Course progression

• Parallels between languages, including
  – sequencing, selection and repetition
  – abstraction mechanisms
    • behavioral abstraction (e.g. functions)
    • data abstraction (e.g. user-defined types)

• Differences between languages, including
  – variables
  – typing
  – extensibility of language (cf. libraries)

• Specific languages
  – Scheme, ML, Prolog, Pascal, C, Java, C#, Fortress, etc.
Functional languages
(e.g. Scheme, Lisp, ML)

- Scheme is a functional language.
- Scheme is based on lambda calculus.
- Lambda abstraction = function definition
- In Scheme, a function is defined using the keyword lambda:
  (lambda (<parameters>) <body>)
- Syntax of Scheme is very simple:
  - primitive expressions
    - numbers, such as 17
    - symbols, such as ’fred
    - names, such as fred
  - complex expressions
    - function applications, such as (+ 3 2) or (addOne x), consist of a function and arguments for the function.
    - special forms, such as (define x y), are evaluated in special ways.

- ML is a functional language.
- ML is based on lambda calculus.
- Lambda abstraction = function definition
- In ML, a function is (typically) defined using the fun keyword:
  fun <name> <param> = <body>
- Syntax of ML is familiar:
  - block-structured syntax
  - infix operators for +, *, etc
  - function names appear before argument lists, not inside parentheses