Unit 2. Programming Fundamentals

uc3m | Universidad Carlos III de Madrid

DEI Interactive Systems Group

Computer Problem Solving

- 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.: "*An ordered and finite set of operations which allows finding the solution of a problem*"
- We use different algorithms every day:
  - Recipies
  - D.I.Y. furniture
  - Explaining to somebody how to get somewhere
  - Driving a car

What is an algorithm?

- An **algorithm** is set of instructions for solving a problem or sub-problem 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 half a 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.

Algorithm representations

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

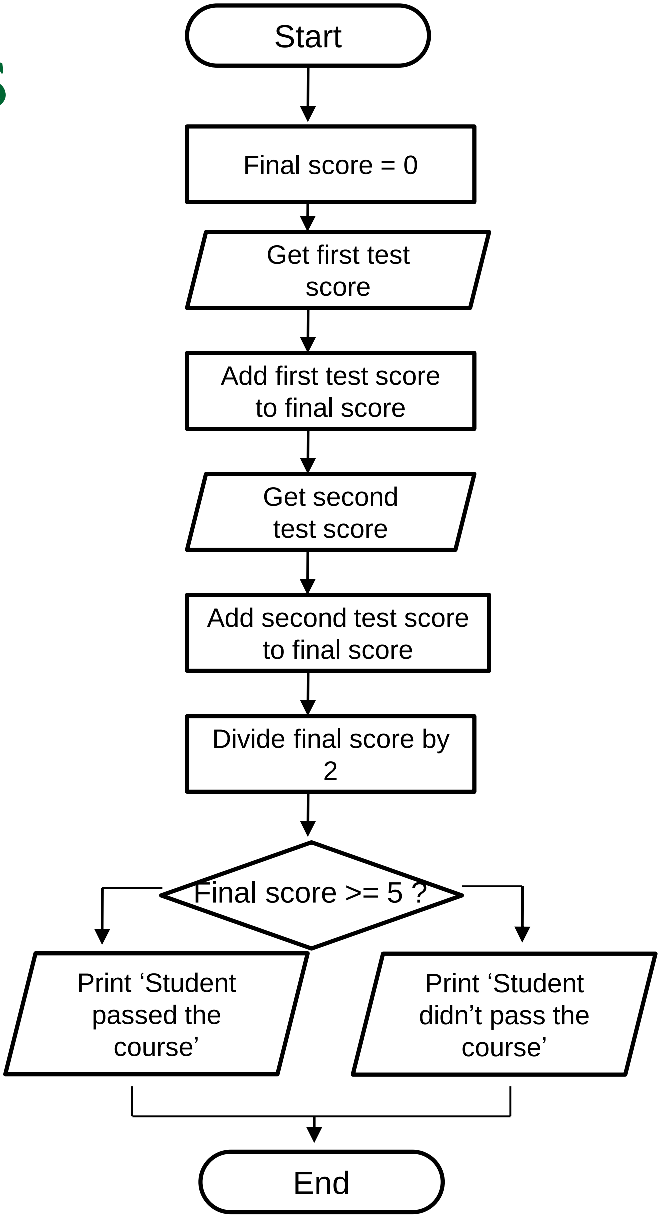
Algorithm representations

- **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.

Algorithm representations

- **Flowchart:** combines symbols and flowlines, to figuratively show the operations of an algorithm
  - Closer 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


Start
Final score = 0
Get first test score
Add first test score to final score
Get second test score
Add second test score to final score
Divide final score by 2
Final score >= 5 ?
Print 'Student passed the course'
Print 'Student didn't pass the course'
End

