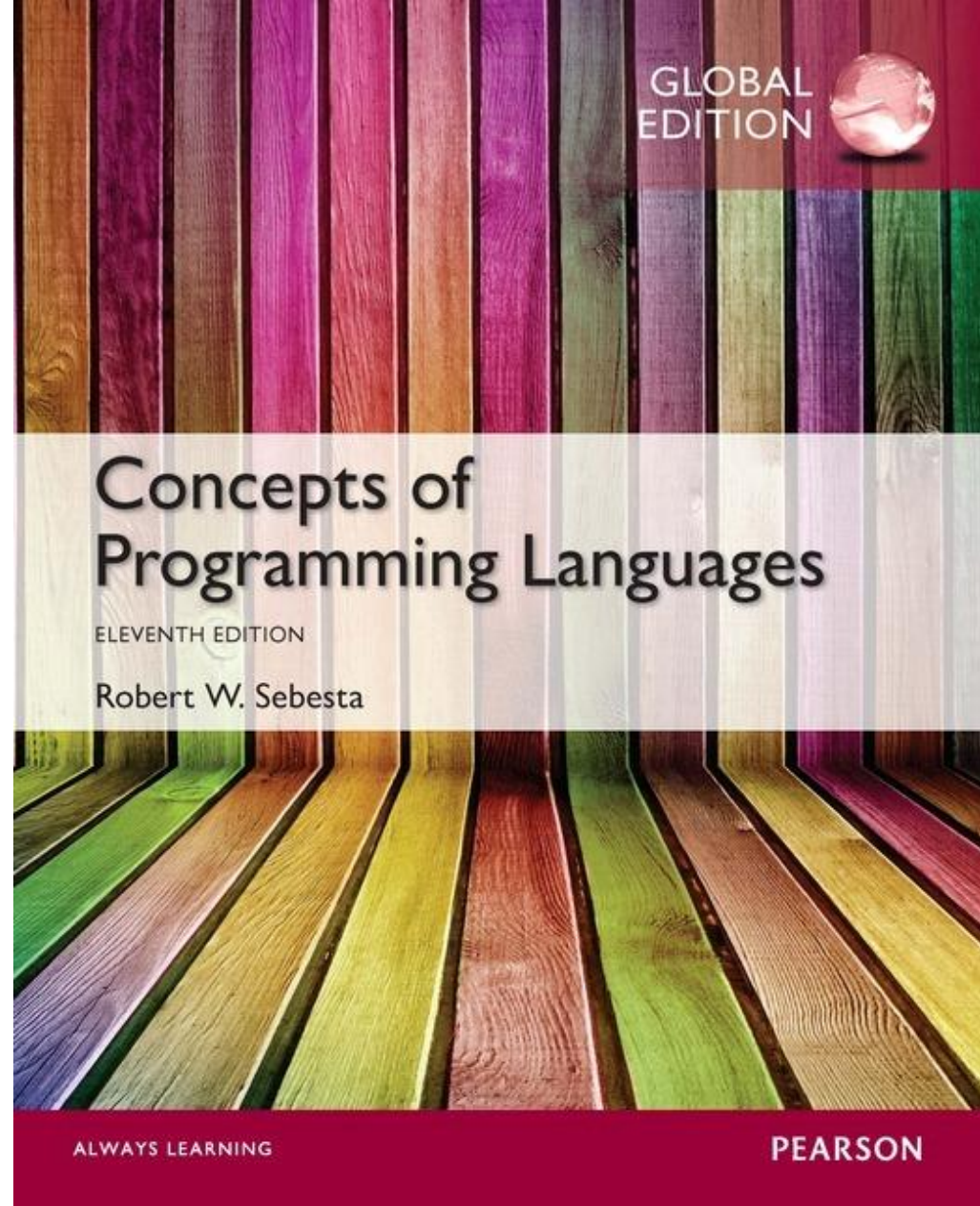


# Chapter 2

## Evolution of the Major Programming Languages



# History

- Early History : The first programmers
- The 1940s: Von Neumann and Zuse
- The 1950s: The First Programming Language
- The 1960s: An Explosion in Programming languages
- The 1970s: Simplicity, Abstraction, Study
- The 1980s: Consolidation and New Directions
- The 1990s: Internet and the Web
- The 2000s: tbd

