~ Lecture I ~

Introduction and motivation

References:

- **Chapter 1** of *Concepts in programming languages* by J. C. Mitchell. CUP, 2003.
- **Chapter 1** of *Programming languages: Design and implementation* (3RD EDITION) by T. W. Pratt and M. V. Zelkowitz. Prentice Hall, 1999.

Practicalities

- Lectures.
  - [1] Introduction and motivation.
  - [2] The first *procedural* language: FORTRAN (1954–58).
  - [3] The first *declarative* language: LISP (1958–62).
  - [4] *Block-structured* procedural languages: Algol (1958–68), BCPL (1967), Pascal (1970), C (1971–78).
  - [5] *Object-oriented* languages — Concepts and origins: Simula (1964–67), Smalltalk (1971–80).
  - [6,7] *Types*, *data abstraction*, and *modularity*: C++ (1983–98), SML (1984–97).
  - [8] The *state of the art*: Java (1996), C# (2000). (⋆ Andrew Kennedy ⋆)

- 2 exam questions.
- Course web page:

  ⟨www.cl.cam.ac.uk/Teaching/2006/ConceptsPL/⟩

  with *lecture slides* and *reading material*.

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

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

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.

Language standards
C

? What does the following mean?

```
#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 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) ;
```



```
int y = 1 ;
int y0 = 0 ;
int ppy() { 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) ; }
```


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
	(cc)	t0=5	u0=6	w0=5	x0=6	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

`9 + 8/3`

mean?

- `11.666...` ?
- Overflow ?
- `1.666...` ?

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

Type rules for DECIMAL in PL/1:

```
DEC(p1,q1) + DEC(p2,q2)
  => DEC(MIN(1+MAX(p1-q1,p2-q2)+MAX(q1,q2),15),MAX(q1,q2))
DEC(p1,q1) / DEC(p2,q2)
  => DEC(15,15-((p1-q1)+q2))
```


For `9 + 8/3` we have:

```
DEC(1,0) + DEC(1,0)/DEC(1,0)
  => DEC(1,0) + DEC(15,15-((1-0)+0))
  => DEC(1,0) + DEC(15,14)
  => DEC(MIN(1+MAX(1-0,15-14)+MAX(0,14),15),MAX(0,14))
  => DEC(15,14)
```

So the calculation is as follows

```
9 + 8/3
  = 9 + 2.66666666666666
  =    11.66666666666666 - OVERFLOW
  =     1.66666666666666 - OVERFLOW disabled
```


History

1951–55: Experimental use of expression compilers.

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.

1981–85: Smalltalk-80, Prolog, Ada 83.

1986–90: C++, SML, Haskell.

1991–95: Ada 95, TCL, Perl.

1996–2000: Java.

2000–05: **C#**, Python, Ruby.

Language groups

- Multi-purpose languages
  - C#, Java, C++, C
  - Haskell, ML, Scheme, LISP
- Scripting languages
  - Perl, TCL, UNIX shell
- Special-purpose languages
  - 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.