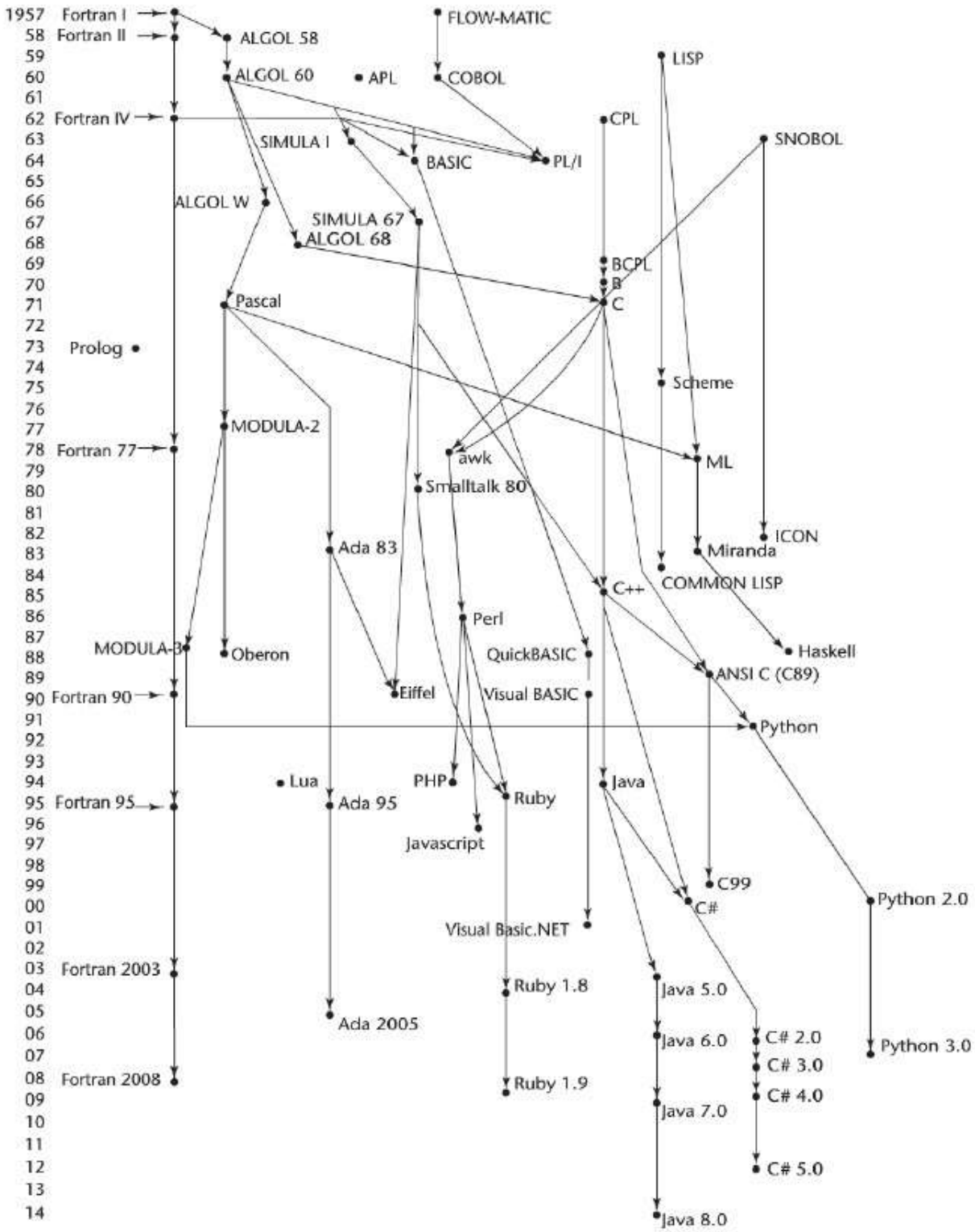
2. EVOLUTION OF THE MAJOR PROGRAMMING LANGUAGES

Genealogy of Common High-Level Programming Languages



Zuse's Plankalkül

• Between 1936 and 1945, German scientist Konrad Zuse built a series of complex and sophisticated computers from electromechanical relays.

Konrad Zuse

• Zuse also developed a language for expressing computations. He named this language Plankalkül (program calculus).

• In a lengthy manuscript dated 1945 but not published until 1972, Zuse defined Plankalkül and wrote algorithms in the language to solve a wide variety of problems.

• Plankalkül was never implemented.

Plankalkül Language Overview

• Data structures in Plankalkül:

The simplest type in Plankalkül was the single bit.
Integer and floating-point numeric types were built from the bit type. The floating-point type used two's-complement notation and employed a "hidden bit" scheme to avoid storing the most significant bit of the normalized fraction part of a value.
Plankalkül included arrays and records. Records could be nested.

• Statements in Plankalkül:

No explicit `goto` statement.
Iterative statement similar to the Pascal `for`.
The `Fin` command had a superscript that indicated an exit out of a specified number of nested loops or to the beginning of a new iteration cycle.
A selection statement was included, but it did not allow an `else` clause.

• Each Plankalkül statement consists of either two or three lines of code.

1. Similar to a statement in a contemporary language.
2. (optional) Subscripts of the array references in the first line.
3. Types for variables mentioned in the first line.

• A statement that assigns the value of the expression `A(4) + 1` to `A(5)`:

```
  | A + 1 => A
V | 45
S | 1.n1.n
```

The row labeled `V` is for subscripts, and the row labeled `S` is for data types. `1.n` means an integer of `n` bits.

• Zuse's programs included assertions (mathematical expressions showing the current relationships between program variables).

Pseudocodes

• In the late 1940s and early 1950s, there were no high-level programming languages or even assembly languages, so programming was done in machine code.

• Problems of machine code:

Programs are hard to read.
Absolute addressing makes modifications tedious and error-prone.

These problems were the primary motivation for inventing assemblers and assembly languages.

• Most programming problems of that time required floating-point arithmetic and indexing to allow the convenient use of arrays. The need for these operations led to the development of somewhat higher-level languages.

Short Code

• Short Code was developed by John Mauchly in 1949 for the BINAC computer and later transferred to the UNIVAC I.

• Short Code consisted of coded versions of mathematical expressions. The codes were byte-pair values, and most equations fit into a word. Sample operation codes:

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

• Variables were named with byte-pair codes, as were locations to be used as constants.

