My Information

- Lecturer: Trần Vĩnh Tân
- Email: tan@cse.hcmut.edu.vn
- Website: http://www.cse.hcmut.edu.vn/~tan
- Office hour: Tuesday 09:00 – 11:00 (subject to change)
- Sakai: http://elearning.cse.hcmut.edu.vn

References

- Concepts of Programming Languages, 8/e, Robert W. Sebesta, Addison Wesley, 2008.
- "Programming Languages – Principles and Practices" – Kenneth C. Louden, Thomson Brooks/ Cole, 2003.
- "Ngôn ngữ lập trình – Các nguyên lý và mô hình" – Cao Hoàng Trụ, 2004.

Assessment

- Tutorials/Labs/On-class Exercises: 10%
- Assignments: 30%, using Scala (http://www.scala-ide.org/)
- Midterm: 20%
- Final: 40%
- **Notice: Fail (zero mark) five tuts/labs will be banned from final exam**

Introduction

Principles of Programming Languages

Outline

- Reasons for Studying Concepts of Programming Languages
- Programming Domains
- Language Evaluation Criteria
- Influences on Language Design
- Language Categories
- Language Design Trade-Offs
- Implementation Methods
- Programming Environments
- Historical Languages

Benefits of Studying

- Increased capacity to express idea
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of the significance of implementation
- Better use of languages that are already known
- Overall advancement of computing

