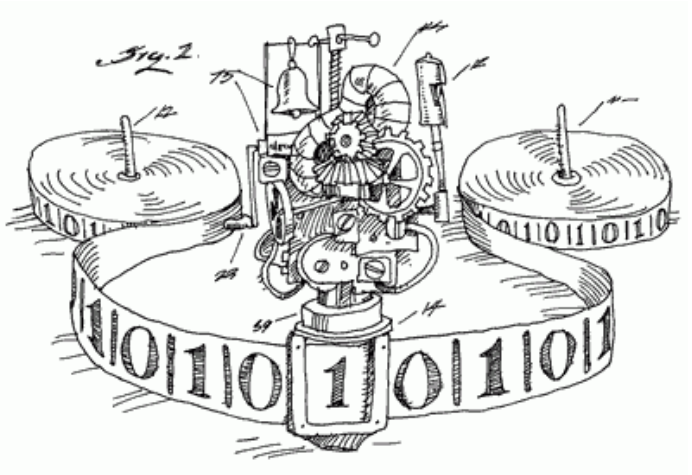
INFO 101 – Introduction to Computing and Security

[2020 - Week 4 / 1]



Prof. Dr. Rui Abreu

University of Porto, Portugal

rui@computer.org

@rmaranhao

Exercise

- [5min] Write a regular expression to capture Chinese license plates
- [5min] Using ***egrep***, find license plates in all .txt files

Exercise

- [5min] Write a regular expression to capture Chinese license plates

Exercise

- [5min] Write a regular expression to capture Chinese license plates

[A-Z][0-9]{3}[0-9A-Z][0-9]

Exercise

- [5min] Write a regular expression to capture Chinese license plates
- [5min] Using ***egrep***, find license plates in all .txt files

Exercise

- [5min] Write a regular expression to capture Chinese license plates
- [5min] Using ***egrep***, find license plates in all .txt files

[A-Z][0-9]{3}[0-9A-Z][0-9]

egrep -n "[A-Z][0-9]{3}[0-9A-Z][0-9]" -o --color=always *.txt

What we are going to discuss…

- Introduction to computer programming and programming languages
- Basic foundations of programming languages (functional, imperative, logic-based, and object-oriented)
- Functional Programming in LISP

Programming languages

- There are so (too) many like:
  - Algol (1958)
  - Cobol (1959)
  - Fortran (1957)
  - Ada (1980)
  - PL/I (1964)
  - Pascal (1968)
  - **LISP (1958)**
  - C, C++
  - Java
  - Java Script
  - C#
  - Scala
  - Python
  - SQL
  - Ruby
  - PHP
  - R
  - Go
  - Swift
  - Prolog

See https://en.wikipedia.org/wiki/Timeline_of_programming_languages

What is a programming language?

- *"… a formal language that comprises a set of instructions used to produce various kinds of output*." (from Wikipedia 2018)

- A program is written in a particular language for implementing an algorithm, i.e., a way of describing a solution for a certain problem.

- Used for communication with machines but also humans
- Programming language contain **abstractions** for specifying data structures or controlling execution and have a certain **expressive power (e.g. being Turing complete).**

Language grammar - Syntax

- Every programming language is composed based on characters forming the language's elements, i.e., its symbols (statements, variables, …)
- Syntax describes the possible combinations of symbols
- Often the Backus-Naur form (BNF) is used for describing the syntax

Backus-Naur Form (BNF)

- A common notation from computer science
- BNF is a meta-language used to describe the grammar of a programming language
- BNF is formal and precise
- BNF is essential in compiler construction
- **Example** (LISP):

```
expression ::= atom | list
atom ::= number | symbol
number ::= [+-]?['0'-'9‘]+
symbol ::= ['A'-'Z''a'-'z‘]+
list ::= '(' expression* ')'
```

Syntax

- The grammar

```
expression ::= atom | list
atom ::= number | symbol
number ::= [+-]?['0'-'9‘]+
symbol ::= ['A'-'Z''a'-'z‘]+
list ::= '(' expression* ')'
```

Examples: 23, ab, (), (abc), (() abc 001) **but not:** _ab, (()), (ab a1)

describes:

- an *expression* is either an *atom* or a *list*;
- an *atom* is either a *number* or a *symbol*;
- a *number* is an unbroken sequence of one or more decimal digits, optionally preceded by a plus or minus sign;
- a *symbol* is a letter followed by zero or more of any characters (excluding whitespace); and
- a *list* is a matched pair of parentheses, with zero or more *expressions* inside it.

