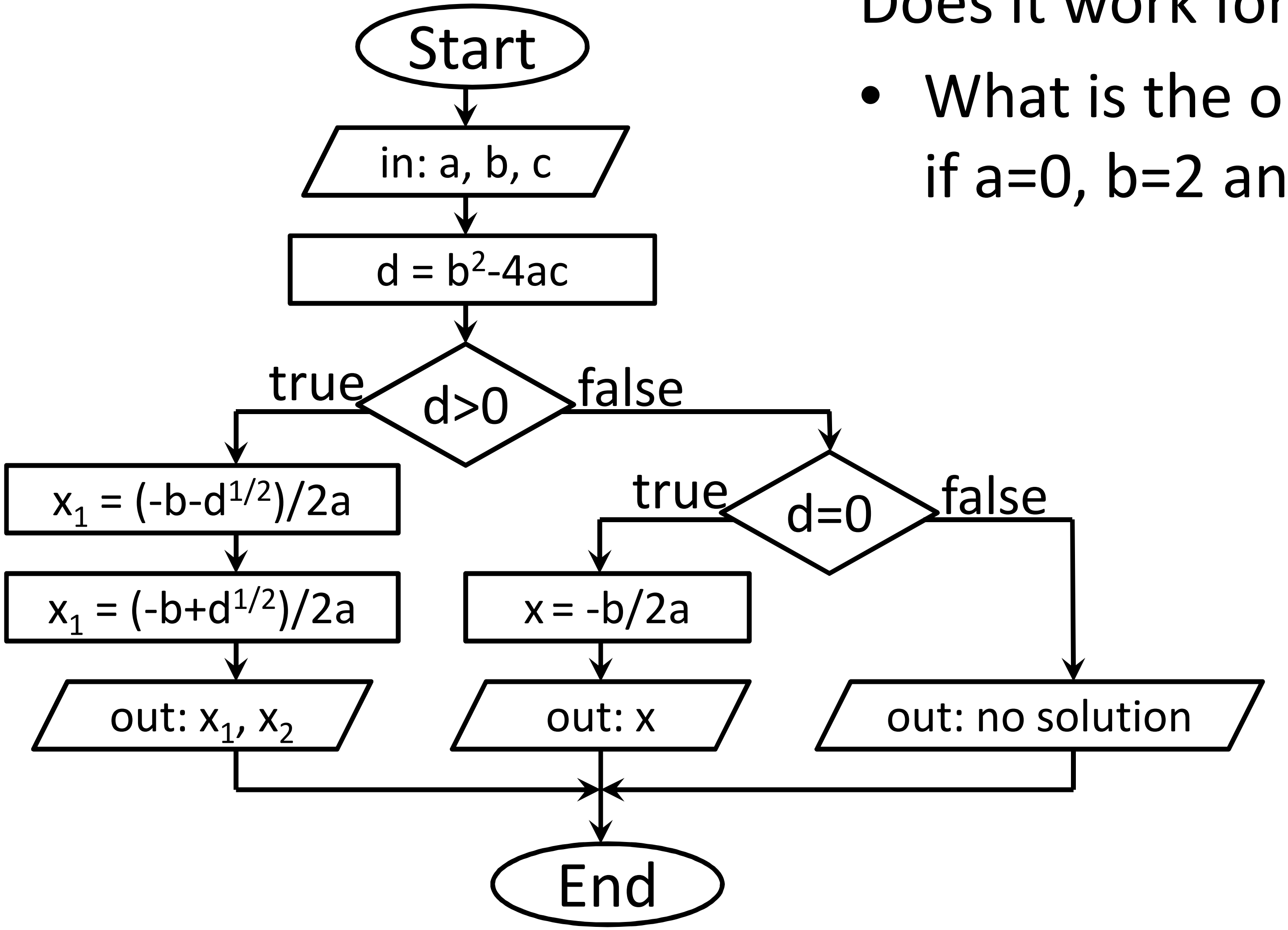
- Solving second degree equation
- General form: $\mathbf{a}x^2 + \mathbf{b}x + \mathbf{c} = 0$
- Input parameters: a, b, c
- Solution: $x_{1,2} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

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?



