CSCI 3336 Organization of Programming Languages

PRELIMINARIES

Topics

- Reasons for Studying Concepts of Programming Languages
- Programming Domains
- Language Evaluation Criteria
- Influences on Language Design
- Language Categories
- Language Design Trade-Offs
- Implementation Methods
- Programming Environments

Reasons for Studying Concepts of Programming Languages

- Increased ability to express ideas
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of significance of implementation
- Better use of languages that are already known
- Overall advancement of computing

Programming Domains

- Scientific applications
  - Large numbers of floating point computations; use of arrays
    - Fortran (1950's)
    - Julia (2012)
- Business applications
  - Produce reports, use decimal numbers and characters
  - Spread sheet systems
    - COBOL (1960's)
- Artificial intelligence
  - Symbols rather than numbers manipulated; use of linked lists
    - LISP (1965)
  - Knowledge based
    - Prolog (1997)

Programming Domains (cont'd)

- Systems programming
  - The computer system and all of the programming support tools
  - Portable for different machines
    - C (1972)
- Web Software
  - Eclectic collection of languages: markup (e.g., XHTML), scripting (e.g., PHP), general-purpose (e.g., Java)
    - Perl – (1987) CGI (Common gateway interphase)
    - HTML (1993)
    - PHP (1995)
    - Javascript (1995)
- Object – Oriented
  - Model organized around "objects" rather then "actions" and data rather then logic
    - Simula (1960's)
    - Java (1995)
    - Ruby (1996)

Language Evaluation Criteria

- **One of the main purpose of this course is to examine carefully the underlying concepts of the various constructs and capabilities of programming languages**
- **Evaluation criteria**
  - **Readability – Easy to understood**
  - **Writability – Easy to write**
  - **Reliability**
  - **Cost**

Evaluation Criteria: Readability

- Code easy/fast – to read and understand
- Language factors that affect readability
  - Overall language simplicity
    - A manageable set of features and constructs
    - Few feature multiplicity (means of doing the same operation)
    - Minimal operator overloading
  - Orthogonality
    - Every possible combination is legal
    - The lack of orthogonality leads to exceptions to the rules of the language
    - Orthogonality = simplicity
  - Control statements built-in
  - Data types & structures built-in
  - Syntax considerations
    - closeness to natural language and/or mathematics

Evaluation Criteria: Writability

- Ease/speed – to create programs
- Language factors that affect writability
  - Simplicity and orthogonality
    - Few constructs, a small number of primitives, a small set of rules for combining them
  - Support for abstraction
    - The ability to define and use complex structures or operations in ways that allow details to be ignored
    - Process abstraction: subprograms
    - Data abstraction: struct
  - Expressivity
    - A set of relatively convenient ways of specifying operations
    - Example: the inclusion of `for` statement in many modern languages

Simplicity improves Read/Writability

- A large language takes more time to learn
  - Programmers might learn only a subset
- Feature multiplicity
  - having more than one way to perform a particular operation) is often confusing

```
x = x + 1; x += 1; x++; ++x;
```

- Operator overloading
  - A single operator symbol has more than one meaning can lead to confusion

```
class myClass {
  public:
    int i;
  public:
    int operator+(int j) { return i-j; }
}
```

- Some languages (assembly) can be "too simple" – too low level. 2, 3, 4, 5 or more statements need to have the effect of 1 statement in a high-level language

Orthogonality improves Read/Writability

- Having fewer constructs and having fewer exceptions increases readability and writability
- Orthogonal languages are easier to learn
  - Examples:
    - Pointers should be able to point to any type of variable or data structure
- However, if a language is too orthogonal, an inexperienced programmer might assume they can do something that makes no sense
  - Example
    - Add to pointers together

Structured control improves Read/Writability

- goto statements were replaced by structured programming in the 1970s
- Most languages now contain sufficient control statements making goto's unnecessary
  - The following are equivalent

```
If ( x < y )
  x ++;
else
  y ++;
```

```
if ( x < y )
  goto L1;
y ++;
goto L2;
L1:  x++;
L2:
```


Data structures improve Read/Writability

- Adequate data types and data structures also aid readability
- A language with Boolean types is easier to read than one without
  - For example:

```
doneReading = 0; // it is more difficult to read than
doneReading = false;
```

Syntax and Read/Writability

- Syntax – the way linguistic elements (e.g. words) are put together to form phrases or clauses/ sentences
- Identifiers forms
  - If too short, reduces readability
  - If too long, reduces writability
- Special word use
  - ADA has end if and end loop, while Java uses { for both.
  - In Fortran 95, Do and End can also be variable names.
- Form and meaning
  - In C, static changes meaning depending on position
    - A static variable inside a function keeps its value between invocations
    - A static global variable or a function is "seen" only in the file is declared.

1-13

Abstraction and Read/Writability

- Abstraction – the ability to define and then use complex structures or operations
  - Allow details to be ignored
  - Allow code to be re-used instead of repeated
- Abstract data types
  - Implementation details are separated from the interface, allowing them to be changed without re-writing all code
- Objects
- Subprograms

1-14

Evaluation Criteria: Reliability

- A reliable program performs to its specifications under all conditions
- Factors that affect reliability
  - Type checking
    - Testing for type errors
  - Exception handling
    - Intercept run-time errors and take corrective measures
  - Aliasing
    - Presence of two or more distinct referencing methods for the same memory location
  - Readability and writability
    - A language that does not support "natural" ways of expressing an algorithm will necessarily use "unnatural" approaches, and hence reduced reliability

1-15

Type checking and Exception Handling Improve Reliability

- Type checking
  - Testing for type errors in a given program
    - For example, if a function is expecting an integer receives a float instead
- Exception Handling
  - Used in Ada, C++, Lisp and Java
    - For example the try and catch blocks of C++ can catch runtime errors, fix the problem, and then continue the program without an "abnormal end"

1-16

Aliases reduces readability and reliability

- Aliasing
  - Referencing the same memory cell with more than one name
    - E.g., in C, both x and y can be used to refer the same memory cell.

```
int x = 5;
int *y = &x;
```

  - Leads to errors
  - Reliability increases with better readability and writability
    - If a program is difficult to read or write, its easier to make mistakes and more difficult to find them

1-17

Evaluation Criteria: Cost

- Total cost – due to many language factors
  - Training programmers (time)
  - Development software and environment (ease to use, time)
  - Compiling programs (time, space)
  - Executing programs (time, space)
  - Language implementation system (interpreter, compiler) availability of free interpreters/ compilers
  - Reliability (poor reliability leads to high costs)
  - Maintenance (time to fix bugs, keep current with new hardware/software)

1-18

Evaluation Criteria: Others

- 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 the language's official definition
- **High expressive power, flexibility**
- **Safety**

1-19

Influences on Language Design

- Computer Architecture
  - Languages are developed around the prevalent computer architecture, known as the *von Neumann* architecture
- Programming Methodologies
  - New software development methodologies (e.g., object-oriented software development) led to new programming paradigms and by extension, new programming languages

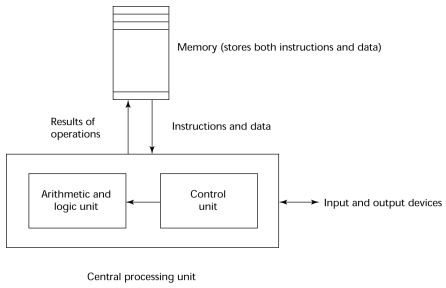
1-20

Computer Architecture Influence

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
  - Data and programs stored in memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
  - Basis for imperative languages
    - Variables model memory cells
    - Assignment statements model piping
    - Iteration is efficient

1-21

The von Neumann Architecture



Programming Methodologies Influences

- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: People efficiency became important; readability, better control structures
  - structured programming
  - top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
  - data abstraction
- Middle 1980s: Object-oriented programming
  - Data abstraction + inheritance + polymorphism

1-23

Language Categories

- Imperative
  - Central features are variables, assignment statements, and iteration
  - Examples: C, Pascal
- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme
- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog

1-24

Language Categories (cont' d)

- Object-oriented
  - Data abstraction, inheritance, late binding
  - Examples: Java, C++
- Markup
  - New; not a programming per se, but used to specify the layout of information in Web documents
  - Examples: XHTML, XML

