Chapter 1

Preliminaries

Topics

- Motivation
- Programming Domains
- Language Evaluation Criteria
- Language Design Trade-Offs
- Influences on Language Design
- Language Categories
- Implementation Methods

What impacts Programming Language Design?

- Application domain
- Evaluation Criteria

  The set of factors that are important to the users of the programming language

- Computer architecture
- Programming methodologies

Language Evaluation Criteria

Characteristics	Criteria		
	Readability	Writability	Reliability
Simplicity / orthogonality	♦	♦	♦
Control structure	♦	♦	♦
Data type and structures	♦	♦	♦
Syntax design	♦	♦	♦
Support for abstraction		♦	♦
Expressivity		♦	♦
Type checking			♦
Exception handling			♦
Restricted aliasing			♦

Evaluation Criteria: Writability

- Writability describes the ease with which a language can be used to create programs for a given domain.
  - Be careful not to compare things which should not be.
- Most of the features that affect readability affects also writability.

Evaluation Criteria: Writability
Factors

- Simplicity and Orthogonality
  - Lack of familiarity with some features leads to misuse and disuse of those features.
    - Misuse could cause bizarre results.
  - Too much orthogonality may produce undetected errors.
    - Any combination of primitive is legal.

```
score = 15 / 3 * 5;
```

Evaluation Criteria: Writability
Factors

- Support for abstraction
  - Ability to define and use complicated structures or operations ignoring all the details.
    - Important for modular programming.
    - Two forms of abstraction
      - Process: subprograms
        - e.g. using a subprogram to implement a search or sort algorithm.
      - Data: data types
        - e.g. trees, arrays, etc.

Evaluation Criteria: Writability
Factors

- Expressivity
  - Aids writability by make it convenient and easy to specify things.
    - e.g. count++ vs. count = count + 1

Evaluation Criteria: Reliability
Factors

- Reliable programs work (according to specifications) under all conditions.
- Type checking
  - Earlier error detection is less expensive to repair
  - Compile-time checking is preferred.
- Exception handling
  - The ability of a program to intercept run-time errors, take corrective measures, and then continue (e.g. C++, Java, Ada).

Evaluation Criteria: Reliability
Factors

- Aliasing
  - Having to or more distinct referencing methods, or names, for the same memory cell.
    - e.g. using pointer in C++, reference in Java
- Readability and Writability
  - The easiest a program is to write, the more likely it is to be correct.
  - Programs that are difficult to read are difficult to both to write and modify.

Evaluation Criteria: Cost

- Cost of learning/teaching a language (programmer training)
- Cost of writing/developing a program (software creation)
- Cost of compiling the program (fast)
- Cost of running the program (fast)
- Cost of the compiler (for free e.g. Java)
- Cost of poor reliability
- Cost of maintaining the program (corrections and modifications to add new capabilities)

Evaluation Criteria: Other

- Portability
  - The ease with which programs can be moved from one implementation to another.
- Generality
  - The applicability to a wide range of applications.
- Well-definedness
  - The completeness and precision of a language's official defining document.

Language Design Trade-Offs

"There are so many important but conflicting criteria, that their reconciliation and satisfaction is a major engineering task."
(Tony Hoare 1973)

Reliability	vs.	Cost (execution)
Expressivity	vs.	Readability
Writability	vs.	Readability
Reliability	vs.	Writability(flexibility)

Language Design Trade-Offs

- Most criteria cannot be defined nor measured precisely.
- The way a language is evaluated is heavily influenced by the point of view and background of the evaluator.
  - Language designer
  - Language implementor
  - Language user

A real designer understands trade-offs and make decisions rather than skirt them.

What impacts Programming Language Design?

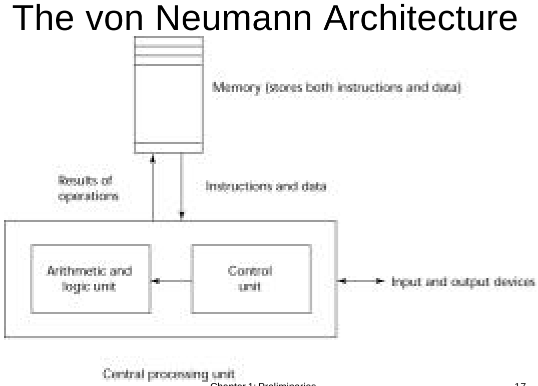
- Application domain
- Evaluation Criteria
- Computer architecture
- Programming methodologies

The programming language should map well to the hardware (computer architecture)

Computer Architecture Influence

- Imperative languages have been designed around the von Neumann architecture
  - Data and programs are stored in memory
  - Central processing unit (CPU) executed the instructions
    - CPU and memory are separated
    - Instructions/data must be transmitted from memory to CPU
    - Results from operations are transmitted back to memory
- Imperative languages map well to this architecture
  - Variables are memory locations
  - Assignments move data back and forth between CPU and memory
  - Iteration for repetition

The von Neumann Architecture



What impacts Programming Language Design?

