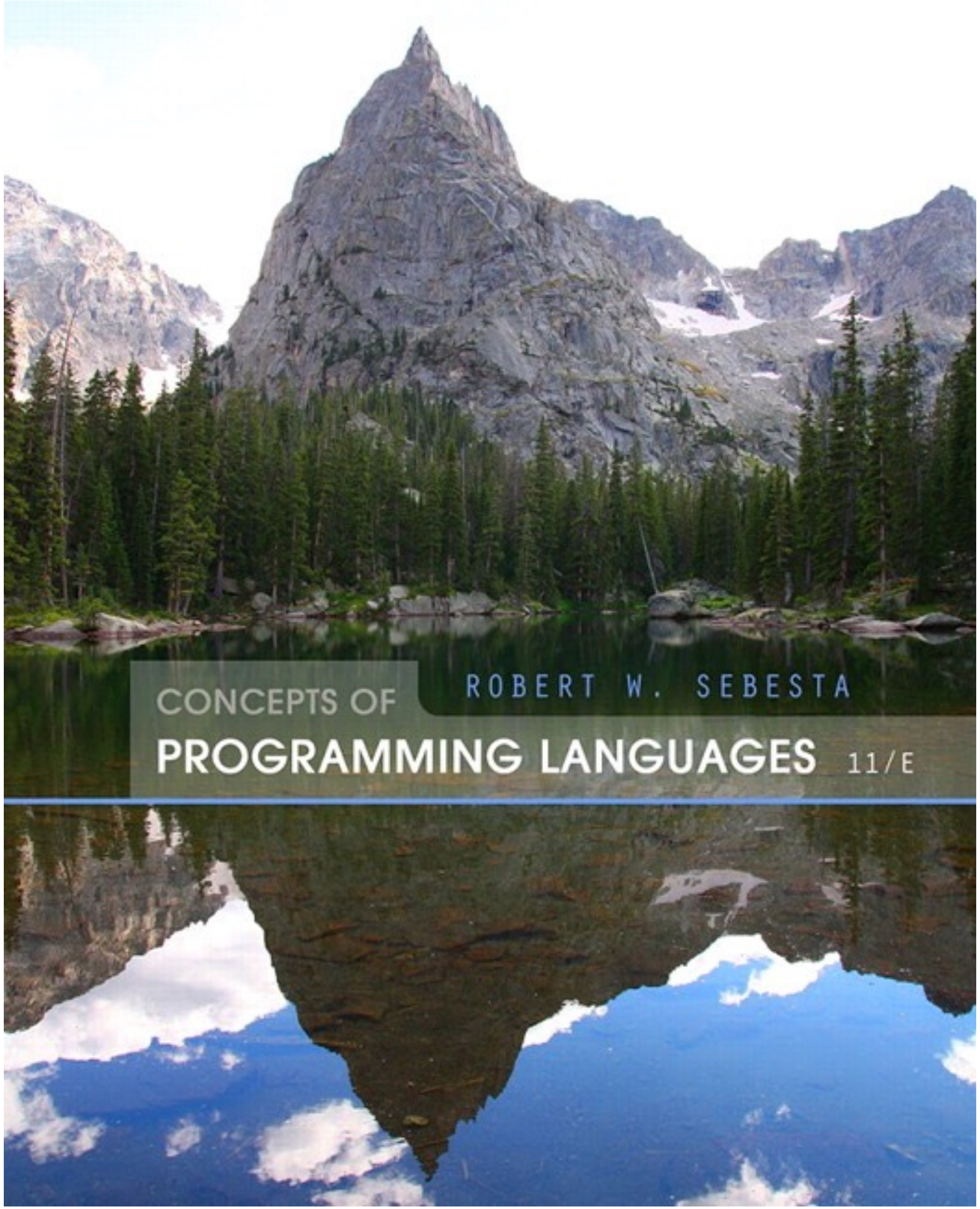
# Chapter 2

## Evolution of the Major Programming Languages




ISBN 0-321-49362-1

# Chapter 2 Topics

- Zuse's Plankalkül
- Minimal Hardware Programming: Pseudocodes
- The IBM 704 and Fortran
- Functional Programming: Lisp
- The First Step Toward Sophistication: ALGOL 60
- Computerizing Business Records: COBOL
- The Beginnings of Timesharing: Basic

