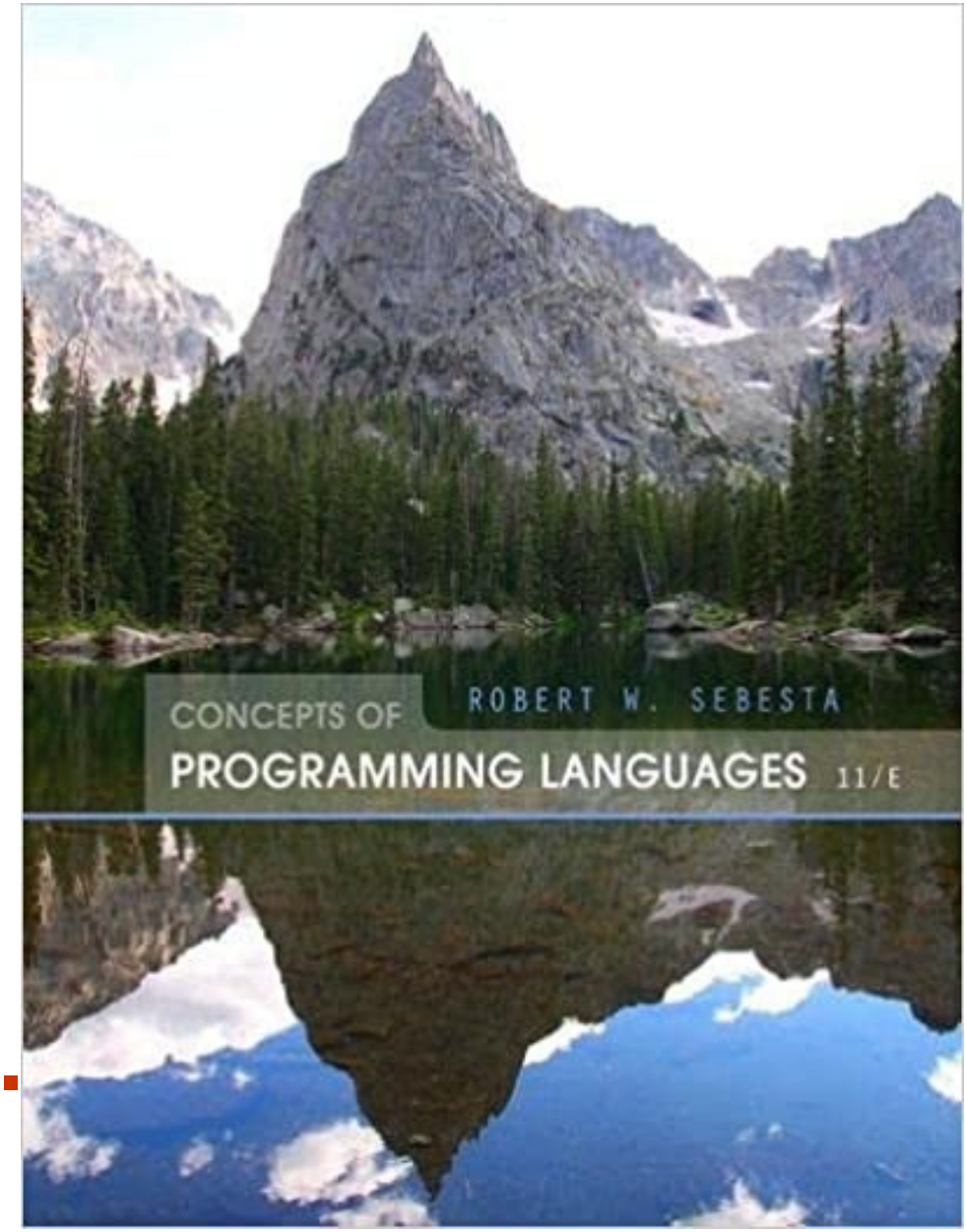
Chapter 2

Evolution of the Major Programming Languages

COMP333: Concepts of Programming Languages

amran Zamanifar, PhD.



