Programming Languages: Introduction

Yuh-Jzer Joung
莊裕澤
Dept. of Information Management
National Taiwan University
February, 2005

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

Memory

(Instruction + Data)

Control

Register

ALU

Controller

Accumulator

K = i + j

Von Neumann bottleneck!

parallel computers

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.

Machine Instructions

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Machine Instructions</th>
<th>Assembly Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>MOV AX, 0</td>
<td>mov ax, 0</td>
</tr>
<tr>
<td>0001</td>
<td>MOV BX, 0</td>
<td>mov bx, 0</td>
</tr>
<tr>
<td>0002</td>
<td>MOV CX, 0</td>
<td>mov cx, 0</td>
</tr>
<tr>
<td>0003</td>
<td>MOV DX, 0</td>
<td>mov dx, 0</td>
</tr>
<tr>
<td>0004</td>
<td>MOV SI, 0</td>
<td>mov si, 0</td>
</tr>
<tr>
<td>0005</td>
<td>MOV DI, 0</td>
<td>mov di, 0</td>
</tr>
<tr>
<td>0006</td>
<td>MOV BP, 0</td>
<td>mov bp, 0</td>
</tr>
<tr>
<td>0007</td>
<td>MOV SP, 0</td>
<td>mov sp, 0</td>
</tr>
</tbody>
</table>

Assembly labguage
**RANDOM-ACCESS MACHINE (RAM)**

```plaintext
PROGRAM
1: M[0] := 0
2: read (M[1])
3: if M[1] ≥ 0 then goto 5
4: goto 7
6: if M[3] ≥ 0 then goto 16
7: writeln (M[1])
8: read (M[2])
10: if M[3] ≥ 0 then goto 12
11: goto 14
13: if M[3] ≥ 0 then goto 8
15: goto 3
16: halt
```

**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.

**Influences on Language Design**

- **Computer Architecture**
  - The Von Neumann Architecture
  - Use of variables to model memory
  - Assignments for changing the content of the memory
  - Simple control flow structures for modeling control
- **Programming Methodologies**
  - Top-down design and stepwise refinement
  - Shift from process-oriented to data-oriented
  - Toward object-oriented programming
- **Software Life Cycle**
  - Maintenance is much more important than coding
  - Readability is the key issue.

**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 launch 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
does 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 out?

Mr. WATT: It was a human error …

Mr. FULTON: Does 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.

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.

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 Domains

- Scientific applications
  - FORTRAN
    - Simple data structures (arrays and matrices)
    - Floating-point arithmetic computation
    - Efficiency
- Business applications
  - COBOL
    - Elaborate reports generation
    - Decimal numbers and character data processing
    - Now generally been replaced by Spreadsheets and Database tools
- Artificial intelligence
  - LISP, Prolog
    - Symbolic computation rather than numeric computation
    - Run-time code generation and execution

Programming Domains (cont.)

- Systems programming
  - e.g., PL1, C
    - Efficiency
    - Low-level features
    - Flexibility, less safety restrictions
- Scripting languages
  - sh, awk, tcl, perl, JavaScript, PHP, Python
    - A script is a file of commands to be executed by the systems
- Special purpose languages
  - e.g., GPSS for simulation
Script Languages --- bash

```bash
# If this is an xterm set the title to user@host-dir
if [ -f /etc/bash_completion ]; then
  . /etc/bash_completion
fi

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```

---

Imperative Programming

```plaintext
factorial (n) = \[
\begin{align*}
&1 & n = 0 \\
&n \times \text{factorial} (n - 1) & n > 1 \\
\end{align*}
\]
```

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

---

Functional Programming

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

---

PROGRAMMING PARADIGMS

- 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.
  - Actions take effect by modifying some memory space (denoted by variables).
  - E.g., Algol, Pascal, C, etc.
- **Functional Programming**
  - Simply put, functional programming is programming without assignments.
  - To mimic mathematical functions to the greatest extent
  - E.g., Lisp, Scheme, ML, etc.

---

OBJECT-ORIENTED Programming

```java
class small {
  private int a, b, c;
  public void draw() { ... }
}
```

```java
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
```

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 also can pinpoint an error in the source text and report it accurately.

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 language is brought up to the level of the machine, by building a higher-level machine called an interpreter (acts as a simulator of the program.)

```
Source program  Compiler  Target code
```

```
Source program  Input  Interpreter  Output
```

Hybrid Implementation System

- The initial implementation of Java were all hybrid. Its intermediate form (byte code) provides portability to any machine that has a byte code interpreter and an associated run-time system (the Java Virtual Machine).
- What is the benefit of hybrid implementation?

The Compilation Process

- Before execution, linking must be performed to link and load other programs called by the main program.

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.

Script languages to add dynamic to Web pages:
- JavaScript: client-side, HTML-embedded scripting language
- PHP: server-side, HTML-embedded scripting language

### 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

### Programming Environments

Programming Environments: The collection of tools used in software development

- UNIX: An old operating system and tool collection
- Borland JBuilder: A PC environment for Java
- Smalltalk: A language processor/environment
- Microsoft Visual C++: A large, complex visual environment