1-25

Language Design Trade-Offs

- Reliability vs. cost of execution
  - Conflicting criteria – costs more to ensure greater reliability
  - Example: Java demands all references to array elements be checked for proper indexing but that leads to increased execution costs
- Readability vs. writability
  - Another conflicting criteria
  - Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
- Writability (flexibility) vs. reliability
  - Another conflicting criteria
  - Example: C++ pointers are powerful and very flexible but not reliably used

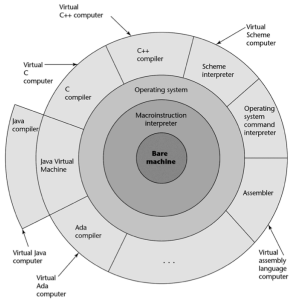
1-26

Implementation Methods

- Compilation
  - Programs are translated into machine language
- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters

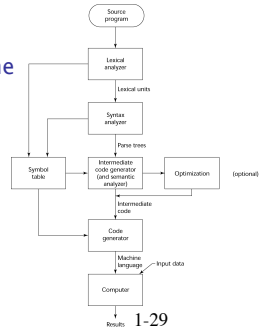
1-27

Layered View of Computer



Compilation

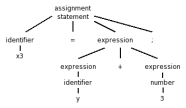
- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution



1-29

Compilation – Phases

- Compilation process has several phases:
  - lexical analysis: converts characters in the source program into lexical units
    - Comments, literals(constants), identifiers
  - syntax analysis: transforms lexical units into *parse trees* which represent the syntactic structure of program



  - Semantics analysis: generate intermediate code
  - code generation: machine code is generated

1-30

Execution of Machine Code

- Fetch-execute-cycle (on a von Neumann architecture)

```
initialize the program counter
repeat forever
  fetch the instruction pointed by the
  counter
  increment the counter
  decode the instruction
  execute the instruction
end repeat
```

1-31

Von Neumann Bottleneck

- Connection speed between a computer's memory and its processor determines the speed of a computer
- Program instructions often can be executed a lot faster than the above connection speed; the connection speed thus results in a *bottleneck*
- Known as von Neumann bottleneck; it is the primary limiting factor in the speed of computers

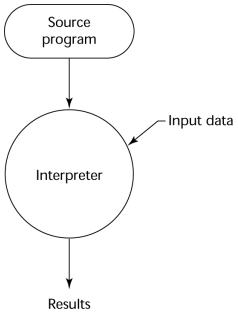
1-32

Pure Interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Becoming rare on high-level languages
- Significant comeback with some Web scripting languages (e.g., JavaScript)

1-33

Pure Interpretation Process



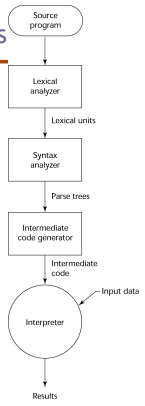
Hybrid Implementation Systems

- A compromise between compilers and pure interpreters
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Faster than pure interpretation

1-35

Hybrid Implementation Process

- Examples
  - Perl programs are partially compiled to detect errors before interpretation
  - Initial implementations of Java were hybrid; the intermediate form, *byte code*, provides portability to any machine that has a byte code interpreter and a run-time system (together, these are called *Java Virtual Machine*)



Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile intermediate language into machine code
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system

1-37

Preprocessors

- Preprocessor macros (instructions) are commonly used to specify that code from another file is to be included
- A preprocessor processes a program immediately before the program is compiled to expand embedded preprocessor macros
- A well-known example: C preprocessor
  - expands `#include`, `#define`, and similar macros

1-38

Programming Environments

- The collection of tools used in software development
- UNIX
  - An older operating system and tool collection
  - Nowadays often used through a GUI (e.g., CDE, KDE, or GNOME) that run on top of UNIX
- Borland JBuilder
  - An integrated development environment for Java
- Microsoft Visual Studio.NET
  - A large, complex visual environment
  - Used to program in C#, Visual BASIC.NET, Jscript, J#, or C++

1-39

Summary

- The study of programming languages is valuable for a number of reasons:
  - Increase our capacity to use different constructs
  - Enable us to choose languages more intelligently
  - Makes learning new languages easier
- Most important criteria for evaluating programming languages include:
  - Readability, writability, reliability, cost
- Major influences on language design have been machine architecture and software development methodologies
- The major methods of implementing programming languages are: compilation, pure interpretation, and hybrid implementation

1-40