Programming Fundamentals

Lecture 01 – Introduction to Programming

Edirlei Soares de Lima
<edirlei.lima@universidadeeuropeia.pt>

Computer Programs

- **Programs are created to solve <u>problems</u> and perform specific tasks.**

- **Problem**: your car has a flat tire… What you do? What are the steps to replace the car's tire?

Computer Programs

- **Problem**: your car has a flat tire… What you do? What are the steps to replace the flat tire?

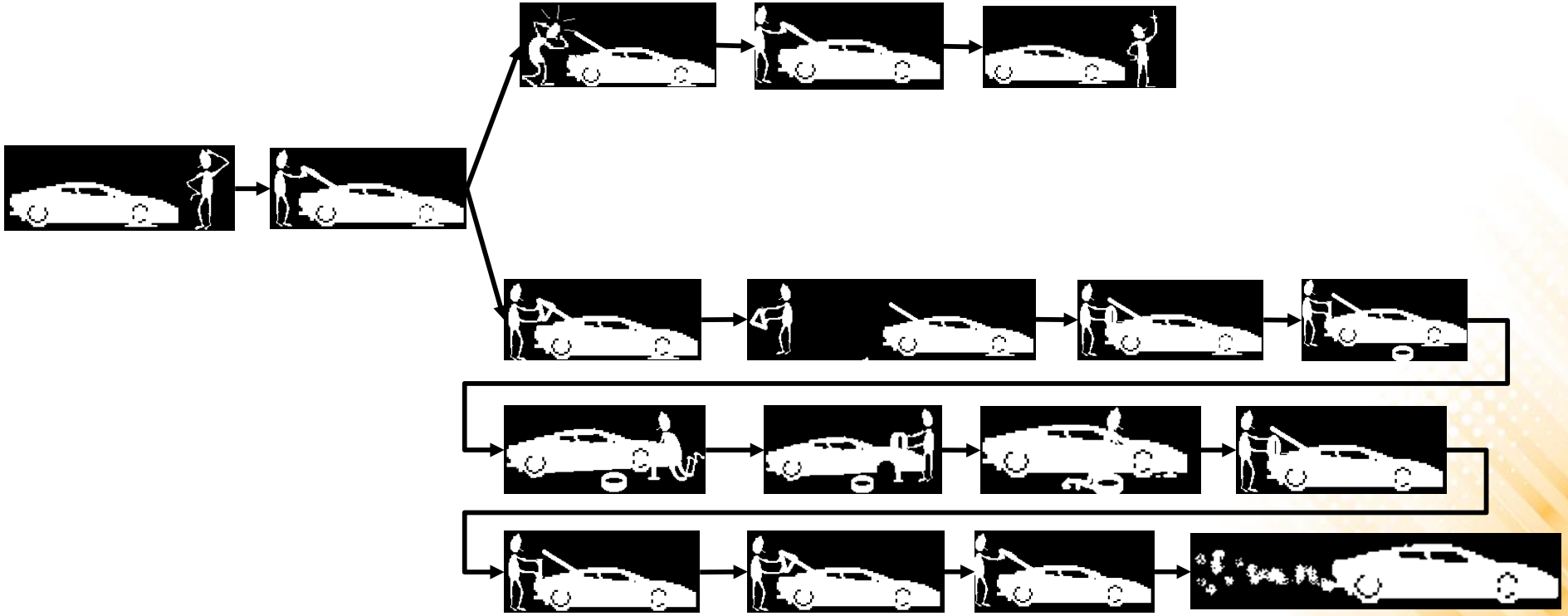
- **Textual solution:**

  - “Open the trunk and make sure that all the equipment is there. If not, close the trunk and ask for a ride. If so, remove the reflective triangle, position it about 30 m from the car. Then, get the new tire and the jack. Place the jack and lift the car ...”

Computer Programs

- **Problem**: your car has a flat tire… What you do? What are the steps to replace the flat tire?

- **Graphical solution:**



Computer Programs

- **Problem**: your car has a flat tire… What you do? What are the steps to replace the flat tire?

- **Algorithmic solution:**

```
open(trunk)
if equipment_is_there == FALSE then
  close(trunk)
  ask_for_ride()
else
  get(triangle_sign, trunk)
  place(triangle_sign, car_position - 30)
  get(jack, trunk)
  get(new_tire, trunk)
  ...
```

Machine Code

- A computer's central processing unit (CPU) only runs machine code.
  - Machine code is a strictly numerical language and is intended to run as fast as possible.
  - The instructions causes the CPU to perform specific task, such arithmetic operations and registry manipulations.
  - Machine code is represented by sequences of binary digits.

