Software II: Principles of Programming Languages

Lecture 2 – A History of Programming Languages

What is a Programming Language?

- A programming language describes computation to be performed by computers.
- Programming languages have a history that parallels the development of computers and a history that is independent of computer development.
  - Konrad's Zuse's ***Plankalkül***
  - Alonzo Church's ***lambda calculus***

Programming Languages Describe Algorithms

- The need to describe calculations is ancient:

  A cistern.
  The length equals the height.
  A certain volume of dirt has been excavated.
  The cross-sectional area plus this volume comes to 120.
  The length is 5. What is the width?
  Add 1 to 5, giving 6.
  Divide 6 into 120, obtaining 20.
  Divide 5 into 20, obtaining the width, 4.
  This is the procedure.

- *Why does this work?*

The Math Behind the Description

Volume + Cross-section

$= L\ W\ H + L\ W$

$= L^2\ W + L\ W$

$= L\ W\ (\ L + 1)$

$= 5\ W\ (\ 5 + 1)$

$= 5\ W\ (\ 6) = 120$

Early History – The First Programmer

- The First Stored-Program Computer to be designed was Charles Babbage's Analytical Engine (the store-program computer to be completed was the UNIVAC in 1951).
- The first computer program to be written was by Babbage's collaborator Ada, Countess of Lovelace in the 1840s.

Zuse's Plankalkül

- Designed in 1945, but not published until 1972
  - Never implemented
  - Advanced data structures
  - Data types included floating point, arrays, records
- Invariants – expressions that would be true during execution at the points in the code where they would appear

Plankalkül Syntax

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

	|	A + 1 => A	
V	|	4 5	(subscripts)
S	|	1.n 1.n	(data type n-bit integers)

The 1950s - The First Programming Languages

Originally, all programming was done using the machine's own language, i.e., the binary code native to the CPU.

- This led to many mistakes which took a great deal of time to locate and correct.
- Eventually, programmers started using symbolic names for opcodes and operands to make it easier to program and then they would hand-translate.

The First Programming Languages (continued)

- Eventually, assemblers were written to automate the translations of these symbolic programs into machine language.
- The success of assembly languages encouraged computer scientists to develop higher-level languages which could further simplify the programming process.

Numerically-Based Languages

- Many of the earliest computers were used almost exclusively for scientific calculations and consequently many of the earliest attempts at languages were for scientific purposes.
- Mauchly's ***Short Code***, Grace Murray Hopper's ***A-0*** and John Backus's ***Speedcoding w***ere designed to compile simple arithmetic expressions.

FORTRAN

- John Backus's team at IBM developed FORTRAN (for ***FOR***mula ***TRAN***slator) in 1955-1957.
- While FORTRAN was designed for numerical computation, it included control structures, conditions and input/output.
- FORTRAN's popularity led to FORTRAN II in 1958, FORTRAN IV in 1962, leading to its standardization in 1966, with revised standards coming out in 1977 and 1990.

A Program In FORTRAN

```
C FORTRAN EXAMPLE PROGRAM
C INPUT:  AN INTEGER, LIST_LEN, WHERE LIST_LEN IS LESS
C         THAN 100, FOLLOWS BY LIST_LEN-INTEGER VALUES
C OUTPUT: THE NUMBER OF INPUT VALUES THAT ARE GREATER
C         THAN THE AVERAGE OF ALL INPUT VALUES
      DIMENSION INTLST(99)
      IMPLICIT INTEGER(A, C, R, S)
      RESULT = 0
      SUM = 0
      READ(5,501) LSTLEN
  501 FORMAT(I3)
      IF (LSTLEN) 106, 106, 101
      IF (LSTLEN - 100) 101, 106, 106
  101 CONTINUE
C READ INPUT DATA INTO AN ARRAY AND COMPUTE ITS SUM
      DO 102 COUNTR = 1, LSTLEN
      READ(5,502) INTLST(COUNTR)
  502 FORMAT(I4)
```

```
      SUM = SUM + INTLST(COUNTR)
  102 CONTINUE
C COMPUTE THE AVERAGE
      AVERGE = SUM / LSTLEN
C COUNT THE VALUES THAT ARE GREATER THAN THE AVERAGE
      DO 103 COUNTR = 1, LSTLEN
      IF (INTLST(COUNTR) - AVERGE)103, 103, 104
  104 CONTINUE
      RESULT = RESULT + 1
  103 CONTINUE
C PRINT THE RESULT
      WRITE(6,503) RESULT
  503 FORMAT(33H NUMBER OF VALUES .GT. AVERAGE IS, I2)
  106 CONTINUE
      WRITE(6,504)
  504 FORMAT(39H ERROR - LIST LENGTH VALUE IS NOT,
     1  6H LEGAL)
      RETURN
     END
```