Algorithm representations

- **Pseudocode**: Natural language constructs modeled to look like statements available in many programming languages
    - Comprenhensible as natural language but unambiguous
    - 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
```

Algorithm 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 humans and to computers
  - Algorithms described in diffferent languages 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 processed 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**

    Example of input and output information for this problem:

    Input sentence: The cat is in the kitchen

    Algorithm output:

First Step: Understand the problem

- What do I know about the problem?
- What is the information that should be processed 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**

    Example of input and output information for this problem:

    Input sentence: The cat is in the kitchen

    Algorithm output: a

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 an alternative?

Second Step: Design the Algorithm

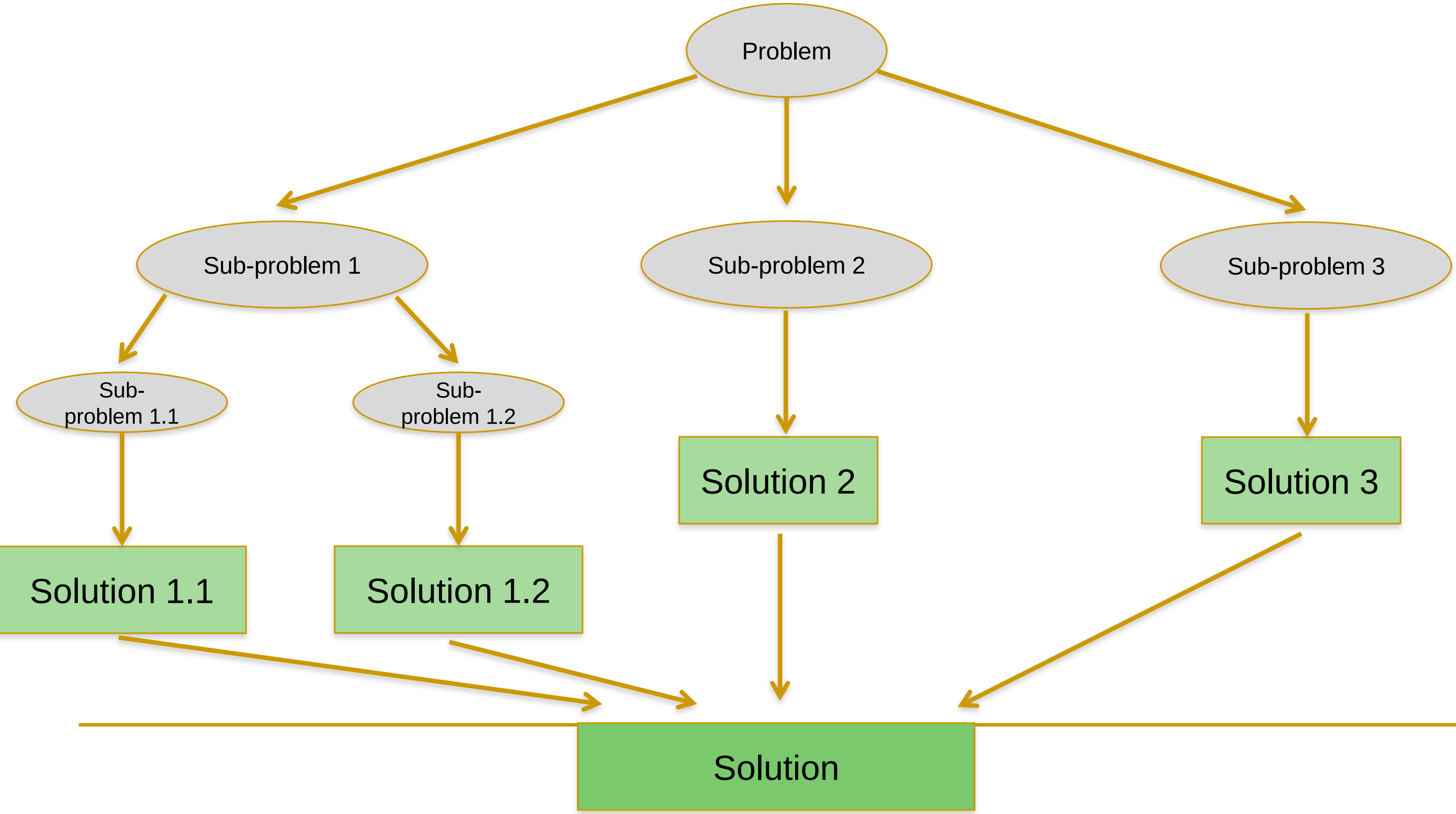
1. Devise a plan:
   - Some techniques to approach the design of the algorithm
     - Look for related problems already solved (pattern matching)
     - Working backwards (reverse engineer)
     - Divide and conquer

Second Step: Design the Algorithm

- “Divide and conquer” method:
  - **Divide** the problem into one or more sub-problems
  - **Conquer** sub-problems by solving them recursively
    - If the problem is simple enough solve it directly
  - As a result a hierarchical structure of problems and sub-problems is obtained
  - The solutions of the sub-problems 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
  - Understand the problem:
    - Budget for the party?
    - Who am I going to invite?
    - Where is the party going to be?
    - What am I going to offer for dinner?

Algorithm Design

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

Algorithm Design

- Exercise: Design an algorithm for planning a birthday party
  - Divide the problem:
    - Problem 1: Obtain the birthday party guest list
    - Problem 2: Book the venue for the party
    - Problem 3: Obtain the shopping list
    - . . .

Algorithm Design