- Application domain
- Evaluation Criteria
- Computer architecture
- Programming methodologies

Programming languages respond to different ways of thinking about programs

Programming Methodologies Influence

- People's needs affect the design of programming languages and paradigms.
  - 1950's and early 1960's
    - Worry about machine efficiency
    - Simple applications
  - Late 1960's
    - Worry about people efficiency
    - Better control structures and improved readability
      - Structured programming
      - Top-down design and step-wise refinement

Programming Methodologies Influence

- Mid-late 1970's
  - Worry about reuse and maintenance
  - Shift from process-oriented to data-oriented
    - Data abstraction
- 1980's
  - Rising complexity and costs
  - Introduction of object-oriented programming
    - Data abstraction + inheritance + polymorphism
- 1990's
  - The Internet
    - Data + network issues + interoperability

Programming Paradigms

- Paradigms are programming styles (a special way to express an idea or algorithm) that embody programming design technology

Imperative			Declarative		
Procedural	OO	Parallel	Logical	Functional	Database
Fortran	C++	Occam	Prolog	Haskell	SQL
	Eiffel	CAML		ML	
	Java	Java		Lisp	
				Scheme	
With blocks	Objects				
Algol	Smalltalk				
Pascal					
C					

Programming Paradigms: Imperative

- Central features are variables, assignment statements, and iterative form of repetition.
- Specific order of execution of the instruction
  - Program = order series of steps
- Separation of data and algorithm
- C, Pascal, Cobol, Fortran

Example

Programming Paradigms: Object-Oriented

- Closely related to imperative
- Program = a set of definitions (data and code that operates on the data encapsulated together)
  - Objects interact with each other by passing messages back and forth
- Other features: inheritance, dynamic binding
- Java, C++, Python, Smalltalk, Eiffel

Example

Programming Paradigms: Functional

- Central features are functions (applied to given parameters)
  - Program = a set of mathematical functions each with an input (domain) and an output (range)
  - No assignments, tons of recursion, and less focus on order
- Lazy evaluation: postpone operand evaluation until operation.
- Lisp, Scheme, Haskell, ML

Example

Programming Paradigms: Logic

- What vs. How
- Rule-based language
- Rules are specified in no particular order
- Program = collection of logical declarations that describe the problem to be solved
  - An inference engine then finds the solution
- It is also called declarative
  - Declare or make assertions
  - No sequence
- Prolog

Example

Programming Example

Greatest Common Denominator (gcd)

C

```
intgcd( intx, inty )
{
   int remainder;
   do {
      remainder = a%b;
      if (remainder != 0) {
         a = b;
         b = remainder;
      }
   } while (remainder);
   return b;
}
```

Scheme

```
(define ( gcd u v)
   (if (= v 0) u
      (gcd v (modulo u v))))
```

Prolog

```
gcd( U, V, U ) :-  V=0
gcd( U, V, X ) :- not( V=0 )
                  Y is U mod V,
                  gcd( V, Y, X )
```

Java

```
public class IntGcd
{
   private intvalue;

   public intIntGcd( intval ){
      value = val;}
   public intGetValue() {
      return value; }
   public intgcd( intv ) {
      intz = value;
      inty = v;
      while ( y != 0 ) {
         int t = y;
         y = z%y;
         z = t; }
      return z; }
}
```


Programming Paradigms: Comparison

	Advantage	Disadvantage
Imperative	Running cost Compilation cost	Reliability Readability
Functional	Writability (asbtract) Readability Reliability Verification	Running cost Compilation cost
Object-oriented	Maintenance cost Reliability Abstraction	Learning cost Compilation cost Running cost

Language Implementation

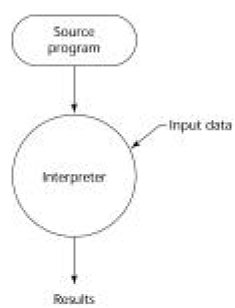
- There are three possible approaches to translating human readable code to machine code
  1. Compilation
  2. Interpretation
  3. Hybrid

Compilation

- Translate high-level program to machine code
- Slow translation
- Fast execution
- Optimization (improve program by making it smaller or faster)
- Slow for development
- Difficult dealing with runtime errors

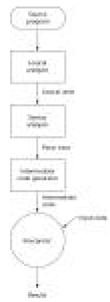
Interpretation

- No translation
- Easier implementation
- Slower execution
- Often requires more space
- Easy run-time error handling
- Becoming rare on high-level languages
- Significant comeback with some Web scripting languages (e.g. JavaScript)



Hybrid

- A compromise between compilers and pure interpreters
- Faster than pure interpretation (medium execution speed)
- A high-level language program is translated to an intermediate language that allows easy interpretation (small translation cost)



Language Implementation: Comparison

	Compiler	Interpreter	Hybrid
Speed (runtime)	++ simple instructions	- complex statements	-
Memory needed	++	- source, symbol table	-
Portability	- reusable backend	-	++ intermediate language
Reliability	- no checks	++ additional checks	++ additional checks

Summary

- Reasons to study concepts of PLs
  - Increase our capacity to use different constructs
  - Enables us to choose languages more intelligently
  - Makes learning new languages easier
- Most important criteria for evaluating PLs
  - Readability, writability, reliability, and cost
- Major influences on language design
  - Machine architecture and software development methodologies
- Major methods of implementing languages
  - Compilation, pure interpretation, and hybrid implementation