Chapter 2 Topics

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

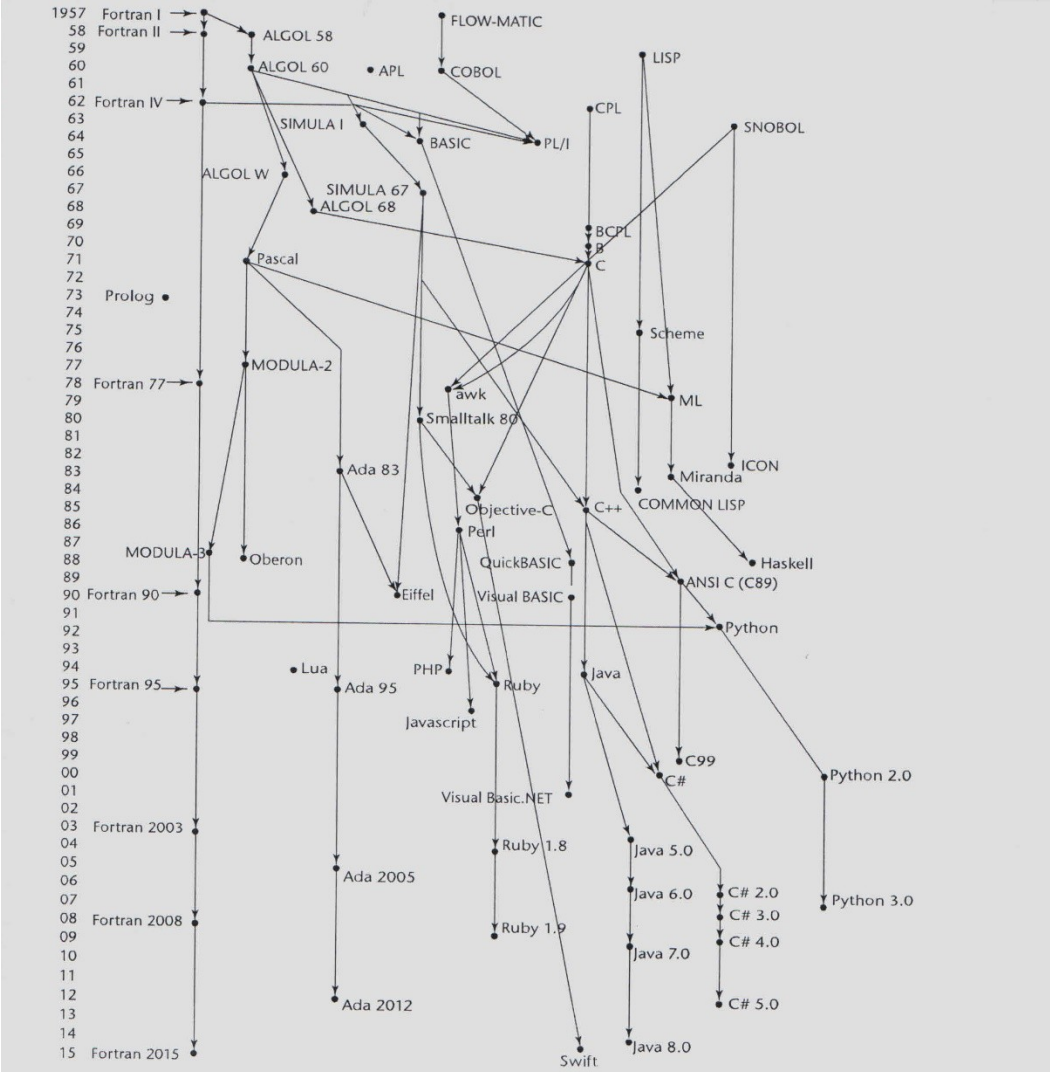
Chapter 2 Topics (continued)

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

Chapter 2 Topics (continued)

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

Genealogy of Common Languages



Zuse's Plankalkül

- Designed in 1945, but not published until 1972
- Never implemented
- Advanced data structures
  - floating point, arrays, records.
  - The records could include nested records.
  - It did include an iterative statement similar to the Ada **for**. It also had the command **Fin** with a superscript that specified an exit out of a given number of iteration loop nesting.

Zuse's Plankalkül Syntax

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

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

Minimal Hardware Programming: Pseudocodes

- First, note that the word pseudocode is used here in a different sense than its contemporary meaning.

- What was wrong with using machine code?
  - Numeric codes for specifying instructions
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies: no indexing or floating point

Pseudocodes: Short Code

- Short Code developed by Mauchly in 1949 for BINAC computers
- Short Code was later transferred to a UNIVAC I computer (**the first commercial electronic computer sold in the United States**)
  - Expressions were coded, left to right
  - Variables were named with byte-pair codes, as were locations to be used as constants.
  - For example, X0 and Y0 could be variables. The statement X0 = SQRT(ABS(Y0)) would be coded in a word as 00 X0 03 20 06 Y0.

Pseudocodes: Speedcoding

- **Speedcoding** developed by Backus in 1954 for IBM 701
  - The system included pseudo instructions for the four **arithmetic operations on floating-point data**, as well as operations such as **square root, sine, arc tangent, exponent, and logarithm**
  - Conditional and unconditional branching
  - Auto-increment registers for array access
  - Slow! : the add instruction took 4.2 milliseconds to execute
  - Only 700 words left for user program
  - Backus claimed that problems that could take two weeks to program in machine code could be programmed in a few hours using Speedcoding

Pseudocodes: Related Systems

- The UNIVAC Compiling System
  - Developed by a team led by Grace Hopper
  - Pseudocode expanded into machine code

- David J. Wheeler (Cambridge University)
  - developed a method of using blocks of re-locatable addresses to solve the problem of absolute addressing