Genealogy of Common Languages

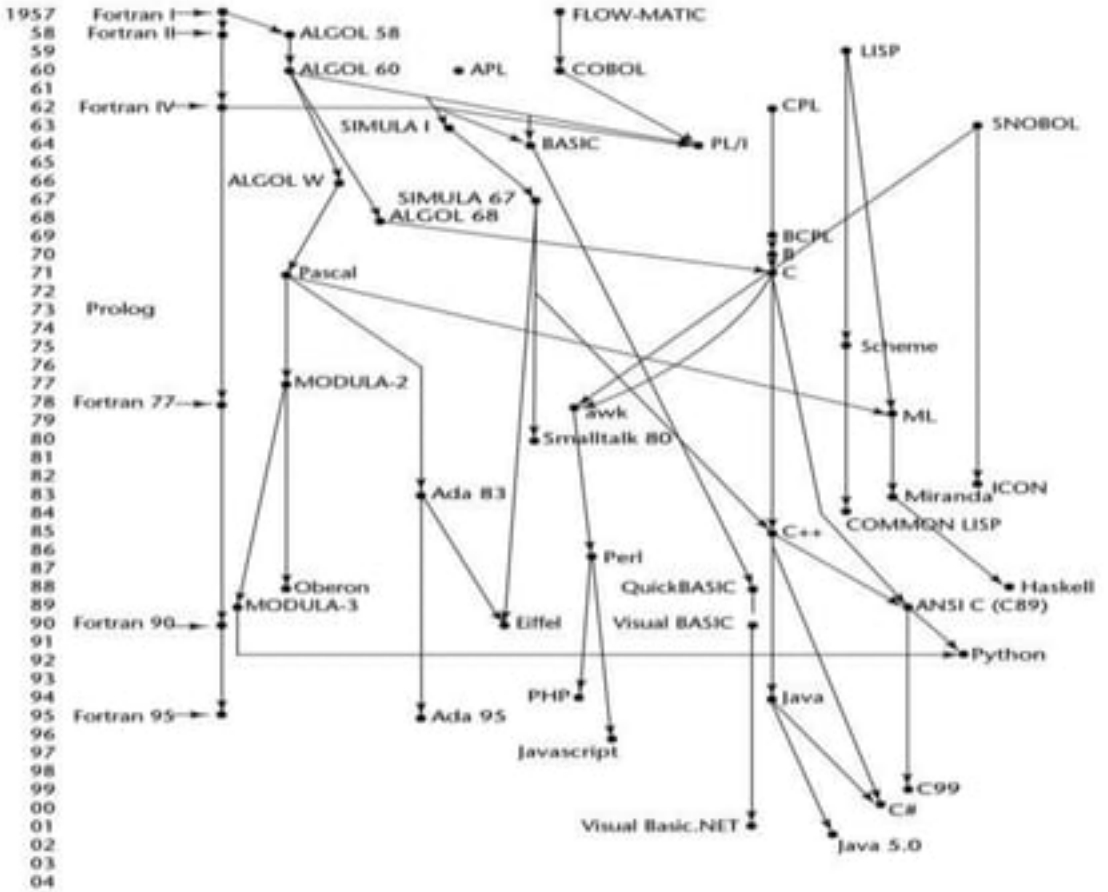


Figure 2.1 Genealogy of common high-level programming languages

Programming Domains

- Scientific Applications
  - Fortran, ALGOL 60
- Business Applications
  - COBOL
- Artificial Intelligence
  - LISP, Prolog, also C
- Systems Programming
  - PL/S, BLISS, Extended ALGOL, and C
- Web Software
  - XHTML; JavaScript, PHP

Language Characteristics

- Simplicity
- Orthogonality
- Control structures
- Data types and structures
- Syntax design
- Support of abstraction
- Expressivity
- Type checking
- Exception handling
- Restricted aliasing
- ....

Language Evaluation

- Readability
- Writability
- Reliability
- Cost

Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Few feature multiplicity (means of doing the same operation)
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal

Orthogonality: Example

IBM Mainframe Assembly

Non-orthogonality

```
A  Reg1, memory_cell
AR Reg1, Reg2
```

VAX superminicomputer

Orthogonality

```
ADDL operand_1, operand_2
```

Evaluation Criteria: Readability

- Control statements
  - The presence of well-known control structures (e.g., `while` statement)
- Data types and structures
  - The presence of adequate facilities for defining data structures
- Syntax considerations
  - Identifier forms: flexible composition
  - Special words and methods of forming compound statements
  - Form and meaning: self-descriptive constructs, meaningful keywords

Control Statements

```
loop1:
   if (incr <= 20) go to out;
loop2:
   if (sum > 10) go to next;
   sum += incr;
   go to loop2;
next:
   incr++;
   go to loop1;
out:
```

Evaluation Criteria: Readability

- Control statements
  - The presence of well-known control structures (e.g., while statement)
- Data types and structures
  - The presence of adequate facilities for defining data structures
- Syntax considerations
  - Identifier forms: flexible composition
  - Special words and methods of forming compound statements
  - Form and meaning: self-descriptive constructs, meaningful keywords

Evaluation Criteria: Writability

- Simplicity and orthogonality
  - Few constructs, a small number of primitives, a small set of rules for combining them
- Support for abstraction
  - The ability to define and use complex structures or operations in ways that allow details to be ignored
- Expressivity
  - A set of relatively convenient ways of specifying operations
  - Example: the inclusion of `for` statement in many modern languages

Evaluation Criteria: Reliability

- Type checking
  - Testing for type errors
- Exception handling
  - Intercept run-time errors and take corrective measures
- Aliasing
  - Presence of two or more distinct referencing methods for the same memory location
- Readability and writability
  - A language that does not support "natural" ways of expressing an algorithm will necessarily use "unnatural" approaches, and hence reduced reliability

Evaluation Criteria: Cost

- Training programmers to use language
- Writing programs (closeness to particular applications)
- Compiling programs
- Executing programs
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

Evaluation Criteria: Others

- Portability
  - The ease with which programs can be moved from one implementation to another
- Generality
  - The applicability to a wide range of applications
- Well-definedness
  - The completeness and precision of the language's official definition

