

Programming Languages

CSC519, Fall 2011

Ubbo Visser

Short introduction

- Ubbo Visser



- Office: Ungar Building, Room No 441
- Phone: 305-284-2254
- Email: visser@cs.miami.edu (preferred)
- Office Hours: by appointment

- Stations

- Münster, Brisbane, Bremen
- Miami
 - Research Associate Professor

- Research interests

- Artificial Intelligence with the focus on knowledge representation and reasoning.
- Application areas: the Semantic Web and Multi-Agent Systems (Games, Robots, RoboCup)

2

More introduction...

- Teaching Assistant
 - Piyali Nath
 - Email: p.nath@umiami.edu
- Contact Hours
 - Each week there are two 75 minutes sessions (Tuesday, Thursday 8am - 9.15am)
 - Classroom: MM 119
- Recommended Text Book
 - Concepts of Programming Languages, 8/E, Robert W. Sebesta, University of Colorado, Colorado Springs, ISBN-10: 0321493621, ISBN-13: 9780321493620, Publisher: Addison-Wesley
- Course Content
 - Chapters 1 to 3, 5 to 12, plus parts of 13, 14, 15, and 16 as time permits. If time is short, some of the concepts in chapters 13-14 will be omitted. Also, you will learn part of some language that is new to most of the students in the class. Course material will be uploaded before the lecture as .pdf files. Check <http://www.cs.miami.edu/~visser> regularly. Content may change slightly during semester.

3

Grading & general issues

- Grading

- will be based on 100 points

Item	Points
Homework	70
Final	30

- Scoring of Homework Assignments

- The score of each homework will be mentioned in it.
- The total score of all homework assignments will be scaled down to 70 points at the end of the semester for the purpose of final grading. For example, if all homework assignments collectively carry 100 points and a student gets 90 out of 100, he/she gets $90 \cdot 70 / 100$ or 63 out of 70 in Homework Assignment component for final grading.

- Class attendance and participation

- Class attendance is not mandatory, although my exams will depend heavily of my lectures. Not all of the material will come from the text. Class participation is also important. Active interest in lectures is the easiest way to learn.

4

General issues (2)

- Plagiarism

- The penalty for copied homework of any kind can be immediate failure in the course. My policy on programs is as follows: There is no reason for two (or more) people handing in identical or nearly identical programs. I will regard such programs as either group-written or simply copied. If I have no hard evidence of copying, such programs will receive NO credit. More serious actions will be taken in cases where there is evidence of cheating.

- Late programs

- Unless otherwise stated, programs will lose 20% of their value for each weekday (Monday through Friday) that they are late, down to a minimum value of 20%. The due date of a program is the latest date on which it can be run to get full credit.

5

General issues (3)

- Dropping the course

- Unless there are extreme extenuating circumstances, I will not allow anyone to drop a course after the drop date. Poor academic performance will never be an acceptable reason for a late drop. The drop date for this course is October 28th.

- Incompletes

- Unless there has been a documentable illness that caused you to miss substantial amounts of class and computer time, I will not give an incomplete grade in this course. Therefore, please do NOT waste my time asking about an incomplete grade unless you have a remarkably good reason.

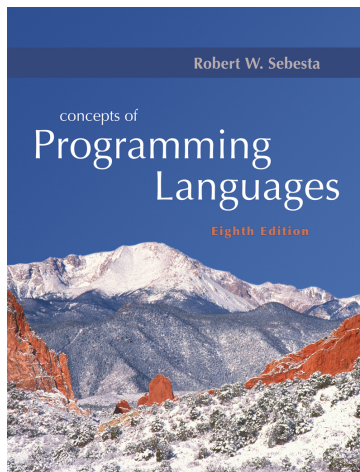
- Make-up exams

- I do not give make-up exams. You simply must show up and take them at the specified times.

6

Chapter 1

Preliminaries



ISBN 0-321-49362-1

7

Chapter 1 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

8

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

9

Programming Domains

- Scientific applications
 - Large numbers of floating point computations; use of arrays
 - Fortran
- Business applications
 - Produce reports, use decimal numbers and characters
 - COBOL
- Artificial intelligence
 - Symbols rather than numbers manipulated; use of linked lists
 - LISP
- Systems programming
 - Need efficiency because of continuous use
 - C
- Web Software
 - Eclectic collection of languages: markup (e.g., XHTML), scripting (e.g., PHP), general-purpose (e.g., Java)

10

Language Evaluation Criteria

- Readability: the ease with which programs can be read and understood
- Writability: the ease with which a language can be used to create programs
- Reliability: conformance to specifications (i.e., performs to its specifications)
- Cost: the ultimate total cost

Table 1.1 Language evaluation criteria and the characteristics that affect them.

