Chapter 2

Evolution of the Major Programming Languages

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

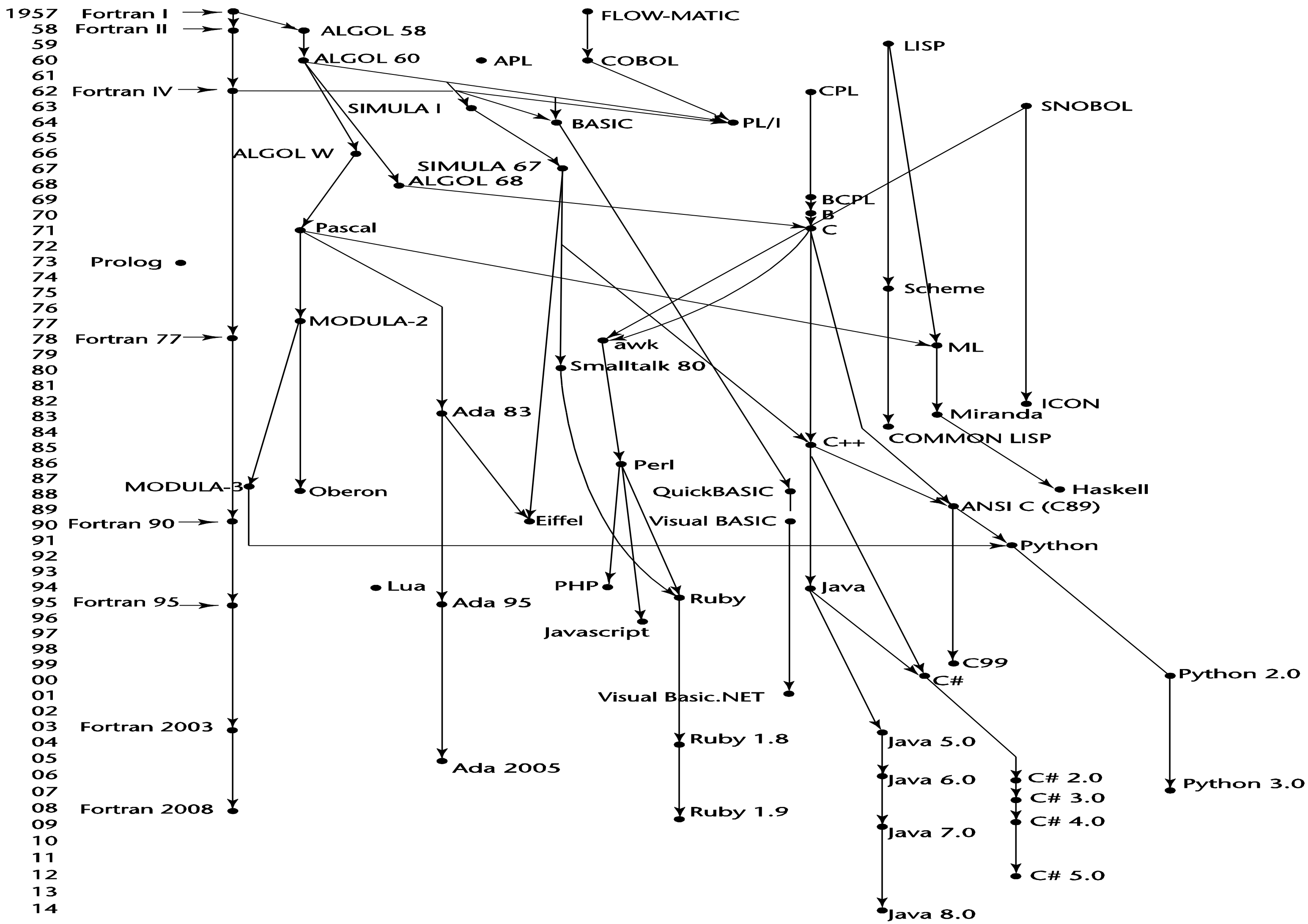
Chapter 2

- This chapter describes the development of a collection of programming languages.
- It explores the environment in which each was designed and focuses on the contributions of the language and the motivation for its development.
- Overall language descriptions are not included; rather, we discuss only some of the new features introduced by each language.
- This chapter does not include an in-depth discussion of any language feature or concept; that is left for later chapters.

Chapter 2

- This chapter discusses a wide variety of languages and language concepts that will not be familiar to many readers.
- These topics are discussed in detail only in later chapters.

Genealogy of Common Languages



Zuse's Plankalkül

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

Minimal Hardware Programming: Pseudocodes

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

Pseudocodes: Short Code

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

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

Pseudocodes: Speedcoding

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

Pseudocodes: Related Systems

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

IBM 704 and Fortran

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

Design Process of Fortran

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

Fortran I Overview

- First implemented version of Fortran
  - Names could have up to six characters
  - Post-test counting loop (**`DO`**)
    - Ex: Do N1 C = 1,10
  - Formatted I/O
  - User-defined subprograms
  - Three-way selection statement (arithmetic **`IF`**)
    - Ex: If(Expr) N1, N2, N3
  - 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

- The Fortran II compiler was distributed in the spring of 1958.
  - Independent compilation
  - placed a practical restriction on the length of programs to about 300 to 400 lines
  - Fixed the bugs

Fortran IV

- A Fortran III was developed, but it was never widely distributed.
- Evolved during 1960–62
- Fortran IV was an improvement over Fortran II in many ways
  - Explicit type declarations
  - Logical selection statement
    - a logical If construct
  - Subprogram names could be parameters
  - ANSI standard in 1966

Fortran 77

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

Fortran 90

- While Fortran 90 included all of the features of Fortran 77.
- Most significant changes from Fortran 77
  - Modules
  - Dynamic arrays
  - Pointers
  - Recursion
  - `CASE` statement
  - Parameter type checking

Latest versions of Fortran

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

Fortran Evaluation

- Highly 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

The Beginnings of Artificial Intelligence : Lisp

- 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.