• Example:

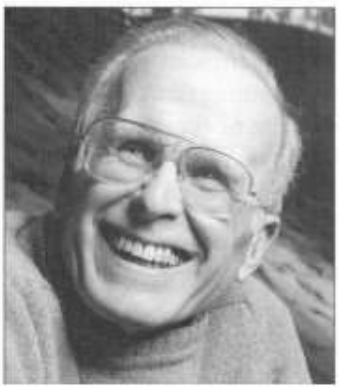
```
X0 = SQRT(ABS(Y0))
```

This statement would be coded in a word as `00 X0 03 20 06 Y0.` The initial `00` was used as padding to fill the word.

• Short Code was not translated to machine code; rather, it was interpreted (known at the time as automatic programming). Short Code interpretation was approximately 50 times slower than machine code.

Speedcoding

• The Speedcoding system developed by John Backus for the IBM 701 extended machine language to include floating-point operations.



John Backus

• The Speedcoding interpreter effectively converted the 701 to a virtual three-address floating-point calculator. Operations included:

Pseudoinstructions for the four arithmetic operations on floating-point data
Mathematical functions such as square root, sine, arc tangent, exponent, and logarithm
Conditional and unconditional branches
Input/output conversions

• Speedcoding was much more efficient than writing in machine language. Matrix multiplication, for example, could be done in 12 Speedcoding instructions.

The UNIVAC "Compiling" System and Related Work

• Between 1951 and 1953, a team led by Grace Hopper at UNIVAC developed a series of "compiling" systems named A-0, A-1, and A-2. These systems expanded a pseudocode into machine code in the same way as macros are expanded into assembly language.

Grace Hopper

• At Cambridge University, David Wheeler developed a method of using blocks of relocatable addresses to partially solve the problem of absolute addressing.

• Maurice V. Wilkes (also at Cambridge) extended the idea to design an assembly program that could combine chosen subroutines and allocate storage.

The IBM 704 and Fortran

- The introduction of the IBM 704 in 1954 prompted the development of Fortran. Unlike earlier computers, the 704 hardware supported both indexing and floating-point instructions.

- Fortran was designed by an IBM group led by John Backus.

- Because of the high cost of computers compared to the cost of programmers, speed of the generated object code was the primary goal of the Fortran I compiler, which was released in April 1957.

- The Fortran development group claimed that the machine code produced by the compiler would be about half as efficient as what could be produced by hand. Despite considerable skepticism, the compiler nearly achieved this goal.

- Fortran was the first widely accepted compiled high-level language, although it was probably not the first compiled high-level language.

Fortran I Overview

• Features of Fortran I:

Input/output formatting
Variable names of up to six characters
User-defined subroutines
`IF` statement
`DO` (loop) statement

• All of Fortran I's control statements were based on 704 instructions.

• There were no variable declarations in Fortran I.

Variables whose names began with `I, J, K, L, M,` and `N` were implicitly of integer type.
All other variables were implicitly floating-point.

Later Versions of Fortran

• Fortran II (1958)

Fixed many of the bugs in the Fortran I compilation system.
Added significant features, including independent compilation of subroutines.

• Fortran III

Was developed but never widely distributed.

• Fortran IV (1960–62)

Added explicit type declarations for variables, a logical `IF` construct, and the ability to pass subprograms as parameters to other subprograms.
Was standardized in 1966 as Fortran 66.

• Fortran 77 (1978)

Added character string handling and an `IF` with an optional `ELSE` clause.

Later Versions of Fortran (Continued)

- Fortran 90 is dramatically different from Fortran 77, adding dynamic arrays, records, pointers, a multiple selection statement, and modules. Also, Fortran 90 allows subprograms to be recursive.

- The Fortran 90 standard included a list of features that were recommended for future removal.

- Fortran 90 made two significant changes to the general form of programs:

  In prior versions of Fortran, each line in a program had a fixed format: the first five characters were reserved for statement labels, and statements could not begin before the seventh character.

  Previous versions of Fortran supported only uppercase letters. Fortran 90 supports lowercase letters as well.

- Fortran 95 added a few new features, including the `Forall` statement.

- Fortran 2003 added support for object-oriented programming along with other new features.

- Fortran 2008 added several new features, including better support for concurrent programming.

Evaluation of Fortran

- As the first widely used high-level language, Fortran dramatically changed the way computers are used.

- The original Fortran design team thought of language design only as a necessary prelude to the critical task of designing the translator.

- In comparison with concepts and languages developed later, early versions of Fortran suffer in a variety of ways.

- Prior to Fortran 90, all storage requirements for a program could be fixed before run time.

  All variables and arrays had fixed sizes.
  No recursion was permitted.

  Having programs use a fixed amount of memory leads to greater run-time efficiency. On the other hand, it imposes a burden on the programmer.

- In spite of the inadequacies of Fortran, it remains one of the most widely used high-level languages.

Example Fortran 95 Program

```
Program Example
! 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
```

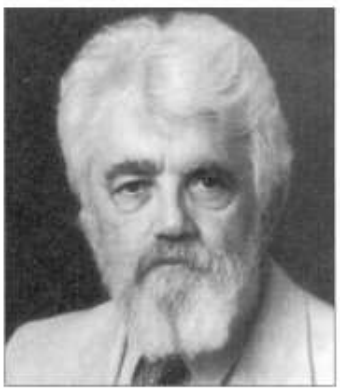
Functional Programming: Lisp

- The first functional programming language was invented to support list processing, which was needed for artificial intelligence (AI) programs.

- The concept of list processing was developed by Allen Newell, J. C. Shaw, and Herbert Simon in the languages IPL-I (Information Processing Language I) through IPL-V.

- In the mid-1950s, IBM added list processing capabilities to Fortran. The result was FLPL (Fortran List Processing Language).

- In 1958, John McCarthy of MIT began investigating symbolic computation, using differentiation of algebraic expressions as an example.



John McCarthy

- McCarthy discovered that FLPL lacked many of features needed for symbolic computation, including recursion, conditional expressions, dynamic storage allocation, and implicit deallocation.

- Later in 1958, McCarthy helped form the MIT AI Project, where Lisp was developed.

Lisp Language Overview

• The first version of Lisp is sometimes called “pure Lisp” because it is a purely functional language.

• Pure Lisp has only two kinds of data structures:

Atoms (symbols and numeric literals)
Lists

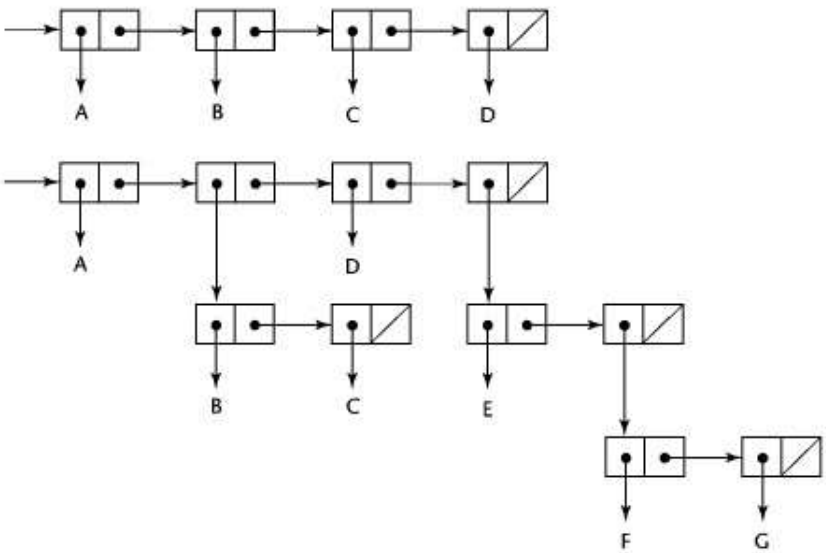
• A list is a series of elements enclosed within parentheses:

```
(A B C D)
```

• The elements of a list can be either atoms or lists:

```
(A (B C) D (E (F G)))
```

• Internally, lists are stored as single-linked list structures, in which each node has two pointers and represents a list element.



Lisp Language Overview (Continued)

- Lisp was designed as a functional programming language:

  All computation in a functional program is accomplished by applying functions to arguments.
  Neither assignment statements nor variables are necessary in functional language programs.
  Recursive function calls can be used for iteration, making loops unnecessary.

- In Lisp, both program code and data are written in list form. For example, the following list could be either data or code:

```
(A B C D)
```

  When interpreted as data, it is a list of four elements. When viewed as code, it is the application of the function named `A` to the three parameters `B, C,` and `D.`

Evaluation of Lisp

• Lisp dominated AI applications for a quarter century.

• Lisp pioneered functional programming, which has proven to be a lively area of research in the field of programming languages.

• An example Lisp function:

```
;  Lisp example function
;  The following code defines a Lisp predicate function
;  that takes two lists as arguments and returns T (true)
;  if the two lists are equal, and NIL (false) otherwise

(DEFUN equal_lists (lisl lis2)
  (COND
    ((ATOM lisl) (EQ lisl lis2))
    ((ATOM lis2) NIL)
    ((equal_lists (CAR lisl) (CAR lis2))
      (equal_lists (CDR lisl) (CDR lis2)))
    (T NIL)
  )
)
```

Two Descendants of Lisp

• Scheme

Developed at MIT in the mid-1970s.
Small size with simple syntax and semantics.

• Common Lisp

Tries to combine several dialects of Lisp into a single language, making it relatively large and complex.
Has a large number of data types and structures, including records, arrays, complex numbers, and character strings.

Related Languages

• ML (MetaLanguage)

Designed in the 1980s by Robin Milner at the University of Edinburgh.
Primarily a functional language, but it also supports imperative programming.
The type of every variable and expression in ML can be determined at compile time.
Types of names and expressions are inferred from context.
Syntax resembles that of an imperative language. Does not use the parenthesized functional syntax of Lisp and Scheme.

• Miranda

Developed by David Turner at the University of Kent in the early 1980s.
Based partly on ML.

• Haskell

Based largely on Miranda.
A purely functional language, having no variables and no assignment statement.
Uses lazy evaluation; no expression is evaluated until its value is required.

• Caml and OCaml

Descended from ML and Haskell.
OCaml supports object-oriented programming.

• F#

Based on OCaml.
One of the .NET languages.
Supports both functional programming and imperative programming.
Fully object-oriented.

The First Step Toward Sophistication: ALGOL 60

• In 1957, several American user groups petitioned ACM to form a committee to design a machine-independent scientific programming language.

• A similar effort had been started in 1955 in Europe by GAMM (a German organization). In 1958, Fritz Bauer of GAMM invited ACM to collaborate on a joint language-design project.

Friedrich (Fritz) L. Bauer

• The first joint meeting was held in Zurich in 1958.

• Goals for the new language:

The syntax of the language should be as close as possible to standard mathematical notation, and programs written in it should be readable with little further explanation.

It should be possible to use the language for the description of computing processes in printed publications.

Programs in the new language must be mechanically translatable into machine language.

• The Zurich meeting involved innumerable compromises. One example: the question of whether to use a comma or a period for the decimal point.

ALGOL 58

• The language designed at Zurich was named the International Algorithmic Language (IAL). During the following year, the name was changed to ALGOL, and the language subsequently became known as ALGOL 58.

• ALGOL 58 generalized many of Fortran's features and added several new constructs and concepts.

• High points of ALGOL 58:

Formalized the concept of data type, although only variables that were not floating-point required explicit declaration.
Added compound statements.
Allowed identifiers to have any length.
Allowed any number of array dimensions.
Allowed lower bound of arrays to be specified by the programmer.
Allowed selection statements to be nested.

• Some European members of the committee were familiar with Zuse's Plankalkül, and wanted assignment to be written

```
expression => variable
```

Because of arguments about character limitations, the greater-than symbol was changed to a colon. Then, largely at the insistence of the Americans, the whole statement was turned around to resemble Fortran:

```
variable := expression
```

• The ALGOL 58 report was only a preliminary document. Nevertheless, three major language efforts used ALGOL 58 as their basis:

MAD (University of Michigan)
NELIAC (U.S. Naval Electronics Group)
JOVIAL (Jules' Own Version of the International Algebraic Language), created by System Development Corporation. For a quarter century, JOVIAL was the official scientific language for the U.S. Air Force.

ALGOL 60 Design Process

• In 1959, John Backus introduced his new notation for describing the syntax of programming languages, which later became known as BNF (Backus-Naur form).

• In January 1960, the second ALGOL meeting was held. Peter Naur of Denmark wrote a description of ALGOL in BNF and handed it out at the meeting.

Peter Naur

• The 1960 meeting resulted in several important changes to ALGOL:

Block structure was added.
Parameters could be passed to subprograms in two ways: by value and by name.
Procedures could be recursive.
Stack-dynamic arrays (arrays whose subscript ranges are specified by variables) were allowed.

• Input and output statements were omitted because they were thought to be too machine-dependent.

• The ALGOL 60 report was published in May 1960.

