Programming Languages: Concepts

(Lectures on High-performance Computing for Economists IV)

Jesús Fernández-Villaverde[1] and Pablo Guerrón[2]

November 21, 2021

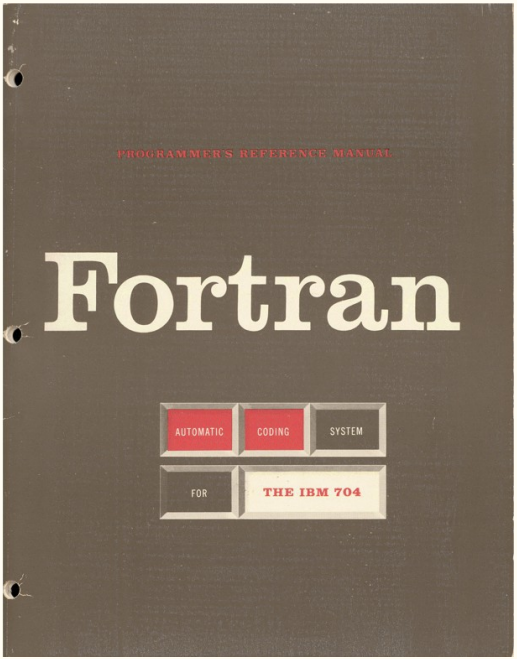
[1]University of Pennsylvania

[2]Boston College

Introduction

Motivation

- Since the invention of `Fortran` in 1954-1957 to substitute assembly language, hundreds of programming languages have appeared.
- Some more successful than others, some more useful than others.
- Moreover, languages evolve over time (different version of `Fortran`).
- Different languages are oriented toward certain goals and have different approaches.
- Our thinking about what is a good programming language has also changed as we accumulate more experience with computers.


PROGRAMMER'S REFERENCE MANUAL
Fortran
AUTOMATIC
CODING
SYSTEM
FOR
THE IBM 704

Some references

- *Programming Language Pragmatics (4th Edition)*, by Michael L. Scott.
- *Essentials of Programming Languages (3rd Edition)*, by Daniel P. Friedman and Mitchell Wand.
- *Concepts of Programming Languages (11th Edition)*, by Robert W. Sebesta.
- `http://hyperpolyglot.org/`

The basic questions

- Which programming language to learn?
- Which programming language to use in *this* project?
- Do I need to learn a *new* language?

Which programming language? I

- Likely to be a large investment.
- Also, you will probably want to be familiar at least with a couple of them (good mental flexibility) plus LaTeX.

Alan Perlis

A language that doesn't affect the way you think about programming is not worth knowing.

- There is a good chance you will need to recycle yourself over your career.

Which programming language? II

- Typical problems in economics can be:
  1. CPU-intensive.
  2. Memory-intensive.
- Imply different emphasis.
- Because of time constraints, we will not discuss memory-intensive tools such as `Hadoop` and `Spark`.

Classification

Classification

- There is no “best” solution.
- But there are some good tips.
- We can classify programming languages according to different criteria.
- We will pick several criteria that are relevant for economists:
  1. Level.
  2. Domain.
  3. Execution.
  4. Type.
  5. Paradigm.

Level

- Levels:
  1. `machine code`.
  2. Low level: `assembly language` like `NASM` (`http://www.nasm.us/`), `GAS`, or `HLA` (*The Art of 64-Bit Assembly*, by Randall Hyde).
  3. High level: like `C`/`C++`, `Julia`, ...
- You can actually mix different levels (`C`).
- Portability.
- You are unlikely to see low level programming unless you get into the absolute frontier of performance (for instance, with extremely aggressive parallelization).

Fibonacci number

Machine code:

```
8B542408 83FA0077 06B80000 0000C383 FA027706 B8010000 00C353BB
01000000 B9010000 008D0419 83FA0376 078BD98B C84AEBF1 5BC3
```

Assembler:

```
ib: mov edx, [esp+8] cmp edx, 0 ja @f mov eax, 0 ret @@: cmp edx, 2 ja @f
mov eax, 1 ret @@: push ebx mov ebx, 1 mov ecx, 1 @@: lea eax, 3 jbe @f mov
ebx, ecx  [ebx+ecx]  cmp edx, mov ecx, eax dec edx jmp @b  @@: pop ebx  ret
```

C++:

```
int fibonacci(const int x) {
        if (x==0) return(0);
        if (x==1) return(1);
        return (fibonacci(x-1))+fibonacci(x-2);}
```

Domain

- Domain:
  1. General-purpose programming languages (GPL), such as `Fortran`, `C/C++`, `Python`, ...
  2. Domain specific language (DSL) such as `Julia`, `R`, `Matlab`, `Mathematica`, ...
- Advantages/disadvantages:
  1. GPL are more powerful, usually faster to run.
  2. DSL are easier to learn, faster to code, built-in functions and procedures.

