Unit 2. Programming Fundamentals

2.1 Computer Solving Problem Phases
2.2 What is an algorithm?
2.3 Algorithms representations
2.4 Algorithms design method
2.5 Algorithms elements

Computer Problem Solving

DOCUMENTATION

- Algorithm development phase
  - Analyze: Understand the problem
  - Design an algorithm
  - Test the algorithm
- Implementation phase
  - Code: translate into a programming language
  - Test the program
- Maintenance phase
  - Maintain: Adapt to new requirements

What is an algorithm?

- R.A.E.: "*And ordered and finite set of operations which allows finding the solution of a problem*"
- We use different algorithms everyday:
  - Recipies
  - D.I.Y. Furniture
  - Explain to somebody how to go somewhere
  - Drive a car

What is an algorithm?

- An **algorithm** is set of instructions for solving a problem or subproblem in a finite amount of time using a finite amount of data

- Properties of an Algorithm:
  - It must be precise and unambiguous
  - It must give the correct solution in all cases
  - It must eventually end

What is an algorithm?

- Is this Spanish Omelet recipe an algorithm?

Cut up the potatoes into cubes a half centimeter in diameter. Fry them using plenty of oil on a low flame. Add onions and fry until transparent. Put the mixture into a separate bowl and set aside to cool. Beat the eggs in a bowl, add some salt and mix well with the potatoes and the onions. Put the mixture in the frying pan again with some more oil. Wait until it sets, turn it upside down and let it set again over a low flame, making sure not to burn it.

Algorithms representations

- Algorithms can be described using
  - Natural language
  - Flowcharts
  - Pseudocode
  - Programming Languages

Algorithms representations

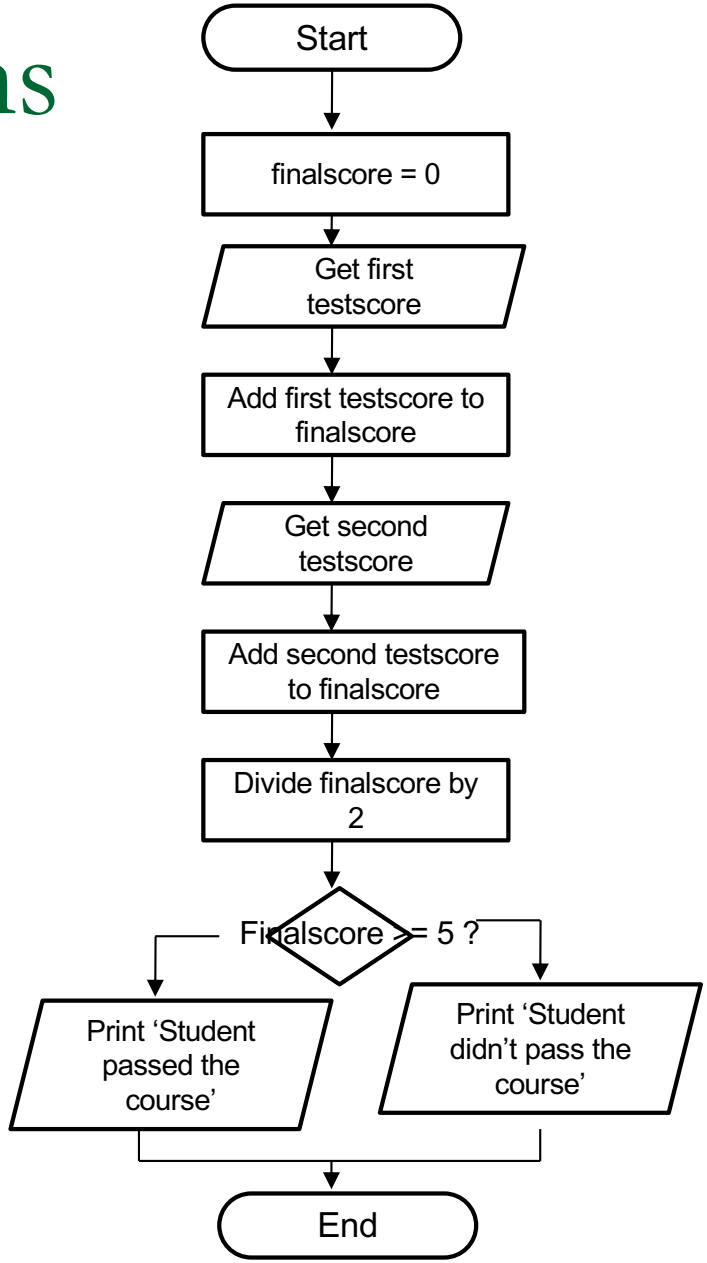
- **Natural Language**:

