CS 403 – Compiler Construction
Week 1

Overview and History of Programming Languages

What is this course about?

- Programming Language Design *(Minor)*
  - Concepts and Paradigms
  - Ideas and philosophy
  - Syntax and Semantics

- Compiler Construction *(Major)*
  - Tools and Techniques
  - Implementation of Compiler
Programming Languages and Compilers are at the core of Computing

- All software is written in a programming language
- Learning about compilers will teach you a lot about the programming languages you already know.
- Compilers are big, so you need to apply all your knowledge of programming.

What is a Programming Language?

- A set of rules that provides a way of telling a computer what operations to perform
- A set of rules for communicating an algorithm
- A linguistic framework for describing computations
- Symbols, words, rules of grammar, and rules of semantics
  - Syntax and Semantics
Levels of Programming Languages

High-level program

```java
class Triangle {
    float surface()
    return b*h/2;
}
```

Low-level program

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

```
0001001001000101
0010010011101100
101011101001...
```

Types of Programming Languages

- **First Generation Languages**
  - Machine Language
    - `0000 0001 0110 1110`
    - `0100 0000 0001 0010`
- **Second Generation Languages**
  - Assembly Language
    - `LOAD x
      ADD R
      R
      R`
- **Third Generation Languages**
  - High-level imperative/object oriented
    - `public Token scan ( ) {
      while (currentchar == ' ' ||
      currentchar == 'n')
      {
      } },}
  - Fortran, Pascal, Ada, C, C++, Java, C#
- **Fourth Generation Languages**
  - Database
    - `select fname, lname
      from employee
      where department='Sales'
    `- `Lisp, SML, Haskel, Prolog`
- **Fifth Generation Languages**
  - Functional Language
    - `fact n = if n==0 then 1 else n*(fact n-1)`
  - Logical Language
    - `uncle(X,Y) :- parent(Z,Y), brother(X,Z).`
Beyond Fifth Generation Languages

- Agent Oriented Programming
- Aspect Oriented Programming
- Intentional Programming

In Java:
```java
for (int i = 1; i <= 10; i=i+2) {
    system.out.print("the number is " + i);
}
```

In Intentional Programming:
```
print the numbers 1 to 10
```

- Natural language programming

The principal paradigms

- Imperative/Procedural Programming
  - Fortran, Pascal, C
- Object-Oriented Programming
  - Simula, SmallTalk, C++, Java, C#
- Logic/Declarative Programming
  - Prolog
- Functional/Applicative Programming
  - Lisp, Scheme, Haskell, SML, F#, Python
- (Aspect Oriented Programming)
  - AspectJ, AspectC#, Aspect.Net
Programming Language Genealogy

History of Programming Languages

- Punch cards
  - Jacquard looms
  - Analytical engine
    (Charles Babbage and Ada Byron Lovelace)
  - US Census data
    (Herman Hollerith)
- Hand-coded machine language programs
  10110000 01100001
- Assembly language programs
  mov AL 19h
- Modern programming languages
Programming Language Generations

- First Generation (late 1940s):
  
  Machine-level programming languages
  
  - Fast and efficient, executed directly on the CPU
  - Consists only of 0s and 1s
  - Difficult for humans to read, write, and debug

Programming Language Generations

- Second Generation (early 1950s):
  
  Symbolic assemblers
  Interpreting routines
  Very early compilers

Assembly languages
  
  - Simple mnemonic instructions <opcode> <operands>
  - **Assembler** translates into machine code
  - Handcoding in assembly only for low-level needs
Programming Language Generations

- **Third Generation**
  (mid 1950s - present):

  **High level, general-purpose**

  - FORTRAN, LISP, COBOL, ALGOL
    (Ada, Basic, C, C++, Java, Pascal, Smalltalk, …)
  - Easier for humans to read, write, debug
  - **Compiler** translates into machine code before running
  - **Interpreter** translates into machine code at runtime

Programming Language Generations

- **Fourth Generation** (1970s - ):
  Specification languages, query languages, report generators, systems engineering
  - Maple, Mathematica, Postscript, SPSS, SQL

- **Fifth Generation** (1980s - ):
  Solve problems using constraints rather than algorithms, used in Artificial Intelligence
  - Prolog
A family tree of languages

Evolution of third-generation Languages

- Begins with FORTRAN in 1954
- Generation of high-level programming languages
- Languages stress expressivity and machine independence
- Programming is procedural
- Includes imperative & functional, and compiler languages
FORTRAN (1954)

- Designed at IBM to efficiently translate mathematical formulas into IBM 704 machine code. Wanted code at least as efficient as hand-coded.
- Language design was secondary to compiler design for optimization
- 1954 Report for a proposed Formula Translating System
- 1957 FORTRAN language manual published
- Translator produced code that in some cases was more efficient than the equivalent hand-coded program.

