

Overview

Reasons for studying concepts of programming languages:

1. To be more expressive
2. Improve ability to choose appropriate languages
3. Improve ability to learn new languages
4. Better understanding of the significance of implementation
5. Increase ability to design new languages
6. Overall advancement of computing

Typical programming domains:

1. Scientific applications
simple data structures, large amount of floating-point arithmetic computations, efficiency
e.g.: Fortran, C/C++, Algol 60
2. Business applications
elaborate input and output facilities, decimal data types, spread-sheet, databases
e.g.: Cobol, SQL
3. Artificial intelligence
symbolic processing, lists as primary data structures
e.g.: Lisp, Prolog, OPS-5
4. System programming
efficiency, low-level features, portable
e.g.: PL/S (IBM), C/C++, Assembly
5. Scripting programming
string processing, pattern matching, closely integration with file systems
e.g.: tcl, awk, bash, sh
6. Web-based applications
platform independent,
e.g.: html, xml, VB-script, java-script, java servlets/JSP
7. Special-purpose
e.g.: RPG: used to produce business reports,
GPSS: used for system simulation

Some Important Language Evaluation Criteria:

1. Readability

- a. Overall Simplicity: in terms of the number of elementary features
Feature Multiplicity: having more than one way to accomplish a particular operation

e.g.: $c = c + 1;$
c++;
++c;
c += 1;

Operator Overloading: a single operator symbol has more than one meaning.

sensible overloading operator symbols is important

b. Orthogonality:

- (1) a small set of primitive constructs that can be combined in a relatively small number of ways to build the control and data structures of the language
- (2) every possible combination is valid and meaningful

The more orthogonal the design of a language, the fewer exception rules the language required. It implies that the language is easier to learn, read, and understand.

e.g.: the type rules of C are not orthogonal, for instance,
functions can return only unstructured types.
files cannot be passed by value.
etc.

Too much orthogonality can also cause problems.

It may lead to unnecessary complexity

It may result in an explosion of combinations.

c. Control Statements:

d. Data Structures:

e. Syntax Design:

2. Writability

A measure of how easily a language can be used to create programs for a chosen problem domain

3. Reliability

A program is reliable if it performs to its specification under all conditions.

Features of a language that may lead to reliable programs:

- Type Checking: testing for type compatibility between two variables or a variable and a constant.
- Exception handling: The ability of a program to intercept run-time errors and other unusual conditions, to take corrective measures, and to continue.
- Aliasing: having two distinct referencing methods or names for the same memory cell. It is a dangerous feature.

4. Cost

- Programmer Training: a function of simplicity, orthogonality, experience
- Program Writing: a function of writability
- Compilation: a function of required computing resources
- Execution: a function of generated code
- Maintenance: a function of reliability and readability

Influences on Language Design

1. Computer Architecture

Von Neumann architecture

- Data and programs are stored in the same memory
- CPU is separated from memory
- Data and instructions must be piped from memory to CPU
- Results of operations are stored back to memory

Related Programming Constructs:

- Variables model memory cells
- Assignment: based on piping operation
- Iterative form of repetition is most efficient, because instructions are stored in adjacent memory cells
- Recursive form of repetition is discouraged

Non-Von Neumann architectures

Data-flow computers: support functional languages

2. Programming Methodologies

Cost of software and hardware:

1970s language deficiencies:

- incompleteness of type checking
- inadequacy of control statements leads to extensive use of goto's
- lack of facilities for exception handling

Shift from operation abstraction to data abstraction:

- Inheritance: reuse
- Dynamic type checking: generic types

Concurrent Programming:

Language Design Trade-Offs

1. Reliability vs. execution efficiency
e.g.: array index range checking
Ada trade efficiency for reliability
2. Writability vs. readability
e.g.: APL trade readability for writability
3. Flexibility vs. safety
e.g.: Pascal variant records: trade safety for flexibility

Language Implementation Methods

Level of abstraction:

Bare Machine

Microinstruction Interpreter

Operating Systems

Assemble, OS shells, Lisp interpreter, Pascal compiler, C/C++ compiler

Bison

1. Compilation

Process of compilation:

Source program: a sequence of strings

Lexical Analyzer (Scanner)

Output is a sequence of tokens (lexical units)

Syntax Analyzer (Parser)

Outputs are parse trees

Intermediate Code Generator

Code Optimizer (Optional)

Code Generator

Output is executable machine code

2. Pure Interpretation

3. Hybrid implementation: