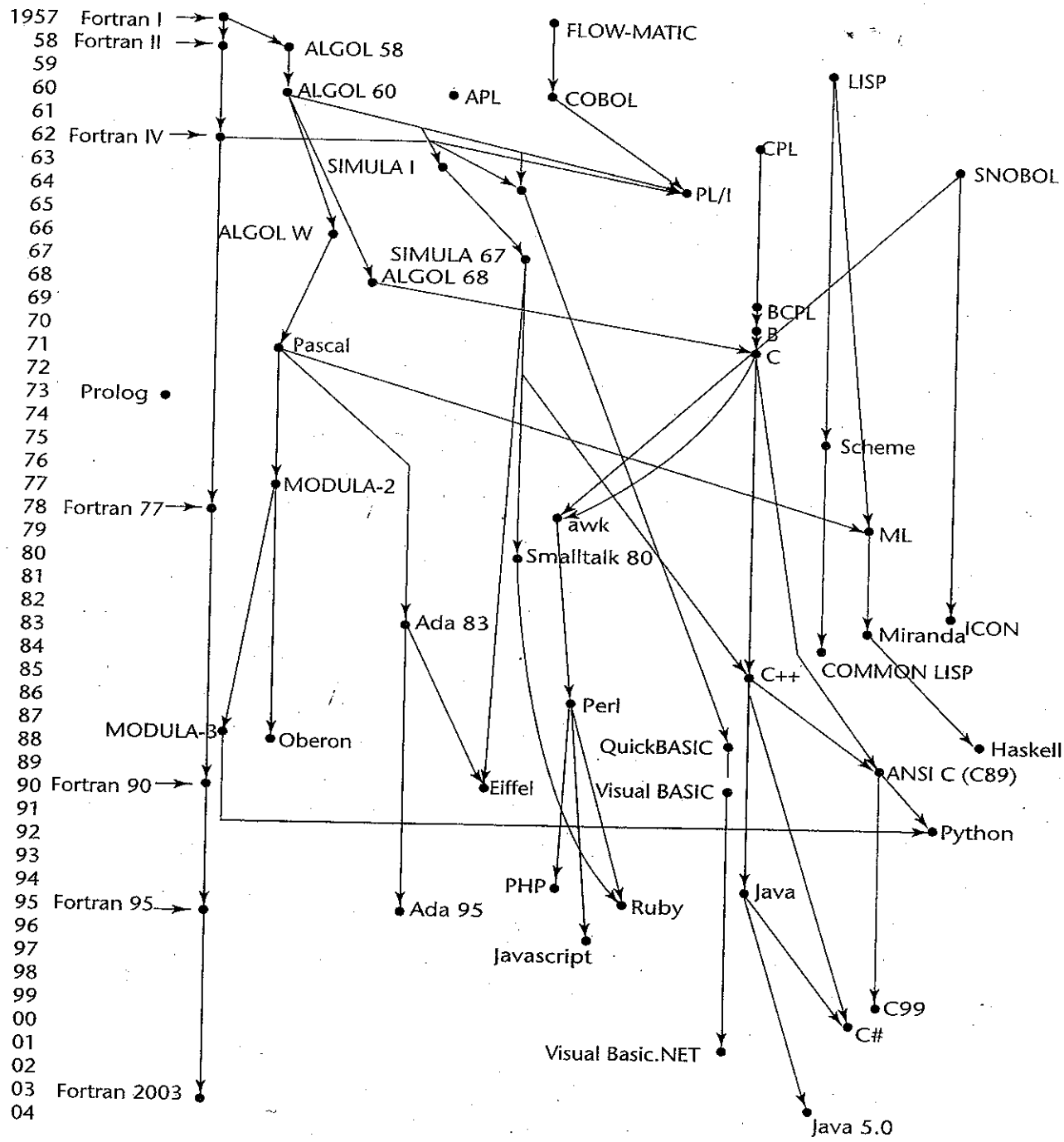



**Figure 2.1** Genealogy of common high-level programming languages

**FORTRAN** *"simplicity" "efficiency"*

**Fortran I** (1954)
- First compiled high level language
- IBM 704 – hardware floating-point number ability
- "The IBM Mathematical FORmula TRANslating system"
- Static memory allocation – no recursion
- 6 characters for variable names
- No explicit declaration → reliability problem
  I,J,K,L,M,N: integer, the rest: real
- IF statement, DO loop
- posttest (vs. pretest)
- column-major
- Blank is ignored → reliability problem

**Fortran II** (1958) - Basic FORTRAN
- Basic Fortran
- Independent compilation of subroutine
- Hollerith literal: 7HMONTANA

**FORTRAN IV** (1962 – 1978) – ANSI FORTRAN
- String: "MONTANA"
- explicit typing

**FORTRAN 77** (1978)
- character string handling, logical loop control statements, if-else

**FORTRAN 8X**
- dynamically allocated array
- case statement

**Fortran 90**
- array manipulation functions (DOTPRODUCT, MATMUL…)
- dynamic array, record, and pointer
- recursion
- free format

**Fortran 95:** forall

**HPF**

**Fortran 2003: support for OOP**

Plan to eliminate arithmetic IF, assigned GOTO, common, equivalence, computed goto

**LISP** (1959)

- McCarthy of MIT
- LISt Processing
- artificial intelligence and theorem proving
- need symbolic data processing in linked list form
- functional programming language → "what" not "how"
- recursion (vs. iteration)
- dynamic scope - interpret
- simple syntax – everything in form of parenthesized lists
  Ex. (A B C)
  1. list of 3 elements
  2. function A with 2 parameters
- dominated the AI area; still most widely used AI language

**Scheme** (1975)

- descendant of LISP
- small in size
- static scope only

**Common LISP** (1984)

- "CLISP"
- combine several dialects → large
- dynamic and static scope
- data types and structures (arrays, records, character strings, …)

**ML**

- meta language for program verification
- functional, but support imperative as well

**Miranda**

- purely functional

**Haskell**

- laze evaluation

**ALGOL** *"simplicity" "elegance"*

**Algol** 58

- universal, machine-independent, ALGOrithmic Language
- descendant of FORTRAN
- compound statement
- identifiers of any length (Fortran: 6)
- No limit on array dimension (Fortran: 3)
- User defined lower bound of array (Fortran: 1)
- for loop (Fortran Do loop)

| American | European |
|---|---|
| 12,300.00 | 12.300,00 |
| var := exp | exp =: var |

**Algol** 60

- BNF (Backus-Naur form)
- concept of block structure
- pass by name and pass by value (Fortran: pass by reference)
- recursion
- stack-dynamic array
- no formatted I/O statements; I/O is implementation dependent
- language for algorithm description
- ancestor of almost all languages since 1960's
- formally defined syntax → led to computer science fields of formal languages, parsing theory, compiler design
- affected computer architecture to implement block and recursion

**Algol** 68 *"orthogonality"*

- orthogonal design
- never widely used
- User-defined data types
- Dynamic arrays

```
flex [1:0] int list
  :
```

list = (3, 5, 6, 2)

**COBOL** (1960)

- COmmon Business Oriented Language)
- FLOW-MATIC for UNIVAC (1957)
- Department of Defense
- ANSI COBOL – 1968, 1974, 1985, 2002

- use English, not mathematic symbols (for managers)
- consists of 4 divisions
  IDENTIFICATION DIVISION
  ENVIRONMENT DIVISION
  DATA DIVISION
  PROCEDURE DIVISION
- Data division is very strong
- file, record – I/O

- hierarchical data structures

```
03 NAME.
   05  FIRST-NAME      PIC X(10).
   05  MIDDLE-NAME  PIC  X(1).
   05  LAST-NAME       PIC X(15).
03  SALARY.
   05  GROSS             PIC  99999V99.
   05   NET                PIC  99999V99.
```

- Naming: FEDERAL-TAX, STATE-TAX
- decimal numbers
- ideal for producing business report

