Course Objectives

- Improve the background for choosing appropriate programming languages for certain types of programming problems
- Be able in principle to program in a *procedural*, an *object-oriented*, a *functional*, and a *logical* programming language
- Understand the significance of an implementation of a programming language in a *compiler* or *interpreter*
- Increase the ability to learn new programming languages
- Increase the capacity to express programming concepts and choose among alternative ways to express things in a programming language
- Simulate useful features in languages that lack them
- Be able in principle to design a new programming language
- Make good use of *debuggers* and related tools

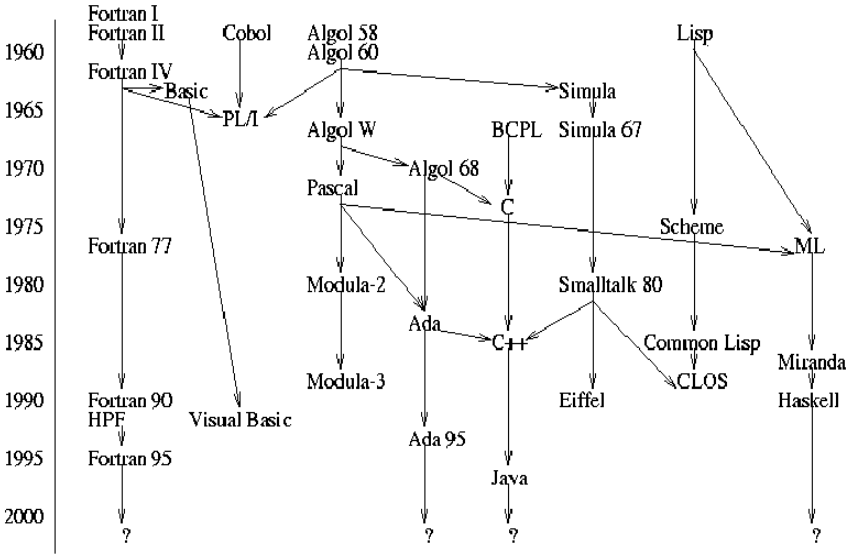
Important Events in Programming Language History

- 1940s: The first electronic computers were monstrous contraptions
  - Programmed in machine code by hand
  - Code not reusable or relocatable
  - Hardware maintainance difficult: cathode tubes regularly burned out
  - The word *bug* denoting programming errors originated from a bug that reportedly roamed around in the hardware making short circuits
- Soon, assembly languages were invented to allow operations to be expressed in mnemonic abbraviations
  - Enables larger, reusable, and relocatable programs
  - Actual machine code produced by assembler
  - Early assembler: one-to-one correspondance between assembly and machine instructions
  - Later: macro expansion into multiple machine instructions to achieve a form of higher-level programming

Important Events in Programming Language History (cont'd)

- Mid 1950s: development of Fortran, the first arguably higher-level language
  - Finally, programs could be developed that were machine independent
  - Main computing activity was numerical computation to solve problems in science and engineering
  - Other high-level languages soon followed:
    - Cobol for Business computing
    - Lisp for symbolic computing and artifical intelligence
    - BASIC for "beginners"
    - C for systems programming
- 1980s: Object-oriented programming
  - Important innovation in software development
  - The concept of a class as a data type abstraction is inherited from Simula 67, a language for discrete event simulation with classes but no inheritance

Programming Language Genealogy



Note: follow this link
`http://www.yahoo.com/Computers_and_Internet/Programmi`
to find a resource of programming languages

Selected Overview of Programming Languages

- Fortran (I, II, IV, 77)
  - Dramatic impact on computing in early days
  - Mainly used for numerical computation
  - No recursion
  - Limited data types (no records and no pointers)
  - Limited type checking
  - Very good compilers are available today
- Fortran (90, 95, HPF)
  - Major revisions, eg. recursion, pointers, and records added
  - New control constructs (eg. `while` loop)
  - Extensive set of array operations
  - HPF (High-Performance Fortran) includes constructs for parallel computation
- Lisp
  - The original functional language developed by McCarthy as a realization of Church's lambda calculus
  - Many dialects, including Common Lisp and Scheme
  - Very powerful for symbolic computation using lists (e.g. for artificial intelligence)
  - Implicit memory management (allocate/deallocate) by "garbage collection"
  - Influenced functional programming languages (ML, Miranda, Haskell)