- Exemple:

```
Adding the registers 1 and 2
and placing the result in
register 6.
```

```
[  op  |  rs |  rt |  rd |shamt| funct]
   0     1     2     6     0     32      decimal
 000000 00001 00010 00110 00000 100000   binary
```

```
Jumping to the address 1024
```

```
[  op  |         target address         ]
   2                  1024                decimal
 000010 00000 00000 00000 10000 000000   binary
```

High-Level Programming Languages

- While it is possible to write programs directly in machine code, it is very complex to manage individual bits and calculate numerical addresses manually.

- Today, the vast majority of programs are written in high-level programming languages.

- In a high-level language, instead of dealing with registers and memory addresses, programmers manipulate variables, functions, loops, boolean expressions, arrays, etc.

Popular Programming Languages

Aug 2018	Aug 2017	Change	Programming Language	Ratings	Change
1	1		Java	16.881%	+3.92%
2	2		C	14.966%	+8.49%
3	3		C++	7.471%	+1.92%
4	5	^	Python	6.992%	+3.30%
5	6	^	Visual Basic .NET	4.762%	+2.19%
6	4	v	C#	3.541%	-0.65%
7	7		PHP	2.925%	+0.63%
8	8		JavaScript	2.411%	+0.31%
9	-	^^	SQL	2.316%	+2.32%
10	14	^^	Assembly language	1.409%	-0.40%
11	11		Swift	1.384%	-0.44%
12	12		Delphi/Object Pascal	1.372%	-0.45%
13	17	^^	MATLAB	1.366%	-0.25%
14	18	^^	Objective-C	1.358%	-0.15%
15	10	vv	Ruby	1.182%	-0.78%

https://www.tiobe.com/tiobe-index/

Example of Code in Lua

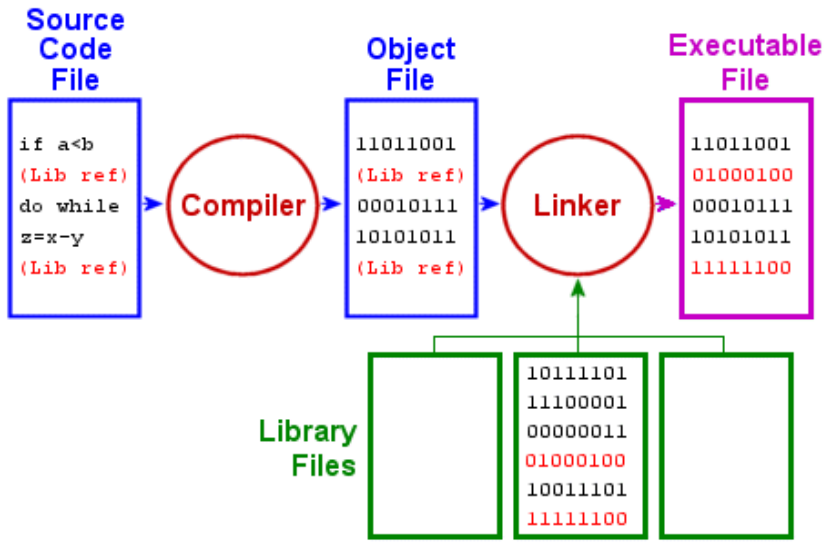
```
local cels
local fahr

io.write("Temperature in Celsius: ")
cels = io.read()

fahr = 1.8 * cels + 32

io.write("Temperature in Fahrenheit: ", fahr, "\n")
```

Compilation Process

- Programs written in high-level languages cannot be directly executed in the CPU.
  - Before running the code, it must be translated to machine code.

- This task (**compilation**) is performed by a program called **compiler**.