**BASIC** (1964) *"time-sharing"*

- "Beginner's All-Purpose Symbolic Instruction Code"
- Dartmouth College – liberal arts i.e., non-science students
- user-friendly
- terminals connected to a remote computer
- fast turnaround for homework
- remote terminal access – time sharing
- Microcomputer (PC) – 70's and 80's
- "limited but easy to learn"
- instruction – easy to learn for beginners
- Fortran + Algol 60
- poor control structure

**Quick BASIC** (1988)

**Visual BASIC** (1990)
- GUI

**Visual Basic .NET**
- OOP

**PL/I** (1963) *"everything for everybody"*

- Programming Language One
- Name change:
  FORTRAN VI → NPL (New Programming Language) → PL/I
  (Note: National Physical Lab in U.K.)

- IBM product
- IBM S/360 needed universal language
- combine best features from FORTRAN, Algol 60, COBOL, LISP and assembly language

  recursion and block structure from ALGOL 60
  syntax from FORTRAN
  data structure, I/O, report generating facilities from COBOL
  Stream I/O - GET, PUT
  Record I/O - READ, WRITE

- Concurrent execution – Task
- Exception handling
- recursion

- Criticism from Dijstra – "too big"
-

Full PL/I
(54K)

Subset PL/I
(12 K)

- PL/C

**APL** (1960)

- A Programming Language
- Kenneth Iversion (IBM)
- Dynamic typing and dynamic storage allocation
- large number of powerful operators
- hard to read
- Very expressive but very hard to read
- special keyboard needed for Greek alphabet
- Arrays can be manipulated like scalar variables

**Snobol** (1960's)

- Farber, Grieswold, Polensky (Bell Lab)
- StriNg Oriented symBOlic Language
- Dynamic typing and synamic storage allocation
- String manipulation
- Lots of pattern matching operators

**SIMULA 67**

- Nygaard and Dahl (Norwegian Computing Center)
- beginning of data abstraction
- Extension of Algol 60 – block structures and control statements
- For simulation

- Coroutine

- Class = data + routine to manipulate
- Class definition is a template and can be instantiated many times
- not widely used

**Pascal (1971)** "simplicity and generality"

- Niklaus Wirth
- ALGOL-W
- Instruction
- Passing parameters by value-result
  case statement from ALGO-W
  user-defined data type from Algol 68
  record from COBOL
- simple, expressive, relatively safe
- set type
- for i: = 1 to n or for i := n downto 1
- structured control structures
  if-then-else
  case
  while do
  repeat until

**C**

- system language
- Unix system (Bell Lab)
- Typed language based on B
- Algol influence
- adequate control structures, data structures
- rich set of operators
- lack type checking

Other Algol descendants

**CLU**
**Modular-2**
**Modular-3**
**Oberon**
**Delphi**

**Prolog (1972)**

- "PROgramming in LOGic" (programmation en logique)
- Programming based on logic
- University of Marseille (Colmerauer, Roussel)
- University of Edinburgh (Kowalski)
- A.I. Database
- formal logic – predicate calculus proposition and resolution
- 2 kinds of statements – facts and rules
- specify goal (not HOW)

**Ada 83**

- History's largest design effort
- embedded system
- There were 450 languages used within DoD
  → lack of compatibility/uniformity
  → standardization
  → Ada

- Identify the requirement for a new DoD language
  Evaluate existing languages for a candidate
  Recommend adaptation or implementation of a language

- Major features
  Packages
  Exception handling
  Generic program units
  Concurrency

- Very big: Ada definition (330 pages)
- No subset or superset

**Ada 95**

- Add inheritance, polymorphism, etc

**Smalltalk** *"object oriented"*

- First pure object oriented language
- Alan Kay foresight of PC – powerful human interface – GUI
- From LOGO, FLEX, SIMULA 67
- Program units are objects
- send message to object, receive reply
- promote windowing system
- promote OOP ideas

**C++** (1980)

- Stroustrup at Bell Lab
- Combine imperative and object-oriented features
- classes
- derived classes, i.e. inheritance
- Inline functions
- Default parameters
- Overloading of operators (and functions)
- Virtual functions for dynamic binding
- multiple inheritance
- Template
- Exception handling
- Kept C's efficiency
- Downward compatible with C
- Too large

