Programming Languages and Compilers

Sumeet Agarwal
Department of Electrical Engineering
IIT Delhi

References:

- Aho, Sethi, and Ullman. *Compilers: Principles, Techniques, and Tools*.
- MacLennan. *Principles of Programming Languages: Design, Evaluation and Implementation.*

Programming Paradigms

- Imperative
  - Procedural
  - Structured/Object-oriented
- Declarative
  - Functional
  - Logic

Imperative vs. Declarative

- Imperative programming uses a state-based model of computation (**Turing machine**); expresses programs in terms of sequences of command statements to change states
- Declarative programming uses a function-based model of computation (**Lambda calculus**); expresses programs as logical or functional statements, without *control flow*

HOW vs. WHAT

Procedural programming

- C, C++, Fortran, Pascal, BASIC
- Break down your task into variables, data structures and subroutines
- Use of procedures, modularity for efficiency and clarity (e.g., *scoping*)
- Allows for development of shared libraries

Structured programming, OOP

- Structured: Extensive use of subroutines, blocks, for/while loops (as opposed to goto); modularity very important
- OOP (Smalltalk, VB.NET, C#, Java, Python, Ruby): Arrange data attributes and methods into *objects*; break down your programming task into a collection of interacting classes of objects
- Control flow less clear in OOP; in this sense less 'imperative'

Encapsulation

- Java example

```
  public class Employee {
  private BigDecimal salary = new BigDecimal(50000.00);

  public BigDecimal getSalary() {
      return salary;
  }

  public static void main() {
      Employee e = new Employee();
      BigDecimal sal = e.getSalary();
  }
}
```

Inheritance

- Python example

```
class SquareSumComputer:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def transform(self, x):
        return x * x

    def inputs(self):
        return range(self.a, self.b)

    def compute(self):
        return sum(self.transform(value) for value in self.inputs())

class CubeSumComputer(SquareSumComputer):
    def transform(self, x):
        return x * x * x
```

Polymorphism

```
abstract class Animal {
    abstract String talk();
}

class Cat extends Animal {
    String talk() {
        return "Meow!";
    }
}

class Dog extends Animal {
    String talk() {
        return "Woof!";
    }
}

void letsHear(Animal a) {
    println(a.talk());
}

void main() {
    letsHear(new Cat());
    letsHear(new Dog());
}
```

Functional programming

- LISP, Scheme, Haskell, SQL, Lex/Yacc
- Computation as evaluation of mathematical functions; implementation left to compiler
- As opposed to 'functions' in procedural languages: no side effects, *referential transparency*
- Used more in academia, not so much in commercial or industrial applications

Logic Programming

- Prolog, Datalog
- Theory of computation based on first-order logic
- Typically uses *Horn clauses* to make declarative statements:

  grandparent(A,B) if **parent**(A,C) and **parent**(C,B)

- Can be seen procedurally as *goal reduction*