IBM 704 and Fortran

- Introduction of the IBM 704 in 1954 was one of the greatest single advances in computing, because its capabilities prompted the development of Fortran.
- Fortran 0: 1954 - not implemented
- Fortran I:1957
  - Designed for the new IBM 704, which had index registers and **floating point hardware**

    - This led to the idea of compiled programming languages, because there was no place to hide the cost of interpretation (no floating-point software)

  - Environment of development
    - Computers had small memories and were unreliable
    - Applications were scientific
    - No programming methodology or tools
    - Machine efficiency was the most important concern

Design Process of Fortran

- The first widely accepted compiled high level language was Fortran I
- Impact of environment on design of Fortran I
  - No need for dynamic storage
  - Need good array handling and counting loops
  - No string handling, decimal arithmetic, or powerful input/output (for business software)

Fortran I Overview

- First implemented version of Fortran
  - Names could have up to six characters
  - DO loop statement
  - Formatted I/O
  - User-defined subprograms
  - Three-way selection statement (arithmetic **IF**)
  - No data typing statements
  - Variables whose names began with I, J, K, L, M, and N were implicitly integer type, and all others were implicitly floating-point.

Fortran I Overview (continued)

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

Fortran II

- Lunched in 1958
  - Independent compilation of subroutines
  - The capability of including precompiled machine language versions of subprograms shortened the compilation process considerably and made it practical to develop much larger programs.
  - Fixed the Fortran I's bugs

Fortran IV

- Evolved during 1960-62
- Fortran IV became one of the most widely used programming languages of its time
  - Logical If construct
  - Explicit type declarations
  - The capability of passing subprograms as parameters to other subprograms.
  - ANSI standard in 1966

Fortran 77

- Became the new standard in 1978
- Fortran 77 retained most of the features of Fortran IV
  - Character string handling
  - Logical loop control statement
  - IF-THEN-ELSE statement

Fortran 90

- Most significant changes from Fortran 77
  - Multiple selection statement, and modules
  - Dynamic arrays
  - Pointers
  - Subprograms could be recursively called.
  - **`CASE`** statement
  - Parameter type checking
  - A new concept that was included in the Fortran 90 definition was that of removing some language features from earlier versions
  - The official spelling of **FORTRAN became Fortran**.The new convention was that only **the first letter of keywords and identifiers would be uppercase.**

Latest versions of Fortran

- **Fortran 95** – relatively minor additions, plus some deletions. A new iteration construct, **Forall**, was added to ease the task of **parallelizing** Fortran programs.
- **Fortran 2003** – support for: **Object Oriented Programing** , procedure pointers, and interoperability with C and C++
- **Fortran 2008** – blocks for local scopes, co-arrays, **DO CONCURRENT construct**, to specify loops without interdependencies
- With the launch of different types of Fortran, parallel languages and High Performance Computing (HPF) were raised.

The following is an example of a Fortran 95 program:

```
! 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
```

Fortran Evaluation

- Highly optimizing compilers (all versions before 90)
  - Types and storage of all variables are fixed before run time
- Dramatically changed forever the way computers are used
- A powerful parallel language for parallel computing
- **Fortran created a turning point in computing**