A Program in FORTRAN IV

```
C FORTRAN EXAMPLE PROGRAM
C INPUT:  AN INTEGER, LISTLEN, WHERE LISTLEN IS LESS
C         THAN 100, FOLLOWS BY LISTLEN-INTEGER VALUES
C OUTPUT: THE NUMBER OF INPUT VALUES THAT ARE GREATER
C         THAN THE AVERAGE OF ALL INPUT VALUES
      INTEGER INTLST(99)
      INTEGER LSTLEN, COUNTR, SUM, AVERGE, RESULT
      RESULT = 0
      SUM = 0
      READ(*,*) LSTLEN
      IF ((LSTLEN .LE. 0) .OR. (LSTLEN .GE. 100))
     1       GOTO 104
C READ INPUT DATA INTO AN ARRAY AND COMPUTE ITS SUM
      DO 101 COUNTR = 1, LSTLEN
      READ(*,*) INTLST(COUNTR)
      SUM = SUM + INTLST(COUNTR)
  101 CONTINUE
```

```
C COMPUTE THE AVERAGE
      AVERGE = SUM / LSTLEN

C COUNT THE VALUES THAT ARE GREATER THAN THE AVERAGE
      DO 102 COUNTR = 1, LSTLEN
      IF (INTLST(COUNTR) .LE. AVERGE) GOTO 103
      RESULT = RESULT + 1
  103 CONTINUE
  102 CONTINUE
C PRINT THE RESULT
      WRITE(6,*) 'NUMBER OF VALUE .GT. AVERAGE IS',
     1        RESULT
  104 CONTINUE
      WRITE(6,*) 'ERROR - LIST LENGTH VALUE IS NOT',
     1     'LEGAL'
      RETURN
      END
```

A Program in FORTRAN 77

```
 Program Example

C Fortran Example program
C Input:  An integer, ListLen, where ListLen is less
C         than 100, follows 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 LISTLEN, COUNTER, SUM, AVERAGE, RESULT
      RESULT = 0
      SUM = 0
      READ *, LISTLEN
      IF ((LISTLEN .GT. 0) .AND. (LISTLEN .LT. 100)) THEN
C Read Input data into an array and compute its sum
        DO 101 COUNTER = 1, LISTLEN
          READ *, INTLIST(COUNTER)
          SUM = SUM + INTLIST(COUNTER)
  101   CONTINUE
```

```
C Compute the average
        AVERAGE = SUM / LISTLEN

C Count the values that are greater than the average
        DO 102 COUNTER = 1, LISTLEN
          IF (INTLIST(COUNTER) .GT. AVERAGE) THEN
            RESULT = RESULT + 1
          ENDIF
  102   CONTINUE
C Print the result
        PRINT *, 'Number of value .GT. Average is', Result
      ELSE
        PRINT *, 'Error - list length value is not legal'
      ENDIF
      END
```

A Program in Fortran 95

```
Program Example
! Fortran Example program
! Input:    An integer, List_Len, where List_Len
!           is less than 100, follows by
!           List_Len-Integer values
! Output:   The number of input values that are
!           greater than the average of all input
!           values
Implicit none
Integer :: Int_List(99)
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 value > Average is', Result
Else
  Print *, 'Error - list length value is not legal'
End If
End Program Example
```

ALGOL

- FORTRAN's success led to fear that IBM would dominate the computer industry.
- GAMM and ACM organized committees to design a universal language which merged and developed ALGOL 58 (which led to ALGOL 60 and ALGOL 62).
- Many later languages are derivatives of ALGOL, including PL/I, C, Pascal and Ada.