Characteristic	CRITERIA		
	READABILITY	WRITABILITY	RELIABILITY
Simplicity/orthogonality	•	•	•
Control structures	•	•	•
Data types and structures	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

11

Evaluation Criteria: Readability

- Overall simplicity
 - A manageable set of features and constructs
 - Minimal feature multiplicity
 - Minimal operator overloading
- Orthogonality
 - A relatively small set of primitive constructs can be combined in a relatively small number of ways
 - Every possible combination is legal
- Control statements
 - The presence of well-known control structures

12

Excursus: Examples of Orthogonality

- Lack of orthogonality: C
 - Although 2 kinds of structured data types (arrays and records), records can be returned from functions, arrays cannot
 - Parameters are passed by value, unless they are arrays in which case they are passed by reference
 - Context dependency:
a + b
- The more orthogonal the design of a language, the fewer exceptions the language rule require. Also easier to learn.
- Too much orthogonality can also cause problems
 - ALGOL 68 is most orthogonal language
 - Unnecessary complexity due to extremely complex structures

13

Evaluation Criteria: Readability

- Overall simplicity
 - A manageable set of features and constructs
 - Minimal feature multiplicity
 - Minimal operator overloading
- Orthogonality
 - A relatively small set of primitive constructs can be combined in a relatively small number of ways
 - Every possible combination is legal
- Control statements
 - The presence of well-known control structures

14

Evaluation Criteria: Readability (2)

- Data types and structures
 - Adequate predefined data types and structures
 - The presence of adequate facilities for defining data structures
- Syntax considerations
 - Identifier forms: flexible composition
 - Special words and methods of forming compound statements
 - Form and meaning: self-descriptive constructs, meaningful keywords

15

Evaluation Criteria: 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
- Expressivity
 - A set of relatively convenient ways of specifying operations
 - Strength and number of operators and predefined functions

16

Evaluation Criteria: 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 require the use of "unnatural" approaches, and hence reduced reliability

17

Evaluation Criteria: Cost

- Training programmers to use the language
- Writing programs (closeness to particular applications)
- Compiling programs
- Executing programs
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

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

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

20

Computer Architecture Influence

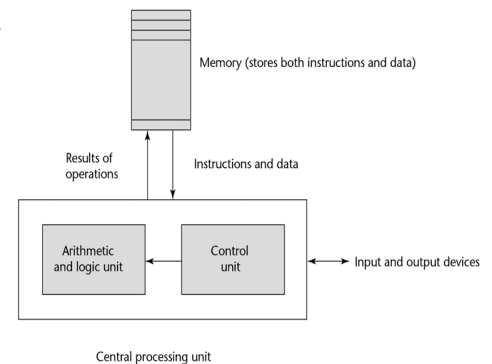
- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
 - Data and programs stored in memory
 - Memory is separate from CPU
 - Instructions and data are piped from memory to CPU
 - Basis for imperative languages
 - Variables model memory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

21

The von Neumann Architecture

Figure 1.1

The von Neumann computer architecture



22

The von Neumann Architecture

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

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

23

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

24

Language Categories

- Imperative
 - Central features are variables, assignment statements, and iteration
 - Include languages that support object-oriented programming
 - Include scripting languages
 - Include the visual languages
 - Examples: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Functional
 - Main means of making computations is by applying functions to given parameters
 - Examples: LISP, Scheme

25

Language Categories (2)

- Logic
 - Rule-based (rules are specified in no particular order)
 - Example: Prolog
- Markup/programming hybrid
 - Markup languages extended to support some programming
 - Examples: JSTL, XSLT

26

Language Design Trade-Offs

- Reliability vs. cost of execution
 - Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs
- Readability vs. writability
 - 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
 - Example: C++ pointers are powerful and very flexible but are unreliable

27

Implementation Methods

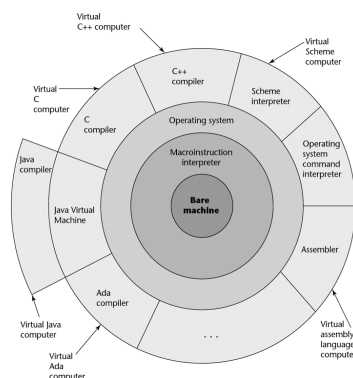
- Compilation
 - Programs are translated into machine language
- Pure Interpretation
 - Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems
 - A compromise between compilers and pure interpreters

28

Layered View of Computer

The operating system and language implementation are layered over machine interface of a computer

Figure 1.2
Layered interface of virtual computers, provided by a typical computer system



29

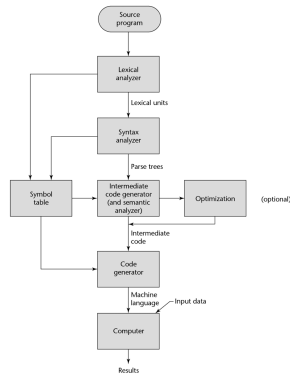
Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution
- Compilation process has several phases:
 - lexical analysis: converts characters in the source program into lexical units
 - 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

30

The Compilation Process

Figure 1.3
The compilation process



31

Additional Compilation Terminologies

- Load module (executable image): the user and system code together
- Linking and loading: the process of collecting system program units and linking them to a user program

32

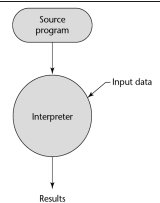
Von Neumann Bottleneck

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

33

Pure Interpretation

Figure 1.4
Pure interpretation



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

34

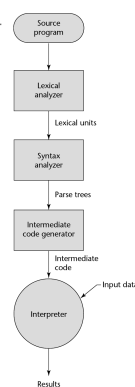
Hybrid Implementation Systems

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

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)

Figure 1.5
Hybrid Implementation system



35

Just-in-Time Implementation Systems

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

36

Preprocessors

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

37

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 runs 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#, and C++

38

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

39