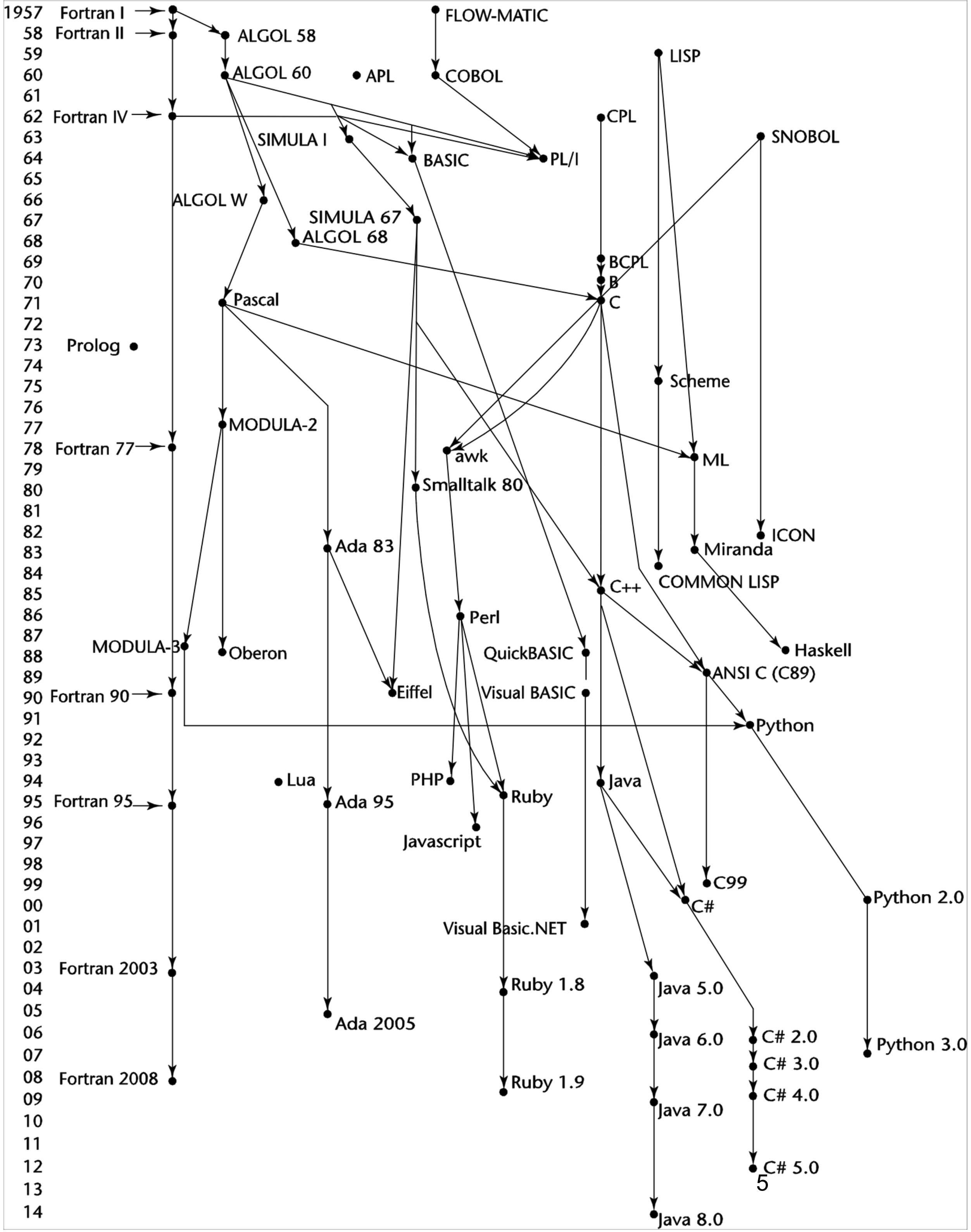
# Chapter 2 Topics (continued)

- Everything for Everybody: PL/I
- Two Early Dynamic Languages: APL and SNOBOL
- The Beginnings of Data Abstraction: SIMULA 67
- Orthogonal Design: ALGOL 68
- Some Early Descendants of the ALGOLs
- Programming Based on Logic: Prolog
- History's Largest Design Effort: Ada

# Chapter 2 Topics (continued)

- Object-Oriented Programming: Smalltalk
- Combining Imperative ad Object-Oriented Features: C++
- An Imperative-Based Object-Oriented Language: Java
- Scripting Languages
- The Flagship .NET Language: C#
- Markup/Programming Hybrid Languages

# Genealogy of Common Languages



# Zuse's Plankalkül

- Designed in 1945, but not published until 1972
- Never implemented
- Advanced data structures
  - floating point, arrays, records
- Invariants

# Plankalkül Syntax

- An assignment statement to assign the expression A[4] + 1 to A[5]

```
            |    A + 1 => A
      V     |   4              5
(subscripts)
      S     |   1.n            1.n
(data types)
```