# Early History: The First Programmer

- Jacquard loom of early 1800s
- Charles Babbage's analytical engine (1830s & 40s)
  - Programs were cards with data and operations
- Ada Lovelace – first programmer

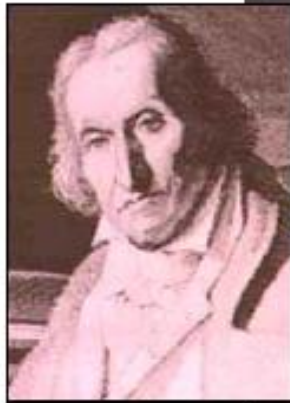
*“The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; And in fact might bring out its results in algebraic notation, were provision made.”*

Source : UMBC CMSC 331



# Jacquard loom of early 1800s

- Worked on automatic weaving machines
- Constructed a loom that used punched cards to encode patterns of weaves (1801)
- Later joined cards into a loop to produce repetitive patterns
- The first stored program!



Joseph-Marie Jacquard  
(1752-1834)



# Charles Babbage's analytical engine (1830s & 40s)

## Babbage's Engines (1)

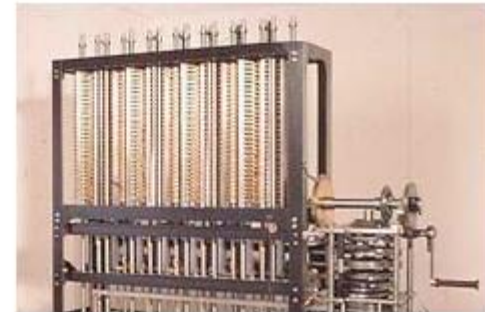
- "I wish to God these tables had been calculated by steam!"
- The Difference Engine (1822)
  - ⇒ Implemented the Method of Differences for calculating mathematical tables
  - ⇒ Never completed in Babbage's lifetime
  - ⇒ Difference Engine No. 2 completed in 1991



Fragment of Difference Engine No. 1 (1832)



Charles Babbage (1791-1871)



Difference Engine No. 2 (1991)

# Charles Babbage's analytical engine (1830s & 40s)

## Babbage's Engines (2)

- The Analytical Engine (1837)
  - ⇒ Conceived as a general-purpose computing machine
  - ⇒ Part of the reason the Difference Engine was never built
  - ⇒ Jacquard cards for programming
  - ⇒ Features
    - ✓ Store (Memory)
    - ✓ Mill (CPU)
    - ✓ Looping
    - ✓ Conditionals
  - ⇒ Never built either



Portion of the Analytical Engine's Mill (1871)

# Ada Lovelace – first programmer

- Worked with Babbage on the design of the Analytical Engine
- Translated Menabrea's "Notions sur la machine analytique de Charles Babbage" into English (1842)
- Added annotations amounting to 3x the length of the original
- Included a "plan" for calculating Bernoulli numbers
- "We may say most aptly that the Analytical Engine weaves algebraical patterns just as the Jacquard loom weaves flowers and leaves"