Design of ALGOL

FORTRAN had been designed to run efficiently on an IBM 701; ALGOL had been designed to meet four different goals:

- ALGOL notation should be close to standard mathematics
- ALGOL should be useful in describing algorithms
- Programs in ALGOL should be compilable into machine language.
- ALGOL should not be tied to a single computer architecture.

Influence of ALGOL

- While ALGOL saw limited use in the US (and only some in Europe), it made many contributions to other languages:
  - Backus and Naur developed the notation still used to express language syntax (BNF), based on Chomsky's context-free language concept.
  - Burrough's use of Lukasiwiez's notation for writing expressions (prefix notation) led to the use of stack-based architectures.

ALGOL-60: Example

```
procedure Absmax(a) Size:(n, m) Result:(y)
                    Subscripts:(i, k);
  value n, m; array a; integer n, m, i, k;
  real y;
comment The absolute greatest element of the
        matrix a, of size n by m is transferred
        to y, and the subscripts of this element
        to i and k;

begin
    integer p, q;
    y := 0; i := k := 1;
```

```
    for p:=1 step 1 until n do
    for q:=1 step 1 until m do
        if abs(a[p, q]) > y then
            begin y := abs(a[p, q]);
            i := p; k := q
            end
end Absmax
```

COBOL

- Commercial data processing was one of the earliest commercial applications of computers.
- The U.S. Defense Dept. sponsored the effort to develop COBOL (***Co***mmon ***B***usiness-***O***riented ***L***anguage), which was standardized in 1960, revised in 1961 & 1962, re-standarized in 1968, 1974, and 1984.
- As of 2000, more lines of source code have been written in COBOL than any other programming language.

Influence of COBOL

- Its popularity is due to:
  - self-documenting style (very English-like)
  - its record structure makes it easy to organize data
  - its PICTURE clauses made it easy to format input and output in different ways.
- COBOL has been a major influence on most database manipulation languages.

A Program in 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.
FD BAL-FWD-FILE
   LABEL RECORDS ARE STANDARD.
   RECORD CONTAINS 80 CHARACTERS.

01 BAL-FWD-CARD.
   05 BAL-ITEM-NO         PICTURE IS 9(5).
   05 BAL-ITEM-DESC       PICTURE IS X(20).
   05 FILLER              PICTURE IS X(5).
   05 BAL-UNIT-PRICE      PICTURE IS 999V99.
   05 BAL-REORDER-POINT   PICTURE IS 9(5).
   05 BAL-ON-HAND         PICTURE IS 9(5).
   05 BAL-ON-ORDER        PICTURE IS 9(5).
   05 FILLER              PICTURE IS X(30).
```

```
FD REORDER-LISTING
   LABEL RECORDS ARE STANDARD.
   RECORD CONTAINS 80 CHARACTERS.

01 REORDER-LINE.
   05 RL-ITEM-NO          PICTURE IS Z(5).
   0F FILLER              PICTURE IS X(5).
   05 RL-ITEM-DESC        PICTURE IS X(20).
   05 FILLER              PICTURE IS X(5).
   05 RL-UNIT-PRICE       PICTURE IS ZZZ.99.
   05 FILLER              PICTURE IS X(5).
   05 RL-AVAILABLE-STOCK  PICTURE IS Z(5).
   05 FILLER              PICTURE IS X(5).
   05 RL-REORDER-POINT    PICTURE IS Z(5).
   05 FILLER              PICTURE IS X(71).
```

```
WORKING-STORAGE SECTION.
01 SWITCHES.
   05 CARD-EOF-SWITCH    PICTURE X.
01 WORK-FIELDS.
   05 AVAILABLE-STOCK    PICTURE 9(5).