**Eiffel**

- C based
- smaller, simpler (vs. C++)

**Delphi**

- Anders Hejlsberg (Turbo Pascal) - Pascal based
- procedural + object-oriented

JAVA

JavaScript

PHP

Python

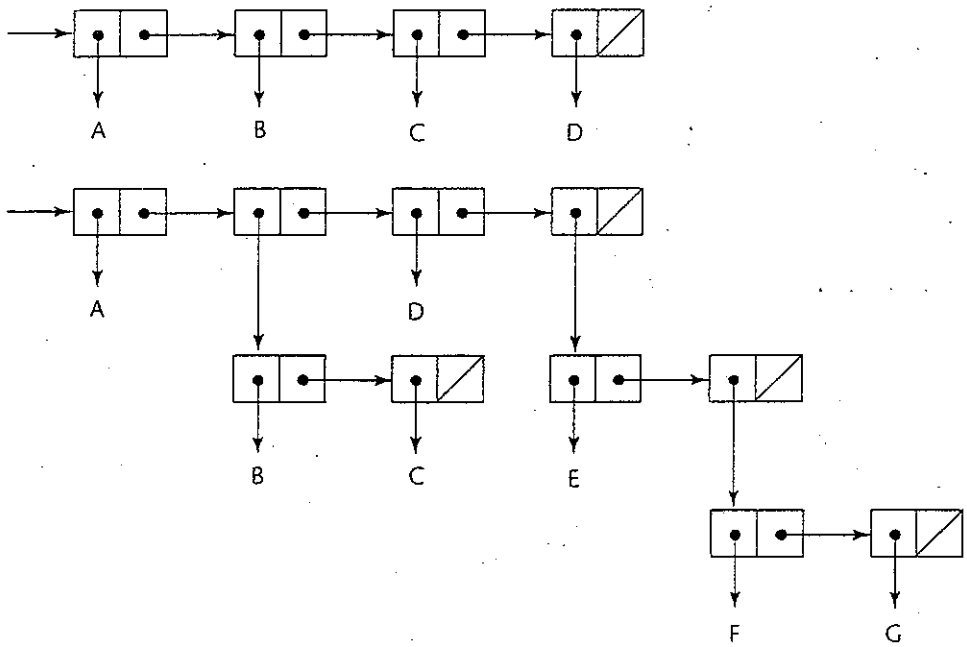
Ruby

C#

```
C  FORTRAN 90 EXAMPLE PROGRAM
C  INPUT:  AN INTEGER, LIST_LEN, WHERE LIST_LEN IS LESS
C          THAN 100, FOLLOWED BY LIST_LEN-INTEGER VALUES
C  OUTPUT: THE NUMBER OF INPUT VALUES THAT ARE GREATER
C          THAN THE AVERAGE OF ALL INPUT VALUES
      INTEGER INTLIST(99)
      INTEGER LIST_LEN, COUNTER, SUM, AVERAGE, RESULT
      RESULT = 0
      SUM = 0
      READ *, LIST_LEN
      IF ((LIST_LEN .GT. 0) .AND.
          (LIST_LEN .LT. 100)) THEN
C  READ INPUT DATA INTO AN ARRAY AND COMPUTE ITS SUM
        DO 10 COUNTER = 1, LIST_LEN
          READ *, INTLIST(COUNTER)
          SUM = SUM + INTLIST(COUNTER)
10      CONTINUE
C  COMPUTE THE AVERAGE
        AVERAGE = SUM / LIST_LEN
C  COUNT THE VALUES THAT ARE GREATER THAN THE AVERAGE
        DO 20 COUNTER = 1, LIST_LEN
          IF (INTLIST(COUNTER) .GT. AVERAGE) THEN
            RESULT = RESULT + 1
          END IF
20      CONTINUE
C  PRINT THE RESULT
        PRINT *, 'NUMBER OF VALUES > AVERAGE IS:', RESULT
      ELSE
        PRINT *, 'ERROR—LIST LENGTH VALUE IS NOT LEGAL'
      END IF
      STOP
      END
```

```
;  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 (CAR lis1) (CAR lis2))
            (equal (CDR lis1) (CDR lis2)))
       (T NIL)
     )
)
```



```
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
```

