Preface

- Purpose of this course
  - Evaluation of existing and future PL and constructs
  - Prepare the concept of the compiler design and construct
- Two approaches of introduction
  - Horizontal : each PL is presented in some depth
  - Vertical: the general concepts and constructs of PL
- This book uses most of the vertical approach(Chap. 4 - 13).

Chapter 1

Reasons to study concepts of PLs(P. 1-5)

1. Increased capacity to express ideas programming concepts(control & d.s.)
2. Improved background for choosing appropriate languages
3. Increased ability to learn new languages(new functions are evolved) Thorough understanding concepts of PL is necessary
4. Understanding the significance of implementation(why lang. design & imple.)
5. Increased ability to design new languages when necessary
6. Overall advancement of computing users were unawareness of the benefits of PL played a important role

Chapter 1

Programming Domains (p. 5-8)

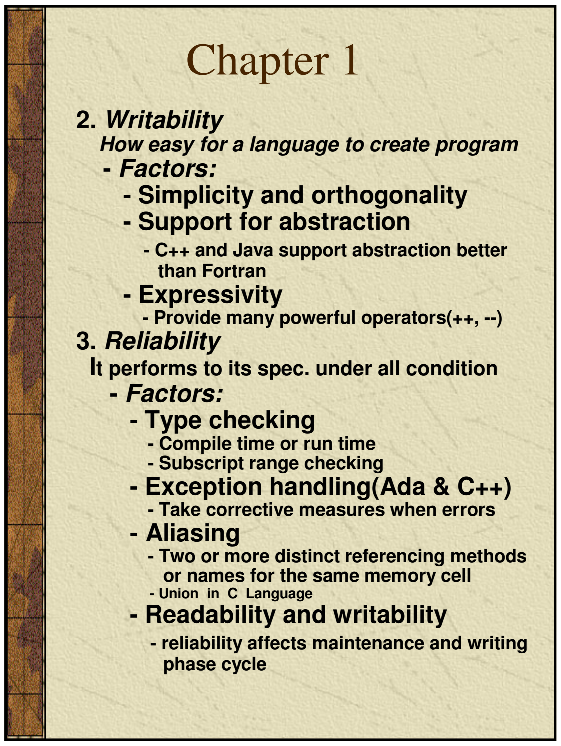
1. Scientific applications
   CPU bounded process – large number of real number computation
   ALGOL 60 & Fortran
2. Business applications
   spreadsheet & database System
   COBOL
3. Artificial intelligence
   symbolic & linked–list computation
   LISP, Prolog, Scheme
4. Systems programming
   OS & programming support tools
   PL/I, BLISS, ALGOL, C
   IBM Digital Burroughs
5. Scripting languages
   shell(c & korn), awk, tcl, Perl for CGI
6. Special purpose languages
   RPG, APT, GPSS

Chapter 1

Language Evaluation Criteria (p. 8-20)

1. ***Readability***
   How easy for a program to be understood
   The most important criterion for maintain
   - *Factors:*
     - Overall simplicity(1)
       - Too many features is bad(many basic component but just used partial)
       - Multiplicity of features is bad(more than one way ➔ ++,--)
       - Operator Overloading is bad(java or C++ provide, +)
     - Orthogonality(2)
       - A relative small set of primitive constructs
       - Makes the language easy to learn and read
       - Meaning is context independent

       Functional language(LISP) offers good 1 and 2
     - Control statements
       - No goto statement
       - Use While loop instead
     - Data type and structures
       - User-defined data type
     - Syntax considerations
       - Identifier forms limitation(short variable length)
       - Special words(begin end)
       - Form and meaning(static variable in C)



Chapter 1

2. ***Writability***

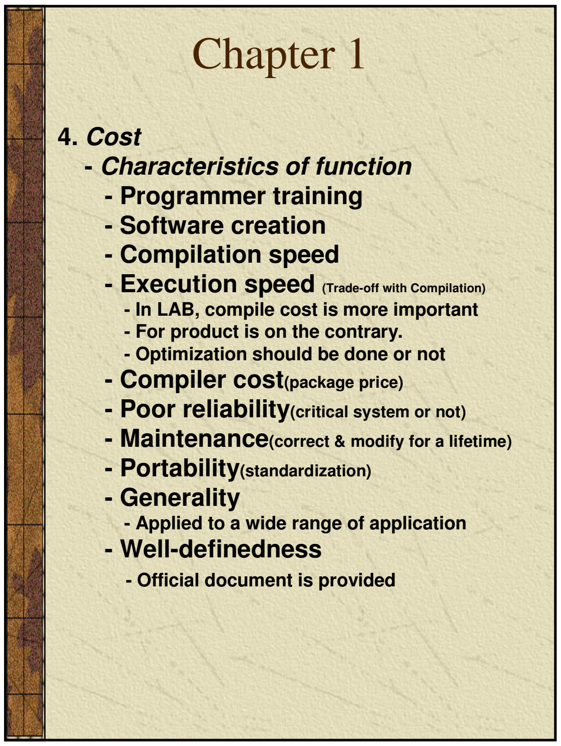
   How easy for a language to create program