# Minimal Hardware Programming: Pseudocodes

- What was wrong with using machine code?
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies--no indexing or floating point

# Pseudocodes: Short Code

- Short Code developed by Mauchly in 1949 for BINAC computers
  - Expressions were coded, left to right
  - Example of operations:

```
    01 -    06 abs value    1n (n+2)nd power
02 )    07 +            2n (n+2)nd root
03 =    08 pause        4n if <= n
04 /    09 (            58 print and tab
```

# Pseudocodes: Speedcoding

- Speedcoding developed by Backus in 1954 for IBM 701
  - Pseudo ops for arithmetic and math functions
  - Conditional and unconditional branching
  - Auto-increment registers for array access
  - Slow!
  - Only 700 words left for user program

# Pseudocodes: Related Systems

- The UNIVAC Compiling System
  - Developed by a team led by Grace Hopper
  - Pseudocode expanded into machine code
- David J. Wheeler (Cambridge University)
  - developed a method of using blocks of re-locatable addresses to solve the problem of absolute addressing

# IBM 704 and Fortran

- Fortran 0: 1954 - not implemented
- Fortran I:1957
  - Designed for the new IBM 704, which had index registers and floating point hardware
    - This led to the idea of compiled programming languages, because there was no place to hide the cost of interpretation (no floating-point software)
  - Environment of development
    - Computers were small and unreliable
    - Applications were scientific
    - No programming methodology or tools
    - Machine efficiency was the most important concern

# Design Process of Fortran

- Impact of environment on design of Fortran I
  - No need for dynamic storage
  - Need good array handling and counting loops
  - No string handling, decimal arithmetic, or powerful input/output (for business software)

# Fortran I Overview

- First implemented version of Fortran
  - Names could have up to six characters
  - Post-test counting loop (**`DO`**)
  - Formatted I/O
  - User-defined subprograms
  - Three-way selection statement (arithmetic **`IF`**)
  - No data typing statements

# Fortran I Overview (continued)

- First implemented version of FORTRAN
  - No separate compilation
  - Compiler released in April 1957, after 18 worker-years of effort
  - Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of 704
  - Code was very fast
  - Quickly became widely used

# Fortran II

- Distributed in 1958
  - Independent compilation
  - Fixed the bugs

# Fortran IV

- Evolved during 1960-62
  - Explicit type declarations
  - Logical selection statement
  - Subprogram names could be parameters
  - ANSI standard in 1966

# Fortran 77

- Became the new standard in 1978
  - Character string handling
  - Logical loop control statement
  - **`IF-THEN-ELSE`** statement

# Fortran 90

- Most significant changes from Fortran 77
  - Modules
  - Dynamic arrays
  - Pointers
  - Recursion
  - **`CASE`** statement
  - Parameter type checking

# Latest versions of Fortran

- Fortran 95 – relatively minor additions, plus some deletions
- Fortran 2003 – support for OOP, procedure pointers, interoperability with C
- Fortran 2008 – blocks for local scopes, co-arrays, `Do Concurrent`