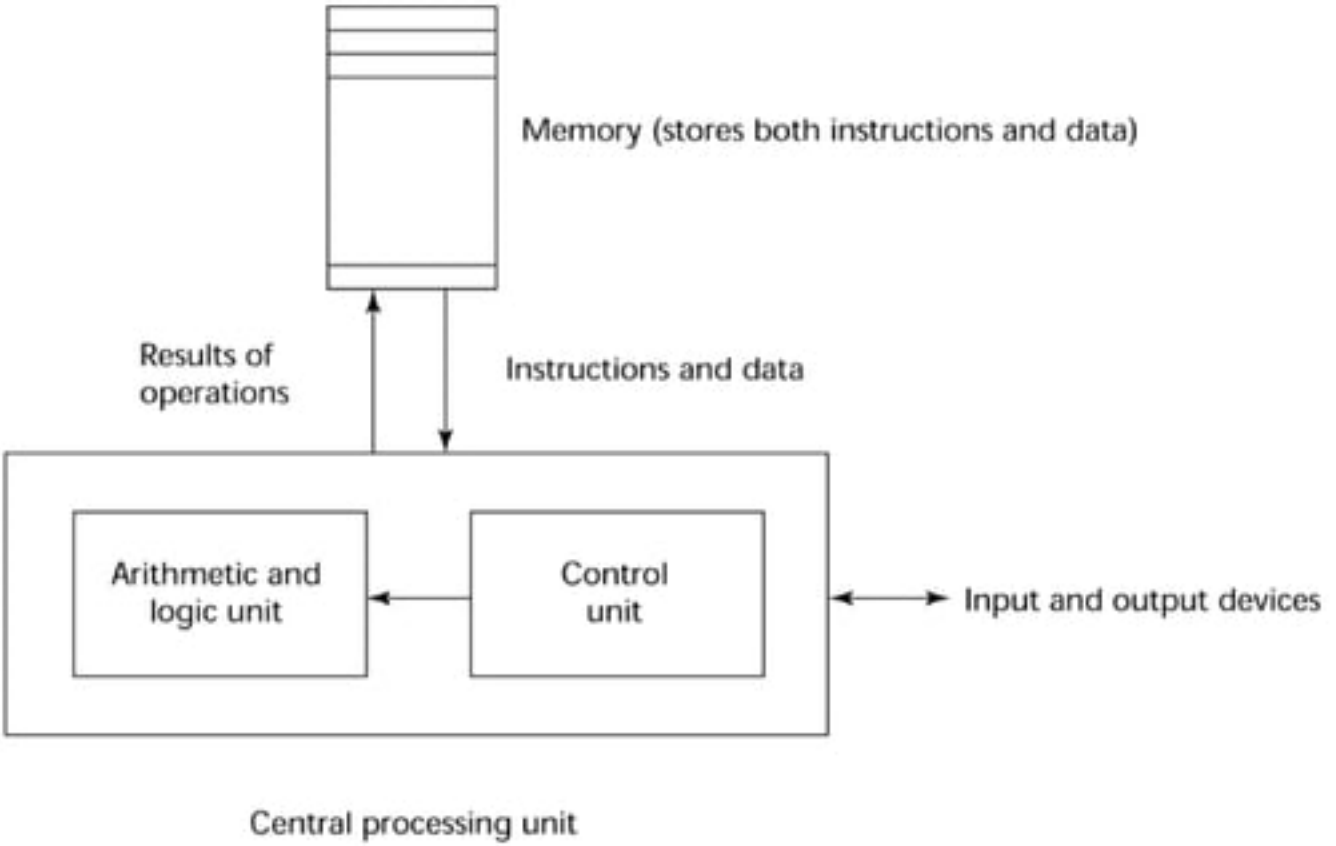
Influences on Language Design

- Computer Architecture
- Programming Methodologies

Computer Architecture Influence

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
  - Data and programs stored in memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
  - Basis for imperative languages
    - Variables model memory cells
    - Assignment statements model piping
    - Iteration is efficient