Syntax (cont.)

- A **Compiler** (e.g., gcc for C) checks whether a certain program fulfills the grammar of its programming language
- In case of no match, we get a **Syntax Error**

- **Notes**:
  - More information about how compilers work, you find in lectures or books about **Compiler Construction**
  - **Regular expressions** are also used for describing (at least parts) of programming languages

Type systems

- A (data) type determines how variables or expressions are classified and which operators are available.

- For **example**: **Integer** (int) represents the whole numbers, -12,.., 0,...,250,…
- There are also other basic data types:
  - **Float** for floating point numbers, e.g., 1.23e-12 meaning $1.23*10^{-12}$
  - **String** for representing sequences of characters, e.g., “this is a text”.
  - A characters (**char**) itself
  - And many others depending on the programming language

Type systems

- There are also user defined types (classes)
- And compositional types like
  - **Arrays** comprising a “list” of elements of one particular type of a given length, e.g., **int**[3] representing an array comprising 3 elements of type **int**.
  - **Records** having fields of different types, e.g., student{**string** name, **int** student_number} representing a record having the two fields name and student_number
- The type system assures that expressions like “123” + 1 are not allowed!

Type system

- Not all programming languages have a type system
  - Untyped languages like assembler
- We distinguish:
  - **Static typing**: A compiler assures that the type system is respected, i.e., that no expressions like ”123” + 1 are used.
  - **Dynamic typing**: The correctness of a type assigned to a variable or expression is checked at runtime (the time when the program is executed).
- We further distinguish:
  - **Weak typing**: E.g. allowing 1.2 + 1 (float + int) or adding a value to a string.
  - **Strong typing**: The opposite of weak typing

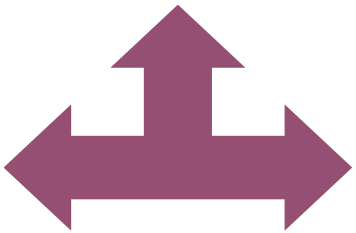
Programming language classification

- Syntactic criteria based on the language grammar
  - How is a program structured?
  - Is the language easy to learn?
  - Is there a language support to develop large software?
  - ...
- Semantics criteria
  - Behavior of programs or how to obtain results?

Semantics?

- Behavior of a program written in a language

„Have a look at this crane.“



Note: Crane is a **homonym** (the same word with mulitple meanings)

Functional programming languages

```
fun test (x,y) =
     if x=0 then 0
     else y + test(x-1,y);
```

What calculation is this SML program performing?

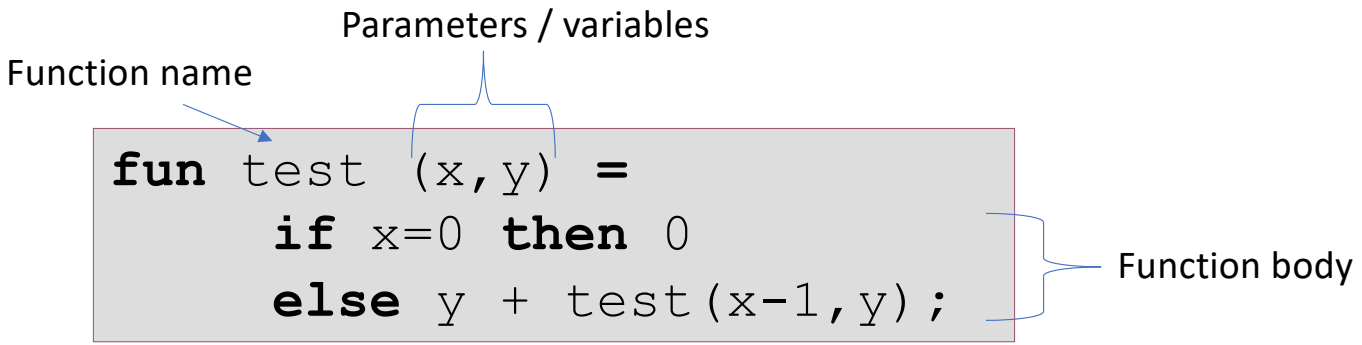
(Inductive) definition of the multiplication