The Beginnings of Artificial Intelligence : Lisp

- Mathematicians were interested in mechanizing certain intelligent processes, such as theorem proving.
- All of these investigations arrived at the same conclusion: Some method must be developed to allow computers to process *symbolic data in linked lists.*

Functional Programming: Lisp

- Lisp is the second-oldest high-level programming language after Fortran and has changed a great deal since its early days, and a number of dialects have existed over its history.
- Lisp was invented by John McCarthy in 1958 while he was at the Massachusetts Institute of Technology (MIT).

Functional Programming: Lisp

- LISt Processing language
  - Designed at MIT by McCarthy
- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)
- Only two data types: *atoms* and *lists*
  - Atoms are either symbols, which have the form of identifiers, or numeric literals
  - The concept of storing symbolic information in linked lists is natural and was used in IPL-II
- Syntax is based on *lambda calculus*

Functional Programming: Lisp

- Such structures allow *insertions* and *deletions* at any point, operations that were then thought to be a necessary part of list processing.
- Simple lists, in which elements are restricted to atoms, have the form

For example, the list (A B C D)

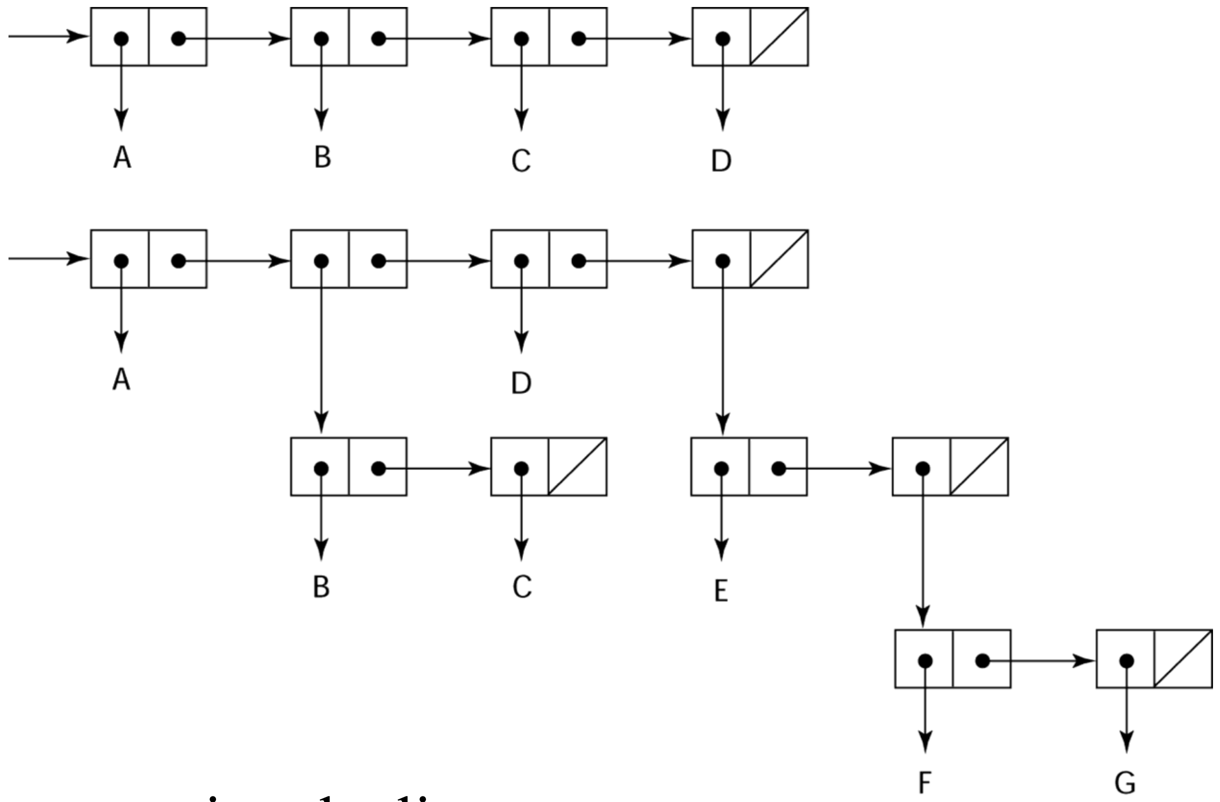
- Nested list structures are also specified by parentheses. For example, the list

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

Functional Programming: Lisp

- Internally, lists are stored as single-linked list structures, in which each node has two pointers and represents a list element.
- A node containing an atom has its first pointer pointing to some representation of the atom, such as its symbol or numeric value, or a pointer to a sublist.
- A node for a sublist element has its first pointer pointing to the first node of the sublist. In both cases, the second pointer of a node points to the next element of the list. A list is referenced by a pointer to its first element.

Representation of Two Lisp Lists



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

Lisp Evaluation

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

Common Lisp

- An effort to combine features of several dialects of Lisp into a single language
- Large, complex, used in industry for some large applications
- Examples:
  - (write-line "Hello World")
  - 7+9+11 → (write (+ 7 9 11))
  - (60 * 9 / 5) + 32 → (write(+ (* (/ 9 5) 60) 32))

- https://www.tutorialspoint.com/lisp/lisp_basic_syntax.htm

example of a LISP program

The following is 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)    ) ))
```

Common Lisp example:

- Examples:

```
(select (:title :author :year)  (from :books)
  (where (:and (:>= :year 1995) (:< :year 2010)))
  (order-by (:desc :year)))
```

Result :

