

# Programming Language Pragmatics

Chapter 5-7

# Chapter 5

- Assembly-Level Computer Architecture
  - Different processor architectures
    - Correctness
    - Speed

# Modern Computers

- Workstation Macro-Architecture
  - von Neumann
- Memory hierarchy
- Data representation

# ISA

- Data movement
- Computation
- Control Transfer
- Special Operations

# Addressing Modes

- Register
- Immediate
- Absolute
- Register Indirect
- Displacement
- Indexed
- Scaled
- Memory Indirect
- Auto-increment/  
decrement

# Compiling for Modern Processors

- Keeping the pipeline full
- Loads
- Branches
- Register Allocation
- Instruction Scheduling
- Impact of subroutine calls

# Questions

- What is the purpose of a cache?
- Why more than one level of caches?
- What is data alignment?
- What is microprogramming?
  - What does this make possible?
- What is the impact of subroutine calls on register allocation?

# Control Flow (Ch6)

- Sequencing
- Selection
- Iteration
- Procedural Abstraction
- Recursion
- Concurrency
- Nondeterminacy

# Expression Evaluation

- Precedence and Associativity
- Assignments
- Orthogonality
  - Features can be used in any combination
- Initialization
- Combination Assignment Operators
- Ordering within expressions
- Short-Circuit Evaluation

# Structured and Unstructured Flow

- Removal of gotos
- Mid-loop exit and continue
- Early returns from subroutines
- Errors and other exceptions

# Sequencing

- Controls the order in which side effects occur in imperative languages

# Selection

- if-then-else statements
- Short-circuit conditions
- Case/Switch Statements

# Iteration

- Enumeration-controlled Loops
  - Changes to loop indices or bounds
  - Empty bounds
  - Access to the Index outside the Loop
  - Jumps
- Combination Loops
- Iterators
  - Enumerating without iterators
- Logically controlled loops
  - Post and mid testing

# Recursion

- Iteration and Recursion
- Applicative- and Normal-Order Evaluation

# Nondeterminacy

- Any ideas???

# Questions

- Why is it expensive to catch all uses of uninitialized variables at run time?
- Why do most languages leave unspecified the order in which the arguments of an operator or function are evaluated?
- What does it mean for a function to be idempotent?
- Why do most languages not allow the bounds or increment of an enumeration-controlled loop to be floating-point numbers?

# Data Types (Ch7)

- Types provide implicit context for many operations
- Types limit the set of operations that may be performed in the semantically valid program.

# Type Systems

- A mechanism for defining types and associating them with certain language constructs
- A set of rules for type equivalence, type compatibility, and type inference

# Type Checking

- Process of ensuring that a program obeys the language's type compatibility rules
- Type Clash
- Strongly Typed: prohibits unintended operations
- Statically/Dynamically typed

# Definition of Types

- Declaration
  - Introduces a name and indicates the scope
- Definition
  - Described the type or objects to which the name is bound

# Views of Types

- Denotational
  - A type is a set of values
- Constructive
  - A type is either a built-in type or a composite type
- Abstraction-based
  - A type is an interface consisting of a set of operations with well-defined and mutually consistent semantics

# Classification of Types

- Bool, Int, Double, Char, etc...
- Enumeration (enum)
- Subrange (type i = 0..100)
- Composite (Constructed)
- Records(Structures), Variant records, Arrays, Sets, Pointers, Lists, Files
- Orthogonality

# Type Checking

- Type Equivalence
  - Structural (Definitions)
  - Name (Lexical occurrence)
- Type Compatibility
  - Languages do not require equivalence
- Type Inference
  -

# Type Conversion

- Structurally equivalent, but the language uses name equivalence
- Different sets of values, but intersecting values are represented in the same way
- Different low-level representations, but we can define some sort of correspondence among their values.

# Records and Variants

- Syntax and Operations
- Memory Layout and Its Impact
- With Statements
- Safety of Variant Records

# Arrays

- Slices and Array Operations
- Dimensions, Bounds, and Allocation
  - Global lifetime, static shape
  - Local lifetime, static shape
  - Local lifetime, shape bound at elaboration time
- Arbitrary lifetime, shape bound at elaboration time
- arbitrary lifetime, dynamic shape
- Memory Layout

# Others

- Strings
- Sets
- Pointers and Recursive Types
  - Dangling References
  - Tombstones (Catch dangling references)
  - Locks and Keys (Alt for Tombstones)
- Garbage collection
  - Reference Counts
  - Mark-and-Sweep Collection
- Lists
- I/O

# Equality Testing and Assignment

- Easy for simple types
- Abstract types are more complicated

# Questions

- What is the purpose of types?
- What prevents C from being strongly typed?
- What is a type clash?