- Examples of compiled languages:
  - C, C++, Basic, Pascal, …

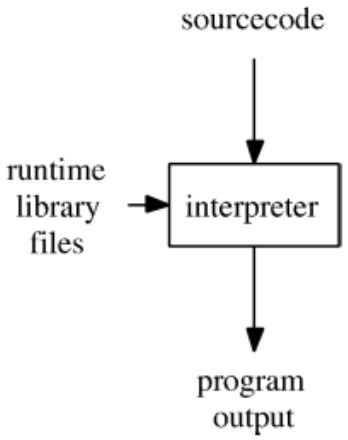


Interpreted Programming Languages

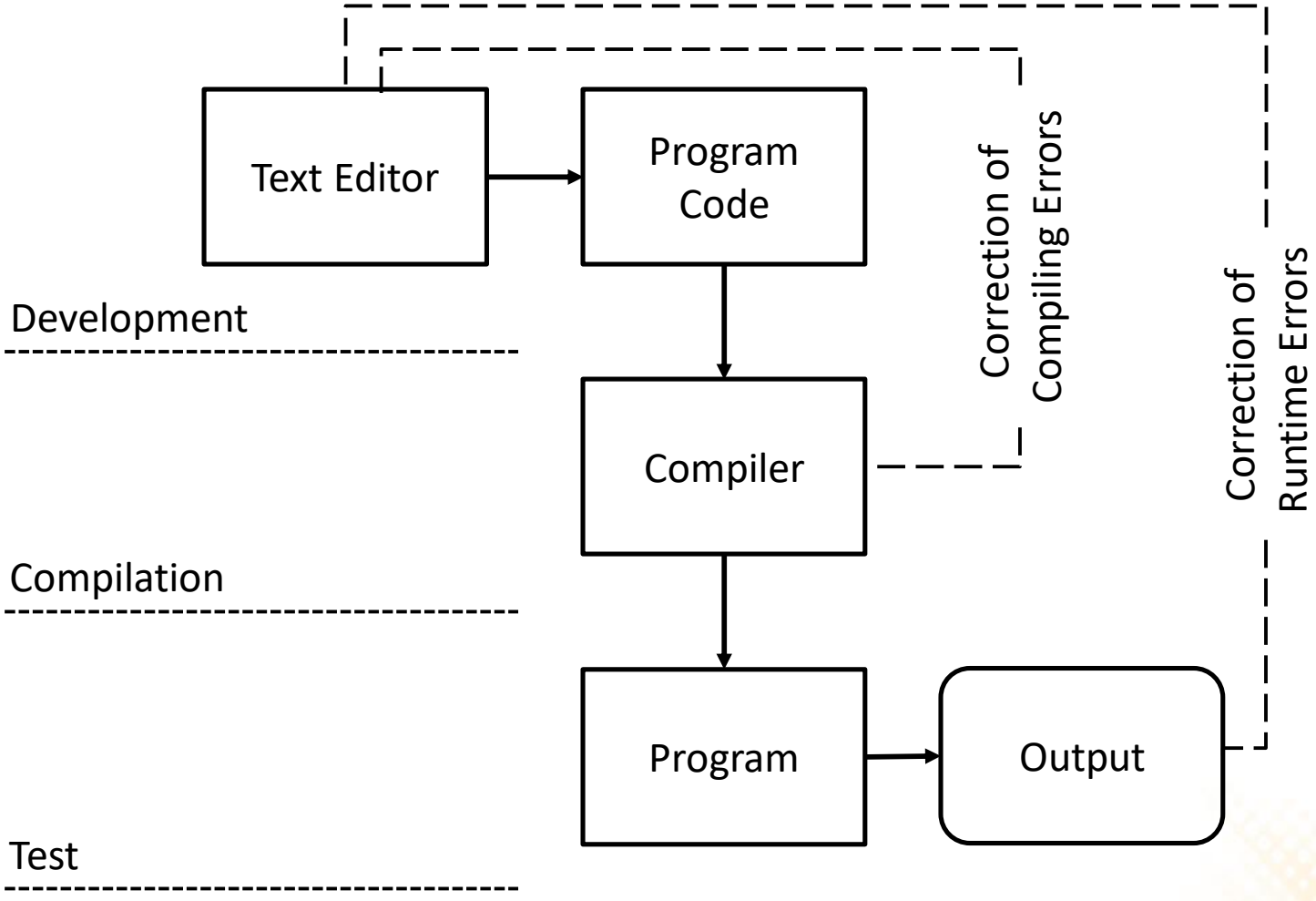
- Not all programming languages are compiled. Some of them are interpreted.

- An interpreted language uses an **interpreter** program to execute the code directly, translating each statement into a sequence of one or more operations that are performed by the interpreter.

