

Programming Evolution

We start with program as a sequence of bits in machine language.

```
For x86 instruction set
(as base 16):
55 89 e5 53 83 ec 04 83
00 00 39 c3 74 10 8d b6
75 f6 89 1c 24 e8 6e 00
```

And then, program in assembly languages

- One-to-one correspondence between mnemonic abbreviations and machine instructions
- Less error-prone for larger programs
- Assemblers translate to machine languages.
- Different assembly languages for different machines

```
x86 assembly:
pushl %ebp
movl %esp, %ebp
pushl %ebx
subl $4, %esp
```

And finally, program in machine independent languages, e.g. Fortran, Lisp, Prolog, Pascal, C, C++

- Closer to natural language and mathematical formulae
- More complicated translation to assembly or machine languages as one-to-one correspondence with machine language no longer exists.

| | | | | | |
|-------------|---------|------------|-------------|------------|--------------|
| A+ | C | Fertile | Lava | Pascal | Scheme |
| ABC | C++ | Forth | LIFE | Perl | Self |
| Ada | C# | Fortran | Limbo | Phantom | SETL |
| Agora | Charity | FPL | LISP | PHP | Simula |
| ALF | CHILL | GNU E | LOGO | Pike | Smalltalk |
| ALGOL | CICS | Guile | Lua | PiXCL | SNOBOL |
| Alice | Clean | Godel | Matlab | PL/B | TADS |
| Amiga E | Clu | Haskell | MCPL | PL/I | Tcl |
| AMPL | COBOL | Hugo | Mercury | Pliant | Tom |
| APL | COMAL | ICI | Miranda | Postscript | UberCode |
| AppleScript | cT | Icon | ML | Prolog | UNITY |
| AspectJ | dBase | Inform | Modula 1-3 | Python | V |
| AWK | Delphi | J | NeoBook | Q | Visual Basic |
| B | Dog | Java | NESL | QuakeC | WebQL |
| BASIC | E | JavaScript | NetRexx | R | Wolfram |
| BCPL | Egg | Juice | Oberon | REBOL | XQuery |
| BETA | Eiffel | K | OO Turing | Rexx | XSLT |
| BLISS | Elastic | Kojo | Objective-C | RPG | Yorick |
| Blockly | Erlang | KRYPTON | Occam | Ruby | ZPL |

and many more...

https://en.wikipedia.org/wiki/List_of_programming_languages

Why So Many?

Evolution We constantly find better way to do things.

- goto-based control flow (e.g. Fortran, COBOL, Basic)
- Structured programming: while loops, case (switch) etc. (e.g. Algol, Pascal, Ada)
- Object-oriented programming (e.g. Smalltalk, C++, Eiffel, Java, C#)

Special purposes

- Lisp - Manipulating list of complex data structure
- C - Low level system programming
- Prolog - Reasoning about logical relationships among data

Personal preference No universally accepted language

Why a few are widely used? (1)

Expressive power Language features have an impact on clear concise and maintainable code.

Ease of use for novice e.g. from Pascal to Java and Python

Ease of implementation

- Implemented for machines with little resource, e.g. Basic
- Implementations are free, e.g. Pascal, Java, Python.

Standardization

- Standard language implementation and libraries ensure portability of code across platforms.
- Different vendors do not need to implement in different ways.

Why a few are widely used? (2)

Open source

- Open-source compiler and interpreter
- Association with popular open source OS (C and Unix/Linux)

Excellent compilers

- Compilers generate very fast code, e.g. Fortran.
- Supporting tools help manage large projects.