PROCEDURE DIVISION.
1000-PRODUCE-REORDER-LISTING.
    OPEN INPUT BAL-FWD-FILE.
    OPEN OUTPUT REORDER-LISTING.
    MOVE "N" TO CARD-EOF-SWITCH.
    PERFORM 1100-PRODUCE-REORDER-LINE
        UNTIL CARD-EOF-SWITCH IS EQUAL TO "Y".
    CLOSE BAL-FWD-FILE
    CLOSE REORDER-LISTING.
    STOP RUN.
```

```
1100-PRODUCE REORDER-LINE.
    PERFORM 1110-READ-INVENTORY-RECORD.
    IF CARD-EOF-SWITCH IS NOT EQUAL TO "Y"
        PERFORM 1120-CALCULATE-AVAILABLE-STOCK
    IF AVAILABLE-STOCK IS LESS THAN
              BAL-REORDER-POINT
        PERFORM 1130-PRINT-REORDER-LINE.

1110-READ-INVENTORY-RECORD.
    READ BAL-FWD-FILE RECORD
       AT END
          MOVE "Y" TO CARD-EOF-SWITCH.

1120-CALCULATE-AVAILABLE-STOCK.
    ADD BAL-ON-HAND BAL-ON-ORDER
        GIVING AVAILABLE-STOCK.
```

```
1130 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.
```

LISP

- John McCarthy of MIT developed LISP (LISt Processor) in the late 1950s to handle list structures in a functional format.
- Only two data types: atoms and lists
- The language pioneered garbage collection and recursive procedures; its dialects include Scheme.
- Because it is not an imperative language, it does not run as efficiently on standard computer architectures. However, there are computer architectures designed for it on which they run more efficiently.
- Many major AI programs have been written in LISP.

APL

- APL (A Programming Language) was designed by Ken Iverson to handle scientific computations, especially those involves vectors and matrices.
- It has a large set of operators and requires a special character set and is extremely difficult to read.

APL Keyboard Layout

Sample APL Program

(~R∈R∘.×R)/R←1↓⍳R

Executed from right to left, this means:

- ⍳R creates a vector containing integers from 1 to R
  - (if R = 6 at the beginning of the program, ⍳R is 1 2 3 4 5 6)
- Drop first element of this vector (↓ function), i.e. 1. So 1↓⍳R is 2 3 4 5 6
- Set R to the new vector (←, assignment primitive), i.e. 2 3 4 5 6

- Generate outer product of R multiplied by R, i.e. a matrix that is the *multiplication table* of R by R (∘.× function), i.e.
- Build a vector the same length as R with 1 in each place where the corresponding number in R is in the outer product matrix (∈, set inclusion function), i.e. 0 0 1 0 1
- Logically negate the values in the vector (change zeros to ones and ones to zeros) (~, negation function), i.e. 1 1 0 1 0
- Select the items in R for which the corresponding element is 1 (/ function), i.e. 2 3 5

The 1960s – An Explosion of Programming Languages

- The 1960s saw the development of hundreds of programming languages, many of them special-purpose languages (for tasks such as graphics and report generation)
- Other efforts went into developing a "universal programming language" that would be suitable for all programming tasks.

PL/I

- IBM developed NPL for its 360 computers (renaming NPPL and later PL/I).
- PL/I used an ALGOL-style syntax and combined featured of both FORTRAN and COBOL.
- PL/I was a complex language that saw some commercial success and its subset PL/C saw some success as a teaching language in the 1970s.

A Program in PL/I

```
/* 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;
   DCL (LISTLEN, COUNTER, SUM, AVERAGE, RESULT) FIXED;
   RESULT = 0
   SUM = 0
   GET LIST (LISTLEN);
   IF (LISTLEN > 0 & LISTLEN< 100) THEN DO;
   /*READ INPUT DATA INTO AN ARRAY AND COMPUTE ITS SUM
   */
      DO COUNTER = 1 TO LISTLEN;
        GET LIST (INTLIST(COUNTER));
        SUM = SUM + INTLIST(COUNTER);
      END;
```