• A third meeting was held in April 1962. The results of this meeting were published under the title “Revised Report on the Algorithmic Language ALGOL 60.”

Evaluation of ALGOL 60

• In some ways, ALGOL 60 was a great success:

It succeeded in becoming the only acceptable formal means of communicating algorithms.
Every imperative programming language designed since 1960 owes something to ALGOL 60.

• The ALGOL 58/ALGOL 60 design effort included several firsts:

First language designed by an international group.
First language designed to be machine-independent.
First language whose syntax was formally described.

• The structure of ALGOL 60 affected machine architecture, especially at Burroughs, whose B5000, B6000, and B7000 computers were designed with a hardware stack to implement ALGOL's block structure and recursive subprograms.

• ALGOL 60 never achieved widespread use in the United States, however. Even in Europe, it never became the dominant language.

• Reasons for ALGOL 60's lack of acceptance:

Some features (such as passing parameters by name) turned out to be too flexible; they made understanding difficult and implementation inefficient.
Lack of input and output statements in the language.
BNF seemed strange and complicated.
Entrenchment of Fortran among users.
Lack of support by IBM.

Example ALGOL 60 Program

```
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

• Prior to COBOL, there were few business languages. The primary one was FLOW-MATIC, which was implemented in 1957 for UNIVAC computers.

• FLOW-MATIC was developed under the direction of Grace Hopper, who believed that languages for business applications should use English statements.

• The first meeting on the subject of a common language for business applications was held at the Pentagon in 1959.

• Goals for the new language:

  Should use English as much as possible.
  Should be easy to use.
  Should not be overly restricted by implementation problems.

• There were early decisions to separate the statements of the language into two categories—data description and executable operations—which would reside in different parts of a program.

• The design committee debated whether or not to support subscripts and arithmetic expressions.

• Versions of COBOL:

  COBOL 60
  Revised COBOL (1961 and 1962)
  COBOL 68 (standard)
  COBOL 74 (standard)
  COBOL 85 (standard)
  COBOL 2002 (standard)

Evaluation of COBOL

- Although COBOL has been used more than any other programming language, it has had little effect on the design of subsequent languages, except for PL/I.

- Novel concepts in COBOL:

  `DEFINE` verb—the first high-level language construct for macros.
  Hierarchical data structures (first implemented in COBOL, although they existed in Plankalkül).
  Use of long names (up to 30 characters) and word-connector characters (dashes).

- The data division is the strong part of COBOL's design:

  Every variable is defined in detail in the data division, including the number of decimal digits and the location of the implied decimal point.
  File records are also described in detail, as are lines to be output to a printer, which makes COBOL ideal for printing reports.

- COBOL's procedure division is relatively weak:

  Functions were not supported in original COBOL.
  Versions of COBOL prior to the 1974 standard also did not allow subprograms with parameters.

- COBOL was the first programming language whose use was mandated by the Department of Defense (DoD). Without this mandate, the language probably would not have survived.

Example COBOL Program

- The following COBOL program reads a file that contains inventory information about a collection of items. Each item record includes the number currently on hand (BAL-ON-HAND) and the item's reorder point (BAL-REORDER-POINT). The reorder point is the threshold number of items on hand at which more must be ordered. The program produces a list of items that must be reordered.

```
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).
```

Example COBOL Program (Continued)

```
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.
```

The Beginnings of Timesharing: Basic

- Basic, like COBOL, has enjoyed widespread use but gotten little respect.

- Like COBOL, in its earliest versions it was inelegant and included only a meager set of control statements.

- Basic was very popular on microcomputers in the late 1970s and early 1980s. This followed directly from two of the main characteristics of early versions of Basic:

  It was easy for beginners to learn, especially those who were not science-oriented.
  Its smaller dialects can be implemented on computers with very small memories.

- Basic became less popular as microcomputers became more powerful.

- A strong resurgence in the use of Basic began with the appearance of Microsoft's Visual Basic in the early 1990s.

Basic Design Process

• Basic (Beginner's All-purpose Symbolic Instruction Code) was designed at Dartmouth College in New Hampshire by two mathematicians, John Kemeny and Thomas Kurtz.

John Kemeny

• Goals of the Basic system:

Be easy for nonscience students to learn and use.
Be "pleasant and friendly."
Provide fast turnaround for homework.
Allow free and private access.
Treat user time as more important than computer time.

• To meet these goals, the system provides time-shared access through terminals.

• Work on the implementation of Basic began in 1963. At 4 a.m. on May 1, 1964, the first program using time-shared Basic was typed in and run.

Basic Overview and Evaluation

- Original Basic had only 14 different statement types and a single data type, floating-point. (It was believed that few of the targeted users would appreciate the difference between integer and floating-point types.)

- Much of the design of Basic came from Fortran, with some minor influence from the syntax of ALGOL 60.

- Basic later grew in a variety of ways, with little or no effort made to standardize it.

- Basic was the first widely used language to allow program development via terminals connected to a computer.

- Basic has been criticized for the poor structure of programs written in it, among other things.

- Early versions of Basic were not meant for writing programs of any significant size. Later versions are better suited to such tasks.

- Visual Basic remains the most widely used version of Basic. Visual Basic .NET, which supports object-oriented programming, is a significant departure from earlier versions of Visual Basic.

Example Basic Program

```
REM Basic Example Program
REM Input: An integer, listlen, where listlen is less
REMthan 100, followed by listlen-integer values
REM Output: The number of input values that are greater
REMthan 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
```

Everything for Everybody: PL/I

• PL/I represents the first large-scale attempt to design a language that could be used for a broad spectrum of application areas.

• By the early 1960s, programmers had settled into two camps:

Scientific programmers used the floating-point data type and arrays extensively. Fortran was their primary language.
Business programmers needed decimal and character string data types as well as elaborate input and output facilities. They mainly used COBOL.

• In early 1963, IBM saw that the two widely separated groups were moving toward each other. IBM's System/360 computers, which supported both floating-point and decimal arithmetic, were designed to appeal to both groups.

• At the same time, IBM decided to develop a programming language that could be used for both business and scientific applications.

• The first version of PL/I was an extension of Fortran called Fortran VI. Later, the name was changed to NPL (New Programming Language), and then, in 1965, to PL/I.

PL/I Language Overview

• PL/I was an attempt to combine the best parts of several existing languages:

ALGOL 60 (recursion and block structure)
Fortran IV (separate compilation with communication through global data)
COBOL 60 (data structures, input/output, and report-generating facilities)

• PL/I added a few new constructs as well:

Programs could create concurrently executing tasks.
It was possible to detect and handle 23 types of exceptions.
Procedures could be called recursively, but recursion could be disabled, allowing more efficient code for nonrecursive procedures.
Pointers were included as a data type.
Cross sections of arrays could be referenced.

• PL/I turned out to be too ambitious:

There were too many features.
There was too little concern about how language features would behave when combined.
Some features were not well designed (pointers, exception handling, and concurrency).

• Still, PL/I was at least a partial success. In the 1970s, it enjoyed significant use in both business and scientific applications and—in subset form—as an instructional vehicle.

Example PL/I Program

```
/* 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

• APL and SNOBOL are quite different from each other. However, they share two characteristics:

  Dynamic typing—A variable acquires a type when it is assigned a value, at which time it assumes the type of the value assigned.
  Dynamic storage allocation—Storage is allocated to a variable only when it is assigned a value.