$$x * y = \begin{cases} 0 & if\ (x = 0) \\ y + ((x-1) * y) & otherwise \end{cases}$$

`test` implements the multiplication!

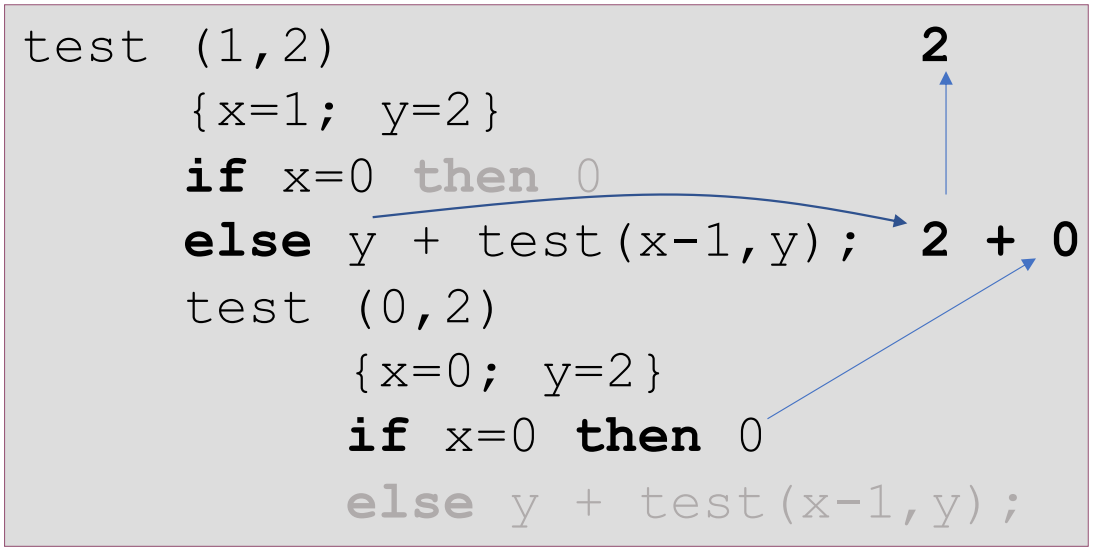
Functional languages

- Programs similar to mathematical functions
- Functions and their parameters
- Definition of functions comprises the function body and parameter variables (set when calling a function)



Example execution

Calling the method test with parameters 1 and 2 returns 2



Logic-oriented programs

- **Predicate**
  `father(X,Y)` ... `X` is father of `Y`
- **Rules** (like implications in propositional logic)
- **Queries** (asking the system whether a predicate can be derived from what we currently know)
- **Example**: „When it rains, the streets are wet. It rains now. Are the streets wet?
  - **Program** in Prolog (which is a language):

```
streets_wet := it_rains.
it_rains.
```

