1. Introduction

Summary of Course Objectives

- Improve the background for choosing appropriate programming languages
- 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 capacity to express general programming concepts and to choose among alternative ways to express things in a particular programming language
- Simulate useful features in languages that lack them
- Be able write programs that parse and translate (programming) languages
- Be able in principle to design a new programming language

Course Outline

1. Introduction: History of programming languages, classification of languages, and overview of languages
2. Functional Programming: Scheme
3. Compilers and Interpreters: How programs are translated to machine code and executed
4. Syntax: How syntax is defined and how syntax can impact the use of languages
5. Semantics: How the meaning and behavior of programming constructs are defined
6. Names, Scopes, and Bindings: How and when bindings for local names are defined in languages with scoping rules
7. Control Flow: How programming constructs define control flow and how these constructs can affect programming style
8. Subroutines and Parameter Passing: How the subrouting calling mechanism is implemented and how and when parameters are passed
9. Exception Handling: How to improve the stability and robustness of your programs
10. Logic Programming: Prolog

Note: This Introduction starts with Chapter 1 of the textbook. Read Chapter 1 Sections 1.1 to 1.3. Study the on-line programming examples and explanations by following the “[read more]” links given in the slides below.

Important Events in Programming Language History

- Programming languages are central to Computer Science
- 1940s: The first electronic computers were monstrous contraptions
  - Programmed in binary machine code by hand
  - Code is not reusable or relocatables
  - Computation and machine maintenance were difficult: cathode tubes regularly burned out
  - The term “bug” originated from a bug that reportedly roamed around in a machine causing short circuits
- Assembly languages were invented to allow machine operations to be expressed in mnemonic abbreviations [read more]
  - Enables larger, reusable, and relocatable programs
  - Actual machine code is produced by an assembler
  - Early assemblers had a one-to-one correspondence between assembly and machine instructions
  - Later: expansion of macros 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 in the 50s: solve numerical problems in science and engineering
  - Other high-level languages soon followed:
    - Algol 58 is an improvement compared to Fortran
    - Cobol for business computing
    - Lisp for symbolic computing and artificial intelligence
    - BASIC for “beginners”
    - C for systems programming
- 1980s: Object-oriented programming
  - Important innovation for software development
  - The concept of a class is based on the notion of data type abstraction from Simula 67, a language for discrete event simulation that has classes but no inheritance
Selected Overview of Programming Languages

- **Fortran (I, II, IV, 77)** [read more]
  - Had a dramatic impact on computing in early days
  - Is 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)** [read more]
  - Major revisions, e.g., recursion, pointers, and records added
  - New control constructs (e.g., `while` loop)
  - Extensive set of array operations
  - HPF (High-Performance Fortran) includes constructs for parallel computation
- **Lisp (Common Lisp and Scheme)** [read more]
  - The original functional language developed by McCarthy as a realization of Church’s lambda calculus.
  - Many dialects exist, including Common Lisp and Scheme
  - Very powerful for symbolic computation with lists (e.g., for artificial intelligence), garbage collection
  - Influenced functional programming languages (ML, Miranda, Haskell)
  - Primarily designed for discrete-event simulation
  - Based on Algol 60
  - Small and simple
  - Introduce concept of coroutines
- **PL/I** [read more]
  - Introduced exception handling
  - First language with pointer data type
  - Poorly designed, too large, too complex
- **Pascal** [read more]
  - Designed for teaching "structured programming"
  - Small and simple
  - Introduce concept for data abstraction
  - Based on Algol 60
  - Primarily designed for discrete-event simulation
  - Introduced concept of coroutines
  - Introduced the class concept for data abstraction
- **Ada (Ada 83)** [read more]
  - 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
- **Basic** [read more]
  - Intended for interactive use (interpreted) and easy for "beginners"
  - Goals: easy to learn and use for non-science students
  - Structure of early basic dialects were similar to Fortran
  - Visual Basic is a popular dialect

- **Algol 60** [read more]
  - The original block-structured language (local variables in a statement block)
  - First use of Backus-Naur Form (BNF) to formally define language grammar
  - All subsequent imperative programming languages are based on it
  - No I/O and no character set, not widely used in US
- **Algol 68**
  - Based on Algol 60 but large and relatively complex
  - Strong influence on Pascal, C, Ada
- **Cobol** [read more]
  - Originally developed by the 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
- **Ada** [read more]
  - Intended for interactive use (interpreted) and easy for "beginners"
  - Goals: easy to learn and use for non-science students
  - Structure of early basic dialects were similar to Fortran
  - Visual Basic is a popular dialect