   - *Factors:*
     - Simplicity and orthogonality
     - Support for abstraction
       - C++ and Java support abstraction better than Fortran
     - Expressivity
       - Provide many powerful operators(++, --)

3. ***Reliability***

   It performs to its spec. under all condition

   - *Factors:*
     - Type checking
       - Compile time or run time
       - Subscript range checking
     - Exception handling(Ada & C++)
       - Take corrective measures when errors
     - Aliasing
       - Two or more distinct referencing methods or names for the same memory cell
       - Union in C Language
     - Readability and writability
       - reliability affects maintenance and writing phase cycle



Chapter 1

4. ***Cost***

   - *Characteristics of function*
     - Programmer training
     - Software creation
     - Compilation speed
     - Execution speed (Trade-off with Compilation)
       - In LAB, compile cost is more important
       - For product is on the contrary.
       - Optimization should be done or not
     - Compiler cost(package price)
     - Poor reliability(critical system or not)
     - Maintenance(correct & modify for a lifetime)
     - Portability(standardization)
     - Generality
       - Applied to a wide range of application
     - Well-definedness
       - Official document is provided

Chapter 1

Primary influences on language design (p. 20-23)

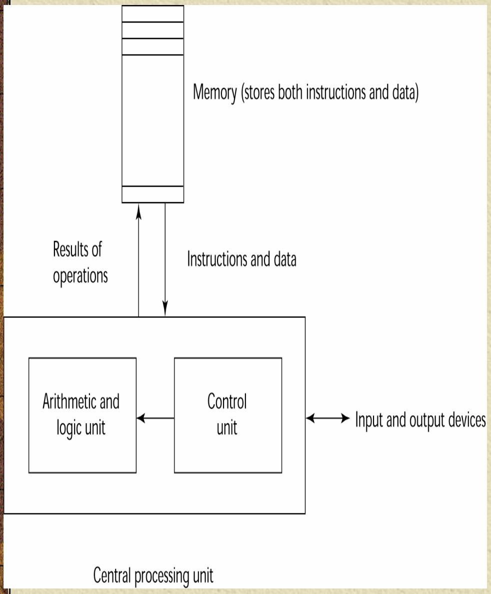
1. *Computer architecture*

- We use **imperative languages**, at least in part, because we use von Neumann machines. Both data and programs are stored in the same memory

2. *Programming methodologies*

- *1950s and early 1960s:* Simple applications; worry about machine efficiency
- *Late 1960s:* People efficiency became important; readability, better control structures(top-down design)
- *Late 1970s:* Data abstraction(data-oriented)
- *Middle 1980s:* Object-oriented programming inheritance & dynamic type binding
- Recently*: process-oriented(concurrency)* Ada and Java

Figure 1.1 The von Neumann computer architecture



Chapter 1

Language Categories
(p. 23-24)

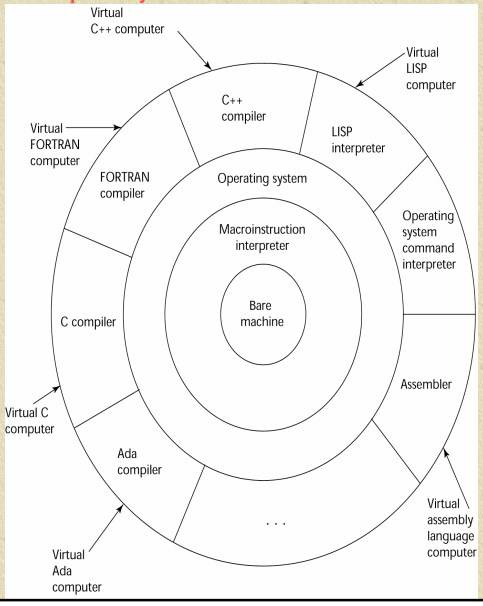
1. Imperative – meet Von-Neumann machine
   detail algorithm P. 20
2. Functional – AI (LISP …) P. 12
   computations are made by applying functions to give parameters
3. Logic – a rule-based language(Prolog)
4. Object-oriented(closely related to imperative) v.s. Procedure-oriented
5. Makeup – HTML, XML, WML

