Language Design

"What's one and one and one and one and one and one and one and one and one and one?"
"I don't know," said Alice. "I lost count."
"She can't do Addition," the Red Queen interrupted. -- Lewis Carroll

- Language evaluation  1.3
  - readability
  - writability
  - reliability
  - cost
- Language implementation  1.7, 1.8
  - language processing
  - compilation/interpretation
  - execution
  - environment

Criteria for Language Evaluation

Why are there so many languages?

What makes a language "better" than any other language?

- Readability
  - maintenance a major consideration in software development
  - maintenance almost always means reading somebody else's code
- Writability
  - a language should invite good programming practice
- Reliability
  - programs should perform as expected under all circumstances
- Cost
  - learning curve, coding, execution, maintenance, etc.

Criteria for Language Evaluation: Readability

- Abstraction
  - procedural abstraction, data abstraction
- Absence of ambiguity
  - and of too much choice
- Orthogonality
  - unrestricted combinations of concepts
- Expressivity of control and data structures
  - tradeoff between power and simplicity
- Appearance
  - intuitive keywords
  - formatting
  - style of comments

31

Criteria for Language Evaluation: Writability

- Abstraction
  - easier to write at a high level, maybe more difficult to learn
- Expressivity
  - what is built-in and what must be explicitly constructed
- Orthogonality
  - freedom to combine concepts in novel ways
- Modularity
  - making more code more manageable
- Support
  - help resources, code resources, language community
- Development tools
  - development environments, visual programming, libraries, debuggers

32

Criteria for Language Evaluation: Reliability

- Type checking
  - automatic checking that the right types are used in operations
- Error and exception handling
  - automatic?
  - graceful?
- Unambiguous naming
  - same name, different scope
  - pointers
  - variable structures
- Compilers and runtime environments
  - tested and bugfree

33

Criteria for Language Evaluation: Cost

- Learning curve
  - programmers may not know the language yet
- Development time
  - ease of programming
  - availability of shared code, reuse
- Efficiency of the language processor
  - compiler, interpreter
  - runtime environment
- Portability and translatability
  - easy to move code to a new platform?
  - easy to translate to a different language
- Maintainability
  - Y2K

34

Language Processing

- *Machine language* is the set of native operations performed by computer hardware.
  - machine language is the *only* set of operations that can be executed by the computer
  - `mov`, `add`, `in`, `rol`, `and`, etc.
- A *programming language* includes a set of more complex operations suitable for problem solving by programmers.
  - computers can be programmed using machine language, but in general, programming languages are the *only* way for humans to program the computer
  - `read`, `write`, `while`, `DrawMenuBar`, `PostMessage`, etc.

35

Language Processing (cont.)

In order to execute programs written in some programming language, the programming language operations must be translated into corresponding machine language instructions

- A *language processor* understands and can execute programs in a programming language.
  - by translating a programming language into machine instructions
  - *translation* is the process of mapping a *source language* into a *target language*
  - often, translation from a high-level programming language to machine language is done in several steps
    - the programming language is translated by a language processor to an intermediate language
    - a second language processor understands and can execute programs in the intermediate language

36

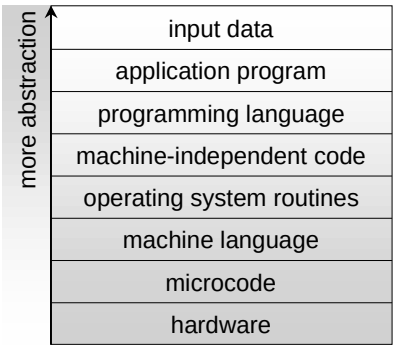
Language Processing (cont.)

- A *virtual machine* is a software simulation of a language processor.
  - a high-level programmer is usually not concerned with any details below the level of the high-level programming language
  - as far as a Java programmer is concerned, the target machine is a *Java machine* (*i.e.* a machine that understand Java directly)
- It is desirable for high-level languages to be independent of any given hardware platform
  - instead of building a separate Prolog virtual machine for every brand of computer, build one that translates to some intermediate language
  - instead of building a separate intermediate virtual machine for all languages, have the different languages use the same intermediate machine

37

Language Processing (cont.)

