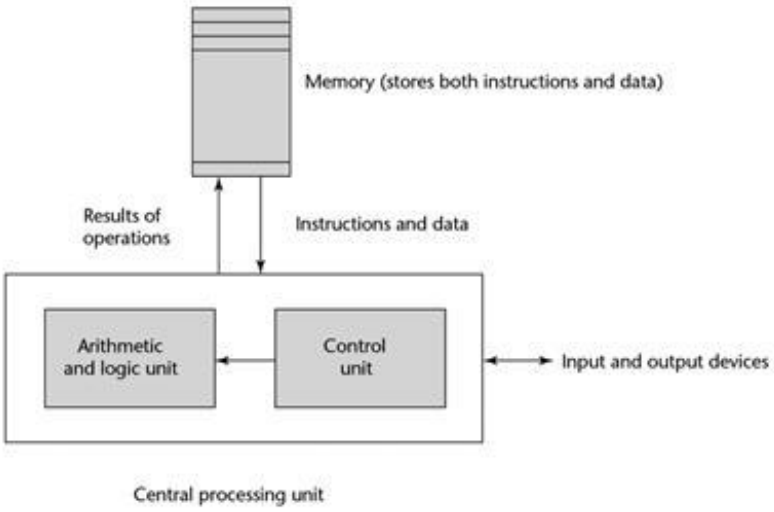
Preliminaries – Chapter 1 (continued)

Influences on Language Design

In addition to the factors previously discussed, the most important factors that influence the basic design of programming languages are computer architecture and program design methodologies.

Computer Architecture

- von Neumann architecture – John von Neumann
  - Past 50 years popular programming languages based upon it – Imperative languages.
  - Data and programs stored in the same memory
  - CPU is separate from memory.
  - Instructions and data must be piped/transmitted from memory to CPU.
  - CPU results are moved back to memory.
  - Nearly all digital computers based on this sine the 1940s.



- "Imperative" languages – central features are:
  - Variables – model memory cells.
  - Assignment statements – based upon piping operation.
  - Iterative repetition – most efficient for this architecture – instructions are stored in adjacent memory cells. Discourages recursion.

- "Functional" or "Applicative" – central features are:
    - Primary means of computation is applying functions to given parameters.
    - Programming done without the kind of variables used with imperative languages, without assignment statements, without iteration.
    - Not likely to displace imperative languages unless the architecture is changed.
    - Parallel architecture machines hold some promise. Although most parallel machines are used for imperative programs, particularly those written in dialects of Fortran.

Programming Methodologies

- Intense analysis of both the software development process and programming language design began in the late 1960s and early 1970s in large part because of the structured programming movement.
- Shift in the major cost of computing from hardware to software – hardware costs decreased and software costs increased.
- Larger and more complex problems being solved by computers.
- Top-down design and stepwise refinement emerged from the research of the 1970s.
- Shift from procedure-oriented to data-oriented program design methodologies – abstract data types – late 1970s.

Data-oriented programming

- SIMULA 67 – first language to provide limited support for data abstraction.
- Most languages designed since the late 1970s support data abstraction.
- Object-oriented design – latest step in the evolution of data-oriented software development.
- Object-oriented methodology begins with data abstraction – encapsulates processing with data objects – controls access to data – adds inheritance and dynamic method binding. Inheritance greatly enhances the reuse of existing software – provides significant increases in software development productivity.
- Smalltalk 1989.
- Object-oriented programming is now part of most popular imperative languages: Ada 95, Java, and C++.
- Object-oriented programming also included in some functional programming, CLOS, and some logical programming, Prolog++.

Procedure-oriented programming

- In a sense, the opposite of data-oriented programming.
- A good deal of research has occurred in the area of concurrency involving procedure-oriented methods.
- Ada, Java, and C include capabilities for creating and controlling concurrent program units.

Language Categories

Programming languages are often separated into four categories: imperative, functional, logical, and object-oriented.

Imperative

- Basic features previously discussed.

Functional

- Basic features previously discussed.

Object-oriented

- Primarily a subcategory of imperative languages.

Visual language

- Subcategory of the imperative languages.
- Visual BASIC – most popular visual language – now Visual BASIC .NET.
- Include capabilities for drag-and-drop code segment generation.
- Provides a simple way to generate graphical user interfaces to programs.
- Formerly referred to as fourth-generation languages.

Logical

- Rule-based language.
- Rules are specified in no particular order.
- Implementation system must choose an execution order that produces the desired result.
- Prolog – most commonly used.