  - **Query**: `streets_wet?`

Prolog program example

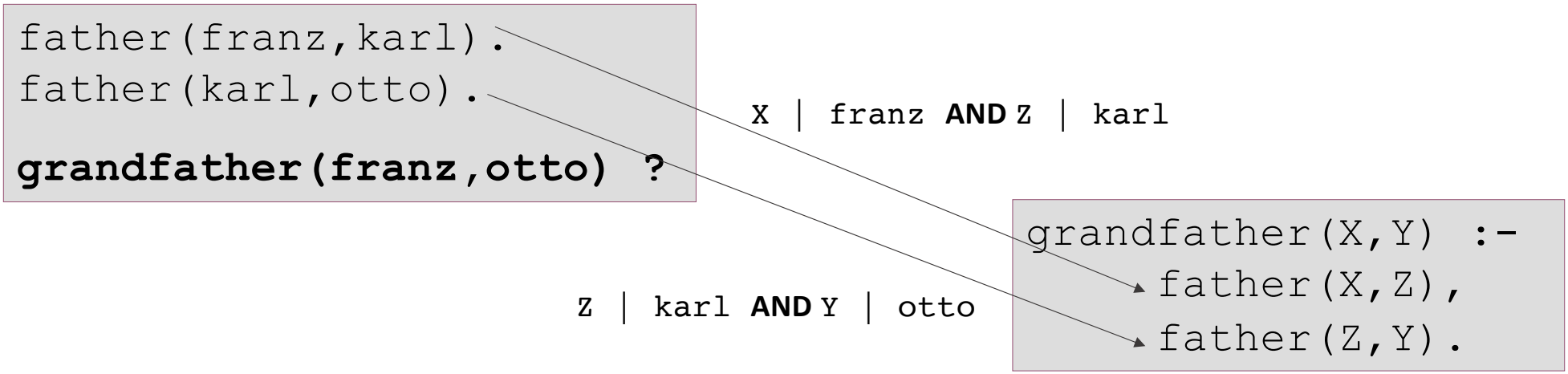
```
grandfather(X,Y) :-
    father(X,Z),
    father(Z,Y).
```

Notes:

1. `X`, `Y`, `Z` are variables
2. The `:-` stands for an implication ←
3. Variable values are assigned using unification (an "extended" version of assignment)

„`X` is the grandfather of `Y` if `X` is the father of `Z` and `Z` the father of `Y`."

Queries in Prolog



- Query `grandfather(franz,otto)` has to return true.

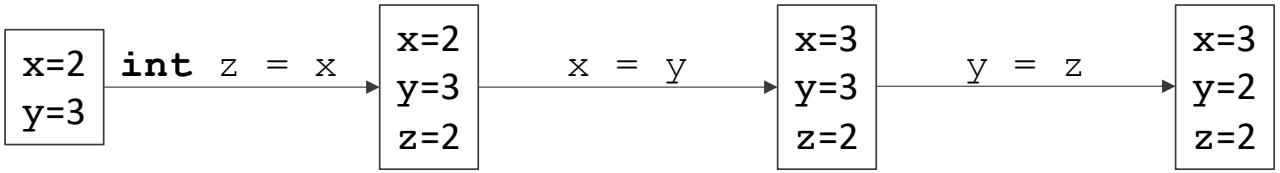
Imperative programming languages

- Also called **assignment languages**
- Assign values to variables
- State of a program given by the current values of all variables
- **Program** = sequence of statements
- Change the state of a program during execution
- Have usually the following **statements**:
  - Variable assignments
  - Conditional statements (IF-THEN-ELSE)
  - Loop statements (WHILE, UNTIL, FOR,…)

Semantics of imperative languages?

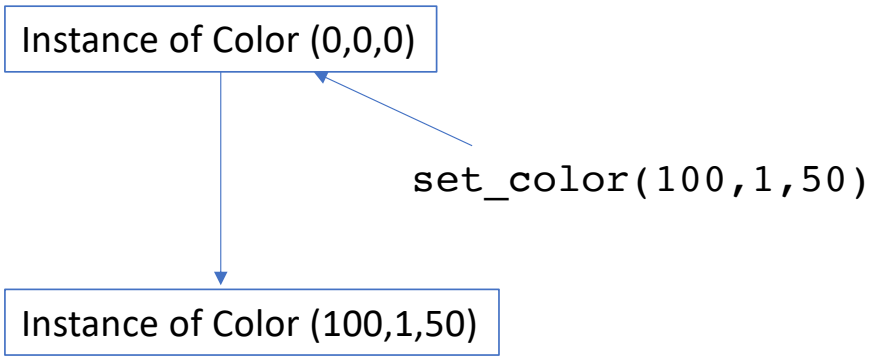
```
void test() {
        int z = x;
        x = y;
        y = z;
}
```

Defined via the interpretation of each single statement. A statement changes the state of the program using assignments.



