[PROGLANG]

NO.
DATE

Why study programming languages?

- increase ability to express ideas
- improve background for choosing language
- increased ability to learn new languages
- better understanding of significance of implementation
- better use of known languages
- overall advancement of computing

Programming Domains

* Scientific (Algol, Fortran)
  - large numbers of floating point computations
  - use of arrays
* Business (COBOL)
  - produce reports
  - use decimal and characters
* Artificial Intelligence (Lisp)
  - symbols rather than numbers manipulated
  - use of linked lists
* Systems Programming (C)
  - needs efficiency because of usage
* Web Software (HTML, PHP, Java)
  - Eclectic collection of languages
  - markup, scripting, general-purpose

Language Evaluation Criteria

1. Readability - can programs be read and understood easily?
2. Writability - can programs be made easily?
3. Reliability - conformance to specifications?
4. Cost - how much does the program cost?

Readability

Overall Simplicity
- manageable set of features and constructs
- minimal feature multiplicity
- minimal operator overloading

Orthogonality
- minimal constructs = minimal combinations
- every possible combination is legal

Data Types
- predefined data types

Syntax Considerations
- identifier forms: flexible composition
- special words + methods
- Form and meaning

Writability

Simplicity and Orthogonality
Support for Abstraction
Expressivity

Reliability

Type Checking
- testing for type errors

Exception Handling
- intercept run-time errors and corrects them

Aliasing
- many referencing methods for same memory

Readability & Writability

Cost

· Training · Writing · Maintenance
· Executing · Reliability

NO.
DATE

Generality
- applicable to a range of applications

Portability
- move from one implementation to another

Well-definedness
- completeness and precision of language

Language Categories
Imperative - sequence of instructions
- C, Java, Perl, .NET, JavaScript
Functional - evaluation of mathematical functions
- Lisp, Scheme, ML, F#
Logic - principles of mathematical logic
- Prolog
Markup / Programming Hybrid
- JSTL, XSLT

Influences on Language Design
Computer Architecture
- Von Neumann architecture
Program Design Methodologies
- software development methodologies
- new programming paradigms
- new programming languages

Computer Architecture Influence
- Imperative language
- Von Neumann computer
→ data stored in memory
→ CPU ≠ RAM
→ instructions are piped from memory to processor

Programming Methodologies Influence
1950 and early 1960
- simple apps, machine efficiency
late 1960
- structured programming, readability
late 1970
- process-oriented to data-oriented
middle 1980
- object-oriented programming

Language Design Trade-Offs
Reliability vs. Cost of Execution
Readability vs. Writability
Flexibility vs. Reliability

Implementation Methods
Compilation - translate into machine code
- large commercial applications
Pure Interpretation -
- small programs / non-efficient
Hybrid Interpretation - compromise
- small and medium systems

Compilation
→ slow translation, fast execution
→ lexical analysis, syntax analysis
semantics analysis, code generation

Pure Interpretation
→ no translation, slower execution
→ requires more space
→ easier implementation of programs

NO.
DATE

Hybrid Implementation
→ high-level translated into intermediate language for easy interpretation
→ faster than pure interpretation

Just-In-Time Implementation
→ delayed compilers
→ translate into intermediate language and compiled into machine code
→ machine code is saved for calls

Preprocessors
→ specify code from another file is to be included

Programming Environment
→ collection of tools used in S.D.
→ UNIX, Visual Studio, Netbeans

SEMANTICS SYNTAX

NO.
DATE

Examining a Programming Language

- form and structure : Syntax
- meaning of the form and structure : Semantics

- semantics should follow directly from syntax.
- a statement's appearance should suggest its usage.

