1 Introduction

What's in This Set of Notes ?

1.1 Idea

- Computer languages are not just a way to get a computer to perform operations
- Computer languages are a novel formal medium for expressing ideas about methodology
  - programs are and must be for people to read
  - only incidental for machines to execute
- Important to understand techniques to control intellectual complexity of large software systems
  - not just syntax
  - not just programming language constructs
  - not just clever algorithms
  - not just analysis

1.2 Computation

- Computational situation involves at least three different types of

entities
    1. a program - a thing that might be edited within a text editor
    2. the process or computation to which the program gives rise, upon being executed
    3. domain or subject matter that the computation is about
- Computation provides framework for "HOW TO" not "What IS"
    - e.g., not declarations
- We will study Computational Process
    - abstract things that inhabit computers
    - spells
    - you must anticipate outcome and consequences

1.3 Why Study Programming Languages

---

1. To improve your ability to develop effective algorithms
    - principles + techniques => compute relative cost of recursion
2. To improve you use of existing languages
    - build more efficient program => how arrays, string are created and manipulated
3. To increase your vocabulary of useful programming constructs
    - e.g., dispatching in OO applied elsewhere
4. To allow better choice of programming languages
    - C++, Java, Smalltalk, Lisp, Scheme, Prolog
5. Better understanding of the significance of implementation
6. To make it easier to learn a new language
7. Increased ability to design a new language
    - Java = Smalltalk + C ++
8. Overall advancement of computing

1.4 Attributes of a Good Language

---

1. Readability and Clarity
    - simplicity

    - minimum number of concepts
    - rules are simple
    - orthogonality
        - being able to combine various language features in all combinations
        - e.g., expression + conditional
    - Control statements
    - Data Types and Structures
2. Writeability
    - support for abstraction
        - gap between abstract data structures + operations and solution
        - C++ is better C with objects
    - expressivity
3. Naturalness for application
    - translate design to implementation
    - OO Design -> OO Language
4. Reliability
    - type checking
    - exception handling
    - aliasing
5. Ease of program verification
    - proven correct by formal verification
    - desk checking (visual inspection)
    - testing
6. Programming Environment
    - editors
    - testing
    - packages
7. Portability
    - Java: write once run anywhere
8. Cost
    - program execution
    - translation (compile vs. interpret)

- creation
- testing
- using
- maintenance

1.5 Language Paradigms

Imperative or Procedural Programming

- Model follows from hardware
- Command drive or statement oriented
- Basic concept is machine state
- Program is sequence of statements
- Execution changes value of one or more locations in memory (new state)
- Syntax

  ...;
  ...;
  ...;
- C, Pascal, Cobol

Applicative or Functional Programming

- Look at functions the program represents rather than its state changes
- Look at results rather than available data
- Focus on the following:

  > What is the function that must be applied to the initial machine state by accessing the initial set of variables and combining them in specific ways in order to get an answer

- Use primitive functions to build more complex ones
- Consider functional transformations
- fn (.. fn(.. (fn (data)))

- Lisp, ML, Scheme
- Assignment free programming

Rule-based or Logic Programming

- Execute by checking for presence of certain enabling conditions and when satisfied execute appropriate actions
- Syntax

  condition -> action

  condition -> action
- Enabling conditions determine execution order
- Like set of filters applied to data
- Prolog

Object-Oriented Programming

- Complex data objects with limited set of functions
- Complex objects built from simple ones
- Important concepts
  - encapsulation
  - inheritance
  - polymorphism
- Combination of imperative and functional programming
- C++, Smalltalk, Java

Access-Oriented Programming

- Attaches side effects or demons to variables so that certain functions will be performed as variables are manipulated

---

Constraint Programming

- Like logic programming
- More declarative than imperative
- Programmer specifies constraints that must always be maintained

during execution (e.g., window should always be 2 * height)
- Underlying constraint satisfaction mechanism

Parallel Programming

- Computation is defined as taking place with multiple processes

---

Visual Programming

- Programs are specified visually rather than textually

1.6 Elements of Programming

- Programming language more than just a means for instructing a computer to perform tasks
- Language serves as a framework within which we organize our ideas about processes
- We must pay particular attention to the means that the language provides for combing simple ideas to form complex ones
- Every powerful language has three mechanisms for accomplishing this:
    1. primitive expressions, represents the simplest entities the language is concerned with
    2. means of combination, how compound elements are built from simpler ones
    3. means of abstraction, how elements can be named and manipulated as units
- In programming we deal with two kinds of elements which are not really so distinct:
    1. procedures
    2. data