Economics, patronage, and inertia

- Backing of powerful sponsor (COBOL, Ada and US Department of Defense; C# and Microsoft; Objective-C and Apple)
- Cost of replacement in terms of software and expertise (COBOL and financial infrastructure)

Classification of Programming Languages (1)

Imperative Specify sequence of steps to execute, i.e. how the computer should do it

- *Von Neumann* Most familiar and widely used, e.g. Fortran, Pascal, Basic, C
 - *Basic means of computation is modification of variables, changing the value of memory.*
- *Scripting* Rapid application development, e.g. csh, JavaScript, PHP, Perl, Python, Ruby
 - *Based on gluing together components that were developed as independent programs*
 - *Written for a special run-time environment that automates the execution of tasks that could alternatively be executed one-by-one by a human operator*
 - *Also refer to dynamic general-purpose languages with small programs “script”*
- *Object-oriented*, e.g. Smalltalk, Eiffel, C++, Java
 - *Based on interactions among semi-independent objects, each of which has both its own internal state and subroutines to manage that state.*

```
//C
int gcd(int a, int b) {
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}
```

Classification of Programming Languages (2)

Declarative Not specify a sequence of steps to execute, but rather the properties of a solution to be found, i.e. what the computer is to do

- *Functional*, e.g. Lisp, Scheme, ML, Haskell
 - *Based on recursive definition of functions.*
 - *Program is a function from inputs to outputs, defined in terms of simpler functions.*

```
(* Ocaml *)
let rec gcd a b =
  if a = b then a
  else if a > b then gcd b (a - b)
       else gcd a (b - a)
```

- *Logic*, e.g. Prolog
 - *Based on an attempt to find values that satisfy certain relationships, using goal-directed search through a list of logical rules.*

```
%Prolog- Search for G that makes gcd(A,B,G) true using the rules
gcd(A,B,G) :- A = B, G = A.
gcd(A,B,G) :- A > B, C is A-B, gcd(C,B,G) .
gcd(A,B,G) :- B > A, C is B-A, gcd(C,A,G) .
```

Why study programming languages?

To choose the most appropriate language for any given task

To make it easier to learn new languages

- Many languages are closely related, e.g. Java, C# and C++
- Basic concepts underlie all programming languages, e.g. types, control flow

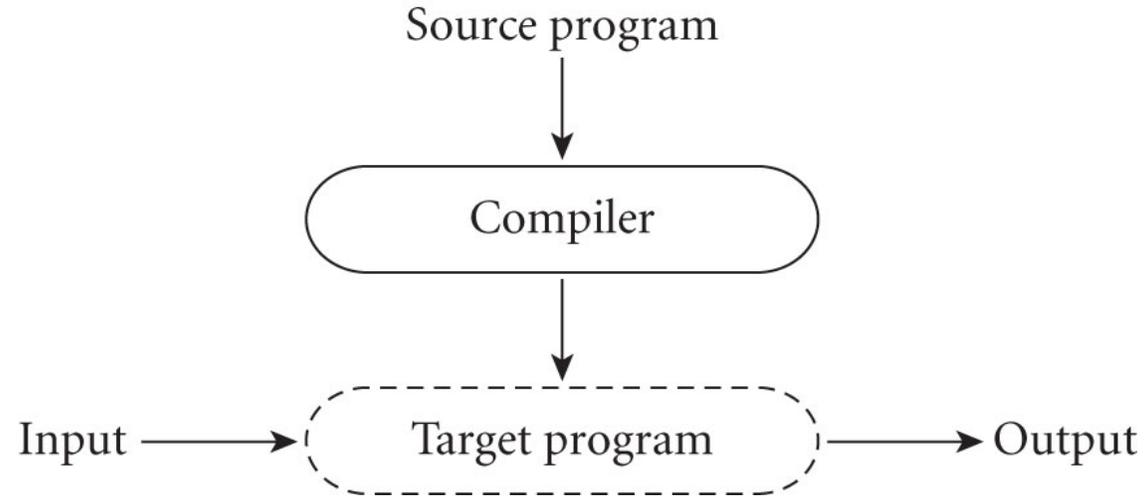
To choose among alternative ways to express things based on knowledge of implementation costs, e.g. nested if vs. switch (case)

To make good use of debuggers, assemblers, linkers, and related tools

- Bugs or system-building problems are easier to handle if implementation detail of language is known.

