

Programming Languages: Introduction

CS216

1

What Is a Programming Language?

- A language (formal notational system)
 - For describing **computations** so that they can be executed on a computer (machine)
- **Human-readable**
- **Machine-readable**

CS216

2

Describing Solvable Computations

- A programming language must be **universal**.
 - Any problem (if it can be solved at all by a computer,) must have a solution that can be programmed (expressed) in the language.

CS216

3

Universal – All Solvable Computations

- A programming language is **universal** if
 - integer values and arithmetic ops
 - variables
 - assignment statement
 - selection statement
 - loop statement/go to statement/recursion

CS216

4

Not-computable Computations (Problems)?

- Is there a not-computable problem at all?
 - Yes
 - The halting problem

CS216

5

Human Readability

- A programming language must be **human-readable**
- **How easy?**
 - Readability
 - Writability
- **How?**
 - Abstractions
 - Data abstraction
 - Control abstraction

CS216

6

Machine Readability

- A programming language must be **implementable** on a computer.
 - Every **well-formed** program in the language must be **executable** on a computer.
- **How efficient?**
 - Time
 - Space

CS216

7


Not-executable?

- Is there an unimplementable language?
 - Yes
 - Specification languages

CS216

8

Two Aspects of Programming Languages

- **Syntax**
 - Structure or form
 - **Semantics**
 - Meaning
- 
- **Human-readable**
 - **Machine-readable**

CS216

9

How to Implement Programming Languages?

- **Compilation**
- **Pure interpretation**
- **Hybrid – compilation + interpretation**

CS216

10

Compilation

- Translated into machine code by a program called a compiler.
- And then executed directly on the computer.
 - Slow translation
 - Fast execution

CS216

11

Phases in Compilation

- In a typical compiler, compilation proceeds through a series of well-defined phases.
- Each phase discovers information or transforms the program into a form of use to later phases.

CS216

12

Phases in Compilation

- Scanner (Lexical analysis)
- Parser (Syntax analysis)
- Semantic analysis
- Intermediate code generation
- Machine-independent code improvement (optimization)
- Target code generation
- Machine-specific code improvement (optimization)

CS216

13

Recognizing the Structure of the Program

- Scanner (Lexical Analysis)
 - A stream of characters
 - A stream of tokens
- Parser (Syntax Analysis)
 - A stream of tokens
 - A parse tree

CS216

14

Discovering the Meaning of the Program

- Semantic Analysis
 - A parse tree (concrete syntax tree)
 - An abstract syntax tree (syntax tree)
- Builds and maintains a symbol table.
 - Symbol table serves as a repository for information about identifiers throughout compilation.
- Intermediate Code Generation
 - An abstract syntax tree (syntax tree)
 - An intermediate code (form)

CS216

15

Translating Into Target Code

- Semantic-preserving translation
- Target Code Generation
 - A modified intermediate code (form)
 - A target (assembly or machine) language

CS216

16

Improving (Optimizing) Code

- Machine-independent code improvement
 - An intermediate code (form)
 - A modified intermediate code (form)
- Machine-dependent code improvement
 - A target (assembly or machine) language
 - A modified target language
- Optional!

CS216

17

The Compilation Process

- The compilation process
 - See fig 1.3

CS216

18

Pure Interpretation

- Executed directly by a program called an interpreter.
 - No translation
 - Slow execution
 - More space
 - Source-level debugging

CS216

19

The Pure Interpretation Process

- The pure interpretation process
 - See Fig. 1.4

CS216

20

Compilation + Interpretation

- Compiled first and then interpreted
 - Hybrid implementation
 - Small translation cost
 - Medium execution speed

CS216

21

The Compilation + Interpretation Process

- The compilation & interpretation process
 - See Fig. 1.5

CS216

22

Programming Language Design

- Primary influences on programming language design:
 - Computer architecture
 - Machine efficiency
 - Programming methodologies
 - Human efficiency

CS216

23

Computer Architecture

- The basic architecture of computers
 - The von Neumann machines
 - A sequential machine
 - See fig 1.1
 - Imperative programming languages – based on variables and assignments

CS216

24

Programming Methodologies/Paradigms

- Imperative programming
- Functional programming
- Object-oriented programming
- Logic programming
- Concurrent programming
- ...