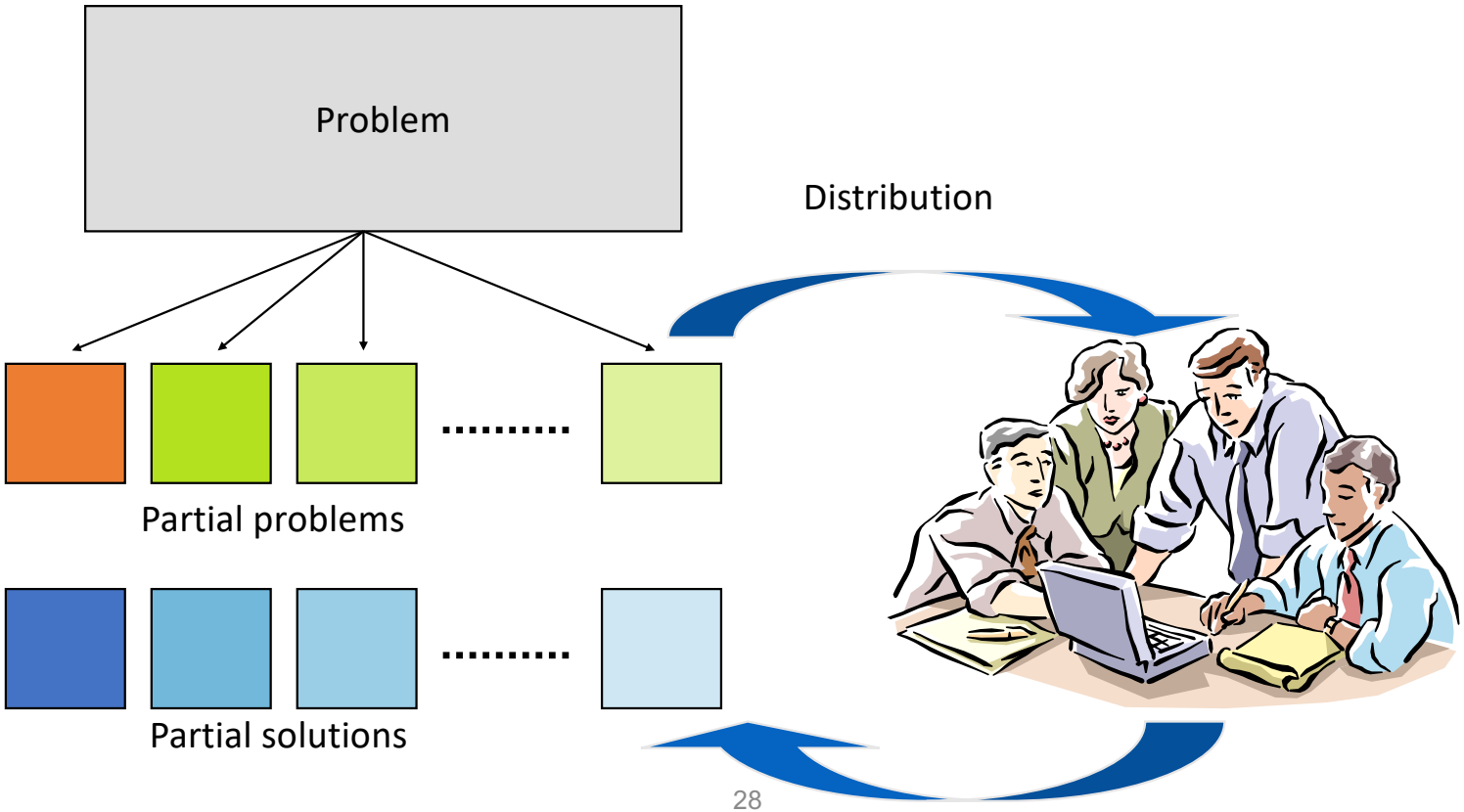
Object-oriented languages

- Are usually imperative languages + objects and classes
- Methods (like functions, or procedures) change the variables stored in classes and objects
- Method body is a sequence of statements like in imperative languages




```
Class Color
      int red, blue, green;
      set_color(int r,b,g) {
            red = r;