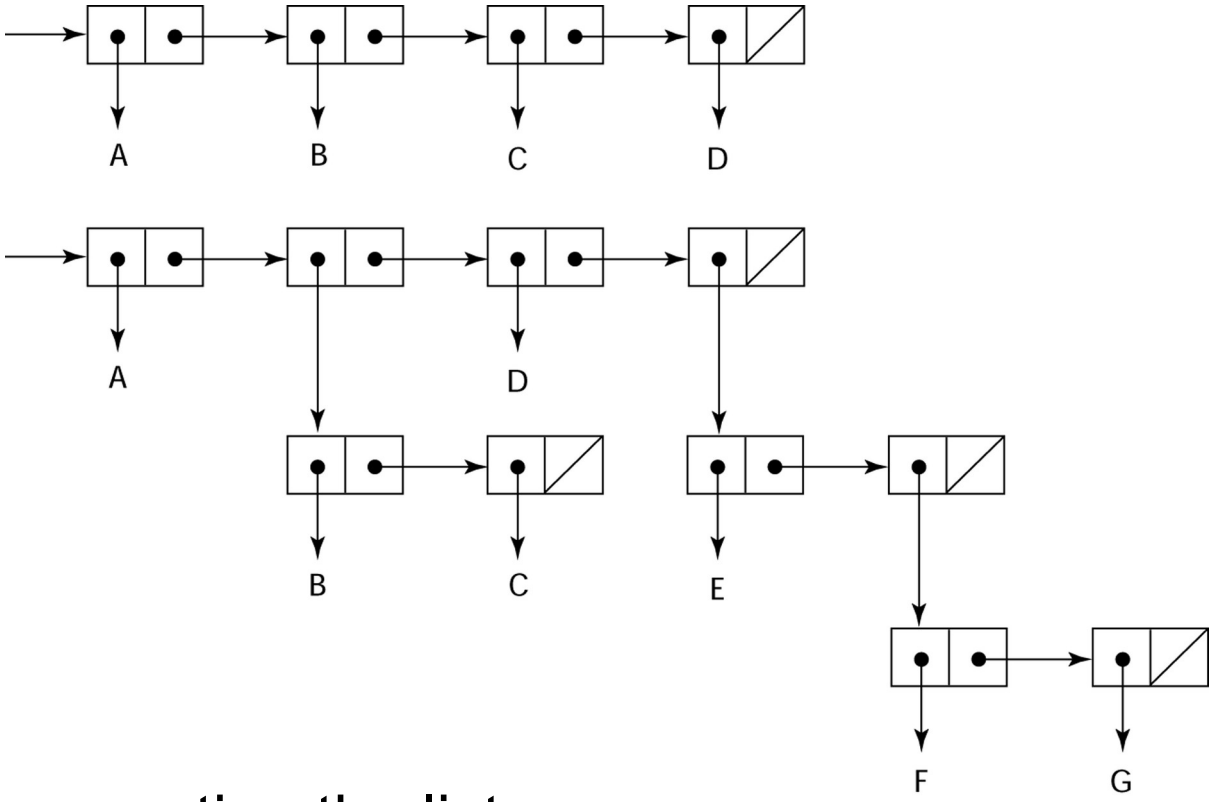
Functional Programming: Lisp

- Interest in AI appeared in the mid-1950s in a number of places. Some of this interest grew out of linguistics, some from psychology, and some from mathematics.
- **Linguists** were concerned with natural language processing.
- **Psychologists** were interested in modeling human information storage and retrieval, as well as other fundamental processes of the brain.
- **Mathematicians** were interested in mechanizing certain intelligent processes, such as theorem proving.
- Some method must be developed to allow computers to process **symbolic data in linked lists**. At the time, most computation was on numeric data in arrays.
- **Fortran List Processing Language (FLPL)** was designed and implemented as an extension to Fortran for list processing.

Functional Programming: Lisp

- Because FLPL did not support recursion, conditional expressions, dynamic storage allocation, or implicit deallocation, it was clear that a new language was needed.
- The first functional programming language was invented to provide language features for **list processing**
- Processing language
  - Designed at MIT by McCarthy
- AI research needed a language to
  - Process data in **lists** (rather than arrays)
  - **Symbolic computation** (rather than numeric)
- Pure LISP has only two data types: atoms and lists - Atoms are either symbols, or numeric literals. - Lists are specified by delimiting their elements with parentheses. (A B C D)
- The internal representations of the two lists are depicted in Figure 2.2.

Fig 2.2: Representation of Two Lisp Lists



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

Processes in Functional Programming: LISP

- LISP was designed as a **functional programming language.**

- All computation in a purely functional program is accomplished by **applying functions to arguments**.

- **Neither the assignment statements nor the variables** that abound in imperative language programs are necessary in functional language programs.

- Repetitive processes can be specified with **recursive function calls**, making iteration (loops) unnecessary.

- These basic concepts of functional programming make it significantly different from programming in an imperative language

The Syntax of LISP

- LISP is very different from the imperative languages. Because it is a functional programming language and because the appearance of LISP programs is so different from those in languages like Java or C++.

- The syntax of Java is a complicated **mixture of English and algebra** While the LISP's syntax is a model of simplicity. Program code and data have exactly the same form: **parenthesized lists**.

- When the list (A B C D) 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.

An example of a LISP program:

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

Lisp Evaluation

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

Scheme

- Developed at MIT in mid 1970s
- Small size
- Extensive use of static scoping
- Functions as first-class entities
- As first-class entities, Scheme functions can be assigned to variables, passed as parameters, and returned as the values of function applications.
- Simple syntax (and small size) make it ideal for educational applications

Common Lisp

- An effort to combine features of several dialects of Lisp into a single language early 1980s, including Scheme.
- Common LISP is a relatively large and complex language. Its basis, however, is pure LISP, so its syntax, primitive functions, and fundamental nature come from that language.
- Common LISP allows both static scoping and dynamic scoping.
- Common LISP has a large number of data types and structures, including **records, arrays, complex numbers, and character strings**
- Large, complex, used in industry for some large applications

Related Languages

- ML was originally designed as a **metalanguage** for a program verification system named **Logic for Computable Functions**
- ML is primarily a functional language, but it also supports imperative programming.
- Unlike LISP and Scheme, ML does not use the parenthesized functional syntax.
- The syntax of ML resembles that of the imperative languages, such as Java and C++

- F# is a relatively new typed language, is a .NET language with direct access to the whole .NET library.
- Being a .NET language also means it can smoothly interoperate with any other .NET language.
- **F# supports both functional programming and procedural programming. It also fully supports object-oriented programming**.