```
⇒ ((:title "Practical Common Lisp"
    :author "Peter Seibel"
    :year 2005)
   (:title "ANSI Common Lisp"
    :author "Paul Graham"
    :year 1995))
```

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
- ALGOL from ALGOrithmic Language

Early Design Process

- ACM and GAMM met for four days for design (May 27 to June 1, 1958)
  - ACM = Association for Computing Machinery; GAMM = German acronym for Association of Applied Mathematics and Mechanics
- Goals of the language
  - Close to mathematical notation
  - Good for describing algorithms
  - Must be translatable to machine code

ALGOL 58

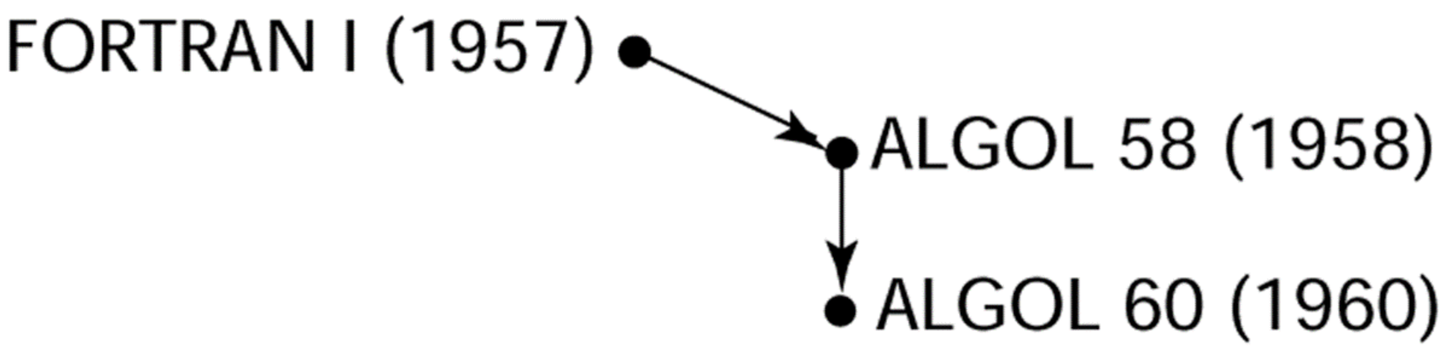
- Concept of type was formalized
- Names could be any length
- Arrays could have any number of subscripts
- Parameters were separated by mode (in & out)
- Subscripts were placed in brackets
- Compound statements (**`begin ... end`**)
- Semicolon as a statement separator
- Assignment operator was :=
- **`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 excited, all support was dropped by mid 1959

ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - Block structure (local scope)
  - Two parameter passing methods
  - Subprogram recursion
  - Stack-dynamic arrays
  - Still no I/O and no string handling



ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative languages are based on it
    - It gave rise to many other programming languages, including CPL, Simula, BCPL, B, Pascal and C.
  - First machine-independent language
  - First language whose syntax was formally defined (BNF)

ALGOL 60 Evaluation (continued)

- Failure
  - Never widely used, especially in U.S.
  - Reasons
    - Lack of I/O and the character set made programs non-portable
    - Too flexible--hard to implement
    - Entrenchment of Fortran
    - Formal syntax description
    - Lack of support from IBM
- algol 68 programming language (not discuss)
- Algol-w programming language (not discuss)

ALGOL 60 Examples:

```
# $run *algolw
= begin

 Compilation begins ...
:    integer I;
:    for I := 1 until 5 do
:        write("Hello, world!");
:end.
:$endfile
```

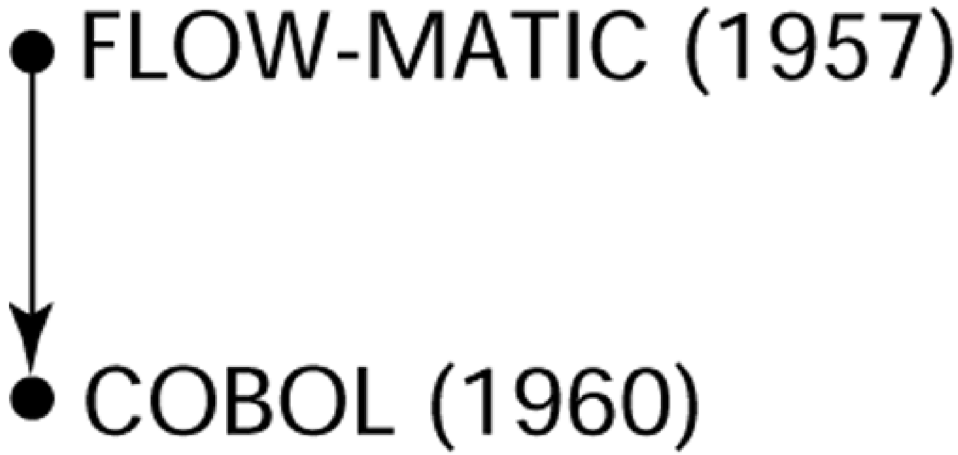
Computerizing Business Records: COBOL

- Environment of development
  - UNIVAC (UNIVersal Automatic **Computer**).

Computerizing Business Records: COBOL

- Environment of development
  - UNIVAC (UNIVersal Automatic **Computer**).

Computerizing Business Records: COBOL



Computerizing Business Records: COBOL

- Environment of development
  - UNIVAC was beginning to use FLOW-MATIC.
  - FLOW-MATIC originally known as B-0 (Business Language version 0), was the first English-like data processing language.
  - It was developed for the UNIVAC I at Remington Rand under Grace Hopper during the period from 1955 until 1959.
  - It had a strong influence on the development of COBOL.
  - **COBOL (Common Business Oriented Language)**

Computerizing Business Records: COBOL

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

COBOL Historical Background

- COBOL Based on FLOW-MATIC
- FLOW-MATIC features
  - Names up to 12 characters, with embedded hyphens
  - English names for arithmetic operators (no arithmetic expressions)
    - + Addition
    - – Subtraction
    - * Multiplication
    - / Division
    - ** Exponentiation
  - ADD Total-1, Total-2 TO Grand-Tot.
  - SUBTRACT COUNT1 FROM 200 GIVING FINAL-CNT.

COBOL Design Process

- Data and code were completely separate
- The first word in every statement was a verb

- First Design Meeting (Pentagon) – May 1959
- Design goals
  - Must look like simple English
  - Must be easy to use, even if that means it will be less powerful
  - Must extend the base of computer users
  - Must not be biased by current compiler problems

COBOL Evaluation

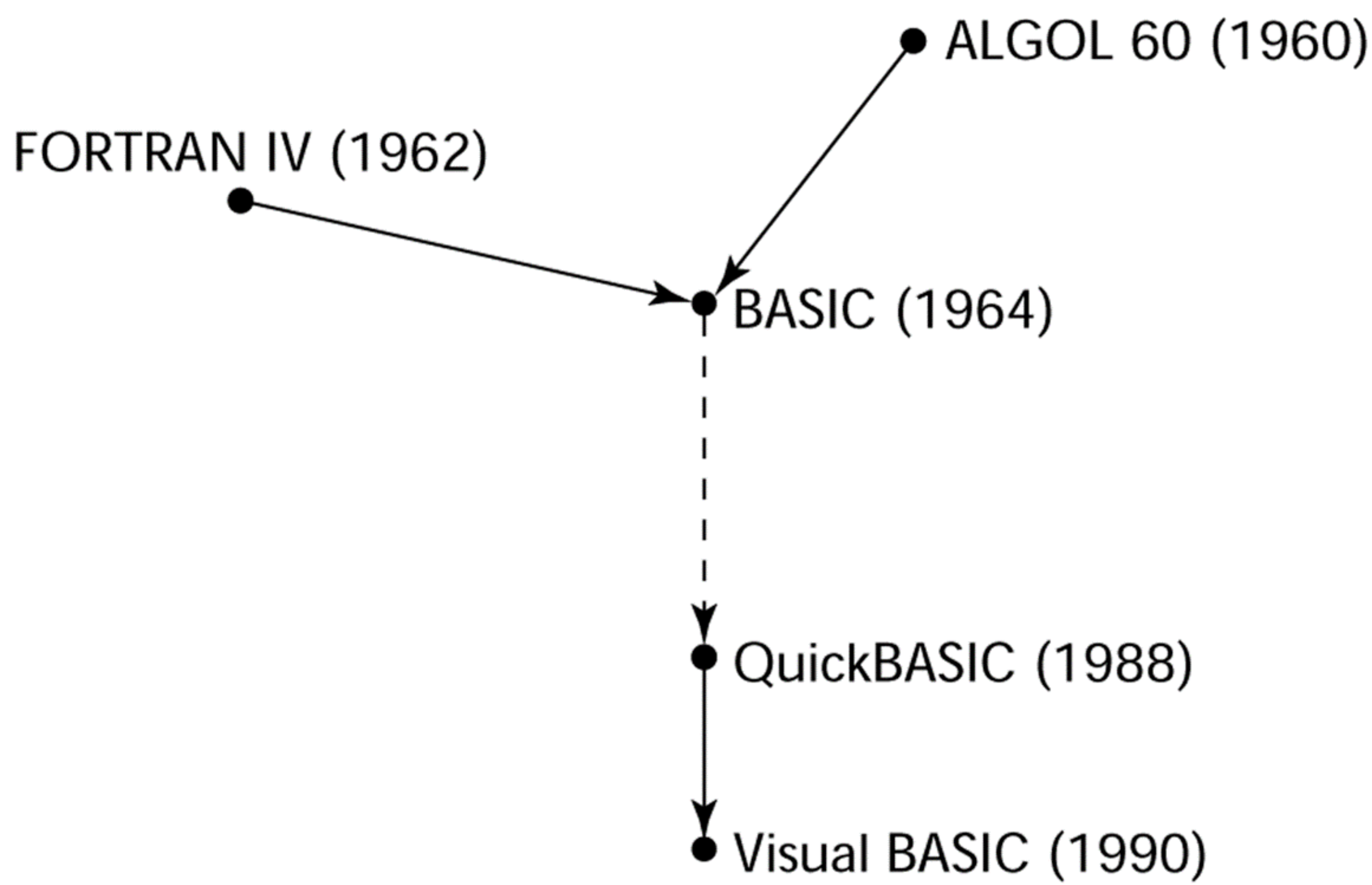
- Contributions
  - First macro facility in a high-level language
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with hyphens
  - Separate data division

COBOL: DoD Influence

- First language required by DoD (Department of Defense)

- Still the most widely used business applications language

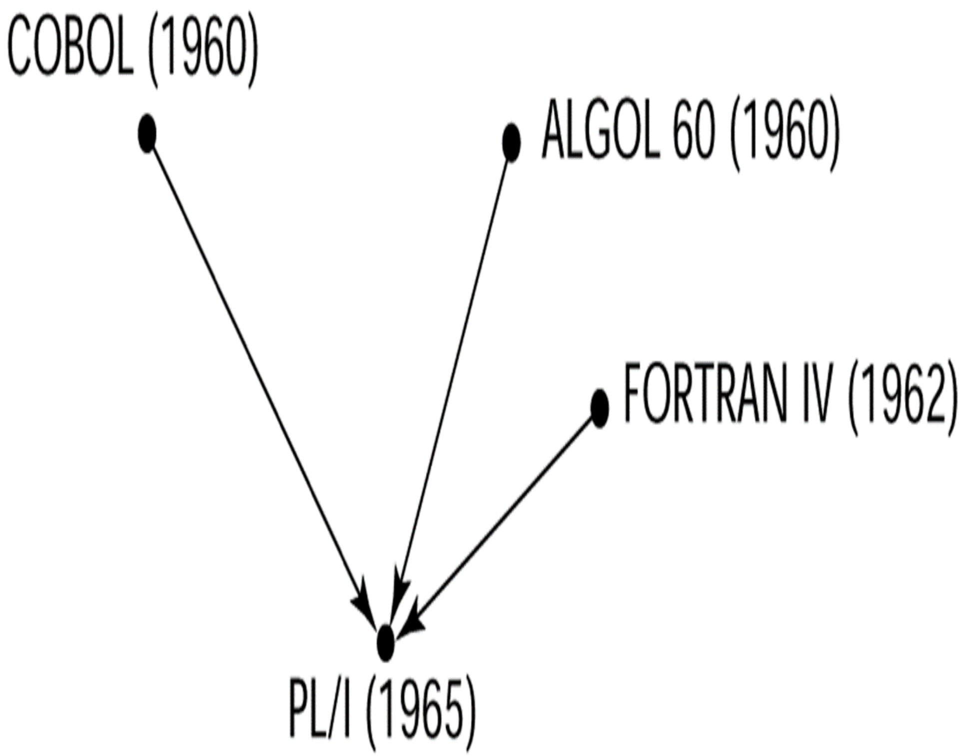
The Beginning of Timesharing: Basic



The Beginning of Timesharing: Basic

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

2.8 Everything for Everybody: PL/I



2.8 Everything for Everybody: PL/I

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

PL/I: Background

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

PL/I: Design Process

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

PL/I: Evaluation

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

Two Early Dynamic Languages: APL and SNOBOL

- Characterized by dynamic typing and dynamic storage allocation
- Variables are untyped
  - A variable acquires a type when it is assigned a value
- Storage is allocated to a variable when it is *assigned a value*
- Examples today: Python; Javascript;
- variable gets its type when assigned value at run time: a=10; a=5.5; etc …

APL: A Programming Language

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

The following program sorts a word list stored in matrix X according to word length:

```
X[⍋X+.≠' ';]
```

The following program finds all prime numbers from 1 to R (presuming an index origin of 1).

$$(\sim R \in R \circ . \times R)/R \leftarrow 1 \downarrow \iota R$$

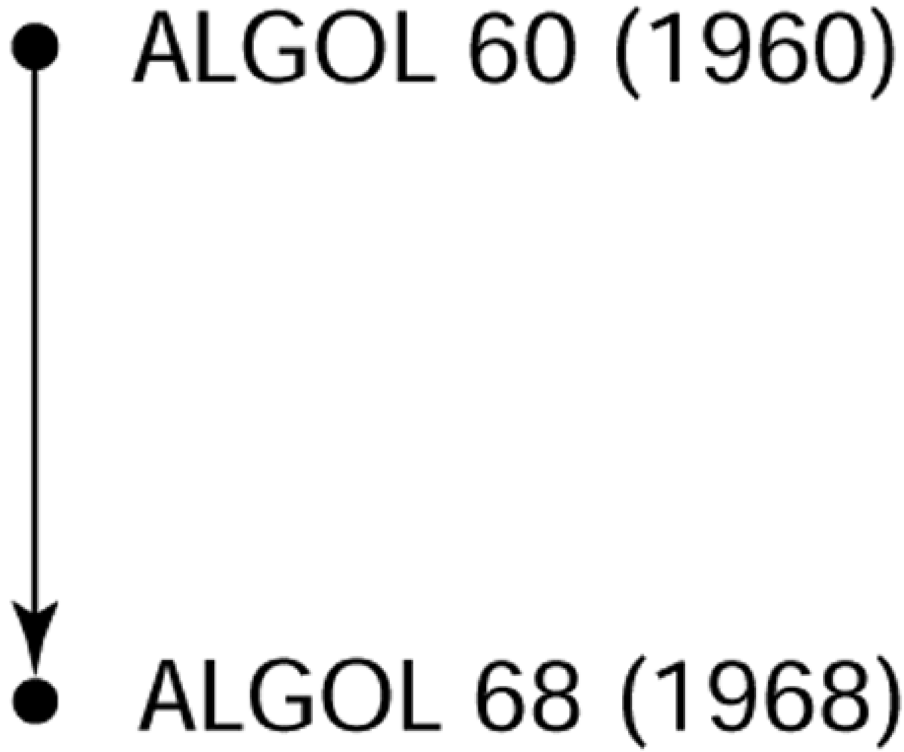
SNOBOL

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

The Beginning of Data Abstraction: SIMULA 67

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

Orthogonal Design: ALGOL 68



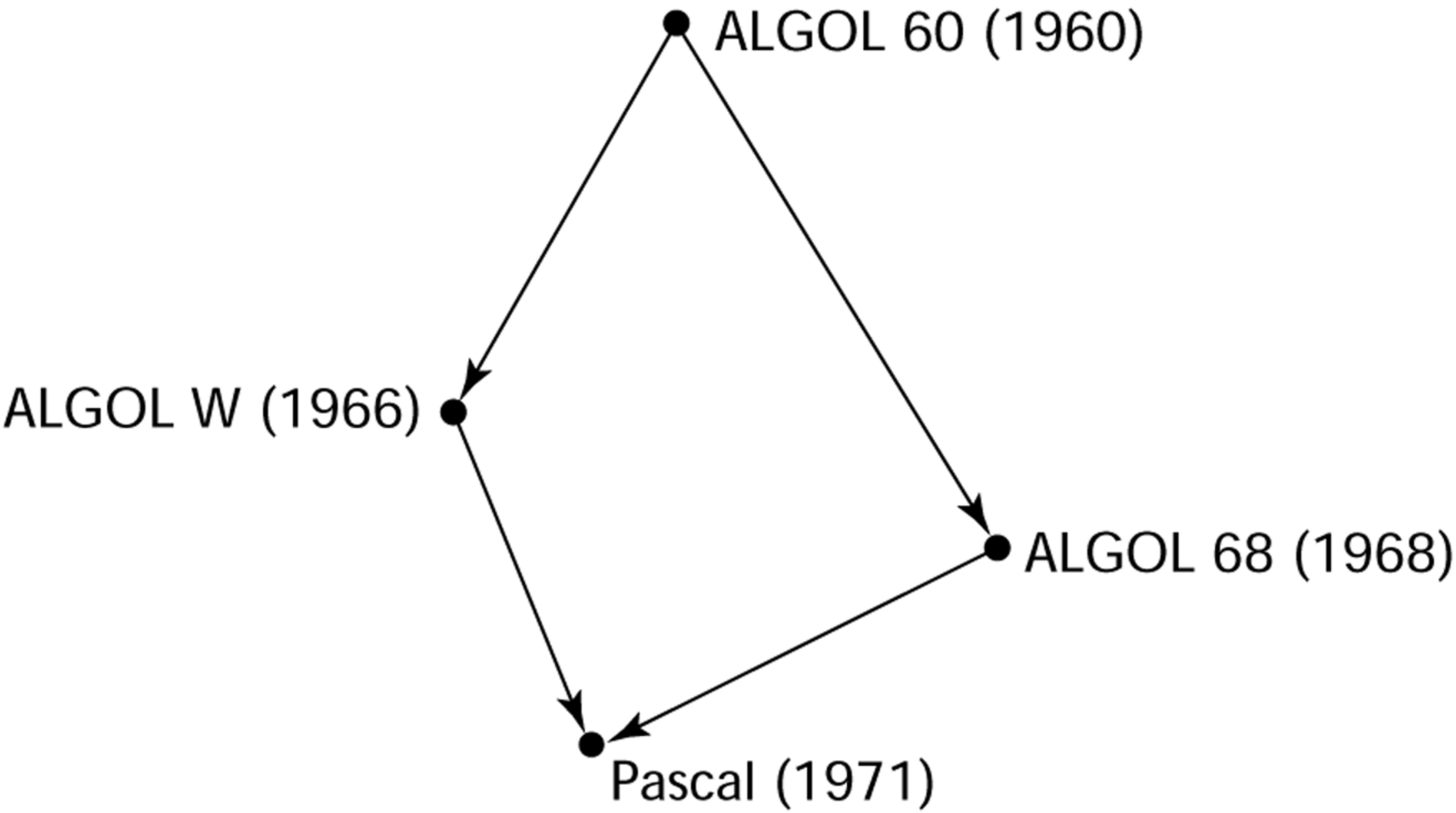
Orthogonal Design: ALGOL 68

- From the continued development of ALGOL 60 but not a superset of that language
- Source of several new ideas (even though the language itself never achieved widespread use)

ALGOL 68 Evaluation

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

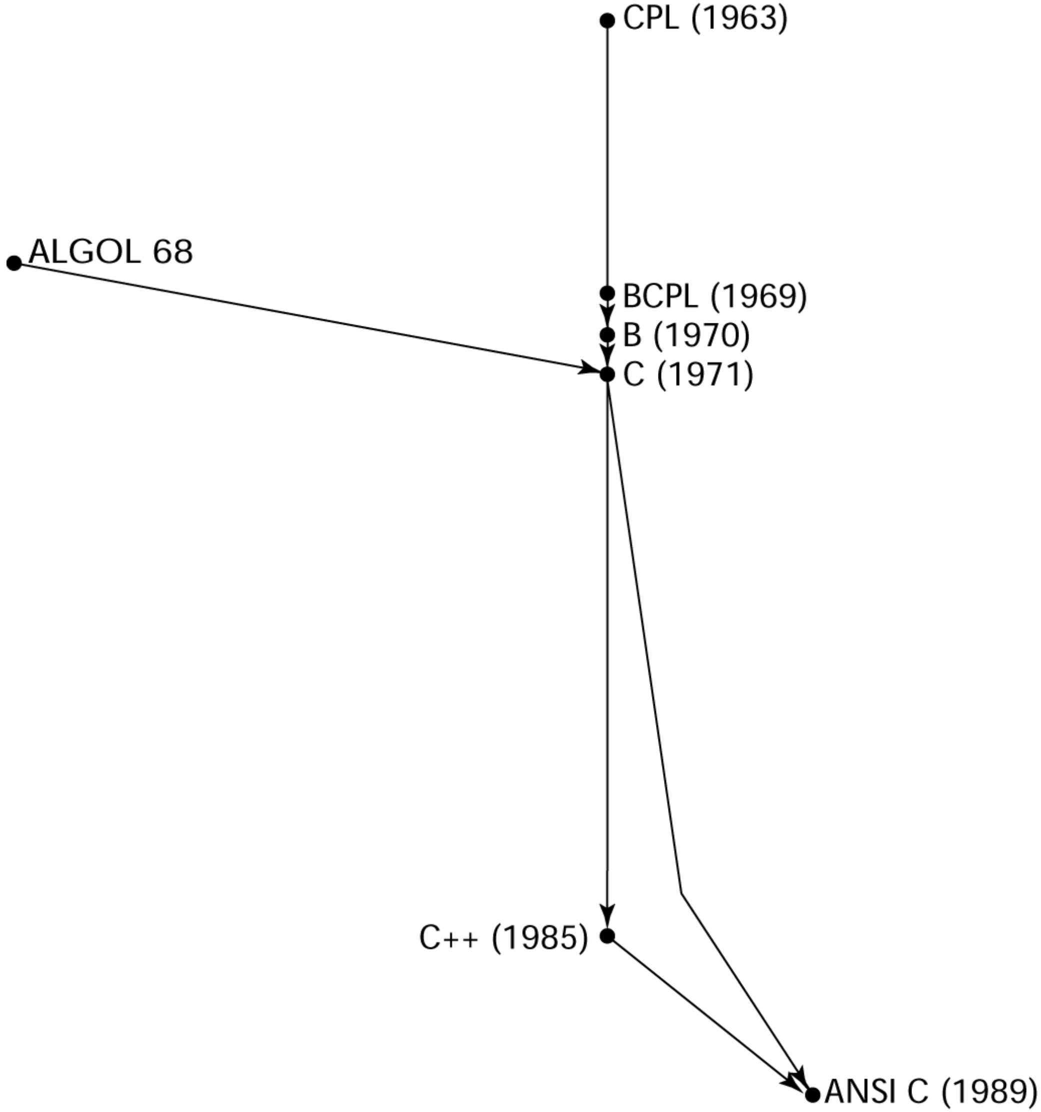
Pascal – 1971



Pascal - 1971

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

C – 1972



C – 1972

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

Programming Based on Logic: Prolog

- Developed, by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski (University of Edinburgh)
- ***PRO***gramming ***LOG***ic
- Uses predicate calculus
- Non-procedural approach: Defines the form of the results to solve a problem and inference rules
- Language is made up of facts and rules
- It stayed highly inefficient
- It has found limited application areas as cretain kinds of DBMS and some areas of AI

History’s Largest Design Effort: Ada

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

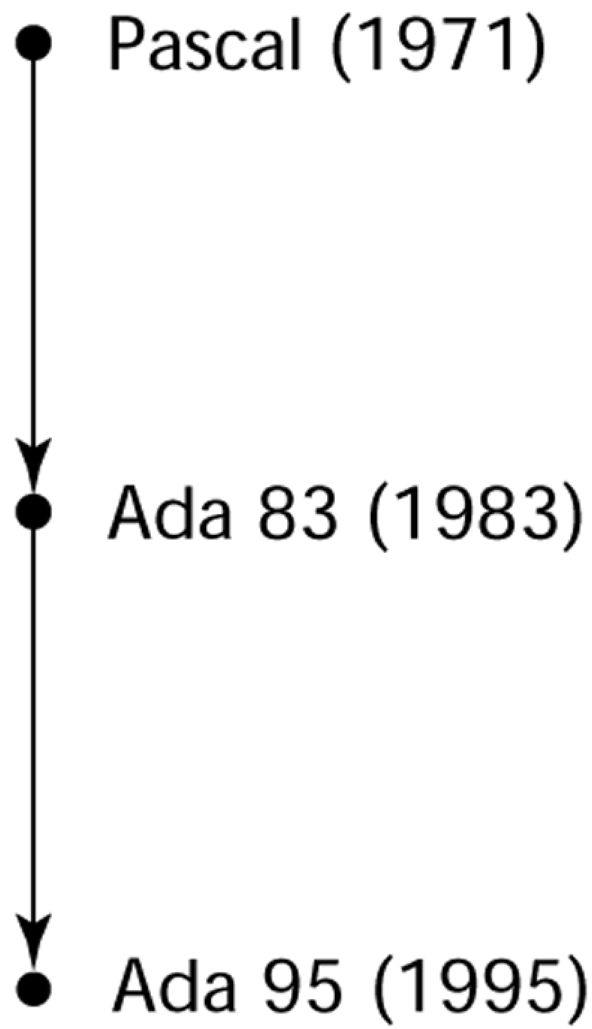
Ada

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

Ada 95

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

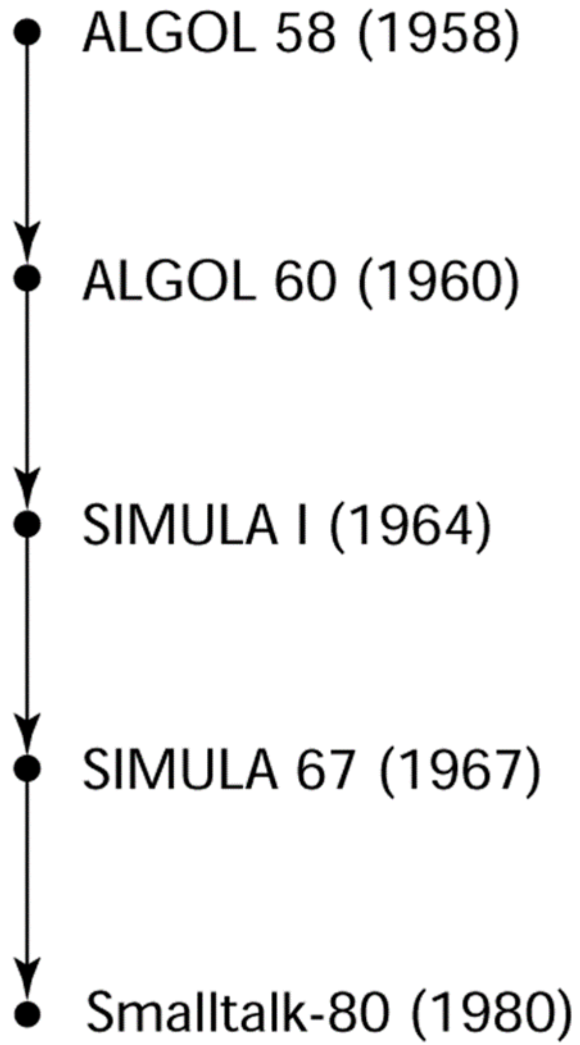
Ada 95