• Neither APL nor SNOBOL had much influence on later mainstream languages.

Origins and Characteristics of APL

- APL was designed around 1960 by Kenneth E. Iverson at IBM.

- It was not originally designed to be an implemented programming language, but rather was intended to be a vehicle for describing computer architecture.

- APL was first described in the book from which it gets its name, A Programming Language, published in 1962.

- In the mid-1960s, the first implementation of APL was developed at IBM.

- APL is famous for its large collection of operators.

  One reason APL has so many operators is that it provides a large number of unit operations on arrays.
  Many of APL's operators are not found on standard keyboards. IBM had to create a special APL print ball for their printing terminals.
  The large collection of operators provides very high expressivity but also makes APL programs difficult to read.

- APL is still used today, although not widely. It has not changed a great deal over its lifetime.

Origins and Characteristics of SNOBOL

- SNOBOL was designed in the early 1960s at Bell Laboratories by D. J. Farber, R. E. Griswold, and F. P. Polensky.

- SNOBOL was designed specifically for text processing. The heart of the language is a collection of powerful operations for string pattern matching.

- One of the early applications of SNOBOL was for writing text editors.

The Beginnings of Data Abstraction: SIMULA 67

• SIMULA I was designed primarily as a simulation language. It was developed by Kristen Nygaard and Ole-Johan Dahl between 1962 and 1964 at the Norwegian Computing Center.

Ole-Johan Dahl

• Nygaard and Dahl then extended SIMULA I by adding features and making changes to make the language useful for more general-purpose applications. The result was SIMULA 67.

• SIMULA 67 is an extension of ALGOL 60. ALGOL lacked coroutines, which are needed for simulations.

• To support coroutines, Nygaard and Dahl developed the class construct and with it the concept of data abstraction.

• SIMULA 67 was the first language to support data abstraction, which provides the foundation for object-oriented programming.

• Although SIMULA 67 never achieved widespread use, some of the concepts it introduced make it historically important.

Orthogonal Design: ALGOL 68

• ALGOL 68 was dramatically different from its predecessor, ALGOL 60.

• One of the most interesting innovations was the language's emphasis on orthogonality, as illustrated by its user-defined data types:

  Primitive data types included boolean, character, integer, real, and complex.
  From these types, references, arrays, structures, and unions could be constructed.

• ALGOL 68 introduced "implicit heap-dynamic" arrays (called flex arrays).
  The declaration of such an array does not specify its length, which is determined later when storage is allocated for the array's elements.

• ALGOL 68 includes a significant number of features that had not been previously used. Its use of orthogonality was revolutionary.

• ALGOL 68 never enjoyed widespread popularity, largely due to problems with the description of the language:

  An unknown metalanguage, van Wijngaarden grammars, was used to describe ALGOL 68. These grammars were far more complex than BNF.
  New terms were created to explain the language. Keywords were indicants, substring extraction was trimming, and the process of procedure execution was coercion of deproceduring, which might be meek, firm, or something else.

Some Early Descendants of the ALGOLs

- All modern imperative languages owe some of their design to ALGOL 60 and/or ALGOL 68.

- Early descendants of the ALGOLs include Pascal and C.

Simplicity by Design: Pascal

• Niklaus Wirth and C. A. R. ("Tony") Hoare belonged to the group that developed ALGOL 68. When their modest proposal for additions and modifications to ALGOL 60 was rejected in 1965, they turned it into the language ALGOL-W.

Niklaus Wirth

• ALGOL-W was implemented at Stanford University and used primarily as an instructional vehicle. The primary contributions of ALGOL-W were the value-result method of passing parameters and the `case` statement.

• Wirth's next major design effort, again based on ALGOL 60, was his most successful: Pascal. The original published definition of Pascal appeared in 1971.

• Features that are often ascribed to Pascal in fact came from earlier languages:

User-defined data types (ALGOL 68)
`case` statement (ALGOL-W)
Records (similar to those of COBOL and PL/I)

Evaluation of Pascal

- The largest impact of Pascal has been on the teaching of programming. By the mid-1980s, Pascal had become the most widely used introductory language.

- Because Pascal was designed as a teaching language, it lacks several features that are essential for some applications. Examples of Pascal's shortcomings:

  Array parameters must have a fixed length.
  No separate compilation capability.

  These deficiencies led to many nonstandard dialects, such as Turbo Pascal.

- Reasons for Pascal's popularity:

  Simplicity combined with expressivity.
  Relative safety, particularly when compared with Fortran or C.

- By the mid-1990s, the popularity of Pascal was on the decline, both in industry and in universities.

Example Pascal Program

```
{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.
```

A Portable Systems Language: C

• Like Pascal, C contributed little to the previously known collection of language features, but it has been very widely used over a long period of time.

• Although originally designed for systems programming, C is suitable for a variety of applications.

• Ancestors of C:

CPL—Developed at Cambridge University in the early 1960s.
BCPL—A simple systems language developed by Martin Richards in 1967.
B—The first high-level language implemented under UNIX. Designed and implemented by Ken Thompson of Bell Laboratories in 1970.
ALGOL 68

• Neither BCPL nor B is a typed language. In an untyped language, all data are considered machine words.

• Problems with this approach led to the development of a new typed language based on B.

• The new language was originally called NB but later named C.

A Portable Systems Language: C (Continued)

• C was designed and implemented by Dennis Ritchie at Bell Laboratories in 1972.

Dennis Ritchie

• The influence of ALGOL 68 is seen in C's `for` and `switch` statements, in its assignment operators, and in its treatment of pointers.