The von Neumann Architecture



Programming Methodologies Influences

- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: People efficiency became important; readability, better control structures
  - structured programming
  - top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
  - data abstraction
- Middle 1980s: Object-oriented programming
  - Data abstraction + inheritance + polymorphism

Language Categories

- Imperative
  - Central features are variables, assignment statements, and iteration
  - Examples: C, Pascal
- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme
- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog
- Object-oriented
  - Data abstraction, inheritance, late binding
  - Examples: Java, C++
- Markup
  - New; not a programming per se, but used to specify the layout of information in Web documents
  - Examples: XHTML, XML

Language Design Trade-Offs

- Reliability vs. cost of execution
  - Conflicting criteria
  - Example: Java demands all references to array elements be checked for proper indexing but that leads to increased execution costs
- Readability vs. writability
  - Another conflicting criteria
  - Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
- Writability (flexibility) vs. reliability
  - Another conflicting criteria
  - Example: C++ pointers are powerful and very flexible but not reliably used

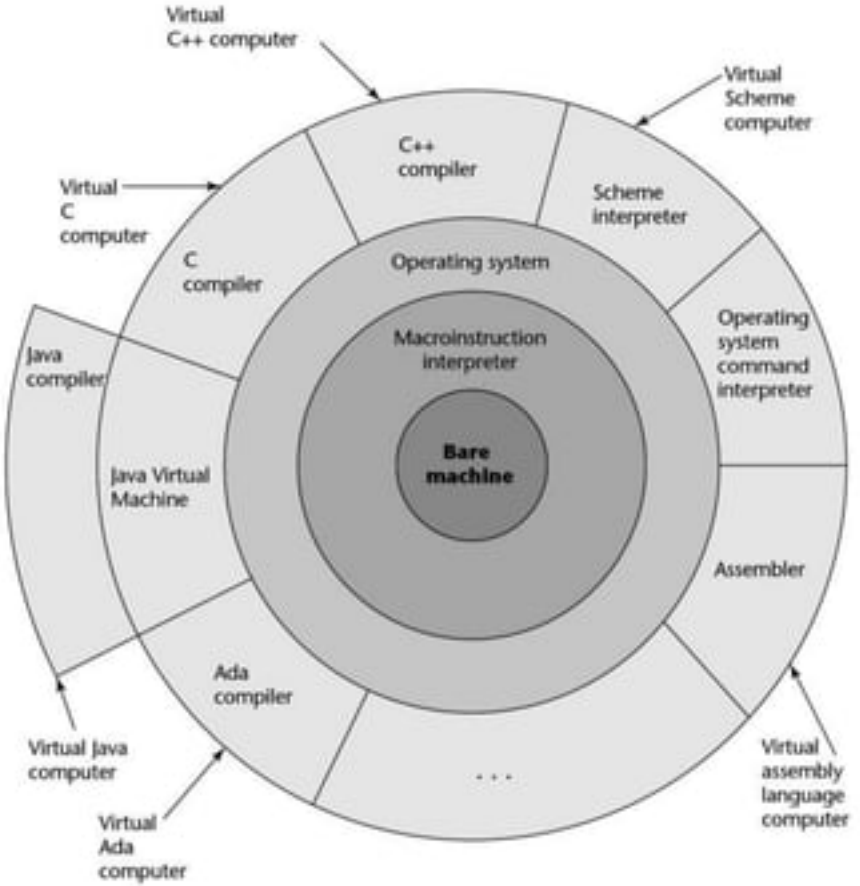
Implementation Methods

- Compilation
  - Programs are translated into machine language
- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters

Layered View of Computer

Figure 1.2

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

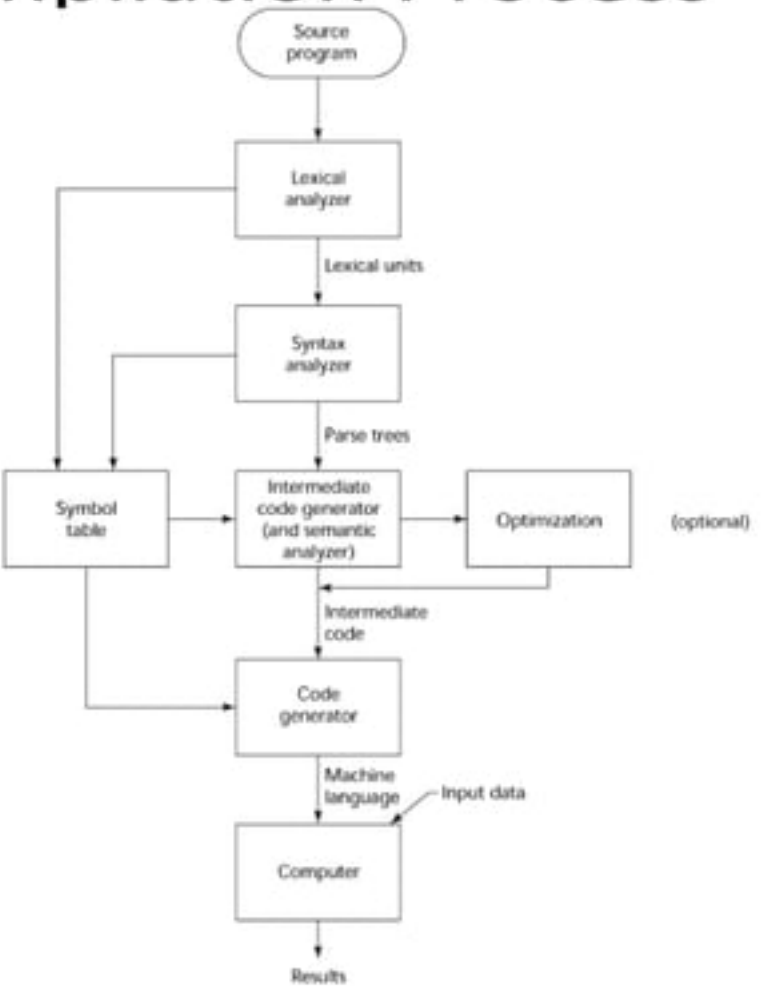


The operating system and language implementation are layered over Machine interface of a computer

Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution
- Compilation process has several phases:
  - lexical analysis: converts characters in the source program into lexical units
  - syntax analysis: transforms lexical units into *parse trees* which represent the syntactic structure of program
  - Semantics analysis: generate intermediate code
  - code generation: machine code is generated

The Compilation Process



Execution of Machine Code

- Fetch-execute-cycle (on a von Neumann architecture)

```
initialize the program counter
repeat forever
  fetch the instruction pointed by the counter
  increment the counter
  decode the instruction
  execute the instruction
end repeat
```

