Topic I

Introduction and motivation

References:

♦ **Chapter 1** of *Concepts in programming languages* by J. C. Mitchell. CUP, 2003.


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.
♦ To simulate useful features in languages that lack them.
♦ To make better use of language technology wherever it appears.
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.
What makes a language successful?

♦ Expressive power.
♦ Ease of use for the novice.
♦ Ease of implementation.
♦ Open source.
♦ Excellent compilers.
♦ Economics, patronage, and inertia.
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>Scientific</td>
<td>FORTRAN</td>
<td>ALGOL, BASIC, APL</td>
</tr>
<tr>
<td></td>
<td>System</td>
<td>Assembler</td>
<td>JOVIAL, Forth</td>
</tr>
<tr>
<td></td>
<td>AI</td>
<td>LISP</td>
<td>SNOBOL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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>Scientific</td>
<td>FORTRAN, C, C++</td>
<td>BASIC, Pascal</td>
</tr>
<tr>
<td></td>
<td>System</td>
<td>BCPL, C, C++</td>
<td>Pascal, Ada, BASIC, MODULA</td>
</tr>
<tr>
<td></td>
<td>AI</td>
<td>LISP, Prolog</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Publishing</td>
<td>(\TeX), Postscript, word processing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>UNIX shell, TCL, Perl</td>
<td>Marvel, Esterel</td>
</tr>
<tr>
<td></td>
<td>New paradigms</td>
<td>Smalltalk, SML, Haskell, Java</td>
<td>Eiffel, C#, Scala</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python, Ruby</td>
<td></td>
</tr>
</tbody>
</table>
Why are there so many languages?

♦ Evolution.

♦ Special purposes.

♦ Personal preference.
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, SML Modules: 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
Classification of programming languages

♦ Imperative

  procedural  C, Ada, Pascal, Algol, FORTRAN, . . .
  object oriented  Scala, C#, Java, Smalltalk, SIMULA, . . .
  scripting  Perl, Python, PHP, . . .

♦ Declarative

  functional  Haskell, SML, Lisp, Scheme, . . .
  logic  Prolog
  dataflow  Id, Val
  constraint-based  spreadsheets
  template-based  XSLT
Theoretical foundations

Examples:

♦ Formal-language theory.
♦ Automata theory.
♦ Algorithmics.
♦ $\lambda$-calculus.
♦ Semantics.
♦ Formal verification.
♦ Type theory.
♦ Complexity theory.
♦ Logic.
Standardisation

♦ Proprietary standards.

♦ Consensus standards.
  ♦ ANSI (American National Standards Institute)
  ♦ IEEE (Institute of Electrical and Electronics Engineers)
  ♦ BSI (British Standard Institute)
  ♦ ISO (International Standards Organisation)
Language standardisation

Consider: \texttt{int }i; \ i = (1 \&\& 2) + 3 ;

\begin{itemize}
  \item Is it valid C code? If so, what’s the value of \texttt{i}?
  \item How do we answer such questions!?
\end{itemize}

\begin{itemize}
  \item Read the reference manual.
  \item Try it and see!
  \item Read the ANSI C Standard.
\end{itemize}
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.
Language standards
PL/1

What does the following mean?

$9 + \frac{8}{3}$

- $11.666\ldots$ ?
- Overflow ?
- $1.666\ldots$ ?
$\text{DEC}(p, q)$ means $p$ digits with $q$ after the decimal point.

Type rules for $\text{DECIMAL}$ in PL/1:

$\text{DEC}(p_1, q_1) + \text{DEC}(p_2, q_2)$

$\Rightarrow \text{DEC}(\text{MIN}(1+\text{MAX}(p_1-q_1, p_2-q_2)+\text{MAX}(q_1, q_2), 15), \text{MAX}(q_1, q_2))$

$\text{DEC}(p_1, q_1) / \text{DEC}(p_2, q_2)$

$\Rightarrow \text{DEC}(15, 15-(\text{(p}_1-\text{q}_1)+\text{q}_2))$
For $9 + \frac{8}{3}$ we have:

$$\text{DEC}(1,0) + \text{DEC}(1,0)/\text{DEC}(1,0)$$
$$\Rightarrow \text{DEC}(1,0) + \text{DEC}(15,15-((1-0)+0))$$
$$\Rightarrow \text{DEC}(1,0) + \text{DEC}(15,14)$$
$$\Rightarrow \text{DEC}(\min(1+\max(1-0,15-14)+\max(0,14),15),\max(0,14))$$
$$\Rightarrow \text{DEC}(15,14)$$

So the calculation is as follows

$$9 + \frac{8}{3}$$
$$= 9 + 2.66666666666666$$
$$= 11.66666666666666 - \text{OVERFLOW}$$
$$= 1.66666666666666 - \text{OVERFLOW disabled}$$
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, Scala.
Language groups

♦ Multi-purpose languages
  ♦ Scala, C#, Java, C++, C
  ♦ Haskell, SML, Scheme, LISP
  ♦ Perl, Python, Ruby

♦ Special-purpose languages
  ♦ UNIX shell
  ♦ SQL
  ♦ LATEX
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.