• Versions of C:

K&R C (the language described in the first edition of Kernighan and Ritchie's The C Programming Language)
C89 (the 1989 ANSI standard version)
C99 (the 1999 standard)

Evaluation of C

- C has adequate control statements and data-structuring facilities to allow its use in many application areas. It also has a rich set of operators.

- One reason that C is both liked and disliked is its lack of complete type checking. For example, functions can be written for which parameters are not type checked. Features such as this provide flexibility at the cost of insecurity.

- A major reason for C's popularity is that it is part of the widely used UNIX operating system.

- An example C program:

```
/* 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 */
int main() {
   int intlist[99], 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 += intlist[counter];
       }
/* Compute the average */
      average = sum / listlen;
/* Count the input values that are > average */
      for (counter = 0; counter < listlen; counter++)
        if (intlist[counterl > average) result++;
/* Print result */
      printf("Number of values > average is: %d\n", result);
    }
   else
    printf("Error-input list length is not legal\n");
   return 0;
}
```

Programming Based on Logic: Prolog

• Logic programming is the use of a formal logic notation to describe computational processes.

• Programming in logic programming languages is nonprocedural. Programs in such languages do not state exactly how a result is to be computed but rather describe the form and/or characteristics of the result.

• In the early 1970s, Alain Colmerauer and Phillippe Roussel of the University of Aix-Marseille, together with Robert Kowalski of the University of Edinburgh, developed the fundamental design of Prolog.

Robert Kowalski

• The first Prolog interpreter was developed at Marseille in 1972.

Prolog Overview and Evaluation

• A Prolog program consists of two kinds of statements, facts and rules:

Fact: `mother(joanne, jake).`
Fact: `father(vern, joanne).`
Rule: `grandparent(X, Z) :- parent(X, Y), parent(Y, Z).`
Rule: `parent(X, Y) :- mother(X, Y).`
Rule: `parent(X, Y) :- father(X, Y).`

These facts and rules form a kind of intelligent database.

• The Prolog database can be interactively queried with goal statements, such as

```
grandparent(vern, jake).
```

When such a query, or goal, is presented to the Prolog system, it uses a process called resolution to attempt to determine the truth of the statement.

• Reasons why logic programming has not become more widely used:

Too inefficient.
Effective only for certain kinds of database management systems and some areas of AI.

History's Largest Design Effort: Ada

- The Ada language is the result of the most extensive and expensive language design effort ever undertaken.

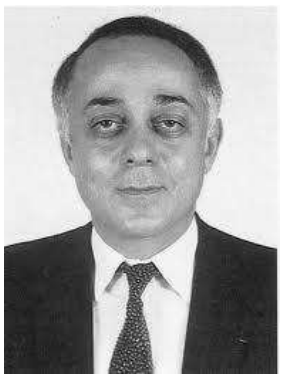
- 1974 – DoD discovers that more than 450 different programming languages were in use for embedded systems development, none of them were standardized by DoD, and none were suitable for embedded systems.

- 1975 – High-Order Language Working Group formed to choose an existing language (or languages) or create a new language for embedded systems.

- 1977 – Four contractors chosen to produce Phase 1 of the language design. All four designs were based on Pascal.

- 1978 – Two finalists chosen to go on to Phase 2.

- 1979 – Cii Honeywell/Bull language design chosen as the winner. The design team was led by Jean Ichbiah.



Jean Ichbiah

The new language was named Ada, after Augusta Ada Byron (1815–1851). A mathematician and daughter of poet Lord Byron, she is generally recognized as the world's first programmer.

- 1980 – Revised version of the language design accepted as MIL-STD 1815.

- 1983 – Standardized by the American National Standards Institute.

Ada Overview

• Major contributions of Ada:

Packages—Provide the means for encapsulating data objects, specifications for data types, and procedures.
Exception handling—Allows the programmer to gain control after any one of a wide variety of exceptions, or run-time errors, has been detected.
Generic packages and subprograms.
Tasks—Can execute concurrently, using the rendezvous mechanism for communication and synchronization.

• Ada was initially criticized as too large and complex. In 1981, Hoare stated that it should not be used for any application where reliability is critical. (He later softened his opinion.)

• Others have praised Ada as the epitome of language design.

Example Ada Program

```
-- 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 Ada.Text_IO, Ada.Integer_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO;
procedure Ada_Ex is
   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;
```

Ada 95 and Ada 2005

- Ada 95 adds two major features to the language:

  An extended type derivation mechanism that supports inheritance and polymorphism.
  Protected objects, which provide a more efficient way to share data among tasks.

- Reasons why Ada 95 is not more popular:

  The DoD stopped requiring its use in military software systems.
  C++ was well-established before Ada 95 was released.

- A major amendment was later added to Ada 95, creating the language known as Ada 2005. New features include interfaces, which are similar to those of Java.

- Ada is widely used in both commercial and defense avionics, air traffic control, and rail transportation, as well as in other areas.

Object-Oriented Programming: Smalltalk

• Language features needed to support object-oriented programming:

  Data abstraction
  Inheritance
  Dynamic binding (polymorphism)

• Smalltalk was the first language that fully supported object-oriented programming.

Smalltalk Design Process

• The concepts that led to the development of Smalltalk originated in the Ph.D. dissertation work of Alan Kay in the late 1960s at the University of Utah.

Alan Kay

• Kay predicted the future availability of powerful computers that would be used by nonprogrammers. Such a computer would be highly interactive and use sophisticated graphics in its user interface.

• Kay envisioned a system called Dynabook. It was based in part on the Flex language, which he had helped design. Flex was based primarily on SIMULA 67.

• Dynabook was based on the paradigm of the typical desk, on which there are a number of papers, some partially covered.

• Kay founded the Learning Research Group at the Xerox Palo Alto Research Center (Xerox PARC).

The group created an “Interim” Dynabook, consisting of a Xerox Alto workstation and the Smalltalk-72 software.
Smalltalk was improved several times in the 1970s, with Smalltalk-80 as the end result.

Smalltalk Language Overview

- The Smalltalk world is populated by nothing but objects, from integer constants to large complex software systems.

- All computing in Smalltalk is done by the same uniform technique: sending a message to an object to invoke one of its methods. The method replies by returning an object.

- Objects are instances of classes, which are similar to the classes of SIMULA 67.

- Smalltalk classes belong to a single class hierarchy.

- The syntax of Smalltalk is unlike that of most other programming languages.

Evaluation of Smalltalk

• Smalltalk has done a great deal to promote two aspects of computing:

Graphical user interfaces
Object-oriented programming

• An example Smalltalk class definition:

```
"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
"Instance methods"
  "Get a pen for drawing polygons"
  getPen
     ourPen <- Pen new defaultNib: 2

  "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
```

Combining Imperative and Object-Oriented Features: C++

• C++ was developed by Bjarne Stroustrup at Bell Laboratories as an enhancement to the C language.