CS216

25

Categories of Programming Languages

- Imperative languages
 - Procedure-oriented
- Functional languages
 - Function-oriented
- Logic languages
 - Rule-based
- Object-oriented languages
 - Closely related to imperative
- Domain-specific languages
 - Scripting, Special-purpose

CS216

26

Programming Languages vs Programming Paradigms

- ?
- Imperative languages for Imperative Programming
- Functional languages for Functional Programming
- Object-oriented languages for Object-oriented programming
- Logic languages for Logic Programming

CS216

27

How to Evaluate Programming Languages?

- Readability
- Writability
- Reliability
- Cost
- Others:
 - Portability, generality, well-definedness

CS216

28

Readability

- The ease with which programs can be read and understood
- Factors:
 - Overall simplicity
 - Too many features is bad.
 - Multiplicity of features is bad.
 - Operator overloading

CS216

29

Readability

- **Orthogonality:** *A relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language.*
 - Lack of orthogonality leads to rule exceptions.
 - Makes the language easy to learn and read.
 - Too much orthogonality?
- Sufficient control statements – No go-to's
- Sufficient data types and structures

CS216

30

Writability

- A measure of how easily a language can be used to create programs for a chosen problem domain.
- Factors:
 - Simplicity and orthogonality
 - Support for abstraction – process and data abstraction
 - Expressivity

CS216

31

Reliability

- A program is reliable if it performs to its specifications under all conditions.
- Factors:
 - Type checking
 - Exception handling
 - Aliasing
 - Readability and writability

CS216

32

Cost

- Cost for
 - Programmer training
 - Software creation
 - Compilation
 - Execution
 - Compiler cost
 - Maintenance

CS216

33

Other Criteria

- **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 defining document

CS216

34

Why Study Concepts of PLs?

- To increase capacity to express programming concepts.
- To improve background for choosing appropriate languages.
- To increase ability to learn new languages.
- To understand the significance of implementation.
- To increase ability to design new languages.

CS216

35

In This Course

- Foundations (syntax + semantics)
- Fundamental Concepts
- Implementations
- Paradigms
- Programming
 - SML
 - Prolog

CS216

36

Project

- Research on four languages supporting different programming paradigms
 - Two languages of your choice

CS216

37

High-level Programming Languages

- How many languages?
 - ?

CS216

38

Major High-level Programming Languages So Far

- | | | |
|-----------|---------------|----------------|
| • FORTRAN | • PASCAL | • C++ |
| • LISP | • C | • ML/ SML |
| • ALGOL | • SCHEME | • Quick BASIC |
| • COBOL | • MODULA | • Visual BASIC |
| • BASIC | • PROLOG | • HASKELL |
| • PL/I | • ADA | • Eiffel |
| • APL | • SMALLTALK | • JAVA |
| • SNOBOL | • ICON | • JavaScript |
| • SIMULA | • COMMON LISP | • C# |
| | | • Perl |
| | | • php |
| | | • Python |
| | | • Ruby |
| | | • ... |

CS216

39

The First High-level Language: **FORTRAN**

- The first high-level programming language
 - **FORTRAN** (**FOR**mula **TRAN**slator) I
 - FORTRAN II
 - FORTRAN 77
 - FORTRAN 90
 - For scientific applications
- First implemented version of FORTRAN
 - Names could have up to six characters
 - User-defined subprograms
 - Three-way selection statement (arithmetic IF)

CS216

40

The First Functional Language: **LISP**

- The first functional language
- **LISP** (**LI**St **P**rocessing language) – 1959
 - For list processing and AI applications
- Pioneered functional programming
 - No need for variables or assignment
 - Control via recursion and conditional expressions

CS216

41

Functional Languages: Descendants of LISP

- **Scheme**
- **COMMON LISP**
- **ML** (**MetaLanguage**)
- **SML** (**Standard ML**)
- **Miranda**
- **Haskell**

CS216

42

The First Step Toward Sophistication: **ALGOL**

- **ALGOL (ALGOrithmic Language) 58** - 1958
- Language Features:
 - Concept of types was formalized
 - Names could have any length
 - Arrays could have any number of subscripts
 - Parameters were separated by mode (in & out)
 - Subscripts were placed in brackets
 - Compound statements (begin ... end)
 - Semicolon as a statement separator
 - Assignment operator was :=
 - if had an else-if clause