The First Step Toward Sophistication: ALGOL 60

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

Early Design Process

- GAMM and ACM each sent four members to the first design meeting. The meeting, which was held in Zurich from May 27 to June 1, 1958, began with the following goals for the new language:

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

ALGOL 58

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

ALGOL 58 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)

- Although IBM was initially enthusiastic to implement ALGOL 58, but all support was dropped by mid 1959

ALGOL 60 Overview

- In January 1960, the second ALGOL meeting was held, this time in Paris. The purpose of the meeting was to debate the 80 suggestions that had been formally submitted for consideration.
- ALGOL 58 was modified at 6-day meeting in Paris
- New features
  - The concept of block structure was introduced.
  - Two different means of passing parameters to subprograms were allowed: pass by value and pass by name
  - Procedures were allowed to be recursive
  - Stack-dynamic arrays were allowed. A stack-dynamic array is one for which the subscript range or ranges are specified by variables, so that the size of the array is set at the time storage is allocated to the array

ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - The only acceptable formal means of communicating algorithms in computing literature
  - All subsequent imperative languages are based on it
  - First machine-independent language
  - It was also the first language whose syntax was formally described by BNF formalism
  - This successful use of the BNF formalism initiated several important fields of computer science: formal languages, parsing theory, and BNF-based compiler design(chapters 3 and 4)

ALGOL 60 Evaluation (continued)

- Failure
  - Never widely used, especially in U.S.
  - Reasons
    - Lack of I/O statements and the character set made programs non-portable
    - Too flexible (high level language) --hard to implement
    - Establishment and acceptability of Fortran among users
    - Formal syntax description
    - **Lack of support from IBM**

An example of an 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
COmmon Business Oriented Language

- COBOL's capabilities meet the needs of its application in business computing

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

- Environment of development
  - **UNIVAC** was beginning to use FLOW-MATIC business language
  - **USA Air Force** was beginning to use AIMACO language, a minor variation of FLOW-MATIC
  - **IBM** was developing COMTRAN (COMmercial TRANslator) language

COBOL Historical Background

- Based on FLOW-MATIC business language

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

  For example the following verbs were used: read, select, open, reorder, calculate, move, and …

COBOL Design Process

- First design meeting by Pentagon after the Zurich ALGOL meeting- May 1959
- Design goals
  - Must look like simple English
  - Must be easy to use, even if that means it will be less powerful
  - Must broaden the base of those who could program computers (easy programming)
  - Must not be restricted by current compiler problems
- Design committee members were all from computer manufacturers and Department of Defense (DoD) branches
- Design Problems: arithmetic expressions? subscripts? Fights among manufacturers

COBOL Evaluation

- Contributions
  - There were early decisions to separate the statements of the language into two categories—data description and executable operations—and to have statements in these two categories be in different parts of programs.
  - First macro facility in a high-level language. For example, the **DEFINE verb of COBOL 60 was the first high-level language construct for macros**.
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with hyphens
  - Separate data division

COBOL: DoD Influence

- First language required by DoD
  - would have failed without DoD
- Still the most widely used business applications language
- An example of COBOL program is shown at your text book. This program reads a file named BAL-FWD-FILE that contains inventory information about a certain collection of items. The program produces a list of items that must be reordered as a file named REORDER-LISTING.

The Beginning of Timesharing: Basic

- BASIC (Beginner's All-purpose Symbolic Instruction Code) was originally designed at Dartmouth College in New Hampshire by two mathematicians, John Kemeny and Thomas Kurtz for art students.
- Design Goals:
  - Easy to learn and use for non-science students
  - Must be "pleasant and friendly"
  - Fast turnaround for homework
  - Free and private access (through terminals)
  - User time is more important than computer time
- First widely used language with time sharing. Many students could run their home work concurrently on just one server through terminals.

Beginner's All-purpose Symbolic Instruction Code: Basic

- Basic was originally designed at Dartmouth College for liberal arts students
- The mail goal of this language was: **user time more important than computer time**. **This goal was indeed a revolutionary concept**
- The resurgence of BASIC in the 1990s was driven by the appearance of Visual BASIC (VB).

- VB became widely used in large part because it provided a simple way of building graphical user interfaces (GUIs), hence the name Visual BASIC.

- Visual Basic .NET, or just VB.NET, is one of Microsoft's .NET languages

- The most important difference between VB and VB.NET is that VB.NET fully supports object-oriented programming.