```
  /* COMPUTE THE AVERAGE */
      AVERAGE = SUM / LISTLEN

  /* COUNT THE VALUES THAT ARE GREATER THAN
                THE AVERAGE */
      DO COUNTER = 1 TO LISTLEN;
        IF (INTLIST(COUNTER) > AVERAGE) THEN
          RESULT = RESULT + 1;
      END;
  /* PRINT THE RESULT */
      PUT SKIP LIST ('NUMBER OF VALUES > AVERAGE IS');
      PUT LIST (RESULT);
   ELSE
      PUT SKIP LIST('ERROR - LIST LENGTH VALUE IS NOT'
           ` LEGAL');
     END PLIEX;
```

Successes and Failures of PL/I

- The PL/I language was so complex that its compiler was huge and slow and the executable code that it created was also huge and slow.
- Because of its complexity, PL/I was a difficult language to master.
- PL/I included some concepts that were ahead of its time, such as exception handling.

SNOBOL

- SNOBOL (***S***tri***n***g ***O***riented Sym***bo***lic ***L***anguage) was develop by R. Griswold at AT&T Bell Labs.
- SNOBOL was the first string-processing language and SNOBOL4 included powerful pattern-matching capabilities.

Simula

- Simula67 was created by Kristen Nygaard and Ole-Johan Dahl at the Norwegian Computing Center in 1965-1967.
- It was designed for computer simulation and introduced to concept of the class, the basis behind object-orientation.

BASIC

- John Kemeny and Thomas Kurtz originally developed BASIC as a language to teach beginning students how to program in a more user-friendly environment.
- BASIC's original design was most heavily influenced by FORTRAN but was later expanded to include many other features.

The 1970s – Simplicity, Abstraction, Study

- Most of the new programming languages are characterized by a move toward simplicity and consistency.
- Developments included Pascal and C. Both languages became extremely popular although adding few new concepts.
- The desire to add mechanisms for data abstraction, concurrency and verification led to the development of languages such as CLU, Euclid and Mesa.

Pascal

- Niklaus Wirth and C. A. R. Hoare developed ALGOL-W and later Pascal, which had a simplified structure.
- Pascal's structure made it a great teaching langauge and a popular language for describing algorithms.
- There were several important features that it lacked including string processing, separate compilation, practical I/O facilities.

C

- Dennis Ritchie of AT&T Bell Labs created C based on earlier languages BCPL and B to use it in writing a revised version of UNIX.
- C is based heavily around the idea of an expression and allows easy conversion between its data types.
- It is regarded as a "middle-level" programming language.

Experiments In Abstraction, Concurrency and Verification

- The most notable attempts to introduce data abstraction, concurrency and verification to programming languages include:
  - CLU – designed at MIT to provide a consistent approach to abstraction mechanism.
  - Euclid – a Pascal derivative that includes abstract data types whose goal was formal verification of programs.
  - Mesa – a Pascal-like language with a module facility, exception handling and mechanisms for concurrency.

The 1980s – New Directions and the Rise of Object-Orientation

- The 1980s began with attempts to introduce ADT mechanisms (Ada and Modula-2).
- The most significant advance in programming languages during the 1980s was the rise of object orientation (Smalltalk and C++).
- Lastly, there was renewed interest in functional and procedural languages (Scheme, ML, FP and Prolog).

Ada

- Ada was designed in 1980 and standardized in 1983 by J. Ichbiah.
- It includes mechanisms of ADT (the ***package***) and concurrency (the ***task***).
- Because of the language's complexity, Ada has been frequently described as "the PL/I of the '80s."

A Program in Ada

```
-- 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 s 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 its
-- sum
     for Counter := 1 .. List_Len loop
        Get Int_List(Counter)):
        Sum = Sum + Int_List(Counter);
     end loop;
-- Compute the average
      Average = Sum / List_Len;
-- Count the values that are greater than the
-- average
      for Counter := 1 .. List_Len loop
        if Int_List(Counter) > Average then
            Result := Result + 1;
        end if;
      end loop;
```

```
-- Print the result
    Put ("The number of values > average is");
    Put (Result);
    New_Line;
  else
    Put_Line("Error – list length value is not",
        " legal");
   end if;
end Ada_Ex;
```

Modula-2

- Modula-2 was more modest in design, expanding the features of Pascal to include ***modules*** (similar to Ada's packages) and a limited form of concurrency called ***coroutines***.
- Because of its restrictive typing and the popularity of Turbo Pascal and C, it never became as popular as expected.

Smalltalk

- Smalltalk was developed at Xerox PARC by Alan Kay et. al. and is considered the purest example of an object-oriented language.
- Its limited success is dues largely to its being tied to a computer and operating system that saw limited success as well as an unusual notation and inefficient implementation.

A Program in Smalltalk

```
"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 defaultNib: 2

  "Instance methods"
  "Draw a polygon"
  draw
  numSides timeRepeat: [ourPen go: sideLength;
                        turn: 360 // numSides
"Set length of sides"
length: len
   sideLength <- len

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

C++

- C++ was developed by Bjarne Stroustrup of AT&T Bell Labs beginning in 1980 as "C with Classes." It was finally standardized in 1998.
- It is a complex language with no compiler yet conforming entirely to the 1998 standard.