Selected Overview of Programming Languages

- Algol 60
  - The original block-structured language (local variables in a block)
  - First use of Backus-Naur Form (BNF) to formally define grammar
  - All subsequent imperative programming languages are based on it
  - No I/O and no character set, not widely used in US
- Algol 68
  - Large and relatively complex
  - Strong influence on Pascal, C, Ada
- Cobol
  - Originally developed by Department of Defense
  - Intended for business data processing
  - Extensive numerical formatting features and decimal number storage
  - Introduced the concept of records and nested selection statements
- Basic
  - Intended for interactive use (intepreted) and easy for "beginners"
  - Goals: easy to learn and use for non-science students
  - Visual Basic is a popular dialect

Selected Overview of Programming Languages

- PL/I
  - First exception handling
  - First pointer data type
  - Poorly designed, too large, too complex
- Pascal
  - Designed for teaching "structured programming"
  - Small and simple
- Simula 67
  - Primarily designed for discrete-event simulation
  - Based on Algol 60
  - Introduced concept of coroutines
  - Introduced the class concept for data abstraction
- Ada
  - Originally intended to be the standard language for all software commissioned by the Department of Defense
  - Very large
  - Elaborate support for packages, exception handling, generic program units, concurrency
- Ada 95
  - Support for object-oriented programming
  - New concurrency features

Selected Overview of Programming Languages

- Smalltalk-80
  - Developed by XEROX PARC
  - First full implementation of an object-oriented language
  - First design and use of window-based graphical user interfaces (GUIs)
- APL
  - Intended for interactive use ("throw-away" programming)
  - Highly expressive functional language makes programs short, but hard to read
  - Many array operations
- Prolog
  - The most widely used logic programming language
  - Non-procedural (declarative: states what you want, not how to get it)
  - Based on formal logic
- Haskell
  - The leading purely functional language, based on Miranda

Selected Overview of Programming Languages

- C
  - One of the most successful programming languages
  - Designed for systems programming
  - Powerful set of operators, but weak type checking and no dynamic semantic checks
- C++
  - The most successful of several object-oriented successors of C
  - Evolved from C and Simula 67
  - Large and complex, because it supports both procedural and object-oriented programming
- Java
  - Developed by Sun Micorsystems
  - Based on C++, but significantly simplified to make it safe
  - Supports only object-oriented programming
  - Safe language (e.g. no pointers but references, strongly typed, and implicit garbage collection)
  - Truly machine-independent (?)

Note: More about Java

So Why is it That There are so Many Programming Languages?

- *Evolution*
  - This course gives you some insight in what constitutes a good or a bad programming construct. (Appendix B of the textbook has a long list of historical mistakes)
  - Early 70s: "structured programming" in which `goto`-based control flow was replaced by high-level constructs such as `while` loops and `case` statements
  - Late 80s: nested block structure gave way to object-oriented structures
- *Special Purposes*
  - Many languages were designed for a specific problem domain. For example
    - Scientific applications
    - Business applications
    - Artificial intelligence
    - Systems programming
- *Personal Preference*
  - The strength and variety of personal preference makes it unlikely that anyone will ever develop a univerally acceptable programming language

What Makes a Programming Language Successful?

- *Expressive Power*
  - All languages are equally powerful in technical sense (i.e. Turing complete)
  - Language features have a huge impact on the programmer's ability to read, write, maintain, and analyze programs
  - Abstraction facilities enhance expressive power
- *Ease of Use for Novice*
  - Low learning curve and often interpreted, eg. Basic and Logo
- *Ease of Implementation*
  - Runs on virtually everything, eg. Basic, Pascal, and Java
  - Freely available
- *Excellent Compilers*
  - Fortran has extremely good compilers (because it lacks recursion and pointers) and is therefore popular for numerical applications
  - Supporting tools to help the programmer manage very large projects, e.g. Visual C++
- *Economics, Patronage, and Inertia*
  - Powerful sponsor: Cobol, PL/I, Ada
  - Some languages remain widely used long after "better" alternatives because of a huge base of installed software and programmer experience

Classification of Programming Languages

Declarative: Implicit solution "What the computer should solve"	*Functional* (Lisp, Scheme, ML, Haskell) *Logic* (Prolog) *Dataflow*
Imperative: Explicit solution "How the computer should solve"	*Procedural* "von Neumann" (Fortran, C) *Object-oriented* (Smalltalk, C++, Java)

- Declarative functional example (Haskell)

```
gcd a b
  | a == b = a
  | a >  b = gcd (a-b) b
  | a <  b = gcd a (b-a)
```

- Declarative logic example (Prolog)

```
gcd(A, A, A).
gcd(A, B, G) :- A > B, N is A-B, gcd(N, B, G).
gcd(A, B, G) :- A < B, N is B-A, gcd(A, N, G).
```

- Imperative procedural example (C)

```
int gcd(int a, int b)
{ while (a != b)
    if (a > b) a = a-b; else b = b-a;
  return a;
}
```

Classification of Programming Languages (cont'