# Fortran Evaluation

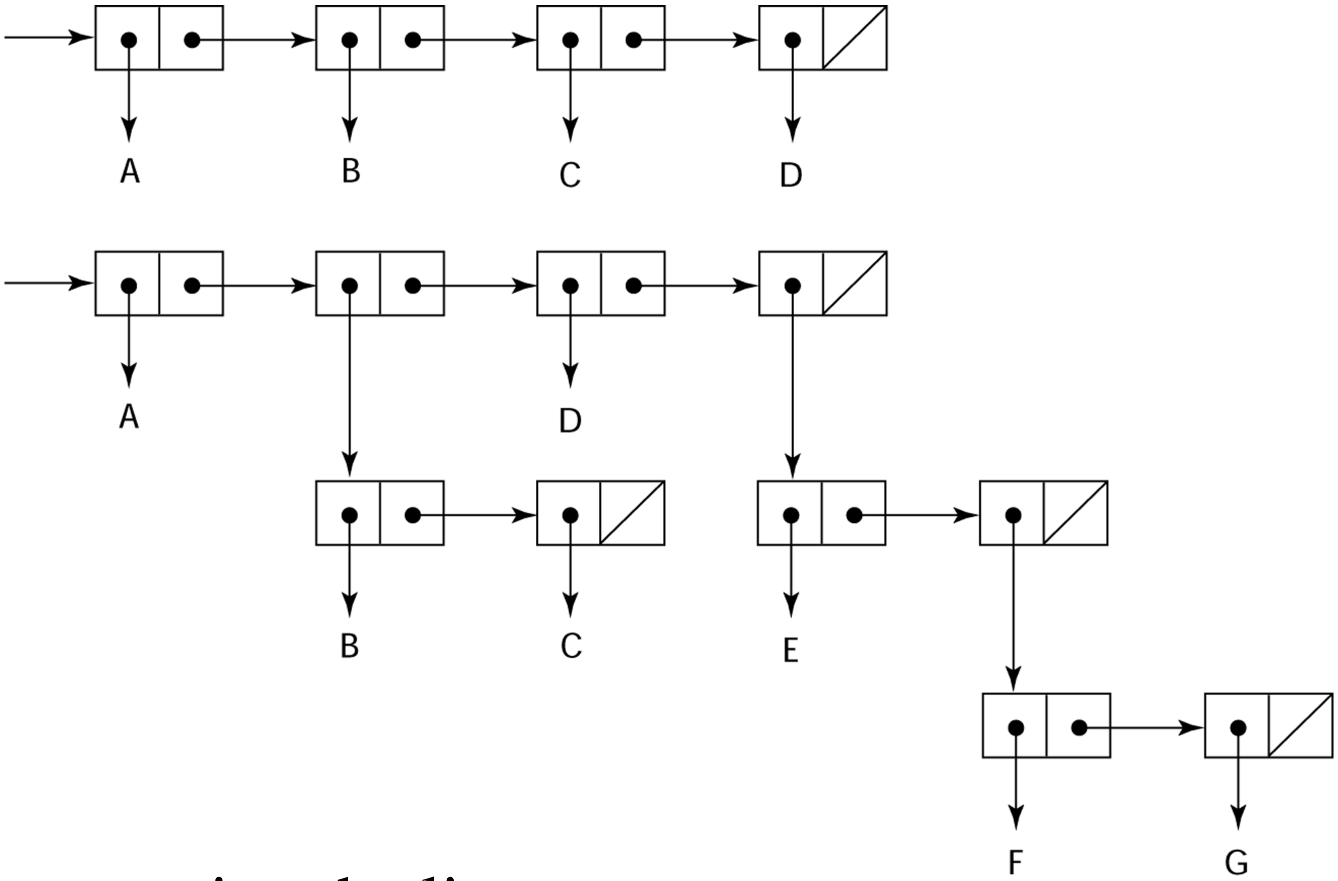
- Highly optimising compilers (all versions before 90)
  - Types and storage of all variables are fixed before run time
- Dramatically changed forever the way computers are used

```
! Fortran 95 Example program
!  Input:  An integer, List_Len, where List_Len is less
!          than 100, followed by List_Len-Integer values
!  Output: The number of input values that are greater
!          than the average of all input values
  Implicit none
  Integer Dimension(99) :: Int_List
  Integer :: List_Len, Counter, Sum, Average, Result
  Result= 0
  Sum = 0
  Read *, List_Len
  If ((List_Len > 0) .AND. (List_Len < 100)) Then
    ! Read input data into an array and compute its sum
      Do Counter = 1, List_Len
        Read *, Int_List(Counter)
        Sum = Sum + Int_List(Counter)
      End Do
    ! Compute the average
      Average = Sum / List_Len
    ! Count the values that are greater than the average
      Do Counter = 1, List_Len
        If (Int_List(Counter) > Average) Then
          Result = Result + 1
        End If
      End Do
    ! Print the result
    Print *, 'Number of values > Average is:', Result
  Else
     Print *, 'Error - list length value is not legal'
  End If
End Program Example
```

# Functional Programming: Lisp

- LISt Processing language
  - Designed at MIT by McCarthy
- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)
- Only two data types: atoms and lists
- Syntax is based on *lambda calculus*

# Representation of Two Lisp Lists




Representing the lists **(A B C D)**
and **(A (B C) D (E (F G)))**

# Lisp Evaluation

- Pioneered functional programming
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Still the dominant language for AI
- Common Lisp and Scheme are contemporary dialects of Lisp
- ML, Haskell, and F# are also functional programming languages, but use very different syntax

```
;  LISP Example function
;  The following code defines a LISP predicate function
;  that  takes two lists as arguments and returns True
;  if the two lists are equal, and NIL (false) otherwise
   (DEFUN equal_lists (lis1 lis2)
    (COND
      ((ATOM lis1) (EQ lis1 lis2))
      ((ATOM lis2) NIL)
      ((equal_lists (CAR lis1) (CAR lis2))
     (T NIL) )
)
```