Markup

- Not a programming language.
- XHTML – most widely used.
- Specifies layout information in Web documents.
- Some programming capabilities in Extensions to XHTML and XML.
    - JSTL – Java Server Pages Standard Tag Library.
    - XSLT – eXtensible Stylesheet Language Transformations.

Special-purpose

- RGP – Report Program Generator – business reports.
- APT – Automatically Programmed Tools – instructing programmable machine tools.
- GPSS – General Purpose Simulation System – systems simulations.

Language Design Trade-offs

Conflicting criteria.

Reliability and Cost

- Java – requires index range checking of all references to array elements – this adds a great deal to the cost of execution – more reliable.
- C – does not require index range checking – fast execution and low execution cost – less reliable.

Writability and Readability

- APL
    - Includes powerful set of operators for array operands – includes a large number of symbols to represent the operators – this can lead to large complex expressions in a very compact program.
    - This high expressivity leads to strong writability and very poor readability.

Writability and Reliability

- C++ – pointers can be manipulated in a variety of ways – this leads to highly flexible addressing of data.
- Java – does not include pointers because of potential reliability problems.

Implementation Methods

Two of the primary components are:

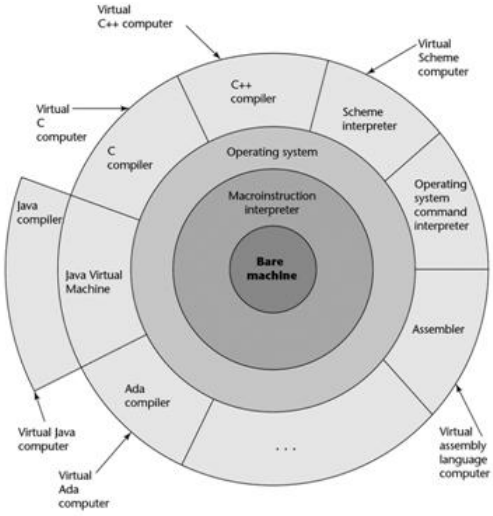
- Internal memory
  - Stores programs and data.
- CPU/Processor
  - Collection of circuits that provide primitive operations or machine instructions, sometimes called macroinstructions.
  - Implemented with lower level instructions called microinstructions.

Machine language

- Computers instruction set.
- Only language that most hardware computers understand.
- Provides the most commonly needed primitive operations.
- Requires system software to create an interface to programs in higher-level languages.

Operating system

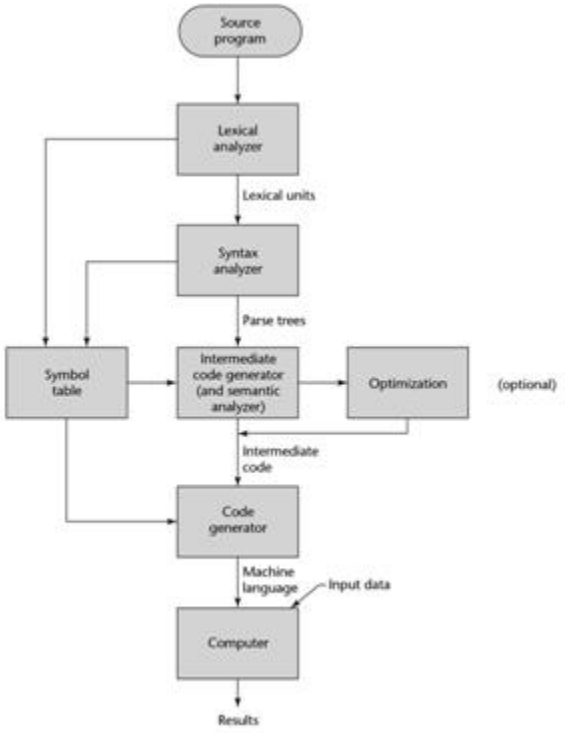
- Provides higher-level primitives than those provided by the machine language.
  - System resource management.
  - Input and output operations.
  - A file management system.
  - Text and/or program editors.
  - A variety of other commonly needed functions.
- Language implementation systems generally interface with the operating system rather than directly to the processor in machine language.



Compilation

Compiler implementation

- Programs in the source language are translated into machine language which can be directly executed.
- Slow translation – Fast execution.
- C, COBOL, and Ada.



Compilation process

- Lexical analysis
  - Groups characters from the source program into lexical units or tokens.
- Syntax analysis or Parsing
  - Constructs a hierarchical structure called a parse tree using the lexical units.
  - Parse tree represents the syntactic structure of the program – parse trees are generally not actually constructed, instead this information is used directly.
- Intermediate code generation
  - Produces a program in a different language at an intermediate level between the source program and the machine language program.
  - Similar to assembly language.