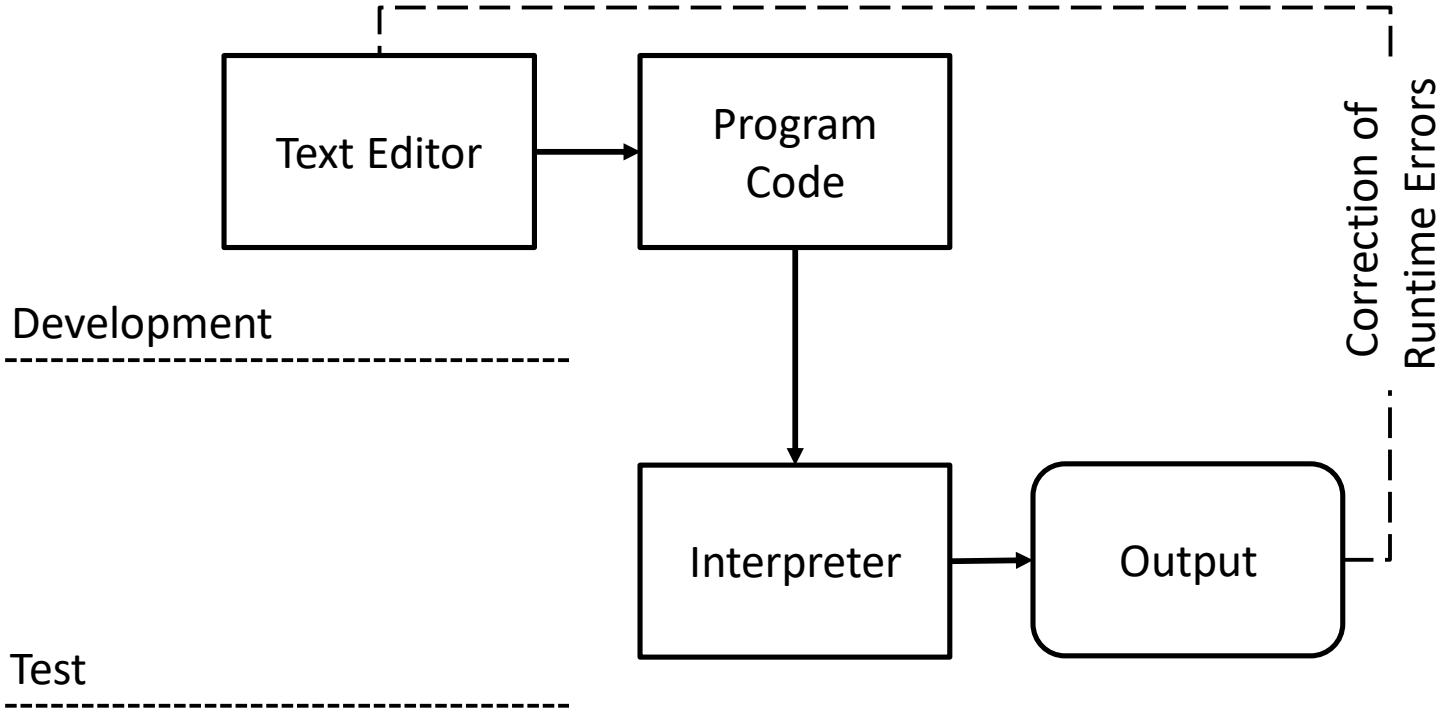
- Examples of interpreted languages:
  - Lua, Java, JavaScript, Python, PHP, …



Development Cycle (Compiled)



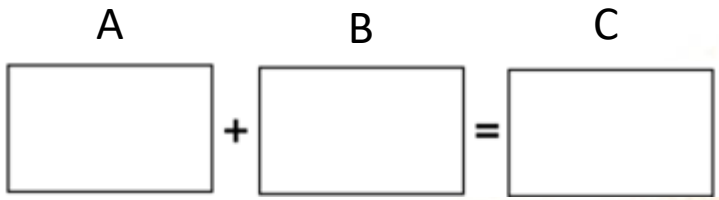
Development Cycle (Interpreted)



Algorithms

- **What is an algorithm?**
  - A detailed sequence of actions to perform in order to accomplish a task or solve a problem.
  - It is an unambiguous specification of how to solve a class of problems.
  - Is independent of programming language.

- Even simple tasks can be described by sequences of actions:
  - ***Example: Add two numbers***
    1) Write the first number in rectangle A.
    2) Write the second number in rectangle B.
    3) Add the number of rectangle A with the number of rectangle B and put the result in rectangle C.



Algorithms: Example 1

- **Algorithm**: calling a friend on the telephone
- **Input**: the telephone number of your friend
- **Output**: none

- **Steps**:
  1. Pick up the phone and listen for a dial tone;
  2. Press each digit of the phone number on the phone;
  3. If busy, hang up the phone, wait 5 minutes, jump to step 1;
  4. If no one answers, leave a message then hang up;
  5. If no answering machine, hang up and wait 2 hours, then jump to step 1;
  6. Talk to friend;
  7. Hang up the phone;

Exercise 1

1) Write an algorithm for an everyday task. Pick a common or interesting task and breaking it down to the level that a computer might understand.
   - Some suggestions: brushing your teeth, taking a shower, a cooking recipe, how to get to the university, …
   - Make it extra specific: pretend you are explaining to someone with no common sense and no knowledge of the task, like an alien or a robot.
   - Format: follow the same format of the example 1 and specify what is the input (like ingredients in a recipe) and the output (if needed).
   - Try to use conditions by using words such as “if”, “else” or “otherwise”.
   - Try to use repetitions by using the word such as “jump to step” when the same step must be repeated.