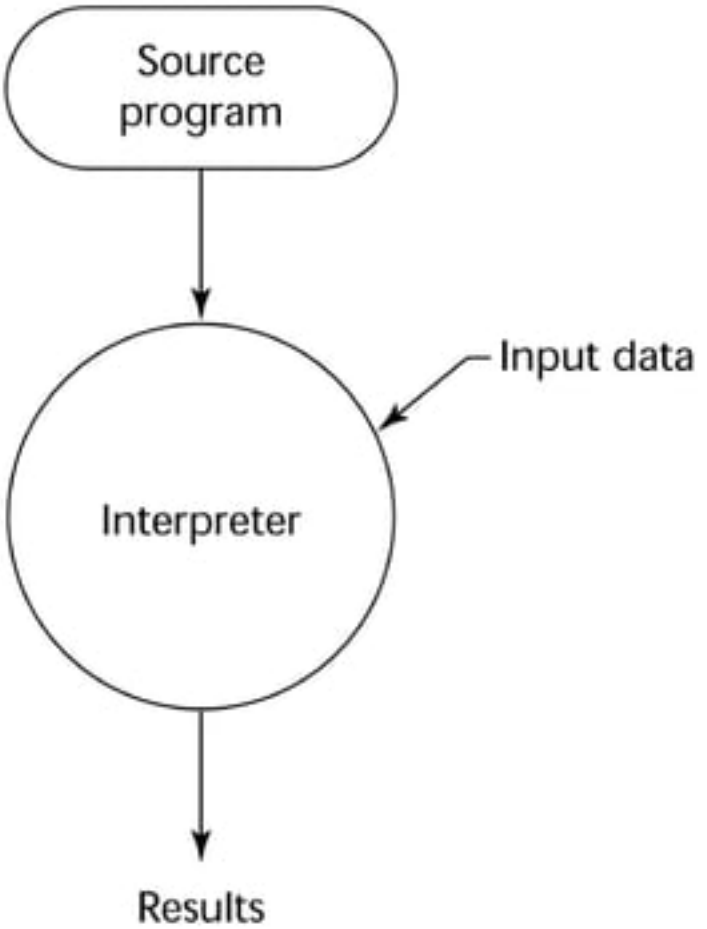
Von Neumann Bottleneck

- Connection speed between a computer's memory and its processor determines the speed of a computer
- Program instructions often can be executed a lot faster than the above connection speed; the connection speed thus results in a *bottleneck*
- Known as von Neumann bottleneck; it is the primary limiting factor in the speed of computers

Pure Interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Becoming rare on high-level languages
- Significant comeback with some Web scripting languages (e.g., JavaScript)

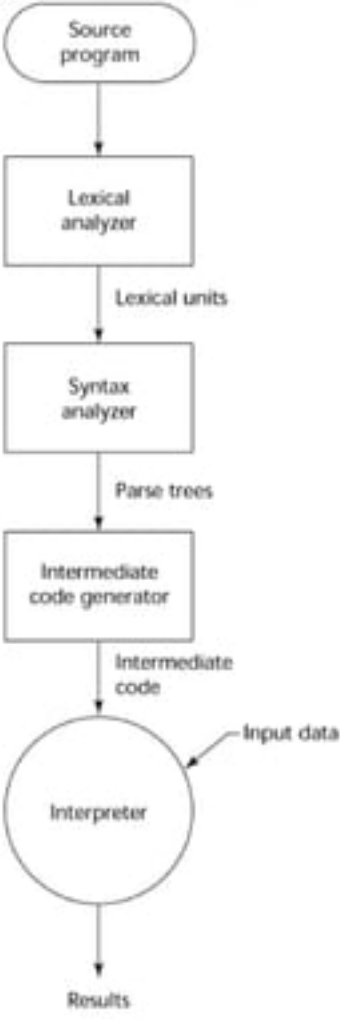
Pure Interpretation Process



Hybrid Implementation Systems

- A compromise between compilers and pure interpreters
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Faster than pure interpretation
- Examples
  - Perl programs are partially compiled to detect errors before interpretation
  - Initial implementations of Java were hybrid; the intermediate form, *byte code*, provides portability to any machine that has a byte code interpreter and a run-time system (together, these are called *Java Virtual Machine*)

Hybrid Implementation Process



Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile intermediate language into machine code
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system

Preprocessors

- Preprocessor macros (instructions) are commonly used to specify that code from another file is to be included
- A preprocessor processes a program immediately before the program is compiled to expand embedded preprocessor macros
- A well-known example: C preprocessor
  - expands `#include`, `#define`, and similar macros

Programming Environments

- The collection of tools used in software development
- UNIX
  - An older operating system and tool collection
  - Nowadays often used through a GUI (e.g., CDE, KDE, or GNOME) that run on top of UNIX
- Eclipse, NetBeans, JBuilder
  - An integrated development environment for Java
- Microsoft Visual Studio.NET
  - A large, complex visual environment
  - Used to program in C#, Visual BASIC.NET, Jscript, J#, or C++