COBOL

```
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.
```

```
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
```

```
/* 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;
```

```
      )CLEAR
      ∇ RES ← PRIMES N; T
[1]   RES ← 2, T ← 3
[2]   →0 X ⍳N < ⍴ RES
[3]   T ← T + 2
[4]   →3 X ⍳ v/0 = RES | T
[5]   RES ← RES, T
[6]   →2
      ∇

      PRIMES 4
2 3 5 7 11

      ∇ PRIMES[2 □ 8]
[2]   →0 X ⍳ N < ⍴ RES
                 /1
[2]   →0 X ⍳N ⍴ RES
                 =
[2]   →0 X ⍳N = ⍴ RES

      [□] ∇
      ∇ RES ← PRIMES N;T
[1]   RES ← 2, T ← 3
[2]   →0 X ⍳N = ⍴ RES
[3]   T ← T + 2
[4]   →3 X ⍳ v/0 = RES | T
[5]   RES ← RES,T
[6]   →2
      ∇
      PRIMES 4
2 3 5 7
```

```
+ / 2 4 ⍴ ⍳3 ; ⍳5
```

```
(2 = (+ / [2] 0 = (⍳N) ∘ . | (⍳N))) / ⍳N
```

```
{Pascal 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 input values }
program pasex (input, output);
  type intlisttype = array [1..99] of integer;
  var
    intlist : intlisttype;
    listlen, counter, sum, average, result : integer;
  begin
  result := 0;
  sum := 0;
  readln (listlen);
  if ((listlen > 0) and (listlen < 100)) then
    begin
{ Read input into an array and compute the sum }
    for counter := 1 to listlen do
      begin
      readln (intlist[counter]);
      sum := sum + intlist[counter]
      end;
{ Compute the average }
    average := sum / listlen;
{ Count the number of input values that are > average }
    for counter := 1 to listlen do
      if (intlist[counter] > average) then
        result := result + 1;
{ Print the result }
    writeln ('The number of values > average is:',
             result)
    end { of the then clause of if (( listlen > 0 ... }
  else
    writeln ('Error-input list length is not legal')
end.
```

```
/* C 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 input values */
void main (){
  int intlist[98], listlen, counter, sum, average, result;
  sum = 0;
  result = 0;
  scanf("%d", &listlen);
  if ((listlen > 0) && (listlen < 100)) {
/* Read input into an array and compute the sum */
    for (counter = 0; counter < listlen; counter++) {
      scanf("%d", &intlist[counter]);
      sum = sum + intlist[counter];
     }
/* Compute the average */
    average = sum / listlen;
/* Count the input values that are > average */
    for (counter = 0; counter < listlen; counter++)
      if (intlist[counter] > average) result++;
/* Print result */
    printf("Number of values > average is:%d\n", result);
   }
  else
    printf("Error-input list length is not legal\n");
 }
```

```
-- Ada 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
with TEXT_IO; use TEXT_IO; -> make GETorPUT visible to compiler.
procedure ADA_EX is
  package INT_IO is new INTEGER_IO (INTEGER);
  use INT_IO;
  type INT_LIST_TYPE is array (1..99) of INTEGER;
  INT_LIST : INT_LIST_TYPE;
  LIST_LEN, SUM, AVERAGE, RESULT : INTEGER;
  begin
  RESULT := 0;
  SUM := 0;
  GET (LIST_LEN);
  if (LIST_LEN > 0) and (LIST_LEN < 100) then
-- Read input data into an array and compute the sum
    for COUNTER in 1 .. LIST_LEN loop
      GET (INT_LIST(COUNTER));
      SUM := SUM + INT_LIST(COUNTER);
    end loop;
-- Compute the average
    AVERAGE := SUM / LIST_LEN;
-- Count the number of values that are > average
    for COUNTER in 1 .. LIST_LEN loop
      if INT_LIST(COUNTER) > AVERAGE then
        RESULT := RESULT + 1;
      end if;
    end loop;
-- Print result
    PUT ("The number of values > average is:");
    PUT (RESULT);
    NEW_LINE;
  else
    PUT_LINE ("Error—input list length is not legal");
  end if;
end ADA_EX;
```

```
"Smalltalk Example Program"
"The following is a class definition, instantiations
 of which can draw equilateral polygons of any number
 of sides"
class name                      Polygon
superclass                      Object
instance variable names         ourPen
                                numSides
                                sideLength
"Class methods"
 "Create an instance"
 new
     ^ super new getPen

 "Get a pen for drawing polygons"
 getPen
     ourPen <- Pen new

 "Instance methods"
 "Draw a polygon"
 draw
     numSides timesRepeat: [ourPen go: sideLength;
                            turn: 360 // numSides]

 "Set length of sides"
 length: len
     sideLength <- len


 "Set number of sides"
 sides: num
     numSides <- num
```

```
// Java Example Program
//  Input: An integer, listlen, where listlen is less
//         than 100, followed by length-integer values
// Output: The number of input data that are greater than
//         the average of all input values

import java.io.*;
class IntSort {
public static void main(String args[]) throws IOException {
  DataInputStream in = new DataInputStream(System.in);
  int listlen,
      counter,
      sum = 0,
      average,
      result = 0;
  int[] intlist = int[99];
  listlen = Integer.parseInt(in.readLine());
  if ((listlen > 0) && (listlen < 100)) {
/* Read input into an array and compute the sum  */
    for (counter = 0; counter < listlen; counter++) {
      intlist[counter] =
            Integer.valueOf(in.readLine()).intValue();
      sum += intlist[counter];
    }
/* Compute the average */
    average = sum / listlen;
/* Count the input values that are > average */
    for (counter = 0; counter < listlen; counter++)
      if (intlist[counter] > average) result++;
/* Print result */
      System.out.println(
          "\nNumber of values > average is:" + result);
    }  //** end of then clause of if ((listlen > 0) ...
    else System.out.println(
            "Error--input list length is not legal\n");
  }  //** end of method main
}  //** end of class IntSort
```