Execution I

- Three basic modes to run code:
  1. Interpreted: `Python`, `R`, `Mathematica`.
  2. Compiled: `Fortran`, `C/C++`.
  3. JIT (Just-in-Time) compilation: `Julia`, `Matlab`.
- Interpreted languages can we used with:
  1. A command line in a `REPL` (Read–eval–print loop).
  2. A script file.
- Many DSL are interpreted, but this is neither necessary nor sufficient.
- Advantages/disadvantages: similar to GPL versus DSL.
- Interpreted and JIT programs are easier to move across platforms.

Execution II

- In reality, things are somewhat messier.
- Some languages are explicitly designed with an interpreter and a compiler (`Haskell`, `Scala`, `F#`).
- Compiled programs can be extended with third-party interpreters (`CINT` and `Cling` for C/C++).
- Often, interpreted programs can be compiled with an auxiliary tool (`Matlab`, `Mathematica`,...).
- Interpreted programs can also be compiled into byte code (`R`, languages that run on the `JVM` -by design or by a third party compiler).
- We can mix interpretation/compilation with libraries.

Types I

- Type strength:
  1. Strong: type enforced.
  2. Weak: type is tried to be adapted.
- Type expression:
  1. Manifest: explicit type.
  2. Inferred: implicit.
- Type checking:
  1. Static: type checking is performed during compile-time.
  2. Dynamic: type checking is performed during run-time.
- Type safety:
  1. Safe: error message.
  2. Unsafe: no error.

Types II

- Advantages of strong/manifest/static/safe type:
  1. Easier to find programming mistakes⇒`ADA`, for critical real-time applications, is strongly typed.
  2. Easier to read.
  3. Easier to optimize for compilers.
  4. Faster runtime not all values need to carry a dynamic type.
- Disadvantages:
  1. Harder to code.
  2. Harder to learn.
  3. Harder to prototype.

Types III

- You implement strong/manifest/static/safe typing in dynamically typed languages.
- You can define variables explicitly. For example, in `Julia`

```
a = 10::Int
```

- It often improve performance speed and safety.
- You can introduce checks:

```
a = "This is a string"
if typeof(a) == String
    println(a)
else
    println("Error")
end
```

Sep 2021	Sep 2020	Change	Programming Language	Ratings	Change
1	1		C	11.83%	-4.12%
2	3	↑	Python	11.67%	+1.20%
3	2	↓	Java	11.12%	-2.37%
4	4		C++	7.13%	+0.01%
5	5		C#	5.78%	+1.20%
6	6		Visual Basic	4.62%	+0.50%
7	7		JavaScript	2.55%	+0.01%
8	14	⇈	Assembly language	2.42%	+1.12%
9	8	↓	PHP	1.85%	-0.64%
10	10		SQL	1.80%	+0.04%
11	22	⇈	Classic Visual Basic	1.52%	+0.77%
12	17	⇈	Groovy	1.46%	+0.48%
13	15	↑	Ruby	1.27%	+0.03%
14	11	↓	Go	1.13%	-0.33%
15	12	↓	Swift	1.07%	-0.31%
16	16		MATLAB	1.02%	-0.07%
17	37	⇈	Fortran	1.01%	+0.65%
18	9	⇊	R	0.98%	-1.40%
19	13	⇊	Perl	0.78%	-0.53%
20	29	⇈	Delphi/Object Pascal	0.77%	+0.24%

Programming Language	2021	2016	2011	2006	2001	1996	1991	1986
C	1	2	2	2	1	1	1	1
Java	2	1	1	1	3	15	-	-
Python	3	5	6	8	25	24	-	-
C++	4	3	3	3	2	2	2	6
C#	5	4	5	7	13	-	-	-
Visual Basic	6	13	-	-	-	-	-	-
JavaScript	7	7	10	9	9	20	-	-
PHP	8	6	4	4	10	-	-	-
SQL	9	-	-	-	37	-	-	-
Assembly language	10	11	-	-	-	-	-	-
Ada	31	28	17	16	18	7	3	2
Lisp	33	27	13	13	17	8	6	3
(Visual) Basic	-	-	7	5	4	3	5	5

Language popularity I

- C family (a subset of the ALGOL family), also known as "curly-brackets languages":
  1. C, C++, C#: 24.74%: 3 out of top 5.
  2. C, Java, C++, C#, JavaScript, PHP: 40.26%: 6 out of top 10.
- Python: position 2, 11.67%.
- Matlab: position 16, 1.02%.
- Fortran: position 17, 1.01%.
- R: position 18, 0.98%.
- Julia: position 27, 0.53%.

Language popularity II

- High-performance and scientific computing is a small area within the programming community.
- Thus, you need to read the previous numbers carefully.
- For example:
  1. You will most likely never use `JavaScript` or `PHP` (at least while wearing with your "economist" hat) or deal with an embedded system.
  2. `C#` and `Swift` are cousins of C focused on industry applications not very relevant for you.
  3. `Java` (usually) pays a speed penalty.
  4. `Fortran` is still used in some circles in high-performance programming, but most programmers will never bump into anyone who uses `Fortran`.

Multiprogramming

- Attractive approach in many situations.
- Best `IDEs` can easily link files from different languages.
- Easier examples:
  1. `ccall` and `PyCall` in `Julia`.
  2. `Rcpp`.
  3. `Mex` files in `Matlab`.