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

PL/I: Background

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

PL/I: Design Process

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

PL/I: Evaluation

- PL/I contributions
  - First unit-level **concurrency**
  - First **exception handling**
  - Switch-**selectable recursion**
  - First **pointer data type**
  - **Cross-sections of arrays** could be referenced. For example, the third row of a matrix could be referenced as if it were a single-dimensioned array
- Concerns
  - Many new features were poorly designed
  - The compiler was too large and too complex

Two Early Dynamic Languages: APL and SNOBOL

- In appearance and in purpose APL and SNOBOL are very different. Neither APL nor SNOBOL had much influence on later mainstream languages
- They share two fundamental characteristics: **dynamic typing** and **dynamic storage** allocation
- Variables are untyped
  - A variable acquires a type when it is assigned a value
  - Storage is allocated to a variable when it is assigned a value

APL: A Programming Language

- Designed as a hardware description language at IBM by Ken Iverson around 1960
- APL has a large number of powerful operators that are specified with a large number of symbols, which created a problem for implementor
  - **Highly expressive** (many operators, for both scalars and arrays of various dimensions)
  - **Programs are very difficult to read**
- Still in use in some specific domains; minimal changes

SNOBOL

- It was designed specifically for text processing
- Designed as a **string manipulation language** at Bell Labs by Farber, Griswold, and Polensky in 1964

- **Powerful operators for string pattern matching**
- **One of the early applications of SNOBOL was for writing text editors**
- Slower than alternative languages
- Still used for a variety of text-processing tasks in several different application areas.

The Beginning of Data Abstraction: SIMULA 67

- Designed primarily for system **simulation** in Norway by Nygaard and Dahl
- Based on ALGOL 60 and SIMULA I
- Simulation requires subprograms that are allowed to restart at the position where they previously stopped.
- Subprograms with this kind of control are known as **coroutines** because the caller and called subprograms have a somewhat equal relationship with each other, rather than the rigid master/slave relationship they have in most imperative languages
- To provide support for coroutines in SIMULA 67, the **class construct** was developed.
- Primary Contributions
  - Coroutines - a kind of subprogram
  - **Classes, objects, and inheritance**

Orthogonal Design: ALGOL 68

- ALGOL 68 (van Wijngaarden et al., 1969), was dramatically different from its predecessor, ALGOL 60

- **Source of several new ideas** (even though the language itself never achieved widespread use)

- Design is based on the concept of **orthogonality**
  - The approach of ALGOL 68 to data structures was to provide a few primitive types and structures and allow the user to combine those primitives into a large number of different structures. **A few basic concepts, plus a few combining mechanisms**
  - **User-defined data types** are valuable because they allow the user to design data abstractions that fit particular problems very closely.

ALGOL 68 Evaluation

- Contributions
  - User-**defined data structures**
  - **Reference types**
  - Assignments to a dynamic array cause allocation of required storage
  - Introduced **Dynamic arrays** (called flex arrays),

  A dynamic array is one in which the declaration does not subscript bounds.

- Comments
  - Less usage than ALGOL 60
  - **Had strong influence on subsequent languages, especially Pascal, C, and Ada**

Some early dependents of ALGOLS: PASCAL

- Developed by Wirth (a former member of the ALGOL 68 committee)
- Designed for **teaching structured programming**
- It lacks several features that are essential for many kinds of applications. The best example of this is the impossibility of writing a subprogram that takes as a parameter an array of variable length
- Pascal's popularity, was based primarily on its remarkable combination of **simplicity** and **expressivity.**
- **Largest impact was on teaching programming**
  - **From mid-1970s until the late 1990s, it was the most widely used language for teaching programming**

A Portable Systems Language: C

- Designed for systems programming (at Bell Labs by Dennis Richie)
- Evolved primarily from BCLP and B, but also ALGOL 68
- Powerful set of operators, but poor type checking

  -Being untyped means that all data are considered machine words, which, although simple, leads to many complications and insecurities. For example, there is the problem of specifying floating-point rather than integer arithmetic in an expression

- Initially spread through UNIX operating systems
- Though designed as a systems language, it has been used in many application areas

- ANSI C was updated in 1999 (ISO, 1999)

C Evaluation

- C has adequate **control statements and data-structuring** facilities to allow its use in many application areas.
- Supports **for** and **switch** statements
- Supports **pointers:** Execution time with pointers is faster because data are manipulated with the address. Memory is accessed efficiently with the pointers. The pointer assigns and releases the memory as well
- Has a rich set of operators that provide a **high degree of expressiveness**
- One of the most important reasons why C is both liked and disliked is its **lack of complete type checking**.
  - Flexibility
- A major reason for its great increase in **popularity** in the 1980s was that a compiler for it was part of the widely used UNIX operating system.

Example for C Code

```
/* 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[counter] > average) result++;
/* Print result */
    printf("Number of values > average is:%d\n", result);
   }
    else
    printf("Error—input list length is not legal\n");
}
```

Programming Based on Logic: Prolog

- Developed, by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski (University of Edinburgh)
- **Based on formal logic: two kinds of statements, facts and rules**
- Non-procedural
- Prolog has only a few kinds of statements, but they can be complex.
- Can be summarized as being an intelligent database system that is used for finding the truth of a given queries.
- Comparatively inefficient
- Few application areas

Prolog Language Overview

- One common use of Prolog is as a **kind of intelligent database**. This application provides a simple framework for discussing the Prolog language.
- The database of a Prolog program consists of two kinds of statements: **facts** and **rules**. The following are examples of **fact statements**:
- mother (joanne, jake). father(vern, joanne). These state that joanne is the mother of jake, and vern is the father of joanne.
- An example of a **rule statement** is -
  grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

  --- This states that it can be deduced that X is the grandparent of Z if it is true that X is the parent of Y and Y is the parent of Z, for some specific values for the variables X, Y, and Z
- **It is effective for certain kinds of database management systems and some areas of AI**.

