Administrative Details

- You must read the course info sheet. It will also be posted on the course website.
  (http://vww.cs.toronto.edu/~shalev/314/14)
- Required text: Sebesta.
- Note additional recommended references.
- All coding assignments must run on CDF in order to receive credit.
- Late policy.
- Plagiarism.

What is a Programming Language?

A programming language is ...

"a set of conventions for communicating an algorithm," Horowitz

Purposes:

- specify algorithm and data
- communicate to other people
- establish correctness

Course Goals

Studying programming languages will help you to

- increase your vocabulary of programming constructs,
- read and understand manuals,
- learn new languages quickly,
- choose the right language for a task, and
- design a new language,
- be a better programmer.

Course Themes

Principles of programming languages, including:

- formalisms for describing the syntax of a language
- issues in designing data type systems
- issues in designing procedures

Programming language paradigms, including:

- functional programming (exemplified by Scheme)
- logic programming (exemplified by Prolog)

Von Neumann Architecture

Most computers have the following basic structure:

```
CPU                                      Main memory
Instructions                   Data
```

(Named after John von Neumann, one of its originators.)

Memory is separate from the CPU, so instructions and data must be moved between memory and CPU.

The fetch-execute cycle

```
Initialize the program counter
Fetch the instruction pointed to by the PC
Increment the PC
Decode the instruction
Fetch needed data from memory, if any
Execute the instruction
Store the result
Jump
```

Executing an instruction is generally much faster than moving things between memory and CPU.

So the speed of this movement limits the speed of the computer.

This is the "Von Neumann bottleneck!"

Levels of programming language

Machine language

- Operations are very simple things like:
  
  ```
  add contents of memory location 0010 to register 7
  add contents of memory location 0011 to register 7
  shift the contents of register 3 left one bit
  jump to program line 1005
  skip the next instruction if register 1 is zero
  ```

- Instructions are encoded as numbers,
- No variables; operands are memory addresses or register numbers,
- Programming requires deep understanding of the machine architecture,
- Programs are not portable because instructions and their encoding are machine-specific,
- Programs are extremely hard to write, debug, and read.
**Assembly language**
- Operations and operands have symbolic names.
- Can use macros as shorthand for common sequences of code.
- An assembler translates into machine code.
- Still machine dependent.
- Almost as hard to write as machine code.

**High-level language**
- Examples: C, Lisp, Java, Fortran ...
- Have higher level constructs, example:
  ```
  if (x == 3) 
    y = 5; 
  else 
    y = 0; 
  
  
  ```
- Language usually supports type checking and other checks that help detect bugs.
- Programs are much easier to write, debug, and read.
- Programs are now machine independent.
- Programs may be “language implementation dependent”.
- Before the first Fortran compiler (1957), it was commonly believed that any compiler would produce code so terribly inefficient as to be useless.

**Translation**

The process of converting a program written in a high-level language into machine language is called **Translation**.

There are two general methods:

**Compilation**: The whole program is translated before execution.

**Interpretation**: Translate and execute, one statement at a time.

**Comparison of the two methods**
- Brings the program down to the level of the machine.
- Can execute translated program many times because the entire translation is produced.
- Program execution is much faster because the translator can do optimization.
- Harder to provide useful feedback when debugging because executing the target code, once compiled, is not easily altered.

**What Makes a Good Language**

**General goals**:
- The language should be easy to learn.
- Programs written in it should be easy to write and to read.

**Properties of a language that help meet these goals**:
- Minimum number of concepts.
- "Orthogonality": concepts combine systematically, with no exceptions.
- Simple syntax.
- No synonyms.
- Meaning of a construct doesn’t depend on context.

**Pseudo-compilation**: A hybrid of compilation and interpretation.

**Language Paradigms**

**Imperative languages**
- Program statements are commands, Example: “Add 17 to x.”
- Key operations: Assignment, looping, fits the von Neumann architecture closely.
- Examples: Fortran, C, Pascal, Turing.

**Functional languages**
- Program statements describe the value of expressions using (essentially) **lambda expressions**, Example: “The reverse of a list is the list followed by the reverse of the rest of the list (or is empty if the list is empty).”
- Key operation: Expression evaluation by applying a function.
- Examples: Lisp, Scheme, ML.

**Logic-based languages**
- Program statements describe facts and rules, Example: “Fact: Doug is Tom’s father, Rule: If x is y’s father and y is z’s father, then x is z’s grandfather.”
- Programs don’t say how to find a solution.
- Key operation: “Unification” (the how).
- Examples: Prolog.

**Object-oriented languages**
- Program describes communication between objects, Example: “Fraction f1, simplify yourself.”
- Key operation: Message passing, inheritance.
- Can be imperative or functional.
- Examples: Simula, C++, Java, CLOS.
• Naturalness for the intended applications.
  Has the control structures, data structures,
  and operations, that are needed, and the
  syntax doesn’t get in the way.
• Language not too concise
  (or programs will be too terse),
• Language concise enough
  (or programs will be too long),
• Has compile-time or run-time checking.
• Support for abstraction and information-
  hiding,
• Can be implemented efficiently.
• Portability,