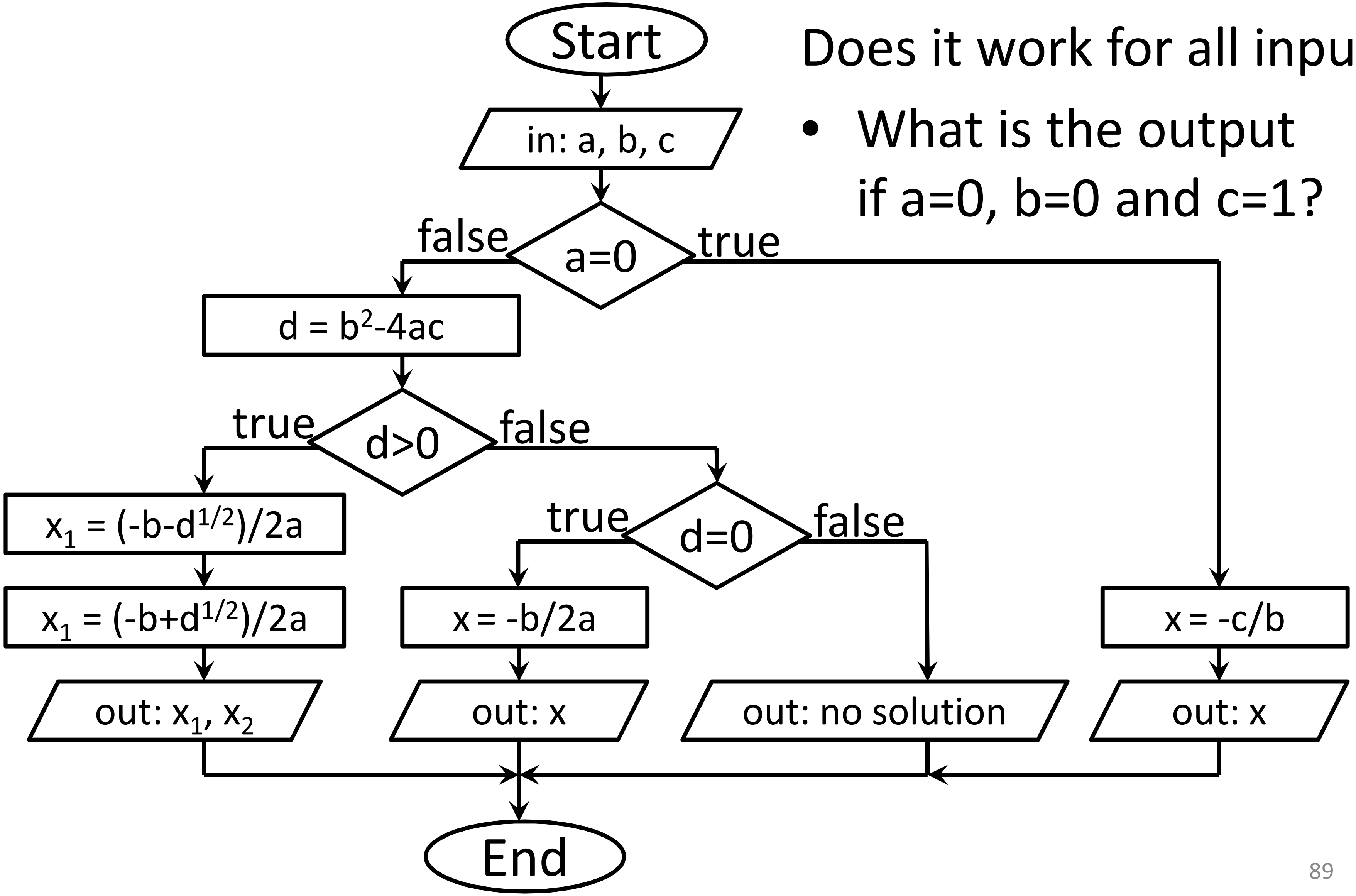
Example of testing strategy



Does it work for all input?

