CSE 341 – Language Families, Autumn 1999

Purpose of programming languages

- Instructing machines.
- Communicating between programmers.
- Expressing high-level designs.
- Notation for algorithms.
- Tool for experimentation.
- Preventing human errors.

Programming style

The programming style we refer to here is not commenting or variable naming conventions, but how abstractions are made. Several styles are:

Procedural: Program consists of procedures that are invoked to change the state of the machine.

Object Oriented: Program consists of objects. Methods on objects are invoked to change the state of that object.

Functional: Program consists of functions that return results. These results are then used for future calculations. The function does not otherwise alter the state of the machine.

While programming languages do not force a particular style, they often encourage one. As well, one style does not dominate the others, rather one style might be more appropriate for a given context.

Families of programming languages

We will be discussing four major families of programming languages and seeing how their constructs facilitate the above styles of programming.

Imperative: Characterized by procedures that change the state of the system. I.e. have *side-effects*.

Procedural: Fortran, (impure) Lisp/Scheme, Algol, Pascal, C.

Object Oriented: Simula, Smalltalk, C++, CLOS.

Declarative: In some sense, describe "what" is wanted, not "how" it is done.

Functional: (pure) Lisp/Scheme, ML, Miranda, Haskell.

Logic/Relational: Prolog, CLP.

Importance

There are many tradeoffs in selecting a language

- reusability
- maintainability
- portability
- performance
- complexity
- expressiveness
- libraries
- ...

Languages

We study functional, object oriented, and logic programming languages. Programs written in languages from these families typically are slower in execution that programs written in procedural languages. However, they are typically more expressive languages and code can be developed faster and easily reused.

Scheme

- Allows side-effects, but encourages a functional style.
- Simple syntax, expression oriented.
- Extensively uses lists (Lisp = LISt Processing).

- Heavy use of recursion.
- Dynamically typed.
- Interpreted (or compiled).

Java

- Object oriented.
- Similar syntax to C++.
- Statically typed, type-safe.
- Extensive class library.
- Interpreted byte-codes.

Miranda

- Pure functional.
- Lazy evaluation.
- Heavy use of lists and recursion.
- Statically typed.

CLP(R)

- Pass by constraint.
- Relations work in both directions.
- Relations combine to restrict solutions.
- Constraint solver searches solution space for solution.
- Can solve linear system of equations.