Organization of Programming Languages

Understand how languages are designed and implemented

- Syntax -- What a program looks like
- Semantics -- What a program means
- Implementation -- How a program executes

Understand most appropriate language for solving specific problems, For example:
- Pascal, C -- procedural, statement oriented
- C++, Java, Smalltalk -- Object oriented
- ML, Lisp -- Functional
- Prolog -- Rule-based
Language Goals

• During 1950s--1960s - Compile programs to execute efficiently.
• There is a direct connection between language features and hardware - integers, reals, goto statements
• Programmers cheap; Machines expensive; Keep the machine busy

But today
• Compile programs that are built efficiently
• CPU power and memory very cheap
• Direct connection between language features and design concepts - encapsulation, records, inheritance, functionality, assertions
Why study programming languages?

- To improve your ability to develop effective algorithms
- To improve your use of your existing programming language
- To increase your 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
Evolution of software architecture

- **1950s** - Large expensive mainframe computers ran single programs (Batch processing)

- **1960s** - Interactive programming (time-sharing) on mainframes

- **1970s** - Development of Minicomputers and first microcomputers. Apple II. Early work on windows, icons, and PCs at XEROX PARC

- **1980s** - Personal computer - Microprocessor, IBM PC and Apple Macintosh. Use of windows, icons and mouse

- **1990s** - Client-server computing - Networking, The Internet, the World Wide Web
Attributes of a good language

• Clarity, simplicity, and unity - provides both a framework for thinking about algorithms and a means of expressing those algorithms

• Orthogonality - every combination of features is meaningful

• Naturalness for the application - program structure reflects the logical structure of algorithm

• Support for abstraction - program data reflects problem being solved
Attributes of a good language (continued)

• Ease of program verification - verifying that program correctly performs its required function

• Programming environment - external support for the language

• Portability of programs - transportability of the resulting programs from the computer on which they are developed to other computer systems

• Cost of use - program execution, program translation, program creation, and program maintenance
Language paradigms

Imperative languages

- Goal is to understand a machine state (set of memory locations, each containing a value)
- Statement oriented languages that change machine state (C, Pascal, FORTRAN, COBOL)
- Syntax: S1, S2, S3, ...

Applicative (functional) languages

- Goal is to understand the function that produces the answer
- Function composition is major operation (ML, LISP)
- Syntax: P1(P2(P3(X)))
- Programming consists of building the function that computes the answer
Language paradigms (continued)

Rule-based languages

- Specify rule that specifies problem solution (Prolog, BNF Parsing)
- Other examples: Decision procedures, Grammar rules (BNF)
- Syntax: Answer $\rightarrow$ specification rule
- Programming consists of specifying the attributes of the answer

Object-oriented languages

- Imperative languages that merge applicative design with imperative statements (Java, C++, Smalltalk)
- Syntax: Set of objects (classes) containing data (imperative concepts) and methods (applicative concepts)
Who defines a language?

Is: \( I = 1 \land \land 2 + 3 \mid 4 \); legal in C? What is assigned to I if it is?

3 ways typically to answer this:

• 1. Read language manual (Problem: Can you find one?)
• 2. Read language standard (Problem: Have you ever seen it?)
• 3. Write a program to see what happens. (Easy to do!)

Most do 3, but current compilers may not give correct answer
Creation of standards

Language standards defined by national standards bodies:

• ISO - International Standards organization
• IEEE - Institute of Electrical and Electronics Engineers
• ANSI - American National Standards Institute

All work in a similar way:
1. Working group of volunteers set up to define standard
2. Agree on features for new standard
3. Vote on standard
4. If approved by working group, submitted to parent organization for approval.
Creation of standards

Standards in the US are voluntary:

- There is no federal standards-making organization.
- NIST – National Institute for Standards and Technology develops standards that are only required on federal agencies, not for commercial organizations.

But: Consensus is the key to standards making:

- Contentious features often omitted to gain consensus
- Only vendors have a vested interest in the results
- Users don't care until standard approved, and then it is too late!
Standards conforming programs

Standards define behavior for a *standards conforming* program – one that meets the rules of the language standard

In general (except for Ada), behavior of non-conforming program is not specified, so any extensions to a standards conforming compiler may still be standards conforming, even though the program is not standards conforming.

Standards supposed to be reviewed every 5 years

Examples:
- Ada 1983, 1995

Not quite 5 years, but at least periodically
When to standardize a language?

Problem: When to standardize a language?

- If too late - many incompatible versions - FORTRAN in 1960s was already a de facto standard, but no two were the same
- If too early - no experience with language - Ada in 1983 had no running compilers
- Just right - Probably Pascal in 1983, although it is rapidly becoming a dead language

Other languages:
- C in 1988
- LISP in 1990 - Way too late
- De facto standards: ML - One major implementation - SML
- Smalltalk - none
- Prolog - none
Internationalization

I18N issue – Internationalization – How to specify languages useful in a global economy?

Characters used internationally:
• Single 8-bit byte; usual format today – 256 character values. A lot in 1963, but insufficient today
• ASCII is a 7 bit 128 character code

What about other languages?
• Additional letters: German umlaut-ä, French accent-é, Scandinavian symbols-ö,
• Russian, other alphabets (Greek, Arabic, Hebrew), ideographs (Chinese, Korean)?
• Unicode – 16 bit code allows for 65K symbols. 8-bit byte is insufficient
Internationalization (continued)

I18N name avoids deciding between internationalization and internationalisation

Some of the internationalization issues:

• What character codes to use?
• Collating sequences? - How do you alphabetize various languages?
• Dates? - What date is 10/12/01? Is it a date in October or December?
• Time? - How do you handle time zones, summer time in Europe, daylight savings time in US, Southern hemisphere is 6 months out of phase with northern hemisphere, Date to change from summer to standard time is not consistent.
• Currency? - How to handle dollars, pounds, marks, francs, euros, etc.
Summary

Language design today must:

• Allow program solution to match physical structure of problem
• Allow for world-wide use
• Be easy to prove solution correct

Rest of course will work on these goals