Problem analysis and resolution methodologies

Introduction to Computer Programming
Bachelor in Data Science

Roberto Guidi

roberto.guidi@supsi.ch

Fall 2021

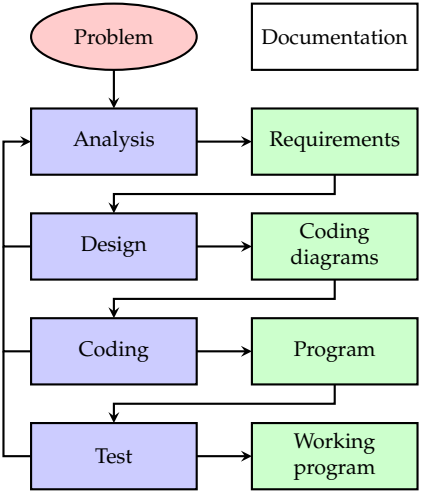
SUPSI University of Applied Sciences and Arts of Southern Switzerland

Content created in collaboration with: L. Grossi, F. Landoni, G. Landolfi, T. Leidi, A.E. Rizzoli, S. Pedrazzini

Problem solving

How should we approach problem resolution in computer science?

1. Problem formulation.
2. Problem analysis.
3. Design.
4. Implementation.
5. Test.
6. Documentation.



Problem solving: problem formulation

The first step necessary before implementing a solution is to understand exactly the problem.

It is not useful to solve the wrong problem.

In practice, most of the times, it is more difficult to understand exactly the nature of the problem than to find a solution.

Problem solving: problem analysis

The purpose of the problem analysis is to clarify, detail and document the function, the services and the features that a software system or a program must offer in order to solve a certain problem in the context in which it will operate.

We must identify:

- data inputs (input).
- desired results (output).
- The techniques to apply to obtain the required results.

The information gathered during the analysis step (requirements specification) is the starting point for the design of a software product and the entire implementation process, validation and maintenance.

Problem solving: design

Based on the requirements specification produced in the analysis step, the design step defines how these requirements will be satisfied, going into details about the structure that the software system to implement must have.

Algorithms to solve the problem are designed by researching suitable formulae and mathematical relationships. the algorithms are described using diagrams, flow charts and/or pseudo-code.

Complex calculation procedures are transformed in a sequence of basic operations (divide et impera).

This step allows to develop a document in which the general structure (high level architecture) and the single component's (modules) features are described.

Problem solving: design testing

Before passing on with the concrete implementation, the logical sequence of operation must be verified theoretically.

The overall structure and single module features are also checked at this point.

Problem solving: coding or implementation

The implementation, also known as development or coding of the software product, is a stage of software creation, that gives concrete form to the software solution through programming, which is the act of writing programs.

The diagrams, flow charts and/or pseudo-code made during the desigh step are translated usign a programming language.

The result is an executable code that can be run on a computer.

Problem solving: implementation testing

This process is used to spot possible problems about the accuracy, the completness and reliability of the software components being developed.

It consists in executing the software to test, alone or in combination with other software tools, in order to evaluate if the software behaviour reflects the requirements identified during the analysis step.

Special cases are identified and verified in order to discover possible bugs.

Problem solving: documentation

Each of planned steps must be documented.

The documentation about the product usage and the description of the methodology used is prepared.

The are various types of documentation:

- development documentation,
- documentation for the customer,
- usage documentation.

Algorithm

An algorithm is a procedure to solve a certain problem through a finite number of basic steps.

The algorithms are formulated using a programming language, whose execution solves the posed problem. The algorithm becomes a program.

The task of a programmer is to produce algorithms (understanding the steps that allows to solve a problem) and to code them into programs (that is making them understandable for a computer).

Algorithms properties

The instructions that form an algorithms must have the following characteristics:

- They must be free of ambiguity.
- Every instruction should terminate in a finite time.
- They should find the solution in a finite number of steps.
- They require a finite amount of input data.
- The execution should produce a unique result.

An algorithm must be formulated so that one executing it gives the same meaning to the instructions as the designer (problem solver).

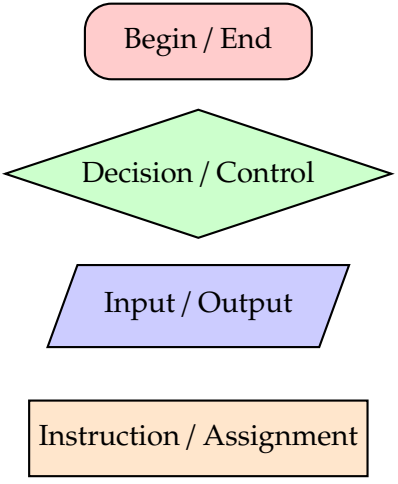
Flow chart

Flow charts make available a set of symbols that helps in the graphical description of the control flow and instructions of an algorithm.