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

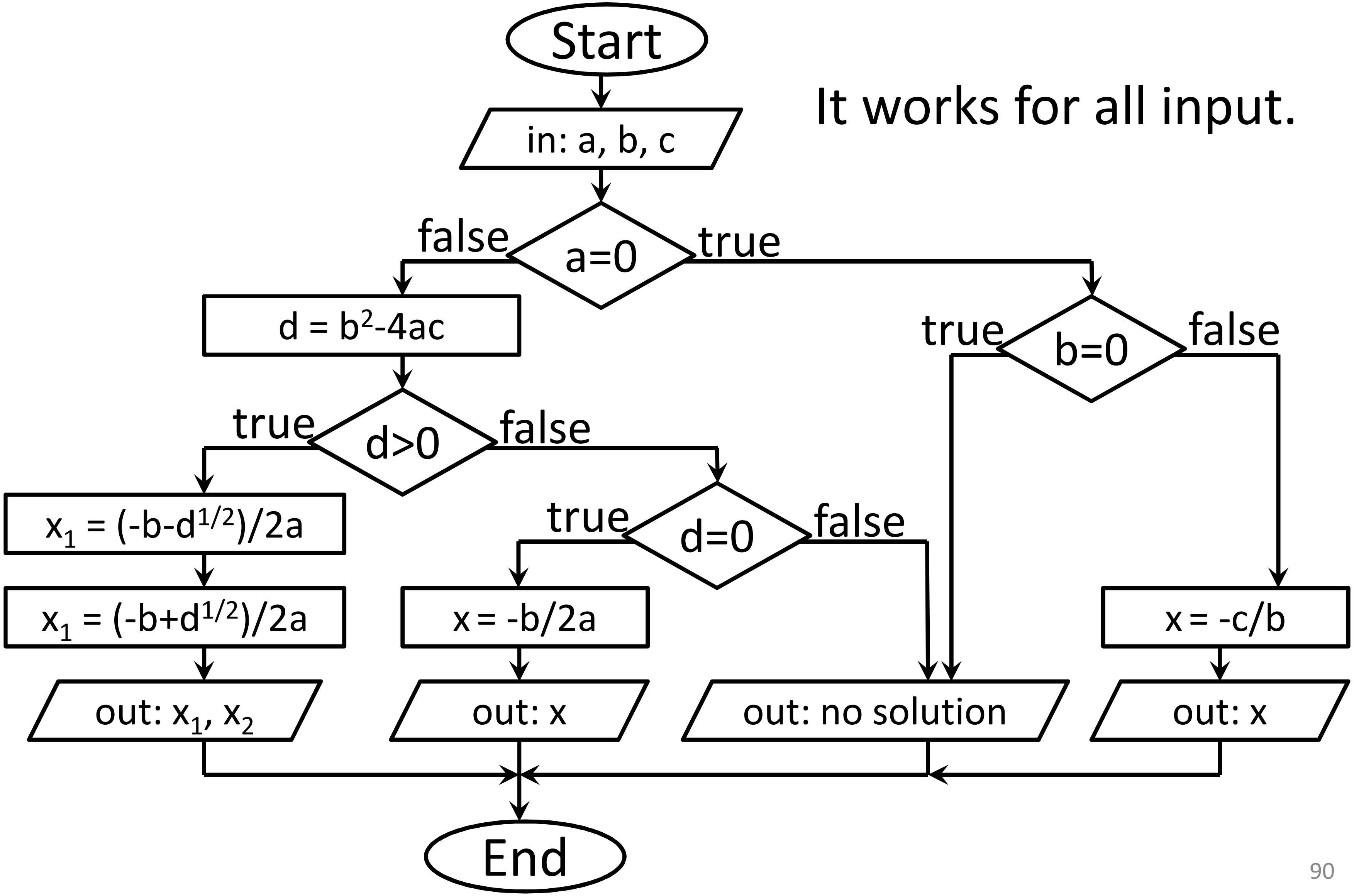
Example of testing strategy



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:

It works for all input.

To reach this state we have had to test the algorithm with more different input combination and then we have had to modify the algorithm.

We have used testing strategy.

```
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
```

The used testing strategy

a	b	c	reason	OK
3	7	2	general case (not zero, d>0)	✓
0	2	3	**a** is zero (first degree)	✓
2	0	5	**b** is zero (x^2=-c/a)	✓
1	2	0	**c** is zero (x[ax+b]=0)	✓
0	0	1	more zeros (not equation)	✓
3	1	9	**d**<0 (no solution)	✓
2	4	2	**d**=0 (only one solution)	✓
-2	-3	-9	negative inputs	✓
2.3	4.2	0.83	not integer values	✓
0.00001	1000000	1	extreme small/large values	✓

Program coding

Creating source code
in real programming language

Syntax and semantics

Syntax: Formal rules of the program text.

Semantics: Does it describe the desired algorithm?