History's Largest Design Effort: Ada

- The Ada language was developed for the Department of Defense (DoD)
- **Huge design effort**, involving hundreds of people, much money, and about eight years
- By 1974, over half of the applications of computers in DoD were embedded systems.
- Named Ada after **Augusta Ada Byron**, the first programmer
- A revised version of the language design was completed in July 1980 and was accepted as the standard Ada Language Reference Manual.
- Another revised version of the Ada Language Reference Manual was released in July 1982. In 1983, the American National Standards Institute standardized Ada.
- The Ada language design was then frozen for a minimum of five years.

Ada Evaluation

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

Ada 95 and Ada 2005

- **Ada 95** (began in 1988)
  - Support for object oriented programing through type derivation
  - Better control mechanisms for shared data through the rendezvous mechanism
  - New concurrency features
  - More flexible libraries
- **Ada 2005**
  - interfaces, similar to those of Java, more control of scheduling algorithms, and synchronized interfaces
- Popularity suffered because the DoD no longer requires its use, but also because of popularity of C++, and the popularity of HPF for parallel programming.

Object-Oriented Programming: Smalltalk

- Developed at Xerox Palo Alto Research Center (Xerox PARC), initially by Alan Kay, later by Adele Goldberg
- **First full implementation of an object-oriented language** (data abstraction, inheritance, and dynamic binding)
- **Pioneered the graphical user interface design**
- The syntax of Smalltalk is unlike that of most other programming language, in large part because of the use of messages, rather than arithmetic and logic expressions and conventional control statements
- Smalltalk has done a great deal to promote two separate aspects of computing: **graphical user interfaces** and **object-oriented programming**.
- The **windowing systems** that are now the dominant method of user interfaces to software systems grew out of Smalltalk.

Combining Imperative and Object-Oriented Programming: C++

- Developed at Bell Labs by Stroustrup in 1980
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67
- A large and complex language, in part because it supports both procedural and object programing programming.
- C++ has both functions and methods grew in popularity, along with OOP
- Both methods and classes can be templated, which means that they can be parameterized. For example, a method can be written as a templated method to allow it to have versions for a variety of parameter types. Classes enjoy the same flexibility
- C++ supports multiple inheritance. It also includes exception handling
- ANSI standard approved in November 1997
- Microsoft released its .NET computing platform, which included a new version of C++, named Managed C++, or MC++

Related Languages: Objective-C & Delphi

- **Objective-C** (Kochan, 2009) is another **hybrid** language with both imperative and object-oriented features.
- After Steve Jobs founded NeXT, he licensed Objective- C and it was used to write the NeXT computer system software
- Apple bought NeXT and used Objective-C to write **MAC OS X.**
- Objective-C is the language of all **iPhone software**
- One characteristic that Objective-C inherited from Smalltalk is the **dynamic binding of messages to objects**.

Delphi

- A **hybrid** language, like C++
  - Began as an object-oriented version of Pascal
  - Designed by Anders Hejlsberg, who also designed Turbo Pascal and C#
  - Delphi, like Visual C++, provides a **graphical user interface (GUI)** to the developers

An Imperative-Based Object-Oriented Language: Java

- Developed at Sun Microsystems in the early 1990s
  - C and C++ were not satisfactory for embedded electronic devices
- In 1990, there was a need for a programming language for embedded consumer electronic devices, such as toasters, microwave ovens, and interactive TV
  - — - Simplicity and Reliability
- Java was found to be a useful tool for Web programming. In particular, Java applets, which are relatively small Java programs that are interpreted in Web browsers and whose output can be included in displayed Web documents.
- Based on C++
  - Significantly simplified (does not include **struct, union, enum**, pointer arithmetic, and half of the assignment coercions of C++)
  - Java provides much of the power and flexibility of C++, but in a smaller, simpler, and safer language.
  - Supports ***only*** **object oriented programming**
  - Has references, but not pointers
  - Includes **support for applets and a form of concurrency**

Java Evaluation

- Eliminated many unsafe features of C++, e.g., pointers
- **Supports concurrency** (using threads), and concurrency control through its synchronized modifier
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Widely used for Web programming
- Use increased faster than any previous language
- Java uses implicit storage deallocation for its objects, often called garbage collection
- Most recent version, 8, released in 2014
- Java is now widely used in a variety of different applications areas.