FORTRAN

- John Backus and others at IBM, 1957
- Replacement for assembly language
- Programming community was sceptical that high-level language could perform adequately
- Cut program size by a factor of 20
- Still used widely for numerical programming

FORTRAN

```
C  99 BOTTLES OF BEER ON THE WALL
   INTEGER BOTTLS
   DO 50 I = 1, 99
   BOTTLS = 100 - I
   PRINT 10, BOTTLS
10 FORMAT(1X, I2, 31H BOTTLE(S) OF BEER ON THE WALL.)
   PRINT 20, BOTTLS
20 FORMAT(1X, I2, 19H BOTTLE(S) OF BEER.)
   PRINT 30
30 FORMAT(34H TAKE ONE DOWN AND PASS IT AROUND,)
   BOTTLS = BOTTLS - 1
   PRINT 10, BOTTLS
   PRINT 40
40 FORMAT(1X)
50 CONTINUE
   STOP
   END
```

FORTRAN Features

- ALL CAPS
- ≤ 6 character variable names (BOTTLS)
- Punch card layout. First six columns for comment marker (C), numeric label
- Loop statement DO 50 I = 1, 99 references label 50 CONTINUE
- PRINT 30 references format 30 FORMAT(...)

LISP

- McCarthy, 1958, at MIT
- Focus on list processing, not numbers
- Used extensively for artificial intelligence (AI)
- Very simple language: lists, functions, recursion
- Grandfather of functional languages ML, OCaml, Miranda, Haskell, ...

```
(define (down n k)
  (if (< n k) '()
    (cons n (down (- n 1) k)))) (

define (plural? n . up)
  (let ((.. string-append)
        (num (number->string n))
        (bot " bottle")
        (ltr (if (null? up) "n" "N")))
    (case n ((0) (.. ltr "o more" bot "s"))
            ((1) (.. num bot))
            (else (.. num bot "s"))

(define (verse n)
  (let ((.. string-append)
        (top (plural? n 1))
        (mid (plural? n))
        (nxt (plural? (if (= n 0)
                          99
                          (- n 1))))
        (beer " of beer")
        (wall " on the wall")
        (actn (if (= n 0)
                  "Go to the store and buy some more, "
                  "Take one down and pass it around, ")))
    `(,(.. top beer wall ", " mid beer ".")
      . ,(.. actn nxt beer wall "."))))
(define (sing verse)
  (let ((n newline))
    (display (car verse)) (n)
    (display (cdr verse)) (n) (n)))

(for-each sing (map verse (iota 99 0)))
```

LISP Features

- Parentheses notation (`fun arg1 arg2 ...`)
- Lists are assembled with `cons` and taken apart with `car`, `cdr`
- Functions can be parameters to other functions (`sing` and `verse` in the last line)
- Blocks (`let ((var1 init1) (var2 init2) ...) ...`)
- Names can contain punctuation marks: `plural?`, ..

ALGOL 60

- Bauer, Naur, Dijkstra, et al. at various European universities
- Clean and formally defined grammar
- Functions, recursion, but not a full functional language
- No I/O (considered an system-dependent implementation detail)

```
'begin'

'comment'
  99 Bottles of Beer on the Wall
;

  'integer' 'procedure' bottles(n);
  'value' n;
  'integer' n;
  'begin'
    'if' n < 1 'then' outstring(1, "no more ") 'else' outinteger(1, n);
    'if' n = 1 'then' outstring(1, "bottle") 'else' outstring(1,
   "bottles");
    outstring(1, " of beer");
  'end';

  'integer' i;

  'for' i := 99 'step' -1 'until' 1 'do' 'begin'
      bottles(i); outstring(1, " on the wall, ");
      bottles(i); outstring(1, "\n");
      outstring(1, "take one down and pass it around, ");
      bottles(i - 1); outstring(1, " on the wall.\n");
  'end';

'end'
```

ALGOL 60 Features

- Assignment := (= means equality, as it did for 400 years in math)
- Keywords enclosed in quotes
- Block structure `'begin' ... 'end'`
- C. A. R. Hoare: "Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors."