Object-Oriented Programming: Smalltalk

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic binding)

Object-Oriented Programming: 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 OO programming
- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November 1997
- Microsoft's version: MC++
  - Properties, delegates, interfaces, no multiple inheritance

A Related OOP Language

- Objective-C (designed by Brad Cox – early 1980s)
- Objective-C (Kochan, 2009) is another hybrid language with both imperative and object-oriented features.
  - C plus support for OOP based on Smalltalk
  - Used by Apple for systems programs

Delphi: Another Related Language

- Delphi is a hybrid language, similar to C++ and Objetive-C in that it was created by adding object-oriented support, among other things, to an existing imperative language, in this case *Pascal*.
- Based on C++ to design a smaller, simpler and more reliable PL for consumer electronics but later became widespread for www
- Has both classes accessed through reference variables and primitive (scalar) types.
- No pointers but object references
- Has no records, union or enumeration type

Delphi: Another Related Language

- Supports only single-inheritance but allows **interface** construct
- Easy to manage *concurrent* processes (***threds***) via shynchronize modifier
- **Garbage collection** mechanism deallocates storage for objects
- Allow widening assignment **type coercions** (conversions)
- Delphi, like Visual C++, provides a graphical user interface (GUI) to the developer and simple ways to create GUI interfaces to applications written in Delphi.

An Imperative-Based Object-Oriented Language: Java

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

Java Evaluation

- Eliminated many unsafe features of C++
- Supports concurrency (***threds***)
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept

- Widely used for Web programming
- Use increased faster than any previous language
- Most recent version, 8, released in 2014

Scripting Languages for the Web

- JavaScript
  - Began at Netscape, but later became a joint venture of Netscape and Sun Microsystems
  - ***A client-side HTML-embedded*** scripting language, often used to create dynamic HTML documents
  - ***Purely interpreted***
  - Related to Java only through similar syntax
- PHP
  - PHP: Hypertext Preprocessor, designed by Rasmus Lerdorf
  - ***A server-side HTML-embedded*** scripting language, often used for form processing and database access through the Web
  - ***Purely interpreted***

Scripting Languages for the Web

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

Scripting Languages for the Web

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

The Flagship .NET Language: C#

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

Markup/Programming Hybrid Languages

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

Markup/Programming Hybrid Languages

- ASP
  - Active Server Pages is a collection of technologies to support dynamic Web documents
  - Uses server-side scripting to generate content that is sent to the client's web browser.
  - The ASP interpreter reads and executes all script code between <% and %> tags
  - These scripts were written using VBScript, Jscript, C# Script.

Summary

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

Some of Examples Codes for different PL

Sample programming in C