Flow charts can be developed to display different levels of detail.

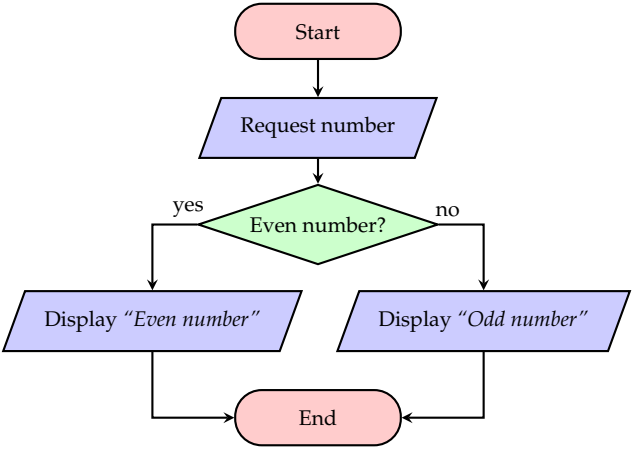
Once used to the analysis of a problems and the language notation, the diagrams are used in particular to study the most complex scenarios.

Flow charts: used symbols



Selection instruction

Algorithm capable of requesting a number and determine if the number is even or odd.

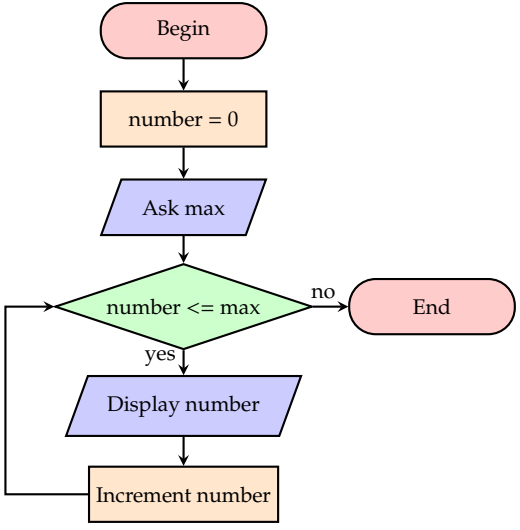


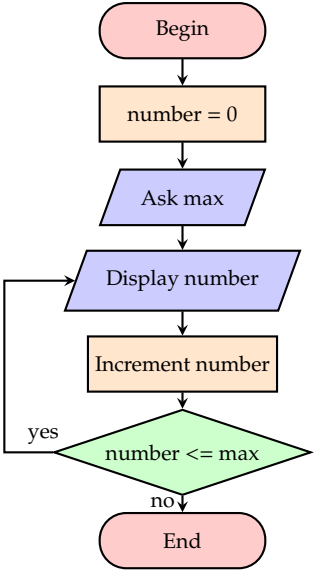
Input:
3
Output:
Odd number

Input:
16
Output:
Even number

Repetitive instructions (loops)

Algorithm that shows numbers between 0 and a value of user choice.





Pseudo-code

Pseudo-code is a language used, as an alternative to the classic flow chart, to represent algorithms.

The writing of pseudo-code may precede the coding of the program with a programming language.

There is not a standard pseudo language; every developer can use its own version.

Algorithm 0 Even or odd number?

1: Display '*Insert a number*'
2: Read the user inserted number
3: Calculate the remainder of the division by 2
4: If the remainder is 0 display '*Even number*' otherwise display '*Odd number*'

Algorithm 1: subtraction of two integer numbers

Problem:

Develop an algorithm able to calculate the difference between two numbers without calculating the result of the arithmetic operation.

Assumptions:

- It is not possible to use the '−' sign.
- We know the preceding number of each integer number.
- It is possible to compare the number with zero.