            blue = b;
            green = g; }
```

Programming languages / Why?



Development of large programs

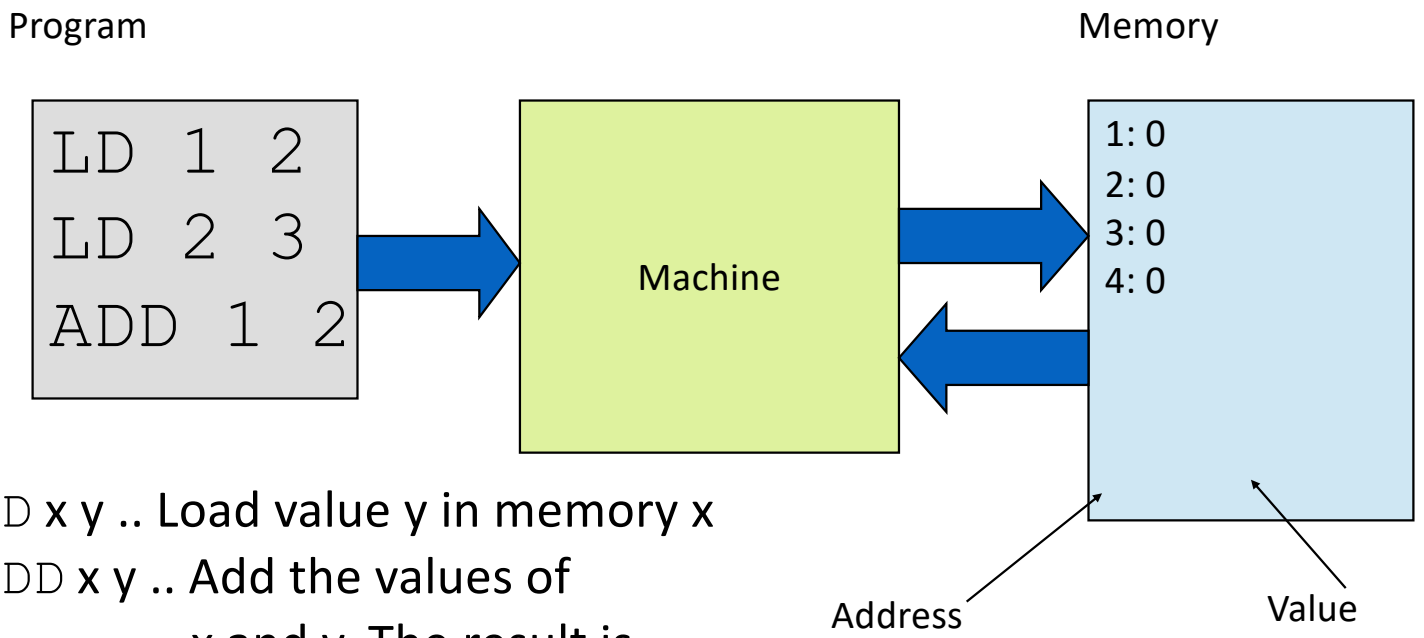
- Problem is to big to be solved by one person
- Team work required
- Separation of parts that can be solved separtely.
- Composition of partial solution to solve the original problem

Is this development process supported in a programming language?