  - Simple
  - Too verbose
  - Too "context-sensitive"- relies on experience of reader
  - Error prone

Add the first score to the second one and divide the total by two. The student passes the exam when the result is greater than 5

Algorithms representations

- **Flowchart**: combines symbols and flowlines, to show figuratively the operations of an algorithm
  - More close to a computer representation
  - Algorithms can be described using a few symbols
  - Non intuitive symbols
  - Text is natural language
  - Large algorithms can be difficult to represent



Algorithms representations

- **Pseudocode**: Natural language constructs modeled to look like statements available in many programming languages
  - Comprenhensible as natural language but unamiguous
  - Independent from the computer

```
Write 'Introduce the first score'
Read score1
Write 'Introduce the second score'
Read score2
sum = score1 + score2
result = sum / 2
If result is greater than 5
    Write "Student passed the course"
else
    Write "Student didn't pass the
        course"
End if
```

Algorithms representations

- **Programming language**:
  a set of pre-defined words that can be combined into statements that a computer can understand and execute

  - Comprenhensible both to human and to computers
  - Algorithms described in diffferent lenguages will look different

```
Score1=input(“Introduce the first score?\n”);
Score2=input(“Introduce the first score?\n”);
sum = score1 + score2;
result = sum / 2;
if (result>5)
    Sprintf( ‘Student passed the course’);
Else
    Sprintf( ‘Student passed the course’);
End if
```

Algorithm development phase

- Algorithm development phase
  - First step: Understand the problem
  - Second step: Design an algorithm

First Step: Understand the problem

- What do I know about the problem?
- What is the information that should be procesed to find the solution?
- What does the solution look like?
- Identify input information and output information.
  - Problem example: **Find first non repeated character in a sentence**

    This could be an example of input and output information for this problem:

    Input sentence: The cat is in the kitchen
    Algorithm output:

Second Step: Design the Algorithm

- Three sub-steps:
  1. Devise a plan: General cases and Special cases
  2. Test the plan for different inputs (trace)
  3. Refine the solution
     - Identify similarities and patterns
     - Make the solution more general
     - Consider algorithm efficiency
     - Is there any alternative?

Second Step: Design the Algorithm

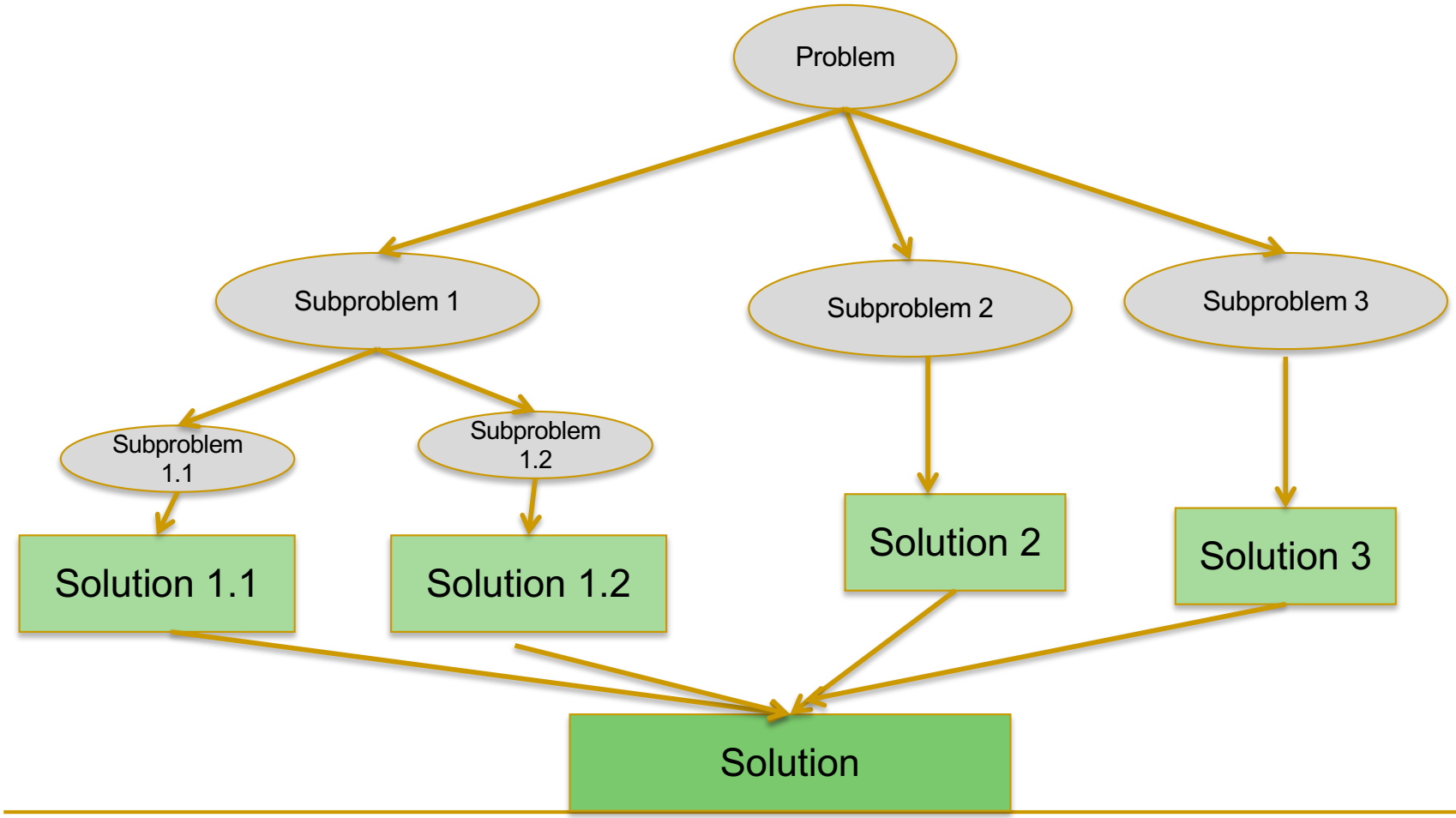
1. Devise a plan:
   - Some techniques to approach the design of the algorithm
     - Look for related problems already solved (pattern matching)
     - Working backwards (reverse engineer)
     - Divide and conquer