- **Simula 67**
  - Based on Algol 60
  - Primarily designed for discrete-event simulation
  - Introduced concept of coroutines
  - Introduced the class concept for data abstraction
- **PL/I** [read more]
  - Introduced exception handling
  - First language with pointer data type
  - Poorly designed, too large, too complex
- **Pascal** [read more]
  - Designed for teaching "structured programming"
  - Small and simple
- **Ada (Ada 83)** [read more]
  - 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 [read more]
  - 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 [read more]
  - The most widely used logic programming language
  - Non-procedural (declarative: states what you want, not how to get it)
  - Based on formal logic
- Haskell [read more]
  - The leading purely functional language, based on Miranda

Selected Overview of Programming Languages

- C [read more]
  - One of the most successful programming languages
  - Primarily designed for systems programming but used more widely
  - Powerful set of operators, but weak type checking and no dynamic semantic checks
- C++ [read more]
  - The most successful of several object-oriented successors of C
  - Evolved from C and Simula 67
  - Large and complex, partly because it supports both procedural and object-oriented programming
- Java [read more]
  - Developed by Sun Microsystems
  - Based on C++, but significantly simplified
  - Supports only object-oriented programming
  - Safe language (e.g. no pointers but references, strongly typed, and implicit garbage collection)
  - Portable and machine-independent with Java virtual machine (JVM)
- C# [read more]
  - Similar to Java and C++, but platform dependent (MS .NET)
  - Common Language Runtime (CLR) manages objects that can be shared among the different languages in .NET

So Why are There so Many Programming Languages?

- Evolution
  - What constitutes a good or a bad programming construct? See Appendix B of the textbook for a list of design considerations
  - 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
    - Internet programming
- Personal Preference
  - The strength and variety of personal preference makes it unlikely that anyone will ever develop a universally accepted 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, e.g. Basic, Pascal, and Java
  - Freely available, e.g. Java
- 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

<table>
<thead>
<tr>
<th>Declarative</th>
<th>Functional</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit</td>
<td>(Lisp, Scheme, ML, Haskell)</td>
<td>(Prolog)</td>
</tr>
<tr>
<td>&quot;What the computer should do&quot;</td>
<td>&quot;How the computer should do it&quot;</td>
<td></td>
</tr>
</tbody>
</table>

- Imperative: Explicit
- Procedural: "von Neumann" (Fortran, C)
- Object-oriented: (Smalltalk, C++, Java)

Classification of Programming Languages (cont'd)

```
<table>
<thead>
<tr>
<th>1900</th>
<th>1950</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperative</td>
<td>Hybrid</td>
<td>Object-oriented</td>
</tr>
<tr>
<td>Ada 95</td>
<td>Ada 95</td>
<td>Ada 95</td>
</tr>
<tr>
<td>C</td>
<td>C5500</td>
<td>C</td>
</tr>
<tr>
<td>Fortran 77</td>
<td>Fortran 90</td>
<td>Fortran 95</td>
</tr>
<tr>
<td>Fortran IV</td>
<td>Fortran IV</td>
<td>Pascal</td>
</tr>
<tr>
<td>Algol 68</td>
<td>Algol 68</td>
<td>Smalltalk</td>
</tr>
<tr>
<td>Simula 67</td>
<td>Simula 67</td>
<td>Simula 67</td>
</tr>
<tr>
<td>Scheme</td>
<td>Scheme</td>
<td>Scheme</td>
</tr>
<tr>
<td>ML</td>
<td>ML</td>
<td>ML</td>
</tr>
<tr>
<td>Lisp</td>
<td>Lisp</td>
<td>Lisp</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>Ada 83</td>
<td>Ada 83</td>
<td>Ada 83</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>C++</td>
<td>C++</td>
<td>C++</td>
</tr>
</tbody>
</table>

Note: Looking for more information on programming languages? Then read the on-line textbook Appendix with a list of programming languages and links to resources related to programming languages.

Exercise 1: Write down the different ways in which program loops are written in MIPS assembly, Fortran, C, Pascal, PL/I, C++, Ada 83, and Java.
Exercise 2: Find the persons who were instrumental to the development of Lisp, Pascal, and Simula 67.
Exercise 3: Which organization(s) developed PL/I and why is PL/I not considered a successful programming language?
Exercise 4: Which organization(s) developed Ada?
Exercise 5: Search the Web for an answer to: "What is the most frequently used programming language?" In what area(s) is this language used?
Exercise 6: Search the Web for an answer to: "What is the most popular programming language?" How did you make sure that you can trust the search result?
Exercise 7: Which language(s) is/are good for manipulating symbolic data and complex data structures according to the textbook?