Algorithm 1: subtraction of two integer numbers

Pseudo-code:

Algorithm 1 Subtraction of two integer numbers

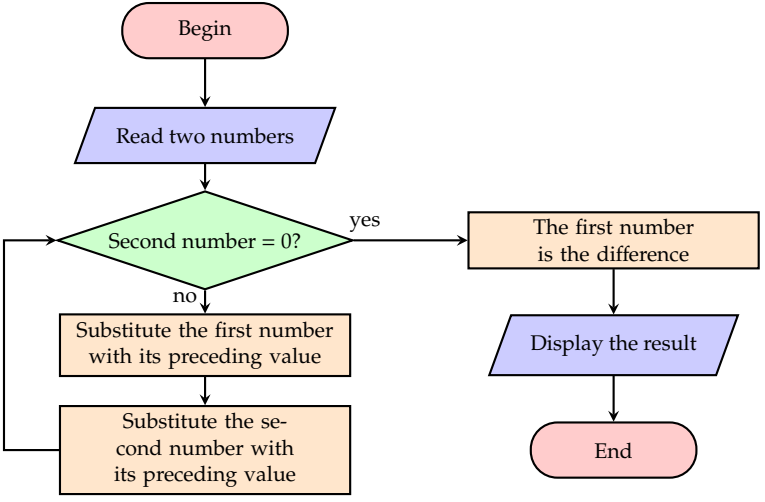
1: Read two numbers
2: the second number is equal to 0? If yes then execute instruction 6, otherwise go on with instruction 3
3: Substitute the first integer number with its preceding value
4: Substitute the the second number with its preceding value
5: Start over from instruction 2
6: The first number represent the difference

Important: there are many different algorithms that solves the same problem.

Question: Do the algorithm always work?

Algorithm 1: subtraction of two integer numbers

Flow chart:



Algorithm 2: square root calculation

Problem:

Develop an algorithm that is able to calculate the square root of a positive real number greater than 0.

The square root of a number a is the number b such that its square is a, that is such that $b^2 = a$.

Geometric interpretation:

- Given a positive number a, its square root can be seen as the side b of a square that has an area equal to a.
- In other words: we must build a square with side b that has an area a.

Algorithm 2: square root calculation

Possible solution: Babylonian method (about 1700 b.C.)

The idea is that of using rectangles with the same area a as the square to obtain, through sequential approximations, exactly the square that we are searching for.

Steps to follow:

1. Build a rectangle of area a having a width equal to $x_0 = a$ and a length equal to $y_0 = 1$.
2. Get close to the square by replacing x_n with the average between x_n and a/x_n (average of the two rectangle sides).
3. Calculate the new value for $y_n = a/x_n$.
4. Repete the procedure as long as the result is $(x_n = y_n)$ or the required precision is obtained.

Algorithm 2: square root calculation

The Babylonian method is defined as an iterative method and can be generalised as follows:

Width of the rectangle:

$$x_{n+1} = \frac{x_n + y_n}{2}$$

Length of the rectangle:

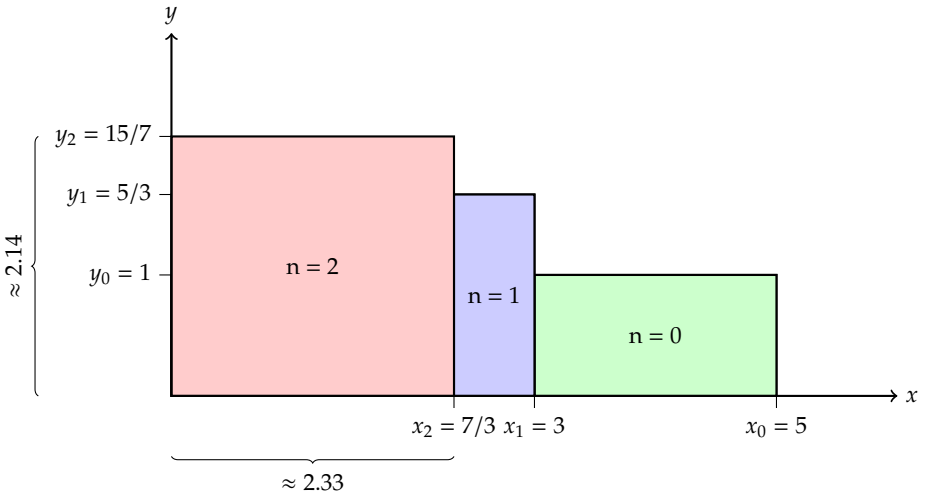
$$y_{n+1} = \frac{a}{x_{n+1}}$$

By combining the two formulae we obtain the following formulae to calculate the rectangle width:

$$x_{n+1} = \frac{1}{2} * \left(x_n + \frac{a}{x_n}\right)$$

Algorithm 2: square root calculation

Graphical representation of the algorithm for $a = 5$ $\left(\sqrt{5} \approx 2.236\right)$



