Programming Languages: Introduction

Yuh-Jzer Joung
莊 裕 澤
Dept. of Information Management
National Taiwan University

February, 2001

WHAT THEY ARE

Program: a specification of a computation

Programming language: a notion for writing programs. More precisely, a notation for specifying, organizing, and reasoning about computations.
WHAT THEY ARE (cont’d)

According to Stroustrup, a programming language is
– a tool for instructing machines,
– a means for communicating between programmers,
– a vehicle for expressing high-level designs,
– a notation for algorithms,
– a way of expressing relationships between concepts,
– a tool for experimentation,
– a means for controlling computerized devices.

WHY STUDY PROGRAMMING LANGUAGE?

• To improve your understanding of a language
• To help you to write a “good” program
• To make it easier to learn a new language
THE Von Neumann MACHINE

CPU

- Controller
- ALU
- Accumulator

Memory
(Instruction + Data)

K ← I + j
retrieve i
retrieve j
perform i+j
store the result to k

LOW LEVEL LANGUAGES

Machine language is the native language of a computer. It is the notation to which the computer responds directly. Assembly language is a variant of machine language in which names and symbols replace the actual codes for machine operations, values, and storage locations, making individual instructions more readable.

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Machine Instructions</th>
<th>Assembly language</th>
</tr>
</thead>
<tbody>
<tr>
<td>0004</td>
<td>BE 0000</td>
<td>mov si, 0</td>
</tr>
<tr>
<td>0007</td>
<td>B0 00</td>
<td>mov al, 0</td>
</tr>
<tr>
<td>0009</td>
<td>02 84 001C R</td>
<td>count: add al, [DATA+si]</td>
</tr>
<tr>
<td>000D</td>
<td>46</td>
<td>inc si</td>
</tr>
<tr>
<td>000E</td>
<td>83 FE 05</td>
<td>cmp si, 5</td>
</tr>
<tr>
<td>0011</td>
<td>75 F6</td>
<td>jne COUNT</td>
</tr>
<tr>
<td>0013</td>
<td>A2 0021 R</td>
<td>mov RESULT, al</td>
</tr>
<tr>
<td>0016</td>
<td>B4 4C</td>
<td>mov AH, 4ch</td>
</tr>
</tbody>
</table>

; clear SI (counts+points to data)
; clear AX (stores total)
; add memory value to AX
; location of memory is DATA plus value in SI
; add one to SI
; compare SI to 5
; if SI is not 5 then goto COUNT
; store result in memory
; return to dos
TOWARD HIGHER-LEVEL LANGUAGES

Language designers seek a balance between two goals:
– making computing convenient for people
– making efficient use of computing machines

Convenience comes first. Without it, efficiency is irrelevant.

Programming languages were invented to make machines easier to use. They thrive because they make problems easier to solve.

Programming languages are designed to be both higher level and general purpose.
– A language is *higher level* if it is independent of the underlying machine.
– A language is *general purpose* if it can be applied to a wide range of problems.
BENEFITS OF HIGHER-LEVEL LANGUAGES

Higher-level languages have replaced machine language and assembly language in virtually all areas of programming, because they provide benefits like the following:

– Readable, familiar notations
– Machine independence (portability)
– Availability of program libraries
– Consistency checks during implementation that can detect errors

The HUMAN ERROR FACTOR

Due to a programming error, the rocket carrying Mariner I, an unmanned probe to the planet Venus, has to be destroyed 290 seconds after lunch on July 22, 1962. The program in the ground computer was supposed to behave as follows:

if not in radar contact with the rocket then
do not correct its flight path

But, in error, the initial not was missing.
**Excerpts from a Hearing in Congress**

The following is a hearing before the Committee on Science and Astronautics, U.S. House of Representatives, July 31, 1962. Mr. Wyatt and Dr. Morrison represented NASA.

CHAIRMAN: Who was responsible for leaving this [not] out?

Mr. WTATT: It was a human error …

Mr. FULTON: Dose NASA check to see that the computers are correctly programmed? Doesn’t any outside inspector check, or is it just up to the programmer and if he does not do it nobody else knows about it?

Dr. MORRISON: This is a minute detail of the program, which I agree should be checked. However, in good management practices, if we followed every detail to this point, we would have a tremendous staff.

Mr. FULTON: …the loss up to $18 or $20 million, plus the time, plus the loss of prestige… would seem to me to require a system of checking to see that the contractor programmed correctly.

