Principles of Programming Languages

CS 492
Lecture 1

Based on Notes by “William Albritton”

Lecture Outline

• Reasons for studying concepts of programming languages
• Programming domains
• Language evaluation criteria
• Influences on language design
• Language categories
• Design trade-offs
• Implementation methods
• Administrative Stuff
Reasons for Studying …

• Increased capacity to express ideas
  – Learning new data structures, algorithms, and other language features will allow a programmer to create more efficient programs
    • Example: after learning object-oriented programming in Java, a C programmer could simulate objects using structures and functions

Reasons for Studying …

• Improved background for choosing appropriate languages
  – Without having studied several different languages, programmers will tend to stick with the first language they learned
    • Example: A C programmer might implement a binary tree with arrays, instead of using the object-oriented capabilities of Java
Reasons for Studying …

• Increased ability to learn new language
  – Knowing the basic concepts of programming languages allows one to learn a new language easier
    • Example: C++ or Ada programmers, who already understand the concept of object-oriented programming, will have an easier time of learning Java than programmers have never used these concepts

Reasons for Studying …

• Better understanding of the significance of implementation
  – Knowledge of the basic concepts of programming languages enables the programmer to make more efficient choices with implementation and to find bugs easier in a program
    • Example: A C programmer who understands pointers can create the binary tree using structures and pointers, instead of with arrays, and will have an easier time debugging programs that use pointers
Reasons for Studying …

• Overall advancement of computing
  – Knowing the advantages & disadvantages of different languages helps “those in charge” to choose better languages over poorer ones
    • Example: In the early 1960s, Fortran was used much more than ALGOL 60, even though ALGOL 60 had much better control statements than Fortran

Programming Domains

• Scientific applications
  – The first computers (1940s) created for scientific applications
  – Used mostly arrays, matrices, counting loops, and selections
  – Most efficient language for this is Fortran (1959)
  – Also ALGOL 60 (1960) and its descendants designed mainly for this purpose
Programming Domains

• Business applications
  – Began to be used for business in 1950s
  – First language for this was COBOL (1960)
  – Used to create reports, storing decimal numbers & character data, and calculation decimal operations
  – More recently, microcomputers are used for spreadsheet systems & database systems

Programming Domains

• Artificial intelligence
  – Uses symbolic rather than numeric computations
  – Uses linked lists of names
  – The functional language LISP was developed for AI applications in 1959
  – In 1970s, logic programming using Prolog appeared
Programming Domains

• Systems programming
  – Operating system + supporting tools = systems software
  – Must be efficient & interface with external drivers
  – In 1960s & 1970s, PL/S, BLISS, & Extended ALGOL were first used for this
  – Currently, the UNIX operating system is written mostly in the C language

Programming Domains

• Scripting languages
  – Early scripting languages were simply a list of commands (called a script) in a file to be executed
  – Scripting languages grew to include variables, control flow statements, functions, etc.
  – Perl is a primitive programming language used for Common Gateway Interface (CGI) programming for the World Wide Web
  – JavaScript & PHP are embedded in HTML documents for use on Web server systems
Language Evaluation Criteria

1. Readability
2. Writability
3. Reliability
4. Cost (Costability?)

• Readability
  – Ease at which a program can be read & understood
  – Factors that affect readability
    • Overall simplicity
    • Orthogonality
    • Control statements
    • Data types & structures
    • Syntax considerations
Overall Simplicity

- A large number of features is difficult to learn
  - So programmers might learn two different subsets of features
- Feature multiplicity (having more than one way to publish a particular operation) is not good
  - For example, in C++ or Java you can decrement a variable in four different ways: `x = x - 1; x-=1; x--; --x`
- Operator overloading (a single operator symbol has more than one meaning) potentially can be bad
- Some languages, such as assembly language, can be too simple

Orthogonality

- A relatively small set of primitive constructs can be combined in the relatively small number of ways to build the control and data structures of the language
  - For example, if the language has three primitive data types, an integer, double, and character, and two type operators, array and pointer, then a large number of data structures can be defined
  - Lack of orthogonality leads to exceptions to rules
  - Too much orthogonality can also be bad, such as if you had the ability to add a pointer to another pointer in C
Control Statements

• In the 1970s, the use of “goto” statements was replaced by structured programming, which could be read from top to bottom
  – Most programming languages now contain sufficient control statements
    • if (x < y) x++; if (x < y) goto L1;     if (x < y) goto L1;
      else y++;                           y++;
      goto L2;                           goto L2;
      L1: x++;                           L1: x++;
      L2:

Data Types & Structures

• Adequate data types and data structures also aids readability
  – A language with Boolean type is easier to read than one without
    • indicatorFlag = 0 is more difficult to read than indicatorFlag = false
Syntax Considerations

• Syntax = the way in which linguistic elements (as words) are put together to form constituents (as phrases or clauses)
  – Identifier forms
    • If too short, reduces readability
  – Special words
    • Ada uses `end if` & `end loop`, while Java just uses `}`
    • In Fortran 95, `Do` & `End` can also be variable names
  – Form & meaning
    • In C, `static` changes meaning depending on position

Language Evaluation Criteria

• Writability
  – Ease at which a language can be used to create programs for a specific problem domain
  – Factors that affect writability
    • Simplicity & orthogonality
    • Support for abstraction
    • Expressivity
Simplicity & Orthogonality

- It is best to have a smaller number of primitive constructs and a consistent set of rules for combining them
  - Having a few constructs with few exceptions is easier to learn for a beginner programmer
  - On the other hand, a language can also be too orthogonal, so the inexperienced programmer could add two pointers together

