Introduction and motivation

References:

2 exam questions.

---

Practicalities

- Lectures.

---

Goals

- Critical *thinking* about programming languages.
  - What is a programming language!?
- *Study* programming languages.
  - Be familiar with basic language *concepts*.
  - Appreciate trade-offs in language *design*.
- Trace *history*, appreciate *evolution* and diversity of *ideas*.
- Be prepared for new programming *methods, paradigms*.
Why study programming languages?

- To improve the ability to develop effective algorithms.
- To improve the use of familiar languages.
- To increase the vocabulary of useful programming constructs.
- To allow a better choice of programming language.
- To make it easier to learn a new language.
- To make it easier to design a new language.

What makes a good language?

- Clarity, simplicity, and unity.
- Orthogonality.
- Naturalness for the application.
- Support of abstraction.
- Ease of program verification.
- Programming environments.
- Portability of programs.

Cost of use.
- Cost of execution.
- Cost of program translation.
- Cost of program creation, testing, and use.
- Cost of program maintenance.

Influences

- Computer capabilities.
- Applications.
- Programming methods.
- Implementation methods.
- Theoretical studies.
- Standardisation.
Applications domains

<table>
<thead>
<tr>
<th>Era</th>
<th>Application</th>
<th>Major languages</th>
<th>Other languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960s</td>
<td>Business</td>
<td>COBOL</td>
<td>Assembler</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORTRAN, Assembler</td>
<td>ALGOL, BASIC, APL</td>
</tr>
<tr>
<td></td>
<td>Scientific</td>
<td>LISP</td>
<td>JOVIAL, Forth</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SNOBOL</td>
</tr>
<tr>
<td>Today</td>
<td>Business</td>
<td>COBOL, SQL, spreadsheet</td>
<td>C, PL/I, 4GLs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORTRAN, C, C++</td>
<td>BASIC, Pascal</td>
</tr>
<tr>
<td></td>
<td>Scientific</td>
<td>Maple, Mathematica</td>
<td>Pascal, Ada, BASIC,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BCPL, C, C++</td>
<td>MODULA</td>
</tr>
<tr>
<td></td>
<td>System</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AI</td>
<td>LISP, Prolog</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Publishing</td>
<td>TeX, Postscript,</td>
<td>Marvel, Esterel</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>word processing</td>
<td>Eiffel, C#</td>
</tr>
<tr>
<td></td>
<td>New paradigms</td>
<td>UNIX shell, TCL, Perl</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smalltalk, ML, Haskell,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java, Python, Ruby</td>
<td></td>
</tr>
</tbody>
</table>

Motivating application in language design

A specific purpose provides *focus* for language designers; it helps to set criteria for making design decisions.

A specific, motivating application also helps to solve one of the hardest problems in programming language design: deciding which features to leave out.

Examples: Good languages designed with a specific purpose in mind.

- **LISP**: symbolic computation, automated reasoning
- **FP**: functional programming, algebraic laws
- **BCPL**: compiler writing
- **Simula**: simulation
- **C**: systems programming
- **ML**: theorem proving
- **Smalltalk**: Dynabook
- **Clu, ML module system**: modular programming
- **C++**: object orientation
- **Java**: Internet applications

Program execution model

Good language design presents *abstract machine*.

- **FORTRAN**: Flat register machine; memory arranged as linear array
- **LISP**: cons cells, read-eval-print loop
- **Algol family**: stack of activation records; heap storage
- **BCPL, C**: underlying machine + abstractions
- **Simula**: Object references
- **FP, ML**: functions are basic control structure
- **Smalltalk**: objects and methods, communicating by messages
- **Java**: Java virtual machine
Theoretical foundations

Examples:

- Formal-language theory.
- Automata theory.
- Algorithmics.
- λ-calculus.
- Semantics.
- Formal verification.
- Type theory.
- Complexity theory.

Standardisation

- Proprietary standards.
- Consensus standards:
  - ANSI.
  - IEEE.
  - BSI.
  - ISO.

Language standardisation

Consider: int i; i = (1 && 2) + 3 ;

?- Is it valid C code? If so, what's the value of i?
?- How do we answer such questions!?

- Read the reference manual.
- Try it and see!
- Read the ANSI C Standard.

Language-standards issues

Timeliness. When do we standardise a language?