Bjarne Stroustrup

• His initial modifications were done in 1980:

Parameter type checking and conversion
Classes (borrowed from SIMULA 67)
Derived classes
Public/private access control of inherited components
Constructors and destructors
Friend classes

• During 1981, more features were added:

Inline functions
Default parameters
Overloading of the assignment operator

The resulting language was called C with Classes.

• Goals of C with Classes:

Support classes and inheritance.
Have little or no performance penalty relative to C.
Could be used for every application for which C could be used.

Combining Imperative and Object-Oriented Features: C++ (Continued)

• By 1984, more features had been added:

Virtual functions
Function and operator overloading
Reference types

This version of the language was called C++.

• In 1985, the first available implementation appeared, a system named Cfront, which translated C++ programs into C programs. This version of Cfront and the version of C++ it implemented were named Release 1.0.

• Important features added to Release 2.0 (1989):

Multiple inheritance
Abstract classes

• Features added in Release 3.0 (1991):

Templates
Exception handling

• ANSI standardization of C++ began in 1990. The standard was published in 1998.

• Microsoft's .NET platform originally included a version of C++ called Managed C++. In 2005, it was replaced by C++/CLI (Common Language Infrastructure).

C++ Overview and Evaluation

• Functions are not required to belong to classes, allowing procedural as well as object-oriented programming.

• Dynamic binding in C++ is provided by virtual functions. When a virtual function is called through a pointer or reference to an object, the correct version of the function is chosen dynamically.

• C++ supports both function templates and class templates.

• Reasons for the popularity of C++:

Availability of good, inexpensive compilers.
Compatibility with C, making it relatively easy for C programmers to learn.
Ability to link C++ code with C code.
At the time it appeared, it was the only object-oriented language suitable for large projects.

• Problems of C++:

Large and complex.
Inherited most of the insecurities of C.

A Related Language: Objective-C

- Objective-C is another hybrid language with both imperative and object-oriented features.

- It was designed by Brad Cox and Tom Love in the early 1980s.

- Initially, Objective-C consisted of C plus the classes and message passing of Smalltalk.

- After Steve Jobs left Apple and founded NeXT, he licensed Objective-C, which was used to write the NeXTSTEP operating system.

- In 1996, Apple bought NeXT and used Objective-C to write MAC OS X. Objective-C later became the language used by iOS developers creating software for the iPhone and iPad.

- One characteristic that Objective-C inherited from Smalltalk is the dynamic binding of messages to objects. If an object cannot respond to a message, an exception is raised at run time.

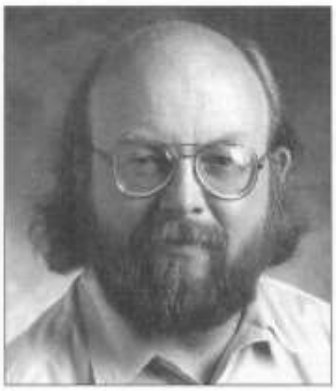
- Objective-C is a strict superset of C, so all of the insecurities of that language are present in Objective-C.

An Imperative-Based Object-Oriented Language: Java

• In 1990, Sun Microsystems examined languages for developing software for consumer electronic devices. Neither C nor C++ provided the desired combination of simplicity and reliability, so Sun developed Java (originally named Oak).

• Java was later found to be a useful tool for Web programming. In particular, Java applets became popular in the middle to late 1990s.

• The Java design team was headed by James Gosling.



James Gosling

Java Language Overview

• Java is based on C++ but was designed to be smaller, simpler, and more reliable.

• C++ features missing from Java:

Pointers
Arithmetic expressions used as control expressions
Stand-alone subprograms
Multiple inheritance
Structures
Unions

• Java allows the creation of concurrent processes, known as threads.

• In Java, all objects are heap allocated and implicitly deallocated via garbage collection. Implicit deallocation makes life easier for the programmer and avoids the problem of memory leaks.

Evaluation of Java

- The designers of Java did well at trimming the excess and/or unsafe features of C++.

- At the same time, the scope of potential applications was increased with the addition of concurrency and class libraries for graphical user interfaces, database access, and networking.

- Java's hybrid implementation makes it slower than compiled language such as C++. However, techniques such as Just-In-Time (JIT) compilation have reduced the speed penalty.

- Reasons for the rapid adoption of Java:

  Ability to support applets.
  Has much of the power of C++ but in a simpler, safer language.
  Compiler/interpreter system for Java is free and easily obtained on the Web.

- Java is now used for a broad range of applications, from writing server-side code to writing software for cell phones.

- Java 8, the most recent version of the language, appeared in 2014.

- Most releases of Java made only minor changes to the language. However, version 5.0 added several significant features, including enumerations, generics, and a new iteration construct, and version 8 added lambda expressions.

Example Java Program

```
// 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 = new 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
```

Scripting Languages

- Early scripting languages were used by putting a list of commands, called a script, in a file to be interpreted.

- The first UNIX shell language, `sh`, began as a small collection of operating system commands.

- Later shells, such as `ksh`, added variables, control flow statements, functions, and other capabilities.

- `awk`, developed at Bell Laboratories, began as a report-generation language but later became a more general-purpose language.

Origins and Characteristics of Perl

• The Perl language, developed by Larry Wall, was originally a combination of `sh` and `awk`.

• Perl is now a powerful, though still somewhat primitive, programming language.

• Variables in Perl are implicitly declared. The first character of a variable name specifies its type:

  Scalar variable names begin with `$`.
  Array names begin with `@`.
  Hash names begin with `%`.

• Perl's arrays have dynamic length and may have gaps between elements.

• Perl's associative arrays are called hashes. Hashes are hash tables in which the keys are strings.

• Perl is a powerful but somewhat dangerous language.

  A scalar variable can store a string or a number. A number may be automatically converted to a string and vice versa. If a string cannot be converted to a number, the result is zero.
  References to nonexistent array elements return `undef`, which is interpreted as zero in a numeric context.

• Perl was initially a UNIX utility for processing text files. Other common uses:

  UNIX system administration
  Computational biology
  Artificial intelligence

  Perl was once widely used for Common Gateway Interface (CGI) scripts, but is now rarely used for that purpose.