C

- Kernighan and Ritchie, AT&T Bell Labs, 1969-73
- Derived from a language called B
- Co-evolved with Unix operating system
- Small language (reaction to the complex PL/1)
- Pointers, registers, inline assembly

C

```
#define MAXBEER (99)

void chug(register int beers) {
    char howmany[8], *s;
    s = beers != 1 ? "s" : "";
    printf("%d bottle%s of beer on the wall,\n", beers, s);
    printf("%d bottle%s of beeeeer . . . ,\n", beers, s);
    printf("Take one down, pass it around,\n");

    if(--beers) sprintf(howmany, "%d", beers); else strcpy(howmany,
   "No more");
    s = beers != 1 ? "s" : "";
    printf("%s bottle%s of beer on the wall.\n", howmany, s);
}

main() {
    int beers;
    for(beers = MAXBEER; beers; chug(beers--))
      puts("");
    puts("\nTime to buy more beer!\n");
}
```

C Features

- Blocks with braces `{ ... }`
- Pointers
- Expressions with side effects: `if (--beers)`
- Expressions in `for (expr1; expr2; expr3)` need not be related
- Rich library (for the time): `printf, strcpy`
- No automatic resource management

C++

- Bjarne Stroustrup, AT&T Bell Labs, 1983
- C with Classes
- Brought object-oriented programming into the mainstream
- Complex feature set: operator overloading, templates, multiple inheritance, virtual base classes, copy constructors/destructors
- New version in progress (C++ 0x), with better support for libraries and multithreading

```
#include <iostream>

template<int I>
class Loop {
public:
  static inline void f() {
    cout << I << " bottles of beer on the wall," << endl
         << I << " bottles of beer." << endl
         << "Take one down, pass it around," << endl
         << I-1 << " bottles of beer on the wall." << endl;
    Loop<I-1>::f();
  }
};

class Loop<0> {
public:
  static inline void f() {
    cout << "Go to the store and buy some more," << endl
         << "99 bottles of beer on the wall." << endl;
  }
};

main() {
  Loop<3>::f();
}
```

C++

- Superset of C (almost)
- Overloaded operators (`cout << ...`)
- Classes
- Very powerful templates
- Very few people understand the whole language

Java

- Gosling, Sun Microsystems, 1995
- Simpler than C++
- Really brought OO into the mainstream
- Garbage collection
- Security model for remote execution
- Very rich library (networking, UI, ...)
- Mobile and Enterprise editions

```
class Verse {
    private final int count;

    Verse(int verse) {
        count = 100-verse;
    }

    public String toString() {
        String c =
            "{0,choice,0#no more bottles|1#1 bottle|1<{0} bottles} of
   beer";
        return java.text.MessageFormat.format(
            c.replace("n","N")+" on the wall, "+c+".\n"+
            "{0,choice,0#Go to the store and buy some more"+
            "|0<Take one down and pass it around},
   "+c.replace("{0","{1")+
            " on the wall.\n", count, (count+99)%100);
    }

    public static void main(String[] args) {
        for (int i = 1; i <= 100; i++)
            System.out.println(new Verse(i));
    }
}
```

Java Features

- C style blocks and control structures
- Classes
- Objects: `new Verse(i)`
- Huge class library: `java.text.MessageFormat`

Summary

- The study of programming languages is valuable for a number of reasons:
  - Increase our capacity to use different constructs
  - Enable us to choose languages more intelligently
  - Makes learning new languages easier
- Most important criteria for evaluating programming languages include:
  - Readability, writability, reliability, cost
- Major influences on language design have been machine architecture and software development methodologies
- The major methods of implementing programming languages are: compilation, pure interpretation, and hybrid implementation