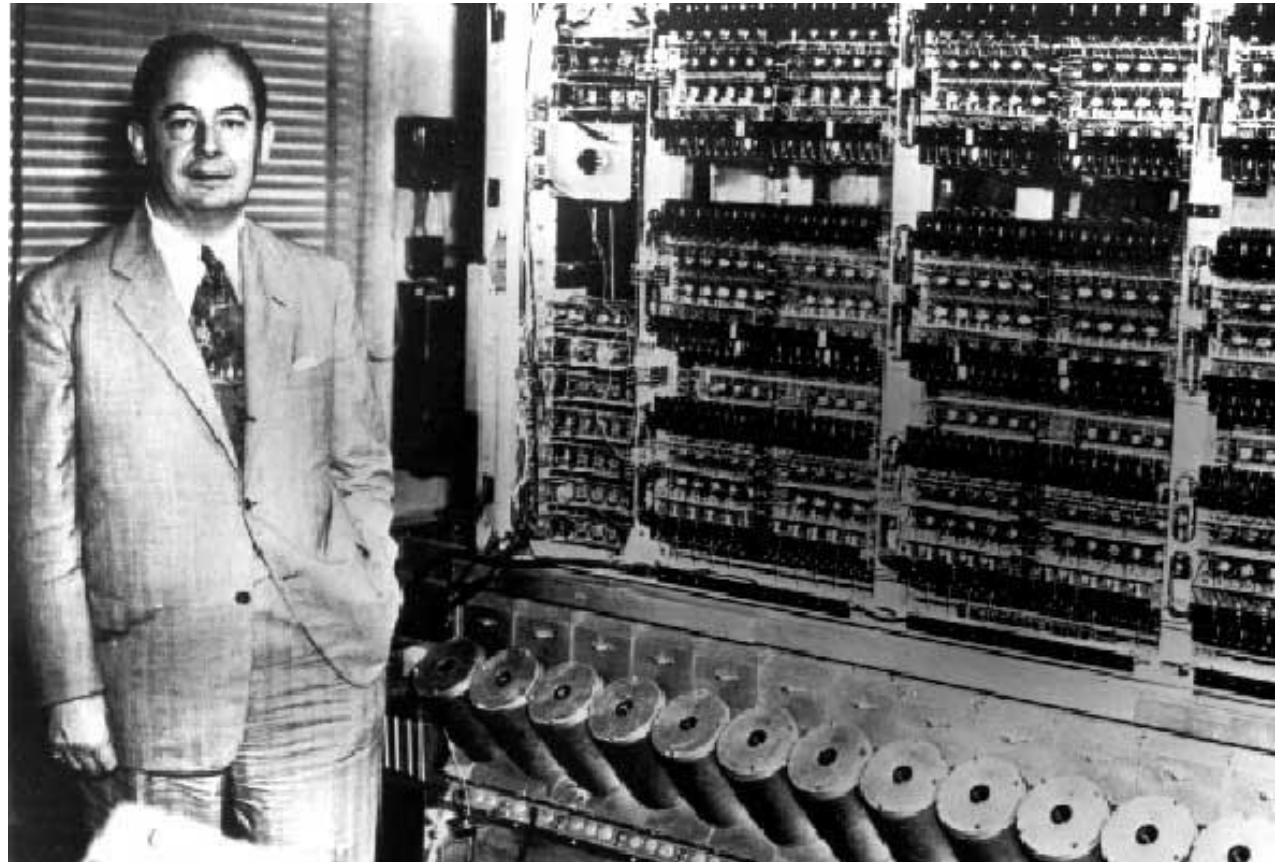
Augusta Ada Byron,  
Countess of Lovelace  
(1816-1852)

