CMSC 330: Organization of Programming Languages

Type Systems, More on Scoping, and Parameter Passing

Language Features Covered Thus Far

- Ruby

– Implicit declarations	{ x = 1 }
– Dynamic typing	{ x = 1 ; x = “foo” }

- OCaml

– Functional programming	add 1 (add 2 3)	
– Type inference	let x = x+1	(x : int)
– Higher-order functions	let rec x = fun y -> x y	
– Static (lexical) scoping	let x = let x = …	
– Parametric polymorphism	let x y = y	(‘a -> ‘a)
– Modules	module foo struct … end	

Programming Languages Revisited

- Characteristics
  - Artificial language for precisely describing algorithms
  - Used to control behavior of machine / computer
  - Defined by its syntax & semantics
- Syntax
  - Combination of meaningful text symbols
    - Examples: if, while, let, =, ==, &&, +
- Semantics
  - Meaning associated with syntactic construct
    - Examples: x = 1 vs. x == 1

Comparing Programming Languages

- Syntax
  - Differences usually superficial

• C / Java	if (x == 1) { … } else { … }
• Ruby	if x == 1 … else … end
• OCaml	if (x = 1) then … else …

  - Can cope with differences easily with experience
    - Though may be annoying initially
  - You should be able to learn new syntax quickly
    - Just keep language manual / examples handy

Comparing Prog. Languages (cont.)

- Semantics
  - Differences may be major / minor / subtle

	Physical Equality	Structural Equality
Java	a == b	a.equals(b)
C	a == b	*a == *b
Ruby	a.equal?(b)	a == b
OCaml	a == b	a = b

  - Explaining these differences a major goal for 330
  - Will be covering different features in upcoming lectures

Programming Language Features

- Paradigm
  - Functional
  - Imperative
  - Object oriented
  - Multi-paradigm
- Higher-order functions
  - Closures
- Declarations
  - Explicit
  - Implicit
- Type system
  - Typed vs. untyped
  - Static vs. dynamic
  - Weak vs. strong (type safe)

Programming Language Features (cont.)

- Names & binding
  - Namespaces
  - Static (lexical) scopes
  - Dynamic scopes
- Parameter passing
  - Call by value
  - Call by reference
  - Call by name
    - Eager vs. lazy evaluation
- Polymorphism
  - Ad-hoc
    - Subtype
    - Overloading
  - Parametric
    - Generics
- Parallelism
  - Multithreading
  - Message passing

Explicit vs. Implicit Declarations

- Explicit declarations
  - Variables must be declared before used
  - Examples
    - C, C++, Java, OCaml
- Implicit declarations
  - Variables do not need to be declared
  - Examples
    - Ruby

Type System Overview

- Typed vs. untyped
- Static vs. dynamic
- Type safety
  - Weak (not type safe) vs. strong (type safe)

Typed vs. Untyped Languages

- Typed language
  - Operations are only valid for specified types
    - 2 * 3 = 6
    - “ice” * “cream” = undefined
  - Helps catch program errors
    - Either at compile or run time
- Untyped language
  - All operations are valid for all values
  - Treat all values as sequences of 0’s and 1’s
  - Example
    - Assembly languages, FORTH