Example: Functional view

- **Example goal**: Develop an interpreter for a programming language
- Program = Sequence of commands
- Machine has a memory and executes a program.
- Commands change the content of the memory.

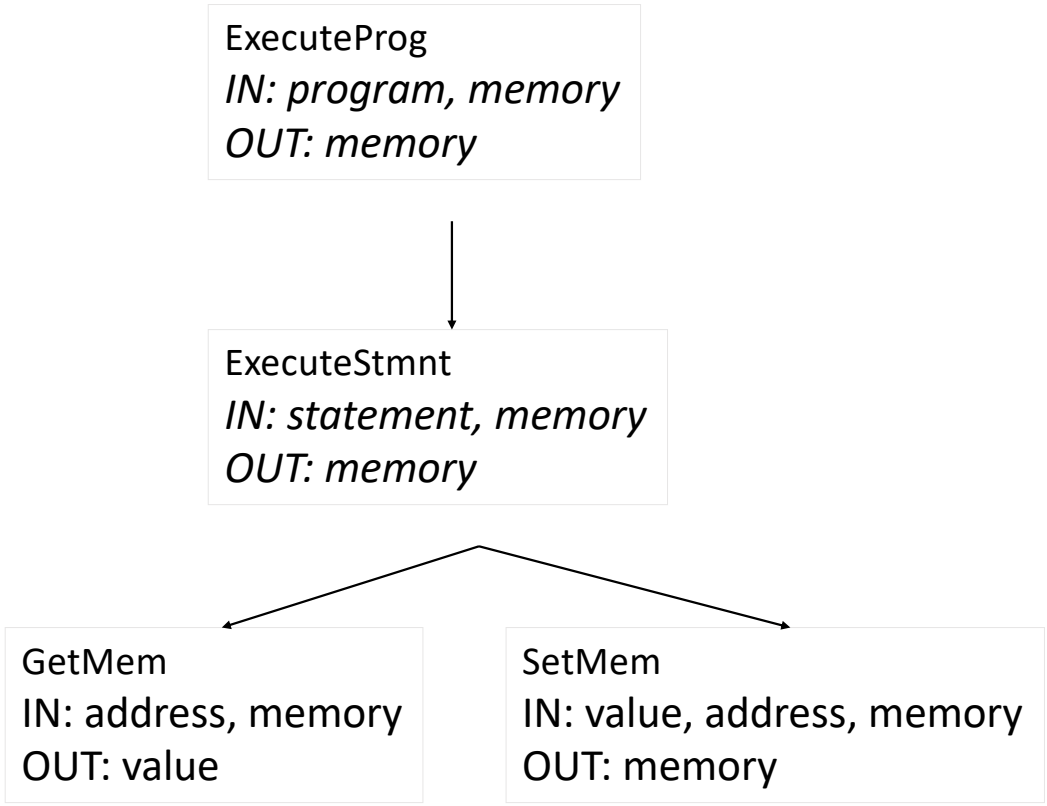
Example – Have a look at different components first



`LD` x y .. Load value y in memory x

`ADD` x y .. Add the values of
x and y. The result is
stored in x.

Example – Functional view



Example – SML program

```
fun executeProg (p,m) =
    if empty(p) then m
    else executeProg(rest(p),
              executeStmnt(first(p),m));

fun executeStmnt (s,m) =
    if loadStmnt(s) then
         executeLoad(s,m)
    else if addStmnt(s) then
         executeAdd(s,m)
    else errorHandling();

....
```

Example – SML program (cont.)

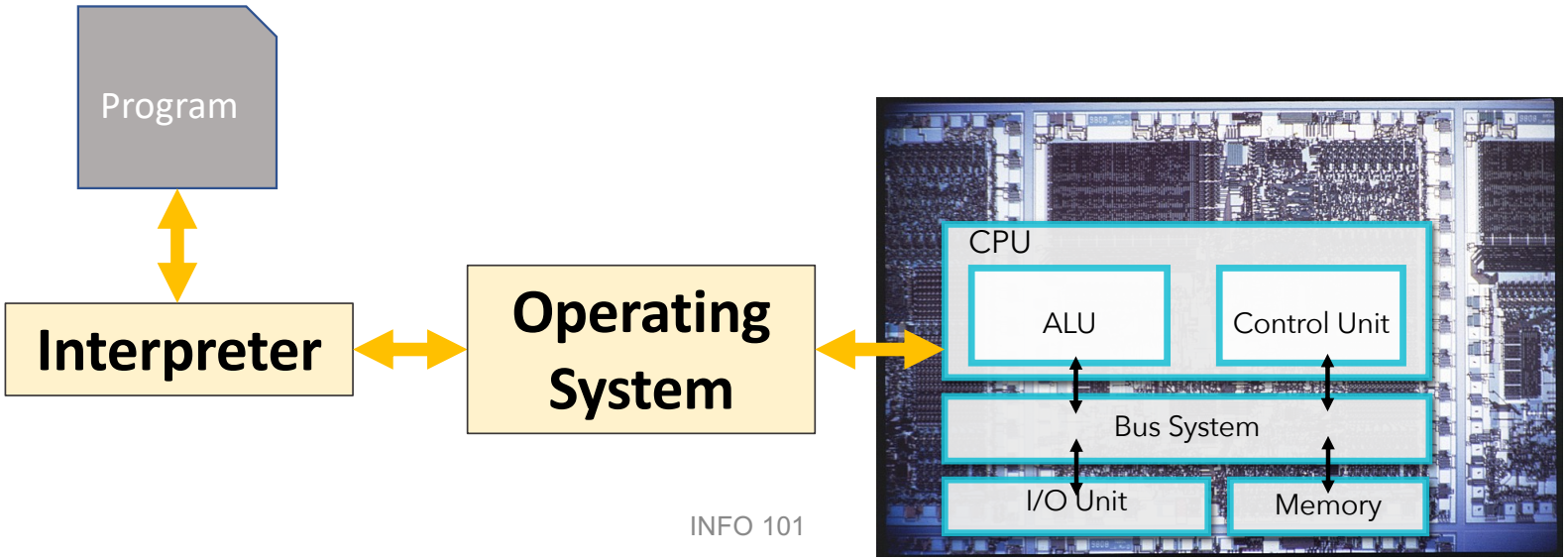
```
Statement ... 3-tupel (Name,Arg1,Arg2)

Memory .. List of tuples (Addr,Val)

fun loadStmnt (s,a,v) = (s=="LD");

fun executeLoad ((s,a,v),m) =
  setMem (v,a,m);

fun setMem (v,a,m) =
  if empty(m) then m

  else

       if #1(first(m))=a then

              (a,v)::rest(m)

       else first(m)::(setMem(v,a,rest(m)));
```

Example – Summary

- A function can be divided into sub-functions
- Separate validation & verification (correctness checks)
- Program can be more easily read and comprehend
- Solution of a problem becomes simpler
- Easier fault localization (debugging)
- Re-use of program parts is supported

Further notes

- Some languages are not compiled to machine code
- Such languages use an **Interpreter** to run the program, e.g., the language Java uses a Java Bytecode Engine.



Summary

- There are at least 3 different kinds of programming languages
  - Imperative languages (or assignment languages)
  - Functional languages
  - Logic-oriented languages
- Languages have a syntax and semantics
  - Syntax is described using regular expressions and the Backus-Naur form
- Languages have a type system
- Compilers are used to convert programs written in a programming language into another language, which is usually a machine readable language (e.g. machine code)

Exercise

- Write a regular expression to capture Chinese's foreign ID card

[A-Z]{3}[0-9]{12}

- Using ***egrep***, find Chinese ID's in all .txt files

egrep -n "[A-Z]{3}[0-9]{12}" -o --color=always *.txt

Advanced stuff

find . -name "*.txt" -or -name "*.tex”

find . -name "*.txt" -or -name "*.tex" | xargs egrep -n "[A-Z]{3}[0-9]{12}" -o --color=always