Often there are several *layers* of virtual machines sitting above the physical hardware



** Not every layer is always present*

38

Compilation and Interpretation

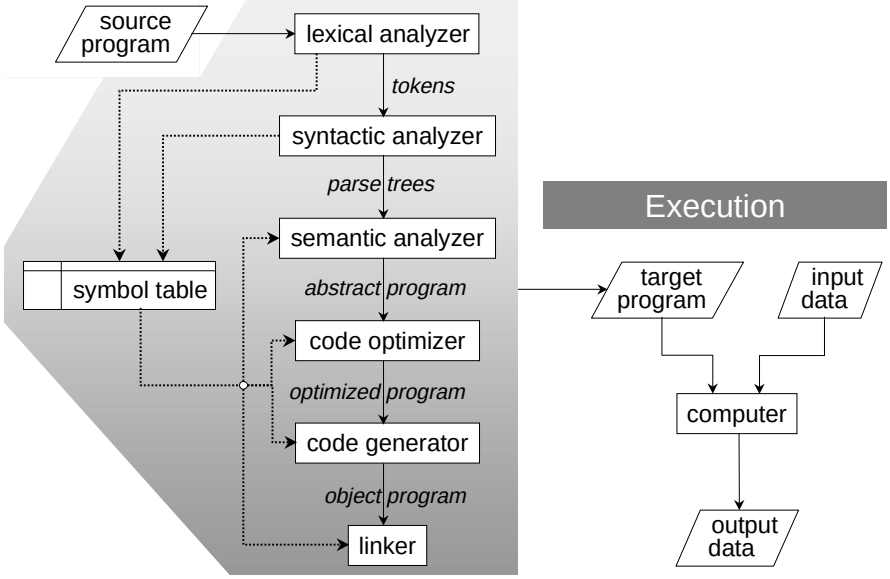
Instructions written in the language of one level must be translated into the language of the level below in order to be understood.

- *Compilation*
  - translate all of the instructions in a program into the language of a lower level virtual machine language
  - execute the translated program later
- *Interpretation*
  - translate some small number of instructions to the target language, then execute immediately, then translate some more.

Often a program is *compiled* into a lower level language and stored; the lower level program is then interpreted.

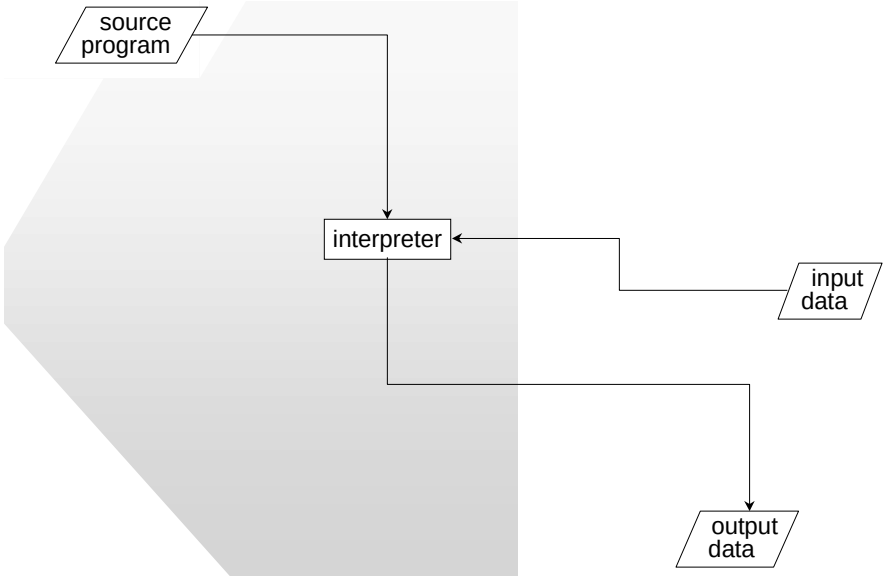
39

Compilation



40

Interpretation (+ Execution)



41

What is Intermediate Code?