Conformance. What does it mean for a program to adhere to a standard and for a compiler to compile a standard?

Ambiguity and freedom to optimise — Machine dependence — Undefined behaviour.

Obsolescence. When does a standard age and how does it get modified?

Deprecated features.
What does the following mean?

```c
#include <stdio.h>
main() {
    int t = 1;
    int t0 = 0;
    t0 = (t=t+1) + ++t;
    printf("t0=%d t=%d\n",t0,t);

    int u = 1;
    int u0 = 0;
    u0 = ++u + (u=u+1);
    printf("u0=%d u=%d\n",u0,u);

    int y = 1;
    int y0 = 0;
    intppy() { return ++y; }
    y0 = ppy() + ppy();
    printf("y0=%d y=%d\n",y0,y);

    int z = 1;
    int z0 = 0;
    z0 = ++z;
    z0 += ++z;
    printf("z0=%d z=%d\n",z0,z);
}
```

```c
int w = 1;
int w0 = 0;
w0 = (w=w+1) + (w=w+1);
printf("w0=%d w=%d\n",w0,w);

int x = 1;
int x0 = 0;
x0 = ++x + ++x;
printf("x0=%d x=%d\n",x0,x);
```

**Answer:**

- **Linux (gcc, cc)**
  - t0 = 5 u0 = 6 w0 = 5 x0 = 6 y0 = 5 z0 = 5

- **Mips (gcc)**
  - t0 = 5 u0 = 5 w0 = 5 x0 = 5 y0 = 5 z0 = 5

- **DEC Alpha and Sun4 (gcc, cc)**
  - t0 = 5 u0 = 5 w0 = 5 x0 = 5 y0 = 5 z0 = 5
Language standards
PL/1

? What does the following mean?

\[9 + \frac{8}{3}\]

- \(11.666...\) ?
- Overflow ?
- \(1.666...\) ?

DEC(p,q) means p digits with q after the decimal point.

Type rules for DECIMAL in PL/1:

\[
\begin{align*}
\text{DEC}(p_1,q_1) + \text{DEC}(p_2,q_2) & \Rightarrow \text{DEC}(\text{MIN}(1+\text{MAX}(p_1-q_1,p_2-q_2)+\text{MAX}(q_1,q_2),15),\text{MAX}(q_1,q_2)) \\
\text{DEC}(p_1,q_1) / \text{DEC}(p_2,q_2) & \Rightarrow \text{DEC}(15,15-((p_1-q_1)+q_2))
\end{align*}
\]

For \(9 + \frac{8}{3}\) we have:

\[
\begin{align*}
\text{DEC}(1,0) + \text{DEC}(1,0)/\text{DEC}(1,0)
\Rightarrow \text{DEC}(1,0) + \text{DEC}(15,15-((1-0)+0)) \\
\Rightarrow \text{DEC}(1,0) + \text{DEC}(15,14) \\
\Rightarrow \text{DEC}(\text{MIN}(1+\text{MAX}(1-0,15-14)+\text{MAX}(0,14),15),\text{MAX}(0,14)) \\
\Rightarrow \text{DEC}(15,14)
\end{align*}
\]

So the calculation is as follows

\[
\begin{align*}
9 + \frac{8}{3}
\Rightarrow 9 + 2.66666666666666 - \text{OVERFLOW}
\Rightarrow 11.66666666666666 - \text{OVERFLOW disabled}
\end{align*}
\]

History


1956–60: FORTRAN, COBOL, LISP, Algol 60.

1961–65: APL notation, Algol 60 (revised), SNOBOL, CPL.

1966–70: APL, SNOBOL 4, FORTRAN 66, BASIC, SIMULA, Algol 68, Algol-W, BCPL.

1971–75: Pascal, PL/1 (Standard), C, Scheme, Prolog.

1976–80: Smalltalk, Ada, FORTRAN 77, ML.
2000–05: C#, Python, Ruby.

Things to think about
♦ What makes a good language?
♦ The role of
  1. motivating applications,
  2. program execution,
  3. theoretical foundations in language design.
♦ Language standardisation.

Language groups
♦ Multi-purpose languages
  ♦ C#, Java, C++, C
  ♦ Haskell, ML, Scheme, LISP
♦ Scripting languages
  ♦ Perl, TCL, UNIX shell
♦ Special-purpose languages
  ♦ SQL
  ♦ \LaTeX