# Scheme

- Developed at MIT in mid 1970s
- Small
- Extensive use of static scoping
- Functions as first-class entities
- Simple syntax (and small size) make it ideal for educational applications

# Common Lisp

- An effort to combine features of several dialects of Lisp into a single language
- Large, complex, used in industry for some large applications

# The First Step Toward Sophistication: ALGOL 60

- Environment of development
  - FORTRAN had (barely) arrived for IBM 70x
  - Many other languages were being developed, all for specific machines
  - No portable language; all were machine-dependent
  - No universal language for communicating algorithms
- ALGOL 60 was the result of efforts to design a universal language

# Early Design Process

- ACM and GAMM met for four days for design (May 27 to June 1, 1958)

- Goals of the language
  - Close to mathematical notation
  - Good for describing algorithms
  - Must be translatable to machine code

# ALGOL 58

- Concept of type was formalised
- Names could be any length
- Arrays could have any number of subscripts
- Parameters were separated by mode (in & out)
- Subscripts were placed in brackets
- Compound statements (**`begin ... end`**)
- Semicolon as a statement separator
- Assignment operator was :=
- **`if`** had an **`else-if`** clause
- No I/O - “would make it machine dependent”

# ALGOL 58 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid 1959

# ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - Block structure (local scope)
  - Two parameter passing methods
  - Subprogram recursion
  - Stack-dynamic arrays

  - Still no I/O and no string handling

# ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative languages are based on it
  - First machine-independent language
  - First language whose syntax was formally defined (BNF)

# ALGOL 60 Evaluation (continued)

- Failure
  - Never widely used, especially in U.S.
  - Reasons
    - Lack of I/O and the character set made programs non-portable
    - Too flexible--hard to implement
    - Entrenchment of Fortran
    - Formal syntax description
    - Lack of support from IBM

```
comment ALGOL 60 Example Program
Input: An integer, listlen, where listlen is less than 100, followed by listlen-integer values
Output: The number of input values that are greater than the average of all the input values ;

begin

   integer array intlist [1:99];
   integer listlen, counter, sum, average, result; sum := 0;
   result := 0;
   readint (listlen);

   if (listlen > 0) ∧ (listlen < 100) then begin

      comment Read input into an array and compute the average;

      for counter := 1 step 1 until listlen do begin
          readint (intlist[counter]); sum := sum + intlist[counter]
      end;

      comment Compute the average;
      average := sum / listlen;

      comment Count the input values that are > average;

      for counter := 1 step 1 until listlen do

         if intlist[counter] > average then result := result + 1;

      comment Print result;
      printstring("The number of values > average is:");
      printint (result)

   end

   else

     printstring ("Error—input list length is not legal";
end
```

# Computerizing Business Records: COBOL

- Environment of development
  - UNIVAC was beginning to use FLOW-MATIC
  - USAF was beginning to use AIMACO
  - IBM was developing COMTRAN

# COBOL Historical Background

- Based on FLOW-MATIC
- FLOW-MATIC features
  - Names up to 12 characters, with embedded hyphens
  - English names for arithmetic operators (no arithmetic expressions)
  - Data and code were completely separate
  - The first word in every statement was a verb

# COBOL Design Process

- First Design Meeting (Pentagon) - May 1959
- Design goals
  - Must look like simple English
  - Must be easy to use, even if that means it will be less powerful
  - Must broaden the base of computer users
  - Must not be biased by current compiler problems
- Design committee members were all from computer manufacturers and DoD branches
- Design Problems: arithmetic expressions? subscripts? Fights among manufacturers

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRODUCE-REORDER-LISTING.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DEC-VAX.
OBJECT-COMPUTER. DEC-VAX.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT BAL-FWD-FILE     ASSIGN TO READER
    SELECT REORDER-LISTING  ASSIGN TO LOCAL-PRINTER.
DATA DIVISION.
FILE SECTION.
FD  BAL-FWD-FILE
    LABEL RECORDS ARE STANDARD
    RECORD CONTAINS 80 CHARACTERS.
01  BAL-FWD-CARD.
    02 BAL-ITEM-NO          PICTURE IS 9(5).
    02 BAL-ITEM-DESC        PICTURE IS X(20).
    02 FILLER               PICTURE IS X(5).
    02 BAL-UNIT-PRICE       PICTURE IS 999V99.
    02 BAL-REORDER-POINT    PICTURE IS 9(5).
    02 BAL-ON-HAND          PICTURE IS 9(5).
    02 BAL-ON-ORDER         PICTURE IS 9(5).
    02 FILLER               PICTURE IS X(30).