LISP (1958)

- Interactive functional language
- Designed for IBM 704 by John McCarthy at Dartmouth 1956-1958
- Language based on lambda calculus. (Mathematical notation for expressing functions.)
- LISP was designed for symbolic formula manipulation. Stands for LiSt Processor.
- Has become standard language of the AI community
ALGOL (1958)

- Designed by international team
- **ALGOrithmic Language**
- Several revisions:
  - ALGOL58
  - ALGOL60
  - ALGOL68
- ALGOL60 had profound influence on programming language design and on computer science. Pascal carries on tradition.
- ALGOL68 was a huge, general purpose language, not widely accepted.
- Language description published in ALGOL60 report
  - First appearance of Backus-Naur Form for programming language definition
  - Widely used as a publication language for algorithms

Cobol (1960)

- US Department of Defense wanted “common” PL for data processing
- CODASYL committee (Conference on Data Systems Languages)
- Result was COBOL in 1960 (COmmon Business-Oriented Language)
- Grace Hopper was involved in development and wrote 1st compiler
- Designed to be machine independent, unlike FORTRAN.
- Influenced by Fortran, ALGOL58, and English.
- Example:
  - Multiply A by B giving C
  - Perform <loop body>
    - Varying J from 2 by 1
    - Until J > N.
APL (early 1960s)

- A Programming Language
- Based on notation developed by Ken Iverson at Harvard 1957-1962.
- Functional, interactive, science-oriented language that assumes the array as the default data structure.
- Suitable for applications with a heavy use of numerical data in large multi-dimensional arrays.
- Used special symbols requiring special keyboard / printer.

```
life=(+/v,v,3 4=+,-1 0 1.0~I 0 10.0<0)
```

BASIC (1964)

- Developed at Dartmouth in 1960’s by Tom Kurtz, John Kemeny, and a succession of undergraduates; first ran in 1964.
- Beginner’s All-purpose Symbolic Instructional Code
- Intended to introduce students in non-scientific disciplines to computing.
- Influenced by FORTRAN and ALGOL.
- Major goal to simplify user interface:
  - Simplicity chosen over efficiency
  - Time sharing over punched cards
  - Distinctions such as int vs real eliminated
  - Automatic defaults for declarations, values, arrays, output format, etc.
  - Clear error messages
  - Students had access to computers at all times
- No universal BASIC standard:
  - ANSI (American National Standards Institute) is a minimal standard.
  - True Basic – Kemeny’s company
PL/1 (1964)

- Planned and designed by IBM as an extension to FORTRAN
- “Extension” departed from FORTRAN specs and was first released as NPL. Renamed PL/1 (Programming Language 1)
- Of interest in academic community because it had every element of language design. Too big and complicated.
- Compiler sold separately from machine
- COBOL and FORTRAN already had huge user bases

ALGOL68

- ALGOL committee produced considerably revised and extended version of ALGOL in 1968.
- Huge, general-purpose language, very different from ALGOL60
- Not well accepted:
  - overly complicated and impractical
  - difficult for compiler writers
- ALGOL68 introduced:
  - User-defined data type
  - Pointer type
  (Both significant features of Pascal)
Pascal (1970)

- Designed by Niklaus Wirth
  - (member of ALGOL committee; he proposed a revision known as ALGOL-W in 1965)
  - Pascal first implemented in 1970.
  - In opposition to trend of PL/1 – ALGOL68 – Ada
  - Named after 17th century French philosopher and mathematician Blaise Pascal.
- **Simple and elegant**
- **Widely used in academic community**
- **Interesting features:**
  - Case statement
  - Facility for user-defined data types
  - Record structure

C (1972)

- Designed by Kenneth Thompson and Dennis Ritchie at Bell Labs in 1972.
- Designed for coding the routines of the UNIX operating system.
- “High level” systems programming language which created the notion of a portable operating system
- Concise syntax – programs somewhat hard to read, understand, debug, maintain
- No built-in operations for handling composite data types such as strings, sets, and lists.
- Not strongly typed. No run-time type checking. Easily leads to programming errors.
- Provides ability to code low-level operations in a high-level language.
Ada

- Designed according to specifications developed by US Department of Defense
- Requirements stressed structural programming methodology and readability over writability
- Development period 1975 – 1985
  - 1975: first requirements documents
  - 1980: complete language proposed
  - 1983: final standardized version
  - 1985: working usable compilers appeared
- Contains virtually all elements of PL design
  - Exception handling
  - Parallel processing
  - Abstract data types

Programming Language Paradigms

- Procedural: procedures, sequential execution of code are basic building blocks of program
  - FORTRAN (FORmula TRANslation; John Backus, IBM, 1950s)
  - ALGOL (ALGOarithmic Language, 1958)
  - COBOL (COmmon Business Oriented Language, 1960)
  - BASIC (Beginner's All-purpose Symbolic Instruction Code, John Kemeny and Thomas Kurtz, Dartmouth, 1963)
  - Pascal (Niklaus Wirth, 1970)
  - C (Dennis Ritchie, Bell Labs, 1972)
Programming Language Paradigms

- **Object-Oriented:** Program is designed around the *objects* required to solve the problem
  - Smalltalk (Alan Kay, Xerox PARC, 1971)
  - Ada (US Dept of Defense, 1975)
  - C++ (Bjarne Stroustrup, Bell Labs, 1983)
  - Java (James Gosling, Sun Microsystems, 1995)
  - C# (Microsoft, 2000)

- **Functional:** Program is designed around the evaluation of *functions*, rather than modifying state
  - LISP (John McCarthy, MIT, 1958)
    - Common Lisp
    - Dylan
    - Logo
    - Scheme
  - ML (Robin Milner et al, Edinburgh, 1970s)
  - Haskell (purely functional language, 1990)
Programming Language Paradigms

- **Logic:** Program is declarative, based on *mathematical logic*
  - Prolog (1972)
    A program lists *facts* and *rules*, program execution is controlled by deduction to answer a *query*.

Programming Language Paradigms

- **Scripting languages** (used for text processing, shells, HTML, CGI)
  - awk (Aho, Weinberger, Kerningham, Bell labs, 1978)
  - Perl (Larry Wall, NASA, 1987)
  - Tcl/Tk (John Ousterhout, 1988)
  - Python (Guido van Rossum, CWI, 1991)
  - PHP (Rasmus Lerdorf, 1995)
  - Ruby (Yukihiro Matsumoto, 1996)