Example (absolute value):

```
input a
if a>0 then
   a=-1*a
enidf
output a
```

Semantic error

Syntax error

Syntax of programing languages

Fortran:

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

Pascal:

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

C:

```
long fakt(long n){
  if (n<=1) return 1;
  else return n*fakt(n-1);
  }
```

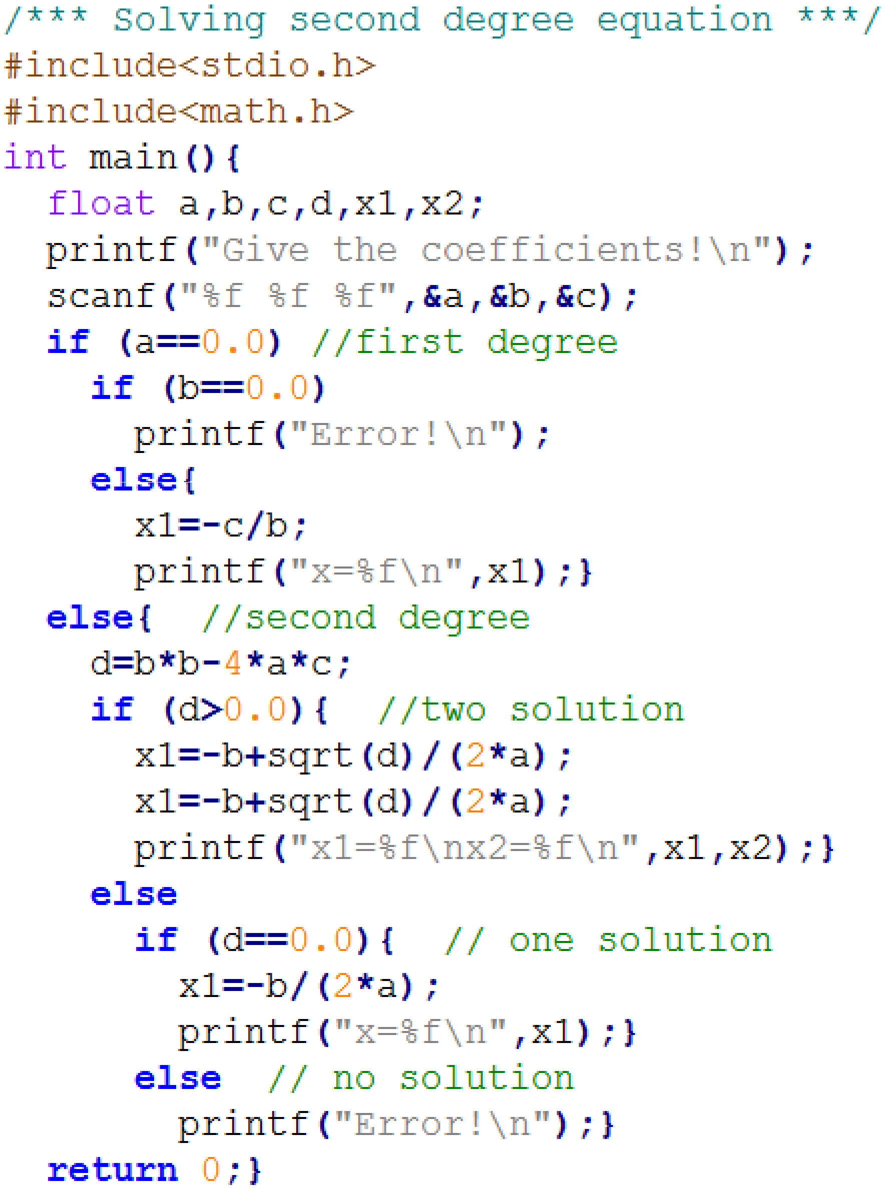
Units and elements of the code

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

Complexity increase

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

C programming language



```c
/*** Solving second degree equation ***/
#include<stdio.h>
#include<math.h>
int main(){
  float a,b,c,d,x1,x2;
  printf("Give the coefficients!\n");
  scanf("%f %f %f",&a,&b,&c);
  if (a==0.0) //first degree
    if (b==0.0)
      printf("Error!\n");
    else{
      x1=-c/b;
      printf("x=%f\n",x1);}
  else{  //second degree
    d=b*b-4*a*c;
    if (d>0.0){  //two solution
      x1=-b+sqrt(d)/(2*a);
      x1=-b+sqrt(d)/(2*a);
      printf("x1=%f\nx2=%f\n",x1,x2);}
    else
      if (d==0.0){  // one solution
        x1=-b/(2*a);
        printf("x=%f\n",x1);}
      else  // no solution
        printf("Error!\n");}
  return 0;}
```

Documentation & Maintenance

Documentation

Complete the documentation:

- Always document everything during the program development.
- What is the solution method?
- What are the solved subproblems?
- What are the necessary inputs and the output?
- How does the implemented algorithm work?
- What are the meaning of the variables? (comments)
- How to use the program? (user manual)
- What are the discovered errors and their solutions.

Maintenance

Maintenance the program:

- If the users need, correct, extend and update you application.
- Make documentation about all changes.

This is the END!

☺