- Exercise: Design an algorithm for planning a birthday party
  - Divide the problem:
    - Problem 1: Obtain the birthday party guest list
      - 1.1 : Send invitations: email? phone?
      - 1.2 : Receive confirmations
      - 1.3 : Write down the final list
    - Problem 2: Book the venue for the party
    - Problem 3: Obtain the shopping list

Algorithm Design

- Exercise: Design an algorithm for planning a birthday party

  - Divide the problem:
    - Problem 1: Obtain the birthday party guest list
    - Problem 2: Book the venue for the party
      - Less than 15 people: at home
      - More than 15: find and book a venue
    - Problem 3: Obtain the shopping list

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 a 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 a 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 an 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

- First solution:
  - 1. Pick a number at random
  - 2. If it is correct: you win and we stop
  - 3. If it is incorrect: repeat the process

Problem Solving Exercises

- A better (refined) solution:
  - 1. Pick a number:
    - 1.1 Add the minimun number to the maximum and divide the result by two, round the result, and pick this number
  - 2. If it is correct: you win and we stop
  - 3. If it is a smaller number, repeat the process but only with the numbers to the left of the one you picked
  - 4. If it is a bigger number, repeat the process but only with the numbers to the right of the one you picked

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 of 50, 20 and 10 euros

Algorithm Design

- Exercise: “Find the average value given a list of numbers”

Algorithm Design

- "Count the numbers"
  - Set counter to 0
  - Read first number and increase the counter
  - Read second number and increase the counter
  - Continue reading and increasing the counter until the end of the list

Algorithm Design

- “Add up the numbers”
  - Set sum result to 0
  - Read first number and add it to sum result
  - Read second number and add it to sum result
  - Continue reading and adding until the end of the list

Algorithm Design

- “Divide the result by numbers counted”
  - If counter is 0 then result is 0
  - If counter is not 0 then result is the sum of the numbers divided by the counter