Scripting Languages for the Web : Perl

- **Early scripting languages** were used by putting a list of commands, called a script, in a file to be interpreted.
- 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
- Perl's arrays , first have dynamic length, meaning that they can grow and shrink as needed during execution, second, **arrays can be sparse**, meaning that there can be gaps between the elements.
- Powerful
- Gained widespread use for **Common Gateway Interface language (CGI)** programming on the Web
- Also used for a replacement for UNIX system administration language
- **Array indexing cannot be checked**, because there is no set subscript range for any array.
- So, there is also **no error detection in array element access**.
- Perl is used as a general-purpose language for a variety of applications, such as computational biology and artificial intelligence

Scripting Languages for the Web : JavaScript

- JavaScript most common use is embedded in Web browsers.
- **JavaScript code is embedded in HTML documents** and interpreted by the browser when the documents are displayed
- Began at **Netscape**, but later became a joint work of Netscape and Sun Microsystems
- JavaScript is related to Java only through the use of similar syntax.
- Java is strongly typed, but **JavaScript is dynamically typed**
- JavaScript's character strings and its arrays have **dynamic length**.
- Java fully supports object-oriented programming, but
- JavaScript supports neither inheritance nor dynamic binding of method calls to methods.
- A client-side HTML-embedded scripting language, often used to create dynamic HTML documents
  - Purely interpreted

Scripting Languages for the Web : PHP

- Its initial motivation was to provide a tool to help track visitors to personal Web site. It was developed as a package called **Personal Home Page** Tools, 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
- PHP code usually produces HTML code as output
- PHP is similar to JavaScript, in its syntactic appearance, **the dynamic nature of its strings and arrays, and its use of dynamic typing**.

- PHP allows simple access to HTML form data, so **form processing is easy with PHP**.
- PHP provides **support for many different database management systems**

Scripting Languages for the Web: Python

- An object oriented programming **interpreted scripting language**
- Python is being used for the same kinds of applications as Perl: **system administration, CGI programming,** and other relatively small computing tasks.

- It is type checked but **dynamically typed, used for form processing**
- Python's syntax is not based directly on any commonly used language.

- Python includes **three kinds of data structures: lists; immutable lists, which are called tuples; and hashes, which are called dictionaries**.

- There is a collection of list methods, such as append, insert, remove, and sort, as well as a collection of methods for dictionaries. **Supports lists, tuples, and hashes**
- Python includes **support for concurrency with its threads**

Scripting Languages for the Web: Ruby

- Designed in **Japan** by Yukihiro Matsumoto
- 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**
- **Both classes and objects in Ruby are dynamic** in the sense that methods can be dynamically added to either.
- **The syntax of Ruby is related to that of Eiffel and Ada**. There is no need to declare variables, because dynamic typing is used Ruby
- Ruby is culturally interesting because it is the first programming language designed in Japan that has achieved relatively widespread use in the United States.

The Flagship .NET Language: C#

- Part of the Microsoft .NET development platform (2000)
- **Based on C++ , Java, Delphi and Visual Basic**
- **The purpose of C# is to provide a language for component-based software development**, specifically for such development in the .NET Framework. In this environment, components from a variety of languages can be easily combined to form systems
- C# **supports multiple inheritance, pointers, structs, enum types, operator** overloading, **and a goto statement**, a **limited kind of dynamic typing**
- In C#, methods can take a variable number of parameters. This is specified by the use of a **formal parameter of array type, preceded by the params reserved word.**
- C# includes a new type, **delegates**, which are both object-oriented and type-safe method references to subprograms
- Among the other features of C# are **rectangular arrays**, and a **foreach statement, which is an iterator for arrays and collection objects**

Markup/Programming Hybrid Languages

- eXtensible Markup Language (XML) is a meta markup language.
- **XML-derived markup languages are used to define data documents** which are called XML documents
- Although XML **documents are human readable**, they are processed by computers. This processing sometimes consists only of **transformations to forms that can be effectively displayed or printed.**

- In many cases transformations are to HTML, which can be displayed by a Web browser.

- **eXtensible Style sheet Language Transformation (XSTL**) transforms XML documents to HTML documents
- **XSLT** is a **markup/programming hybrid language**.
- XSLT was defined by the World Wide Web Consortium (W3C) in the late 1990s.

- XSLT **also has programming constructs at a lower level**. For example:
  looping construct is included, which allows repeated parts of the XML document to be selected.
- There is also a sort process.
- These **lower-level constructs are specified with XSLT tags, such as <for-each>.**

Markup/Programming Hybrid Languages: JSP

- JSP is a collection of technologies designed to support **dynamic Web documents.**
- When a JSP document, **which is often a mixture of HTML and Java**, is requested by a browser, the JSP processor program, which resides on a Web server system, converts the document to a **servlet**.
- The servlet produced by the JSP processor is run by the servlet container
- **Servlets are commonly used for form processing and for database access**.
- **A servlet is an instance of a Java class** that resides on and is executed on a Web server system.
- **JSTL**, **a JSP library**, **includes programming constructs in the form of HTML elements**
- **The JSTL defines a collection of XML action elements that control the processing of the JSP document on the Web server.**

Summary

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