---

**Excerpts from a Hearing in Congress (Cont’d)**

Mr. MORRISON: This is true. I would like to point out there were 300 runs made of this [programmed; the error was not uncovered]…

Mr. FULTON: My point is that we know of one [error], but we do not know if there were others …

Mr. WAGGONNER: … I share your concern there. I would have to be reluctant to say we hire enough personnel to check every programmer. That would mean… two people doing every job - a man checking every man …

CHAIRMAN: …I feel that I have a vague knowledge of what you are talking about, but we certainly should be able to devise some system for checks that will not allow this type of error to creep in.
A ROLE FOR PROGRAMMING LANGUAGES

Code inspection and program testing are two common techniques for detecting program errors.

But as Dijkstra said, program testing can be used to show the presence of bugs, but never to show their absence.

Programming verification is to prove correctness of a program.

However, programming verification does not scale well. Instead, we must organize the computations in such a way that our limited powers are sufficient to guarantee that the computation will establish the desired effect.

In a nutshell, a role for programming languages is to provide ways of organizing computations.

---

PROBLEMS OF SCALE

Techniques for dealing with small programs do not necessarily scale up.

Any one change to a program is easy to make. But, the effect of a change can ripple through the program, perhaps introducing errors or bugs into some forgotten corner.

Structure and organization are the key to managing large programs.

Programming languages can help in two ways:

- Their readable and compact notations reduce the likelihood of errors.
- They provide ways of organizing computations so that they can be understood one piece at a time.
Programming paradigms are ways of thinking about programming.

**Imperative Programming**

Imperative languages are action oriented; that is, a computation is viewed as a sequence of actions.

E.g., Algol, Pascal, C, etc.

**Functional Programming**

Simply put, functional programming is programming without assignments.

E.g., Lisp, Scheme, ML, etc.

**Object-Oriented Programming**

Central to object-oriented programming is the concept of objects and their classification into classes and subclasses.

E.g., Smalltalk, C++, Java, etc.

**Logic Programming**

A kind of rule-based programming.

E.g., Prolog

---

**Imperative Programming**

\[
\text{factorial } (n) = \begin{cases} 
1 & n = 0 \\
 n \times \text{factorial } (n-1) & n > 1 
\end{cases}
\]

```
int factorial (int n) {
    int i, fact=1;
    for (i=1; i<=n; i++)
        fact = fact * i;
    return fact;
}
```
**Functional Programming**

\[
\text{factorial}(n) = \begin{cases} 
1 & n = 0 \\
 n \times \text{factorial}(n-1) & n > 1 
\end{cases}
\]

\[
\text{fun factorial}(n) = \\
\text{if } n=0 \text{ then } 1 \text{ else } n \times \text{factorial}(n-1)
\]

**OBJECT-ORIENTED Programming**

```java
class small {
    private int a, b, c;
    public void draw( ) { ... }
}
class large extends small {
    private int a, b, c;
    public void draw( ) { ... }
    public void sift( ) { ... }
}
```
Logical Programming

mother(nancy,joe).
father(jonson,nancy).
parent(x,y) :- mother(x,y).
parent(x,y) :- father(x,y).
grandparent(x,z) :- parent(x,y), parent(y,z).
sibling(x,y) :- mother(m,x), mother(m,y), father(f,x), father(f,y)

> grandparent(jonson,joe).
> yes

LANGUAGE IMPLEMENTATION

There are two basic approaches to implementing a program in a higher-level language:

• Compilation
  – The language is brought down to the level of the machine, using a translator called a **compiler**.

• Interpretation
  The machine is brought up to the level of the language, by building a higher-level machine called an **interpreter**.
Compilation vs. Interpretation

Compilation is biased toward static properties, while interpretation can deal with dynamic properties. They can be compared as follows:

Compilation can be more efficient than interpretation

Unlike a compiler, which translates the source program once and for all, an interpreter examines the program repeatedly.

Interpretation can be more flexible than compilation

An interpreter allows programs to be changed "on the fly" to add features or correct errors. It can also pinpoint an error in the source text and report it accurately.

Language Development
A goal in the late 1980s was to make the retrieval of information easy. The breakthrough came in 1989 at CERN, for the development of WWW and HTML.

The Web poses new issues to programming languages:

- security
- performance
- platform independence

Some successful achievements are Java, HTML, and XML.
WHAT MAKES A GOOD LANGUAGE?

• Clarity, simplicity, and unity
• Naturalness for the application
• Support for abstraction
• Ease of program verification
• Programming environment
• Portability
• Cost of use
  – cost of program execution
  – cost of program translation
  – cost of program creation, testing, and use
  – cost of program maintenance