Support for Abstraction

- Abstraction = the ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored
  - Using the same function several times in the program
  - To implement a binary tree in Fortran 77 requires arrays, while C++ uses nodes with two pointers and an integer
Expressivity

• A language has relatively convenient, rather than cumbersome, ways for specifying computations
  – For example, in C `x++` is shorter & more convenient than `x=x+1`

Language Evaluation Criteria

• Reliability
  – A reliable program performs to its specifications under all conditions
  – Factors that affect reliability
    • Type checking
    • Exception handling
    • Aliasing
    • Readability & writability
Reliability

• Type checking
  – Testing for type errors in a given program
    • For example, can have errors if a function that is expecting an integer receives a float instead

• Exception handling
  – Used in Ada C++ and Java, but not in C and Fortran
    • For example, the try & catch blocks of C++ catch run-time errors, fix the problem, and then continue the program without an “abnormal end”

Reliability

• Aliasing
  – Referencing the same memory cell with more than one name
    • For example, in this C code both x & y can be used to the same memory cell
      \[
      \text{int } x = 5; \\
      \text{int } *y = &x;
      \]

• Readability & writability
  – If a program is difficult to read or write, its easy to make mistakes and difficult to find them
Language Evaluation Criteria

• Cost
  – The total cost of a programming language is based on many of its characteristics
  – Factors that affect cost
    • Training programmers
    • Writing programs
    • Compiling programs
    • Executing programs
    • Language implementation system
    • Poor reliability
    • Maintaining programs

Cost

• Most important contributors to language costs
  – Program development
  – Maintenance
  – Reliability

• Most important evaluation criteria in respect to language costs
  – Writability
  – Readability
Other Evaluation Criteria

- **Portability**
  - Ease with which programs can be moved from one implementation to another
  - Influenced by a degree of standardization
    - Time-consuming and difficult (C++ 10 yrs)
    - BASIC not standardized
- **Generality**
  - Applicability to a wide range of applications
- **Well-definedness**
  - Completeness and precision of the language’s official defining document

-----Class Discussion-----

- **C vs. Java:** Evaluate the C & Java languages using the criteria just described. What kind of problem-solving domain would C be useful for? What about Java?
  1. Readability
  2. Writability
  3. Realiability
  4. Costability
Influences on Language Design

- **Computer architecture**
  - von Neumann ("von Noyman") architecture
    - Data & programs are stored in memory
    - CPU executes instructions
    - Instructions & data are piped between memory & CPU
  - Imperative languages designed around this
    - Variables model memory cells
    - Assignment statements model piping operation
    - Iterative form is most efficient form of iteration
Influences on Language Design

- **von Neumann bottleneck**
  - Fetch-execution cycle
  - Fetch the instruction pointed to by program counter
  - Increment program counter
  - Decode the instruction
  - Execute the instruction
  - Instructions can be executed faster than they can be moved to the CPU for execution

Influences on Language Design

- **Programming methodologies**
  - Type checking & control statements were developed in early 1970s
    - Used procedure-oriented programming
  - Abstract data types were developed in late 1970s
  - Object-oriented programming developed in the early 1980s
Language Categories

1. Imperative
   - Uses variables, assignments statements, & iteration
2. Functional
   - Uses functions, parameters, & recursion
3. Logic
   - Uses facts & rules (written in no specific order) and a language system which produces the desired result
4. Object oriented
   - Uses data abstraction, inheritance, & dynamic method binding

Language Design Trade-offs

- Some criteria conflict with one another
- For example, reliability & cost of execution
- In C, the index ranges of arrays are not checked
  - So executes fast, but it not so reliable
- On the other hand, Java checks all references to array elements
  - So executes slower, but is more reliable
Implementation Methods

• Compilation
  – A compiler is a program that reads a program written in one language (the source language) and translates it into an equivalent program in another language (the target language). A part of this process is to report to its user the presence of errors in the source program.

  source program => compiler => target program
  |  
  error messages

8 Phases of a Compiler

• 6 main phases
  1. Lexical analyzer (identify words, or tokens)
  2. Syntax analyzer (identify sentences, use parsing)
  3. Semantic analyzer (type checking)
  4. Intermediate code generator
  5. Code optimizer
  6. Code generator
• Two more activities (interact with all 6 phases)
  1. Symbol table manager
  2. Error handler
Compilation Process

- Linking the program
  - To create the executable (.exe file) program from the object (.obj) file, the linker (link-editor) links:
    1. The object files from the program modules, and
    2. Any additional library files
  - Usually this includes the use of relocatable addresses within the program
    • Allows the program to run in different memory locations
Post-Compiler

- **Loading the executable (.exe file) program**
  - Also called the load module or executable image
  - The loader
    - Identifies a memory location in which the program can be loaded, and
    - Alters the relocatable machine code addresses to run in the designated memory location
  - A program is loaded into memory each time it is run

Implementation Methods

- **Pure interpretation**
  - High-level language statements are immediately decoded into machine code & executed
    - Easy to debug
    - Slow to decode (10 to 100 times slower)
    - Requires more space for symbol table & source program
Pure Interpretation

- **Source program**
  - **Input data**

**Interpreter**

**Results**

Implementation Methods

- **Hybrid implementation systems**
  - Cross between compilers & interpreters
    - High-level language programs are translated to an intermediate language, which is easily interpreted
  - Perl
    - Partially compiled to detect errors, then interpreted
  - Java
    - Intermediate “byte code” is created from a program, then it is interpreted
Hybrid Implementation System

Administrative Issues

- Class Timings (mistake in quarter schedule)
- Course Website
  - [http://suraj.lums.edu.pk/~cs492](http://suraj.lums.edu.pk/~cs492)
- Office Hours (Today)
- Next Class (and reading)
  - Handouts for Next Class