```
#include <stdio.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Output of program:
Hello World

Sample programming in C

```
#include <stdio.h>
 int main()
{
  int x;
  printf("Input an integer\n");
  scanf("%d", &x); // %d is used for an integer
 printf("The integer is: %d\n", x);
   return 0;
}
```

Sample programming in C++

```
#include <iostream>

int main()
{
    cout << "Hello, World!";
    return 0;
}
```

Sample programming in C++

```
#include <iostream>
int main()
{
    int x;
  cout<<"Input an integer\n";
  cin>>x;
  cout<<"The integer is: ", x;
   return 0;
}
```

Sample programming in Java

```
/* HelloWorld.java
 */
public class HelloWorld
{
        public static void main(String[] args) {
                System.out.println("Hello World!");
        }
}
```

Sample programming in Java

```
import java.util.Scanner;
 class InputNumbers{
   public static void main(String args[])  {
      int x;
      System.out.println("Enter integer NO");
      Scanner in = new Scanner(System.in);
      x = in.nextInt();
System.out.println(" The integer is: = " + z);
   }
}
```

Sample programming in Java

```
import java.util.Scanner;
 class AddNumbers{
   public static void main(String args[]) {
      int x, y, z;
      System.out.println("Enter two integers to calculate their sum");
      Scanner in = new Scanner(System.in);
      x = in.nextInt();
      y = in.nextInt();
      z = x + y;
           System.out.println("Sum of the integers = " + z);
   }
}
```

Sample programming in C#

```
// Hello World! program
  public class Hello {
      public static void Main(string[] args)
      {
          System.Console.WriteLine("Hello World!");
      }
   }
}
```

Sample programming in C#

```
using System;

public class Hello {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Sample programming in C#

```
using System;
public class Test{
    public static void Main(string[] args){
            int value = 10;  // Variable
            Console.WriteLine(value);
            Console.WriteLine(50.05);
        }
    }
```

Sample programming in C#

```
using System;
  class MyClass{
        public static void Main(string[] args){
                string userInput;
                int intVal;
                double doubleVal;
        Console.Write("Enter integer value: ");
        userInput = Console.ReadLine();
        intVal = Convert.ToInt32(userInput); /* Converts to integer type */
        Console.WriteLine("You entered {0}",intVal);
        Console.Write("Enter double value: ");
        userInput = Console.ReadLine();
        doubleVal = Convert.ToDouble(userInput);/* Converts to double*/
        Console.WriteLine("You entered {0}",doubleVal);
        }
}
```

Sample programming in PHP

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
   echo "Hello World!";
?>

</body>
</html>
```

Sample programming in php

```
<!DOCTYPE html>
<html>
<body>
  <?php
     $txt = "Hello world!";
     $x = 5;
     $y = 10.5;
echo $txt;
echo "<br>";
echo $x;
echo "<br>";
echo $y;
?>
</body>
</html>
```

Sample programming in php

https://www.w3schools.com/Php/php_examples.asp

Sample programming in python

```
# This program prints Hello, world!

print('Hello, world!')
print("Hello, world!" )
```

Sample programming in python

```
# Add two numbers
num1 = 3
num2 = 5
sum = num1+num2
print(sum)
```

Sample programming in python

```
# This program adds two numbers
  num1 = 1.5
  num2 = 6.3
# Add two numbers
 sum = float(num1) + float(num2)
# Display the sum
  print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

Sample programming in python

```
# Store input numbers

num1 = input('Enter first number: ')
num2 = input('Enter second number: ')

# Add two numbers
sum = float(num1) + float(num2)

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

Sample programming in python

```
# Store input numbers

num1 = float(input('Enter first number: '))
num2 = float(input('Enter second number: '))
# Add two numbers
sum = num1 + num2

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```