FD  REORDER-LISTING
    LABEL RECORDS ARE STANDARD
    RECORD CONTAINS 132 CHARACTERS.
01  REORDER-LINE.
    02 RL-ITEM-NO           PICTURE IS Z(5).
    02 FILLER               PICTURE IS X(5).
    02 RL-ITEM-DESC         PICTURE IS X(20).
    02 FILLER               PICTURE IS X(5).
    02 RL-UNIT-PRICE        PICTURE IS ZZZ.99.
    02 FILLER               PICTURE IS X(5).
    02 RL-AVAILABLE-STOCK   PICTURE IS Z(5).
    02 FILLER               PICTURE IS X(5).
    02 RL-REORDER-POINT     PICTURE IS Z(5).
    02 FILLER               PICTURE IS X(71).
```

```
WORKING-STORAGE SECTION.
01  SWITCHES.
    02 CARD-EOF-SWITCH      PICTURE IS X.
01  WORK-FIELDS.
    02 AVAILABLE-STOCK      PICTURE IS 9(5).
PROCEDURE DIVISION.
000-PRODUCE-REORDER-LISTING.
    OPEN INPUT BAL-FWD-FILE.
    OPEN OUTPUT REORDER-LISTING.
    MOVE "N" TO CARD-EOF-SWITCH.
    PERFORM 100-PRODUCE-REORDER-LINE
        UNTIL CARD-EOF-SWITCH IS EQUAL TO "Y".
    CLOSE BAL-FWD-FILE.
    CLOSE REORDER-LISTING.
    STOP RUN.
100-PRODUCE-REORDER-LINE.
    PERFORM 110-READ-INVENTORY-RECORD.
    IF CARD-EOF-SWITCH IS NOT EQUAL TO "Y"]
        PERFORM 120-CALCULATE-AVAILABLE-STOCK
        IF AVAILABLE-STOCK IS LESS THAN BAL-REORDER-POINT
            PERFORM 130-PRINT-REORDER-LINE.
110-READ-INVENTORY-RECORD.
    READ BAL-FWD-FILE RECORD
        AT END
            MOVE "Y" TO CARD-EOF-SWITCH.
120-CALCULATE-AVAILABLE-STOCK.
ADD BAL-ON-HAND BAL-ON-ORDER
    GIVING AVAILABLE-STOCK.
130-PRINT-REORDER-LINE.
    MOVE SPACE              TO REORDER-LINE.
    MOVE BAL-ITEM-NO        TO RL-ITEM-NO.
    MOVE BAL-ITEM-DESC      TO RL-ITEM-DESC.
    MOVE BAL-UNIT-PRICE     TO RL-UNIT-PRICE.
    MOVE AVAILABLE-STOCK    TO RL-AVAILABLE-STOCK
    MOVE BAL-REORDER-POINT  TO RL-REORDER-POINT.
    WRITE REORDER-LINE.
```

# COBOL Evaluation

- Contributions
  - First macro facility in a high-level language
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with hyphens
  - Separate data division

# COBOL: DoD Influence

- First language required by DoD
  - would have failed without DoD
- Still the most widely used business applications language

# The Beginning of Timesharing: Basic

- Designed by Kemeny & Kurtz at Dartmouth
- Design Goals:
  - Easy to learn and use for non-science students
  - Must be "pleasant and friendly"
  - Fast turnaround for homework
  - Free and private access
  - User time is more important than computer time
- Current popular dialect: Visual Basic
- First widely used language with time sharing

```
REM  BASIC Example Program
REM  Input:  An integer, listlen, where listlen is less
REM          than 100, followed by listlen-integer values
REM  Output: The number of input values that are greater
REM          than the average of all input values
  DIM intlist(99)
  result = 0
  sum = 0
  INPUT listlen
  IF listlen > 0 AND listlen < 100 THEN
REM  Read input into an array and compute the sum
    FOR counter = 1 TO listlen
      INPUT intlist(counter)
      sum = sum + intlist(counter)
    NEXT counter
REM  Compute the average
    average = sum / listlen
REM  Count the number of input values that are > average
    FOR counter = 1 TO listlen
      IF intlist(counter) > average
        THEN result = result + 1
    NEXT counter
REM  Print the result
    PRINT "The number of values that are > average is:";
           result
  ELSE
    PRINT "Error—input list length is not legal"
END IF END
```

# 2.8 Everything for Everybody: PL/I

- Designed by IBM and SHARE
- Computing situation in 1964 (IBM's point of view)
  - Scientific computing
    - IBM 1620 and 7090 computers
    - FORTRAN
    - SHARE user group
  - Business computing
    - IBM 1401, 7080 computers
    - COBOL
    - GUIDE user group

# PL/I: Background

- By 1963
  - Scientific users began to need more elaborate I/O, like COBOL had;  business users began to need floating point and arrays for MIS
  - It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly
- The obvious solution
  - Build a new computer to do both kinds of applications
  - Design a new language to do both kinds of applications

# PL/I: Design Process

- Designed in five months by the 3 X 3 Committee
  - Three members from IBM, three members from SHARE
- Initial concept
  - An extension of Fortran IV
- Initially called NPL (New Programming Language)
- Name changed to PL/I in 1965

# PL/I: Evaluation

- PL/I contributions
  - First unit-level concurrency
  - First exception handling
  - Switch-selectable recursion
  - First pointer data type
  - First array cross sections
- Concerns
  - Many new features were poorly designed
  - Too large and too complex