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1 //EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11-strict.dtd">
<!-- example.html
     Input: An integer, listLen, where listLen is less
            than 100, followed by listLen-numeric values
    Output: The number of input values that are greater
            than the average of all input values.
    -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head><title> Example </title>
</head>
<body>
<script type = "text/javascript">
<!--
var intList = new Array(99);
var listLen, counter, sum = 0, result = 0;

listLen = prompt (
        "Please type the length of the input list", "");
if ((listLen > 0) && (listLen < 100)) {
// Get the input and compute its sum
   for (counter = 0; counter < listLen; counter++) {
      intList[counter] = prompt (
                    "Please type the next number", "");
      sum += parseInt(intList[counter]);
   }
// Compute the average
   average = sum / listLen;
// Count the input values that are > average
   for (counter = 0; counter < listLen; counter++)
      if (intList[counter] > average) result++;
// Display the results
   document.write("Number of values > average is: ",
                  result, "<br />");
} else
   document.write(
       "Error - input list length is not legal <br />");
// -->
</script>
</body>
</html>
```

```
// C# 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 input values.
using System;
public class Ch2example {
  static void Main() {
    int[] intlist;
    int listlen,
        counter,
        sum = 0,
        average,
        result = 0;
    intList = new int[99];
    listlen = Int32.Parse(Console.readLiine());
    if ((listlen > 0) && (listlen < 100)) {
// Read input into an array and compute the sum
      for (counter = 0; counter < listlen; counter++) {
        intList[counter] = Int32.Parse(Console.readLine());
        sum += intList[counter];
      } //- end of for (counter ...
// Compute the average
      average = sum / listlen;
// Count the input values that are > average
      foreach (int num in intList)
        if (num > average) result++;
// Print result
      Console.WriteLine(
         "Number of values > average is:" + result);
    } //- end of if ((listlen ...
    else
      Console.WriteLine(
         "Error--input list length is not legal");
  } //- end of method Main
} //- end of class Ch2example
```