Terminology

Sentence - a string of characters over alphabet
Language - a set of sentences
Lexeme - lowest level syntactic unit
Token - a category of lexemes

Formal Definition of Languages

Recognizer (parser)	Generator (grammar)
- reads input strings	- generate sentences
- syntax analysis - determines if input belongs to a language	- comparison determination

Backus - Naur Form / Context - Free Grammar

- Noam Chomsky (1950s)
- Language generators, describe syntax of natural L.
- John Backus, BNF = CFG

BNF Fundamentals

Metalanguage - language used to describe another
Rule / Production - the entire definition

LHS	RHS
abstraction defined	definition

Grammar - finite nonempty set of rules
Abstractions - nonterminal
Lexemes / Tokens - terminal
Derivation - repeated application of rules

Parse Tree

- hierarchical representation of a derivati
- leaf node → terminal
- internal node → nonterminal

Extended BNF

increased readability and writability

1. optional part of RHS [ ] brackets
2. implied iteration of Tokens { } braces
3. multiple choice options (|)

Attribute Grammars

Static Semantics — before execution compile tim
- legal forms of programs (syntax)

Dynamic Semantics
- deals with the meaning during executio

1. Attributes - similar to variables, have va
2. Attribute computation Function - how values are computed
3. Predicate Functions - static semantic
   - rules of the language
   - specify conditions needed to compute the values of attributes

NO.
DATE

Why need a methodology and notation for describing semantics
- programmers must know the use of statements
- compiler writers must know what constructs do
- Correctness proofs would be possible
- Compiler generators would be possible
- designers could discover inconsistencies

Describing Programming Semantics
Operational
- method of describing the meaning of language constructs of their effects
Axiomatic
- based on formal logic for proving the correctness of programs
Denotational
- mathematical objects with recursive functions are used to represent meanings.

Operational Semantics
- executes compiled version on a pc
- sequence of state changes
- translator and simulator
- Natural ; focuses on final result
- Structural ; complete sequence of changes

Denotational Semantics
- mathematical objects denote the meaning of syntactic entities
- mathematical object → function → entity
- used to prove correctness of programs
- little use to language users, used in compiler generation.

Axiomatic Semantics
- formal program verification
- inference rules are used
- program verification and semantics specs
- precondition - constraints, relations
- postcondition - assertion after a statement
weakest; least restrictive for post condition
- good for correctness proofs
- excellent framework for reasoning

LEXICAL SYNTAX ANALYSIS

No.
DATE

Lexical Analysis & Syntax Analysis

Simplicity - lexical is less complex
Efficiency - lexical can be optimized due to compilation time
Portability - lexical is platform dependent
syntax is platform independent

LEXICAL ANALYSIS

Lexical Analyzer

- pattern matcher for character strings
- front-end of the parser (input)
- Lexemes : substrings that belong together
- token : lexical category → output
- scans pure HLL line by line, removes comments and whitespaces

Building a Lexical Analyzer

1. write a formal description of the tokens and use a software tool that constructs table-driven lexical analyzers using
1. Design a state-diagram that describes the tokens and write a program that implements the state diagram
3. Design a state-diagram that describes the tokens and hand-construct a table driven implementation of the state diagram

Parsing Problem (Syntax Analyzer)

1. Check for syntax errors and produce a diagnostic message and recover
2. Produce a complete parse tree, or trace the structure of the complete parse tree

Top-down Approach
Bottom-up Approach

Time Complexity

Unambiguous Grammar = $O(n^3)$
Commercial compilers = $O(n)$

Recursive-Descent Parsing

- also known as top-down
- subprogram for each nonterminal
- EBNF is suited for such because it minimizes number of nonterminals

1. Left-to-right
2. If with options, Left-to-right } Left Recursio
3. Epsilon is arrow to right

Pairwise Disjoint Test

- cannot top-down parse if fail
- compare nonterminals that share common prefix
- if the first set of alternatives in the suffix appears in both, the test fails.

NO.
DATE

Bottom-Up Parsing

- expand right-most nonterminal at each step (rightmost derivation)
- Handle : string of symbols to be replaced at each stage of parsing

LR Parsing Algorithm

1. left-to-right scanning of input string
2. start with right most derivation

Advantage :

- work on almost all grammars
- work on larger larger class of grammars
- detect syntax errors ASAP
- LR is a superset of LL

Knuth's Insight

- bottom-up can use the entire history of the parse to make decisions

Shift-Reduce Algorithm

- used by bottom-up parsers
- Reduce : replacing the handle on top of the stack with its LHS
- Shift : moving next token to top of the parse stack.