Functional Languages

- Scheme was actually developed in the late 1970s but gained popularity in the 1980s due to Abelson and Sussman's book "Structure and Implementation of Computer Programs."
- ML (for ***M***eta***l***anguage) is a functional language with a Pascal-like syntax.
- John Backus developed FP (***F***unctional ***P***rogramming) that was heavily influenced by APL.

Prolog

- Prolog is a declarative language developed by Colmerauer, Roussel and Kowalski in 1972 based on predicate calculus and mathematical logic.
- Prolog became popular in the 1980s because it was useful in expert systems development and the Japanese Fifth-Generation Project.

The 1990s – Consolidation, the Internet, Libraries and Scripting

- Programming language development in the 1990s was dominated by a few factors: Java, scripting languages and greater focus on libraries.
- Haskell, a purely functional language like ML and Ada matured with the Haskell98 and Ada95 standards (which added OOP and parallelism to Ada).

Java

- Java was developed by James Gosling of Sun Microsystems for embedded systems.
- Its popularity is due to its relative simplicity, portability, large library of windowing, networking and concurrency utilities and (of course) the World Wide Web.
- Java is a proprietary language with Sun maintaining tight control over it.

Libraries

- In the past, libraries were frequently an afterthought in the design of programming languages.
- Increasingly, libraries are important to the success of programming languages, e.g.,
  - Java API
  - C++ STL

Scripting Languages

- Scripting languages became increasingly popular in the 1990s.
- A scripting language is a special-purpose language which ties together utilities, library components and operating systems commands into complete programs.
- Examples include AWK, Perl, TCL, Javascript, Rexx, and Python.

Scripting Languages for the Web

- Scripting languages that have become popular (at least in part) because of the Web include:
  - Perl
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Lua

Perl

- Designed by Larry Wall—first released in 1987
- Variables are statically typed but implicitly declared
- Three distinctive namespaces, denoted by the first character of a variable's name
- Powerful, but somewhat dangerous
- Gained widespread use for CGI programming on the Web
- Also used for a replacement for UNIX system administration language

JavaScript

- Began at Netscape, but later became a joint venture of Netscape and Sun Microsystems
- A client-side HTML-embedded scripting language, often used to create dynamic HTML documents
- Purely interpreted
- Related to Java only through similar syntax

PHP

- PHP: Hypertext Preprocessor, designed by Rasmus Lerdorf
- A server-side HTML-embedded scripting language, often used for form processing and database access through the Web
- Purely interpreted

Python

- An OO interpreted scripting language
- Type checked but dynamically typed
- Used for CGI programming and form processing
- Dynamically typed, but type checked
- Supports lists, tuples, and hashes

Ruby

- Designed in Japan by Yukihiro Matsumoto (a.k.a, “Matz”)
- Began as a replacement for Perl and Python
- A pure object-oriented scripting language
  - All data are objects
- Most operators are implemented as methods, which can be redefined by user code
- Purely interpreted

Lua

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

The Flagship .NET Language: C#

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

Markup/Programming Hybrid Languages

- These include
  - XSLT (eXtensible Stylesheet Language Transformation)
  - JSP (Java Servlet Pages)

XSLT

- eXtensible Markup Language (XML): a metamarkup language
- eXtensible Stylesheet Language Transformation (XSTL) transforms XML documents for display
- Programming constructs (e.g., looping)

JSP

- Java Server Pages: a collection of technologies to support dynamic Web documents
- JSTL, a JSP library, includes programming constructs in the form of HTML elements

The Future

- The past demonstrated that it is extremely difficult to predict the direction of future programming language development.