Language Design Trade-offs
(p. 24-25)

1. Reliability versus cost of execution(index)
   Ada C
2. Writability versus readability(power operator)
   APL (APL is poor)
3. Flexibility versus safety(pointer operation)
   Pascal (Pascal is poor)

Figure 1.2

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



Chapter 1

Implementation Methods
(p. 25-31)

Macro-instruction + Micro-instruction
= machine instruction
OS + C Compiler = Virtual C Computer

1. Compilation
   - Translate high-level program to machine code
     Lexical Analyzer, Syntax Analyzer,
     Intermediate code generator(Semantics Analyzer) ,
     Optimization , Code Generation ➔ use symbol table
   - Slow translation + Linker
   - Fast execution
   - von Neumann bottleneck
connection between computer's memory and CPU
   C, COBOL, Ada, Fortran

2. Pure interpretation for source-code debugger
   - No translation
   - fetch-cycle
   - Slow execution(every-time statement decoding)
   - Becoming rare
   APL, LISP, shell in UNIX, javascript

3. Hybrid implementation systems
   - Small translation cost
   - Medium execution speed
   Perl – sh, awk

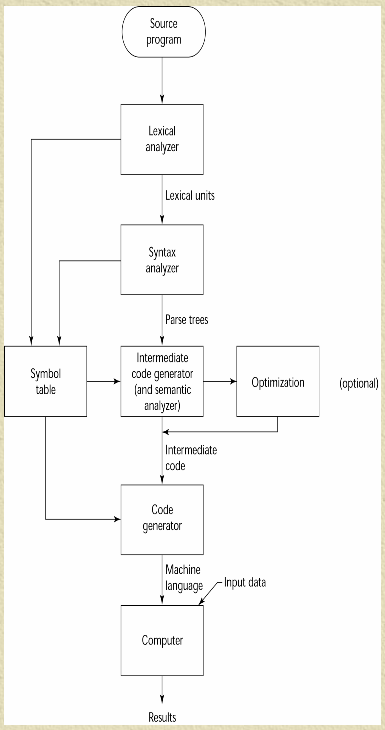


Figure 1.3
The compilation process

Figure 1.4
Pure Interpretation

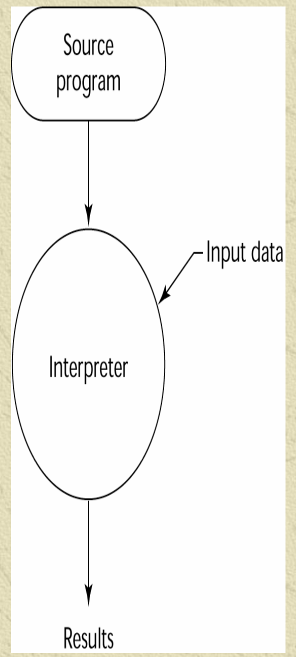
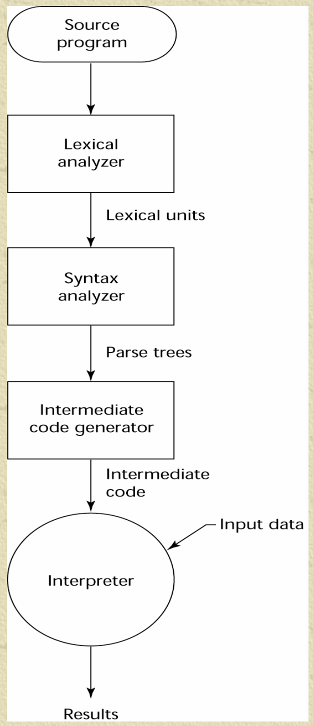


Figure 1.5
Hybrid Implementation system



Chapter 1

Programming Environments
(p. 31-33)

-The collection of tools used in software development –

a file system, Text editor, linker, and a compiler

1. UNIX
   - An old operating system and tool collection(Common Desktop Envi.)

2. Borland C++
   - A PC environment for C and C++

3. Smalltalk
   - A language processor/environment

   1st system to use window system & mouse

4. Microsoft Visual C++
   - A large, complex visual window environment
5. VB, Dephi, Jbuilder …