Second Step: Design the Algorithm

- "Divide and conquer" method:
  - **Divide** the problem into one or more sub-problems
  - **Conquer** subproblems by solving them recursively
    - If the problem is simple enough solve it directly
  - As a result a hierarchical structure of problems and subproblems is obtained
  - The solutions of the subproblems can then be **combined** to solve the original problem

Second Step: Design the Algorithm

- "Divide and conquer" method:



Second Step: Design the Algorithm

- Advantages of the “Divide and conquer” method:
  - Smaller problems are easier to comprehend.
  - Solutions to smaller problems are easier to test.
  - Sub-solutions tend to be simpler than when considered as a whole.
  - Different designers can work in different parts of the problem in parallel
  - The program will be easier to maintain
  - Reuse of sub-solutions for other problems

Algorithm Design

- Exercise: Design an algorithm for planning a birthday party

Algorithm Design

- Exercise: Design an algorithm for planning a birthday party
  - Understand the problem

Algorithm Design

- Exercise: Design an algorithm for planning a birthday party
  - Divide the problem:

Second Step: Design the Algorithm

2. Test the plan for different inputs (trace):
   - Consider general cases and special cases
     - Problem example: **Find first non repeated character in a sentence**

       Example of an special case for the previous problem:

       Input sentence:
       Algorithm output:

Second Step: Design the Algorithm

2. Test the plan for different inputs (trace):
   - Consider general cases and special cases
     - Problem example: **Find first non repeated character in a sentence**

       Example of an special case for the previous problem:

       Input sentence: blablabla
       Algorithm output: none

Algorithm Design

3. Refinement:
   - Identify similarities and patterns
   - Make the solution more general
   - Consider algorithm efficiency
     - Is there any alternative?

Problem Solving Exercises

- Problem: Try to guess a number in the minimum amount of tries
  - I can only tell you if you are right, too high or too low

    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Problem Solving Exercises

- Design an algorithm for obtaining the average value given a list of numbers
- Design an algorithm for an ATM: the user will introduce the amount required and the machine will only dispense notes in denominations of 50, 20 and 10 euros.

Algorithm Design

- Exercise: “Given a list of numbers find the average value”

Problem Solving Exercises

- Design an algorithm for an ATM: the user will introduce the amount required and the machine will only dispense notes in denominations of 50, 20 and 10 euros.
  - *The ATM only allows quantities in multiples of 10*
  - *The ATM only allows quantities greater than 10*

Problem Solving Exercises

- Game:
  - https://www.brainpop.com/games/blocklymaze/
  - You have to tell the man how to reach the goal by putting in the right order the different blocks/lego pieces