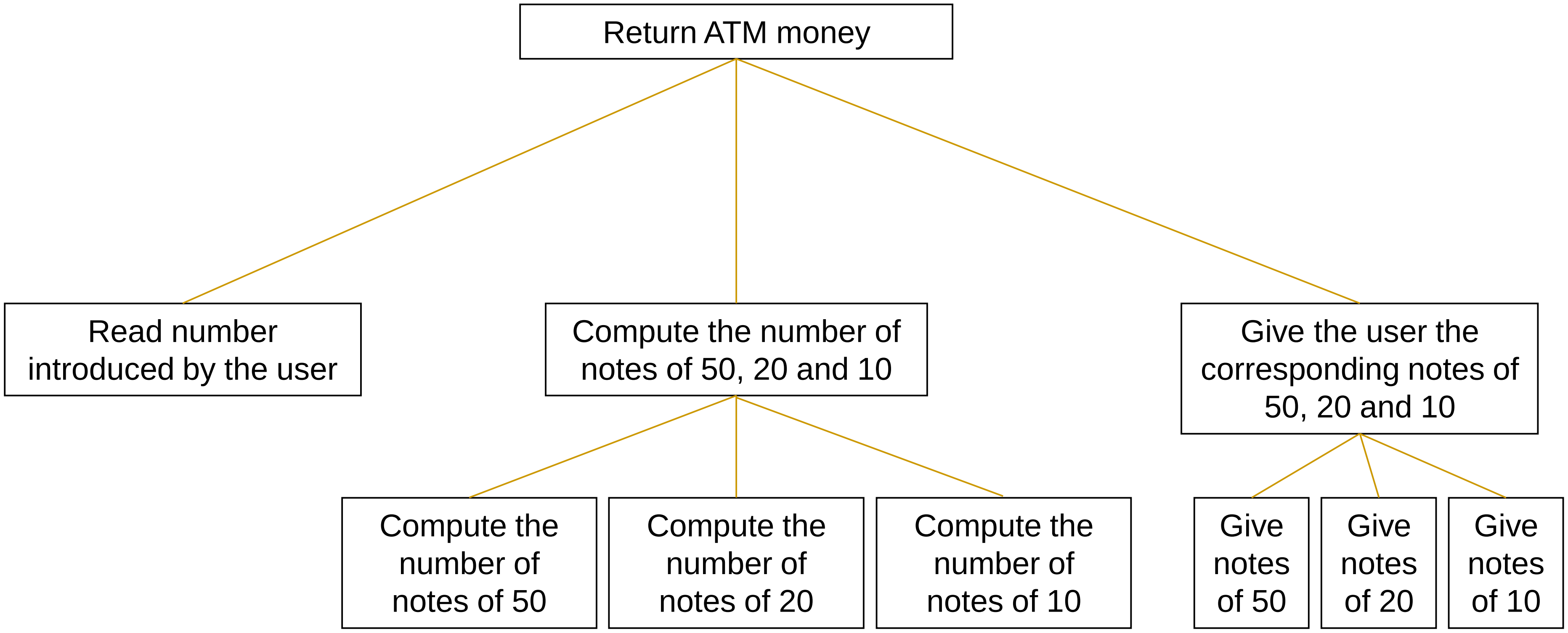
Algorithm Design

- Solution:
  - Set counter and sum result to 0
  - Try to read a number
  - If a number has been read add it to the sum result and increase the counter
  - Repeat the two previous steps until no more numbers can be read
  - If counter is 0 then the result is 0
  - If counter is not 0 then result is the sum of the numbers divided by the counter

Problem Solving Exercises

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

Algorithm Design


Return ATM money
Read number introduced by the user
Compute the number of notes of 50, 20 and 10
Give the user the corresponding notes of 50, 20 and 10
Compute the number of notes of 50
Compute the number of notes of 20
Compute the number of notes of 10
Give notes of 50
Give notes of 20
Give notes of 10

Algorithm Design

- "Compute the number of notes of 50"
  - If quantity is greater or equal to 50
    - notesOf50 = quantity / 50
    - quantityFor20 = remainder of quantity / 50
  - If quantity is less than 50
    - notesOf50 = 0
    - quantityFor20 = quantity

Algorithm Design

- “Compute the numbers of notes of 20”
  - If quantityFor20 is greater or equal to 20
    - notesOf20 = quantityFor20 / 20
    - quantityFor10 = reminder of quantityFor20 / 20
  - If quantityFor20 is less than 20
    - notesOf20 = 0
    - quantityFor10 = quantity

Algorithm Design

- "Compute the numbers of notes of 10"
  - If quantityFor10 is greater or equal to 10
    - notesOf10 = quantityFor10 / 10

Algorithm Design

- Read the quantity
- notesOf50 = 0, notesOf20=0, notesOf10= 0
- If quantity is greater or equal to 50
  - notesOf50 = quantity / 50
  - quantityLeft = remainder of quantity / 50
- If quantity is less than 50
  - notesOf50 = 0
  - quantityLeft = quantity
- If quantityLeft is greater or equal to 20
  - notesOf20 = quantityLeft / 20
  - quantityLeft = remainder of quantityLeft / 20
- If quantityLeft is less than 20
  - notesOf20 = 0
  - quantityLeft = quantityLeft
- If quantityLeft is greater or equal to 10
  - notesOf10 = quantityLeft / 10
- Give the user *notesOf50* notes of 50
- Give the user *notesOf20* notes of 20
- Give the user *notesOf10* notes of 10

Problem Solving Exercises

- Game:
  - https://www.brainpop.com/games/blocklymaze/