CS216

43

ALGOL60

- New Features:
 - **Block structure (local scope)**
 - **Two parameter passing methods: pass by value & pass by name**
 - **Subprogram recursion**
 - **Stack-dynamic arrays**
 - First language whose syntax was formally defined (using BNF).
- **All subsequent imperative languages are based on it.**
 - **“Algol-like” programming languages**

CS216

44

The First Language For Business Application: **COBOL**

- **COBOL (COmmon Business Oriented Language)** - 1960
 - Designed for business applications.
- Contributions:
 - First macro facility in a high-level language
 - Hierarchical data structures (records)
 - Nested selection statements
 - Long names (up to 30 characters), with hyphens
 - Data Division
- Still the most widely used business applications language

CS216

45

The Beginning of Timesharing: **BASIC**

- **BASIC (Beginner's All-purpose Symbolic Instruction Code)** - 1964
- For:
 - Easy to learn and use for non-science students
 - Extremely simple syntax and semantics
- Current popular dialects:
 - **QuickBASIC**
 - **Visual BASIC**

CS216

46

The First Language For “Everything For Everybody”: **PL/I**

- **PL/I (Programming Language/I)**- 1965
- Contributions:
 - First unit-level **concurrency**
 - First **exception handling**
 - Switch-selectable recursion
 - First **pointer** data type

CS216

47

The Beginnings of Data Abstraction: **SIMULA67**

- **SIMULA 67** - 1967
 - For system simulation
 - Based on ALGOL 60
- Contributions:
 - **Coroutines** - a kind of subprogram
 - Implemented in a structure called a **class**
 - Classes are the basis for data abstraction
 - Classes are structures that include both local data and functionality

CS216

48

Orthogonal Design: **ALGOL68**

- **ALGOL 68** - 1968
 - Based on the concept of **orthogonality**
- Contributions:
 - **User-defined data structures**
 - Reference types
 - **Dynamic arrays**
- Comments:
 - Had even less usage than ALGOL 60.
 - Had strong influence on subsequent languages, especially Pascal, C, and Ada.

CS216

49

Simplicity by Design: **PASCAL**

- **Pascal** - 1971
 - For teaching **structured programming**
 - Small, simple, nothing really new
 - Most widely used language for teaching programming in colleges

CS216

50

A Portable System Language: **C**

- **C** - 1972
 - For **systems programming**
 - Evolved primarily from B, but also ALGOL 68
 - Powerful set of operators, but **poor type checking**.
 - Initially spread through UNIX.

CS216

51

The First Language Based on Logic: **PROLOG**

- **PROLOG** (**PRO**gramming in **LOGic**)- 1972
 - Based on formal logic
 - Non-procedural
 - Being an intelligent database system that uses an inferencing process to infer the truth of given queries

CS216

52

History's Largest Design Effort: **ADA**

- **Ada** - 1983
 - Huge design effort, involving hundreds of people, much money, and about eight years
- Contributions:
 - **Packages** - support for data abstraction
 - Exception handling - elaborate
 - **Generic** program units
 - Concurrency - through the tasking model
- Included all that was then known about software engineering and language design.

CS216

53

Object-Oriented Language: **SMALLTALK**

- **Smalltalk** – 1980
- First full implementation of an object-oriented language
 - data abstraction, inheritance, and dynamic type binding
- Pioneered the **graphical user interface** everyone now uses

CS216

54

Combining Imperative and OO Features: C++

- Developed at Bell Labs by Stroustrup in 1985
- Facilities for object-oriented programming, taken partially from SIMULA 67, were added to C.
- A large and complex language
- Rapidly grew in popularity, along with OOP.
- ANSI standard approved in November, 1997.

CS216

55

Programming the WWW: JAVA

- Programming the World Wide Web.
- Developed at Sun in the early 1990s.
- Based on C++
 - Significantly simplified.
 - Supports only OOP.
 - Has references, but not pointers.
 - Includes support for applets and a form of concurrency.

CS216

56

Programming Future: Library & Scripting

- Growing importance on libraries
 - Interface with OS and Hardware
 - A rich library
 - Integrated with the programming languages
- Scripting languages
 - Ties utilities together

CS216

57

Summary: High-level Programming Languages

- A big picture:
- See Fig. 2.1!

CS216

58