Example Perl Program

```
# Perl Example Program
# Input: An integer, $listlen, where $listlen is less
#than 100, followed by $listlen-integer values.
# Output: The number of input values that are greater than
#the average of all input values.
($sum, $result) = (0, 0);
$listlen = <STDIN>;
if (($listlen > 0) && ($listlen < 100)) {
# Read input into an array and compute the sum
  for ($counter = 0; $counter < $listlen; $counter++) {
    $intlist[$counter] = <STDIN>;
    $sum += $intlist[$counter];
  } #- end of for (counter ...
# Compute the average
  $average = $sum / $listlen;
# Count the input values that are > average
  foreach $num (@intlist) {
    if ($num > $average) { $result++; }
  } #- end of foreach $num ...
# Print result
  print "Number of values > average is: $result \n";
} #- end of if (($listlen ..
else {
  print "Error--input list length is not legal \n";
}
```

Origins and Characteristics of JavaScript

- When use of the Web exploded in the mid-1990s, new languages arose to allow computation to take place on a web server or within a browser.

- JavaScript was developed by Brendan Eich at Netscape. Originally named Mocha, the language was renamed LiveScript by the time it was released.

- In late 1995, LiveScript became a joint venture of Netscape and Sun Microsystems and its name was changed to JavaScript.

- A version of JavaScript was standardized in 1997 by the European Computer Manufacturers Association (ECMA). The standard language is known as ECMAScript.

- Microsoft's version of JavaScript is named JScript.

- JavaScript interpreters are usually included in Web browsers. JavaScript code is embedded in HTML documents and interpreted by the browser when the documents are displayed.

- Primary uses of JavaScript:

  Validating form data
  Dynamically creating and modifying HTML documents

- JavaScript is related to Java only through the use of similar syntax.

- Some major differences between JavaScript and Java:

  JavaScript is dynamically typed.
  Array subscripts are not checked in JavaScript.
  JavaScript supports neither inheritance nor dynamic binding of method calls to methods.

Example JavaScript Program

```
// example.js
//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

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 />");
```

Origins and Characteristics of PHP

- PHP was developed by Rasmus Lerdorf, a member of the Apache Group, in 1994.

- PHP was originally an acronym for Personal Home Page but later came to stand for PHP: Hypertext Preprocessor.

- PHP is an open-source product. PHP processors are resident on most Web servers.

- PHP code is interpreted on the Web server when an HTML document in which it is embedded has been requested by a browser. PHP code often produces HTML code as output, which replaces the PHP code in the HTML document.

- PHP is similar to JavaScript, in its appearance, the dynamic nature of its strings and arrays, and its use of dynamic typing.

- PHP did not originally support object-oriented programming, but that support was added in the second release.

- Advantages of PHP:

  Allows simple access to HTML form data.
  Supports many database management systems, making it a useful language for building programs that need Web access to databases.

Origins and Characteristics of Python

• Python is an interpreted scripting language designed by Guido van Rossum in the early 1990s.

• Python is being used for the same kinds of applications as Perl: system administration, CGI programming, and other relatively small tasks.

• Python characteristics:

Syntax is not based directly on any commonly used language.
Dynamically typed.
Has three kinds of data structures: lists, tuples (immutable lists), and dictionaries.
Object-oriented.
Includes pattern-matching capabilities.
Has exception handling.
Provides garbage collection.

• Python supports concurrency and network programming. It also has more support for functional programming than other nonfunctional programming languages.

Origins and Characteristics of Ruby

- Ruby was designed by Yukihiro Matsumoto in the early 1990s and released in 1996.

- Ruby is a pure object-oriented language, like Smalltalk.

- Both classes and objects are dynamic in the sense that methods can be dynamically added to either. Thus, different instances of the same class can behave differently.

- There is no need to declare variables, because dynamic typing is used.

- The scope of a variable is specified in its name:

  Local variable names begin with a letter.
  Instance variable names begin with `@`.
  Global variable names begin with `$`.

- Ruby is the first programming language designed in Japan to have achieved widespread use in the U.S.

Origins and Characteristics of Lua

- Lua (Portuguese for "moon") was designed in the early 1990s by Roberto Ierusalimschy, Waldemar Celes, and Luiz Henrique de Figueiredo at the Pontifical Catholic University of Rio de Janeiro in Brazil.

- Lua supports imperative and functional programming, with a design that was influenced by Scheme, Icon, and Python.

- Like JavaScript, Lua does not support object-oriented programming but was influenced by it.

- Like Scheme, Lua has only one primary data structure. In Lua's case, it is the table (an associative array).

- Lua uses garbage collection for its objects, which are heap-allocated.

- Like most other scripting languages, Lua is dynamically typed.

- Lua is relatively small and simple, having only 21 reserved words.

- Lua programs are compiled into intermediate code, which is then interpreted.

The Flagship .NET Language: C#

- C# is designed to be used for component-based software development in the .NET framework. Both C# and .NET were released by Microsoft in 2002.

- C# is based on C++ and Java, but includes some ideas from Delphi and Visual Basic. The lead designer of C#, Anders Hejlsberg, previously developed Turbo Pascal and Delphi.

- C# is one of several languages supported by .NET. These languages share a common class library and are compiled into the same intermediate form, called Intermediate Language (IL).

- A Just-In-Time compiler is used to translate IL into machine code before it is executed.

C# Overview

- C# restores many C++ features removed from Java, including pointers, structures, enumeration types, operator overloading, and the `goto` statement.

- In several cases, the C# version of a C++ feature has been improved.

  The `enum` types of C# are safer because they are never implicitly converted to integers.
  Structures are significantly different in C#.
  The C# `switch` statement is safer.

- C# includes a new type, delegates, which are used for implementing event handlers, controlling the execution of threads, and callbacks.

- A C# method can take a variable number of parameters of the same type if its formal parameter has an array type and is preceded by the word `params`.

- C# can implicitly convert primitive values into objects ("boxing") and vice-versa ("unboxing").

- Other features of C#:

  Rectangular arrays
  `foreach` statement
  Properties (data members with get and set methods that are implicitly called when references and assignments are made to the associated data members)

- C# has changed several times since its initial release. The most recent version is C# 5.0.

Example C# Program

```
// 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.ReadLine());
      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
```

Markup-Programming Hybrid Languages

- A markup-programming hybrid language is a markup language in which some of the elements can specify programming actions.

- XSLT and JSP are examples of such languages.

- XML documents can be transformed to HTML (for display in a browser) with another markup language, eXtensible Stylesheet Language Transformations (XSLT), which can specify programming-like operations.

- JSP (Java Server Pages) is a collection of technologies that support dynamic Web documents.