To understand interactions of languages with operating systems and architectures

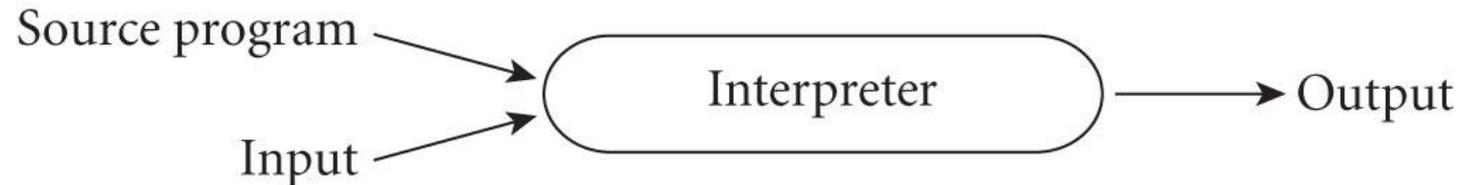
Program Translation: Compiler



Better performance because

- It generates machine code once and that can be executed many times.
- Decision made at compile time does not need to be made at run time.
 - *Compile time decision (e.g. x at location 49378) is put into machine code (e.g. referring to x in source program will access location 49378).*

Program Translation: Interpreter



It reads statements one at a time, line by line, and executes them as it goes along.

- e.g. when + operation in statement is recognized, interpreter's add function is called which then executes machine code ADD instruction.

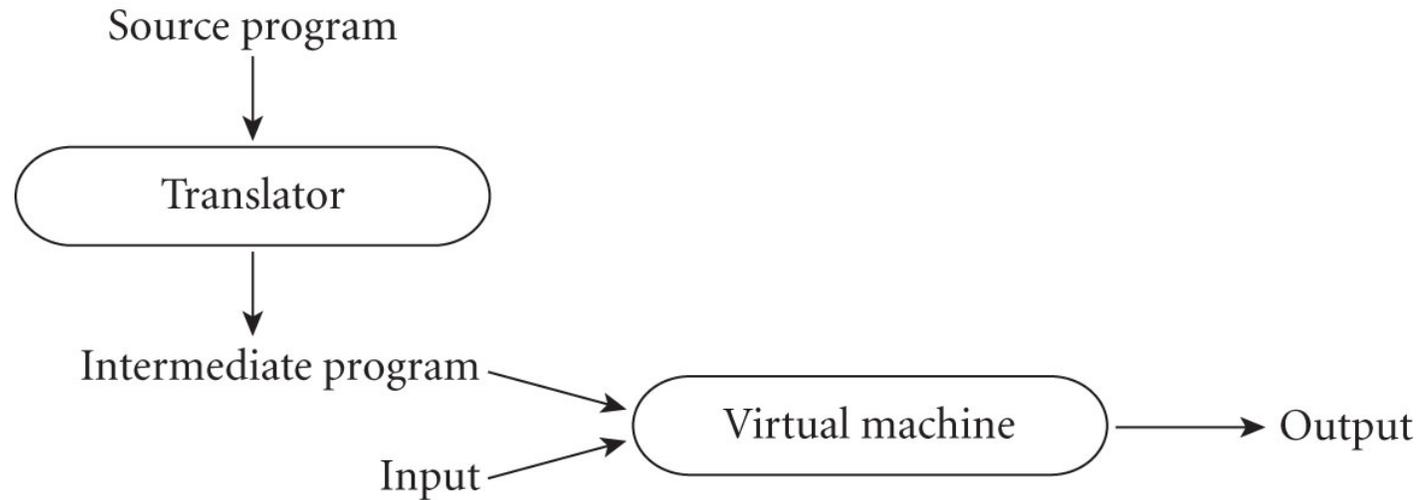
Better error messages and more flexible

- Source-level debugger since it executes source code
- Can determine characteristics, e.g. type, size of variables, at run time depending on input data.

Slower

- e.g. table lookup for location of variable x is needed every time it is accessed.

Combination of Compiler and Interpreter



Found in most language implementations

Default in Java

- Byte code is machine-independent intermediate format for distribution.
- Source program is thoroughly compiled and kept.

Just-in-Time Translation

Found in modern language implementations for better performance.

Intermediate program is translated into machine language immediately before each execution of the program.

- Frequently used portion, rarely in full