Pascal

```
if I < N + 1 then
   K := I
```

Intermediate Code

```
    reg1 = I
    reg2 = N
    reg3 = reg2 + 1
    reg4 = reg1 - reg3
    if reg4 >= 0 goto L1
    reg5 = I
    K = reg5
L1:…
```

Optimized Intermediate Code

```
    reg1 = I
    reg2 = N
    reg2 = reg2 + 1
    reg2 = reg2 - reg1
    if reg2 <= 0 goto L1
    K = reg1
L1:…
```

Object Code

```
    mov AX, I
    mov BX, N
    inc BX
    sub BX, AX
    js  L1
    mov K, AX
L1:…
```

42

Programming Environments

The *programming environment* is the collection of tools available to a programmer using a particular language:

- editor
- compiler
- linker
- runtime environment
- debugger
- libraries, library manager
- help files
- graphical development tools
- command line interpreter

43

Integrated (and not-so-Integrated) Environments

Different languages (and different implementations of the same language) integrate more or fewer of these tools into a single package.

- "Disintegrated"
  - assembly
    - no editor, no runtime environment, no tools, no interpreter, no debugger (though sometimes the OS has one), libraries rare
    - online documentation: ya right!
  - Microsoft C for DOS
    - no editor (use your fav), no runtime environment, no graphical development tools, no interpreter
    - compiler, linker, debugger: separate DOS programs
    - libraries included as separate object files, managed using linker
    - ASCII text help files, no browser (or standard DOS browser)
    - graphics schmaphics!

44

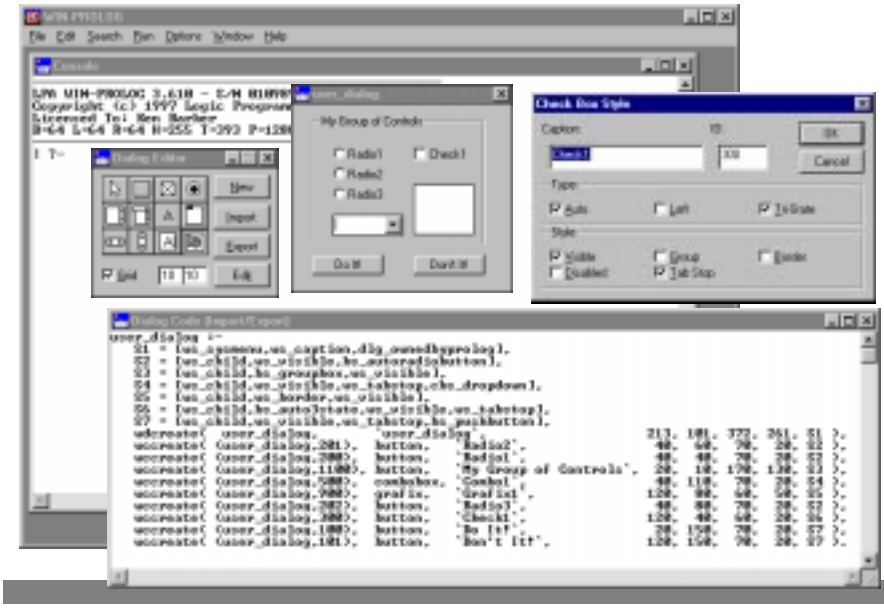
Integrated Environments (cont.)

- Towards integration
  - Unix
    - C compiler combines compiler and linker
    - Emacs editor facilitates C programming
  - Turbo C for DOS
    - shipped with proprietary editor
    - compiler, linker: separate programs that could be invoked from within the editor separately or together
    - debugger integrated with editor
    - help files online in editor along with help browser

Integrated Environments (cont.)

- Full integration
  - Smalltalk
    - editor, language processor, runtime environment, class browser all integrated
    - windows? we invented windows!
- Visual programming
  - Delphi, JDK, MS Visual C++, WinProlog, etc.
    - full integration of editor, language processor, class browser, etc.
    - plus code generators for drag'N'drop GUI development

Integrated Environments (cont.)



47

Environments for Interpreters

Interpreters (as opposed to compilers) often have their own special environment features (in addition to the tools already discussed):

- command line interpretation of single commands
- commands to load programs from disk
- runtime intervention
- built-in “compiler”
  - ▲ usually saves the entire language processor (interpreter) to an executable file that contains the user program as internal data
  - ▲ huge executable programs
  - ▲ not usually much more efficient than using the full interpreter
  - ▲ better for distribution

48