- Semantic analysis
  - Integral part of intermediate code generation.
  - Checks for errors that are difficult to detect during syntax analysis – type checking.
- Optimization
  - Often and optional part of compilation.
  - Improves programs by making them smaller or faster or both – intermediate code version.
- Code generator
  - Translates the optimized intermediate code program into machine language coder program.
- Symbol table
  - Serves as a database for the compilation process.
  - Stores identifiers and their attributes.
  - Added to during lexical analysis and syntax analysis.
  - Read from during semantic analysis and code generation.

Linking and loading

- The process of collecting system and user programs together – uses a system program called the linker.
- Before a user program can be executed it must be linked with other required operating system programs – called the load module or executable image.
- The linker is also responsible for linking users programs found in libraries.

Fetch-execute cycle

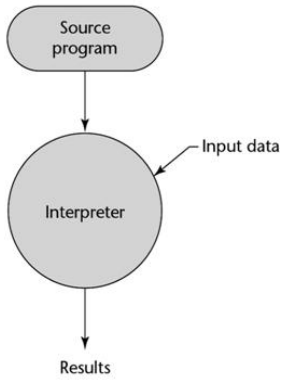
- The process of machine code program execution – von Neumann architecture.
- Program counter – stores the address of the next instruction to be executed.

```
initialize the program counter
repeat forever
        fetch the instruction pointed to by the program counter
        increment the program counter to point at the next instruction
        decode the instruction
        execute the instruction
end repeat
```

The speed of the computer is determined by the connection speed between internal memory and the processor. This connection is called the von Neumann bottleneck.

Pure Interpretation

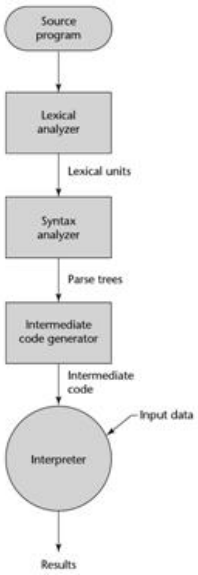
- Programs are interpreted by a program called an interpreter – no translation.
- Interpreter acts as a software simulation of a machine that deals with high-level language program statements rather than machine instructions – provides a virtual machine for the language.
- Allows easy implementation of many source-level debugging operations – better error messages.
- Execution is 10 to 100 times slower than compiled systems.
    - Caused by decoding high-level language statements instead of machine language instructions
    - Statements must be decoded every time regardless of the number of times executed.
- Statement decoding is the bottleneck.
- Often requires more space – source program and symbol table must be present.
- 1960s – APL, SNOBOL, LISP.
- Recently – JavaScript, PHP.



Hybrid Implementation Systems

Some languages are a compromise between compilers and pure interpreters.

- Translate high-level language programs to an intermediate language designed for easy interpretation.
- Faster than pure interpretation because the source language statements are decoded once.



- Perl is implemented with a hybrid system.
- Initial implementations of Java were all hybrid – byte code intermediate form – interpreter – combined referred to as Java Virtual Machine.
- Just-in-Time – initially translates programs to an intermediate language.
    - Compiles intermediate language methods into machine code when called during execution – machine code kept for subsequent calls.
    - Java, .NET languages

Preprocessors

- Program that processes a program immediately before it is compiled.
- Preprocessor instructions are embedded in programs.
- Essentially a macro expander.
- Commonly used to specify that code from another file is to be included.

The following statement copies the contents of the listed file to the position of the #include.

#include myLib.c

The following statement is used to define symbols that represent expressions.

#define max(A, B) ((A) > (B)) ? (A) : (B))

x = max(2 * y, z / 1.73) → x = ((2 * y) > (z / 1.73)) ? (2 * y) : (z / 1.73)

Programming Environments

Collection of tools used in the development of software – file system, text editor, linker, compiler.

- UNIX
  - First distributed in the middle 1970s.
  - Built around a portable multiprogramming operating system.
  - Provides powerful support tools for software production and maintenance of a variety of languages.
  - In the past it lacked a uniform interface among its tools.
  - Now commonly used through a GUI that runs of top of UNIX – Solaris CDE, GNOME, KDE.
- Borland JBuilder
  - Provides an integrated compiler, editor, debugger, and file system – all tools are accessed through a graphical interface.
  - Complex and powerful system for creating Java software.
- Microsoft Visual Studio.NET
  - Includes a large collection of software development tools through a windowed interface.
  - Used to develop Visual BASIC, C++, C#, J#, JScript (Microsoft's JavaScript) software.