```
/* PL/I PROGRAM EXAMPLE
        INPUT:  AN INTEGER, LISTLEN, WHERE LISTLEN IS LESS THAN
                100, FOLLOWED BY LISTLEN-INTEGER VALUES
        OUTPUT: THE NUMBER OF INPUT VALUES THAT ARE GREATER THAN
                THE AVERAGE OF ALL INPUT VALUES   */
      PLIEX: PROCEDURE OPTIONS (MAIN);
         DECLARE INTLIST (1:99) FIXED.
         DECLARE (LISTLEN, COUNTER, SUM, AVERAGE, RESULT) FIXED;
         SUM = 0;
         RESULT = 0;
         GET LIST (LISTLEN);
         IF (LISTLEN > 0) & (LISTLEN < 100) THEN
    DO;
         /* READ INPUT DATA INTO AN ARRAY AND COMPUTE THE SUM */
             DO COUNTER = 1 TO LISTLEN;
               GET LIST (INTLIST (COUNTER));
               SUM = SUM + INTLIST (COUNTER);
             END;
         /* COMPUTE THE AVERAGE */
             AVERAGE = SUM / LISTLEN;
         /* COUNT THE NUMBER OF VALUES THAT ARE > AVERAGE */
             DO COUNTER = 1 TO LISTLEN;
               IF INTLIST (COUNTER) > AVERAGE THEN
                 RESULT = RESULT + 1;
             END;
         /* PRINT RESULT */
             PUT SKIP LIST ('THE NUMBER OF VALUES > AVERAGE IS:');
             PUT LIST (RESULT);
             END;
           ELSE
             PUT SKIP LIST ('ERROR—INPUT LIST LENGTH IS ILLEGAL');
END PLIEX;
```

# Two Early Dynamic Languages: APL and SNOBOL

- Characterized by dynamic typing and dynamic storage allocation
- Variables are untyped
  - A variable acquires a type when it is assigned a value
- Storage is allocated to a variable when it is assigned a value

# APL: A Programming Language

- Designed as a hardware description language at IBM by Ken Iverson around 1960
  - Highly expressive (many operators, for both scalars and arrays of various dimensions)
  - Programs are very difficult to read
- Still in use; minimal changes

# SNOBOL

- Designed as a string manipulation language at Bell Labs by Farber, Griswold, and Polensky in 1964
- Powerful operators for string pattern matching
- Slower than alternative languages (and thus no longer used for writing editors)
- Still used for certain text processing tasks

# The Beginning of Data Abstraction: SIMULA 67

- Designed primarily for system simulation in Norway by Nygaard and Dahl
- Based on ALGOL 60 and SIMULA I
- Primary Contributions
  - Coroutines - a kind of subprogram
  - Classes, objects, and inheritance

# Orthogonal Design: ALGOL 68

- From the continued development of ALGOL 60 but not a superset of that language
- Source of several new ideas (even though the language itself never achieved widespread use)
- Design is based on the concept of orthogonality
  - A few basic concepts, plus a few combining mechanisms

# ALGOL 68 Evaluation

- Contributions
  - User-defined data structures
  - Reference types
  - Dynamic arrays (called flex arrays)
- Comments
  - Less usage than ALGOL 60
  - Had strong influence on subsequent languages, especially Pascal, C, and Ada

# Pascal - 1971

- Developed by Wirth (a former member of the ALGOL 68 committee)
- Designed for teaching structured programming
- Small, simple, nothing really new
- Largest impact was on teaching programming
  - From mid-1970s until the late 1990s, it was the most widely used language for teaching programming

# C - 1972

- Designed for systems programming (at Bell Labs by Dennis Richie)
- Evolved primarily from BCLP and B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX
- Though designed as a systems language, it has been used in many application areas

# Programming Based on Logic: Prolog

- Developed, by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski ( University of Edinburgh)
- Based on formal logic
- Non-procedural
- Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries
- Comparatively inefficient
- Few application areas

# History's Largest Design Effort: Ada

- Huge design effort, involving hundreds of people, much money, and about eight years
- Sequence of requirements (1975-1978)
  - (Strawman, Woodman, Tinman, Ironman, Steelman)
- Named Ada after Augusta Ada Byron, the first programmer

# Ada Evaluation

- Contributions
  - Packages - support for data abstraction
  - Exception handling - elaborate
  - Generic program units
  - Concurrency - through the tasking model
- Comments
  - Competitive design
  - Included all that was then known about software engineering and language design
  - First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed

# Ada 95

- Ada 95 (began in 1988)
  - Support for OOP through type derivation
  - Better control mechanisms for shared data
  - New concurrency features
  - More flexible libraries
- Ada 2005
  - Interfaces and synchronizing interfaces
- Popularity suffered because the DoD no longer requires its use but also because of popularity of C++

# Object-Oriented Programming: Smalltalk

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic binding)
- Pioneered the graphical user interface design
- Promoted OOP

# Combining Imperative and Object-Oriented Programming: C++

- Developed at Bell Labs by Stroustrup in 1980
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67
- A large and complex language, in part because it supports both procedural and OO programming
- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November 1997
- Microsoft's version: MC++
  - Properties, delegates, interfaces, no multiple inheritance

# A Related OOP Language

- Objective-C (designed by Brad Cox – early 1980s)
  - C plus support for OOP based on Smalltalk
  - Uses Smalltalk's method calling syntax
  - Used by Apple for systems programs

# An Imperative-Based Object-Oriented Language: Java

- Developed at Sun in the early 1990s
  - C and C++ were not satisfactory for embedded electronic devices
- Based on C++
  - Significantly simplified (does not include **`struct`**, **`union`**, **`enum`**, pointer arithmetic, and half of the assignment coercions of C++)
  - Supports *only* OOP
  - Has references, but not pointers
  - Includes support for applets and a form of concurrency

# Java Evaluation

- Eliminated many unsafe features of C++
- Supports concurrency
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Widely used for Web programming
- Use increased faster than any previous language
- Most recent version, 8, released in 2014

# Scripting Languages for the Web

- Perl
  - Designed by Larry Wall—first released in 1987
  - Variables are statically typed but implicitly declared
  - Three distinctive namespaces, denoted by the first character of a variable's name
  - Powerful, but somewhat dangerous
  - Gained widespread use for CGI programming on the Web
  - Also used for a replacement for UNIX system administration language
- JavaScript
  - Began at Netscape, but later became a joint venture of Netscape and Sun Microsystems
  - A client-side HTML-embedded scripting language, often used to create dynamic HTML documents
  - Purely interpreted
  - Related to Java only through similar syntax
- PHP
  - PHP: Hypertext Preprocessor, designed by Rasmus Lerdorf
  - A server-side HTML-embedded scripting language, often used for form processing and database access through the Web
  - Purely interpreted

# Scripting Languages for the Web

- Python
  - An OO interpreted scripting language
  - Type checked but dynamically typed
  - Used for CGI programming and form processing
  - Dynamically typed, but type checked
  - Supports lists, tuples, and hashes
- Ruby
  - Designed in Japan by Yukihiro Matsumoto (a.k.a, “Matz”)
  - Began as a replacement for Perl and Python
  - A pure object-oriented scripting language
    - All data are objects
  - Most operators are implemented as methods, which can be redefined by user code
  - Purely interpreted

# Scripting Languages for the Web

- Lua
  - An OO interpreted scripting language
  - Type checked but dynamically typed
  - Used for CGI programming and form processing
  - Dynamically typed, but type checked
  - Supports lists, tuples, and hashes, all with its single data structure, the table
  - Easily extendable

# The Flagship .NET Language: C#

- Part of the .NET development platform (2000)
- Based on C++ , Java, and Delphi
- Includes pointers, delegates, properties, enumeration types, a limited kind of dynamic typing, and anonymous types
- Is evolving rapidly

# Markup/Programming Hybrid Languages

- XSLT
  - eXtensible Markup Language (XML): a metamarkup language
  - eXtensible Stylesheet Language Transformation (XSTL) transforms XML documents for display
  - Programming constructs (e.g., looping)
- JSP
  - Java Server Pages: a collection of technologies to support dynamic Web documents
  - JSTL, a JSP library, includes programming constructs in the form of HTML elements

# Summary

- Development, development environment, and evaluation of a number of important programming languages
- Perspective into current issues in language design