Algorithm 2: square root calculation

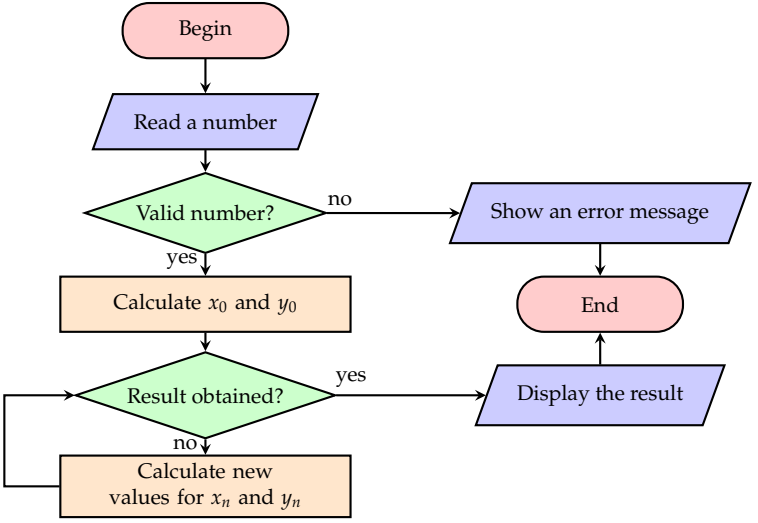
Pseudo-code:

Algorithm 2 Square root calculation

1: Read a number
2: Check if the number inserted is valid. If not, display an error message and finish.
3: Calculate the initial values for x_0 and y_0
4: Check if the result has been obtained. If yes, go on with instruction 7. If not, go on with instruction 5
5: Calculate the new values for x_n and for y_n
6: Start over from instruction 4
7: Display the result

Algorithm 2: square root calculation

Flow chart:



Algorithm 3: automatic drinks machine

Problem:

Develop an algorithm able to simulate an automatic drinks machine.

Assumptions:

Desired features:

1. Choice of the drink.
2. Insert coins.
3. Return the change.
4. Provide the drink.

Algorithm 3: automatic drinks machine

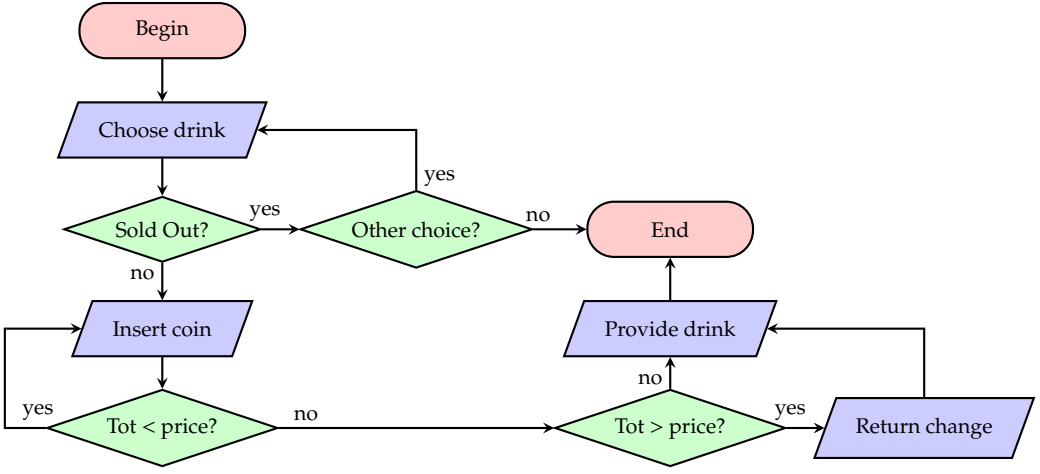
Pseudo-code:

Algorithm 3 Automatic drinks machine

1: Request the drink choice
2: Check if the drink is available. If not, go on with instruction 8, otherwise go on with instruction 3
3: Ask for coins
4: Check if the total amount inserted is enough. If yes, go on with instruction 5, otherwise go on with instruction 3
5: Check if there is a change to give back. If yes, go on with instruction 6, otherwise execute instruction 7
6: Calculate and return the change
7: Provide the selected drink and finish
8: Request if another drink is desired. If yes, go back to instruction 1, otherwise finish.

Algorithm 3: automatic drinks machine

Flow chart:



Summary

- Problem solving
- Algorithms
- Flow charts
- Pseudo-code
- Algorithm examples
  - Subtraction of two integer numbers
  - Square root calculation
  - Automatic drinks machine