12	-	${}^1V_{10} - {}^1V_1$	${}^2V_{10} \dots\dots$	$\left\{ \begin{array}{l} {}^1V_{10} = {}^2V_{20} \\ {}^1V_1 = {}^1V_1 \end{array} \right.$	$= n - 2 (= 2) \dots\dots$	
13	}	-	${}^1V_6 - {}^1V_1$	${}^2V_6 \dots\dots$	$\left\{ \begin{array}{l} {}^1V_6 = {}^2V_6 \\ {}^1V_1 = {}^1V_1 \end{array} \right.$	$= 2n - 1 \dots\dots$
14		+	${}^1V_2 + {}^1V_7$	${}^2V_7 \dots\dots$	$\left\{ \begin{array}{l} {}^1V_2 = {}^1V_2 \\ {}^1V_7 = {}^2V_7 \end{array} \right.$	$= 2 + 1 = 3 \dots\dots$
15		÷	${}^2V_6 \div {}^2V_7$	${}^1V_1 \dots\dots$	$\left\{ \begin{array}{l} {}^2V_6 = {}^2V_6 \\ {}^1V_7 = {}^2V_7 \end{array} \right.$	$= \frac{2n-1}{3} \dots\dots$
16		×	${}^1V_6 \times {}^3V_{11}$	${}^4V_{11} \dots\dots$	$\left\{ \begin{array}{l} {}^2V_6 = {}^0V_6 \\ {}^3V_{11} = {}^4V_{11} \end{array} \right.$	$= \frac{2n}{3} \cdot \frac{2n-1}{5} \dots\dots$
17	}	-	${}^2V_6 - {}^1V_1$	${}^2V_6 \dots\dots$	$\left\{ \begin{array}{l} {}^1V_6 = {}^2V_6 \\ {}^1V_1 = {}^1V_1 \end{array} \right.$	$= 2n - 2 \dots\dots$
18		+	${}^1V_2 + {}^2V_7$	${}^2V_7 \dots\dots$	$\left\{ \begin{array}{l} {}^1V_7 = {}^2V_7 \\ {}^1V_1 = {}^1V_1 \end{array} \right.$	$= 3 + 1 = 4 \dots\dots$
19	}	÷	${}^3V_6 \div {}^3V_7$	${}^1V_1 \dots\dots$	$\left\{ \begin{array}{l} {}^1V_6 = {}^3V_6 \\ {}^1V_7 = {}^3V_7 \end{array} \right.$	$= \frac{2n-2}{4} \dots\dots$
20		×	${}^1V_6 \times {}^4V_{11}$	${}^5V_{11} \dots\dots$	$\left\{ \begin{array}{l} {}^3V_6 = {}^0V_6 \\ {}^4V_{11} = {}^5V_{11} \end{array} \right.$	$= \frac{2n}{5} \cdot \frac{2n-1}{8} \cdot \frac{2n-2}{4} =$
21		×	${}^1V_{20} \times {}^5V_{11}$	${}^6V_{12} \dots\dots$	$\left\{ \begin{array}{l} {}^3V_{20} = {}^1V_{20} \\ {}^6V_{12} = {}^2V_{12} \end{array} \right.$	$= D_3 \cdot \frac{2n}{5} \cdot \frac{2n-1}{8} \cdot \frac{2n}{4}$

# The 1940s: Von Neumann and Zuse

John Von Neumann led a team that built computers with stored programs and a central processor

ENIAC, however, was also programmed with patch cords.



Von Neuman with ENIAC



# ENIAC

- One of the first electronic digital computers
- Eckert and Mauchly at U. Penn (1945)
- Designed to compute ballistic firing tables
- Originally programmed by switches and cables
- von Neumann suggested stored-program design (1948)
- Other early digital computers
  - ⇒ Zuse's Z1 (1938) and Plankalkül (1945)
  - ⇒ Atanasoff at Iowa State (1942)
  - ⇒ Aiken's Mark I at IBM (1943)



Eckert



Mauchly



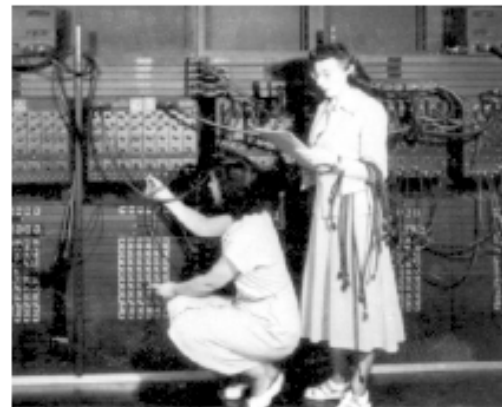
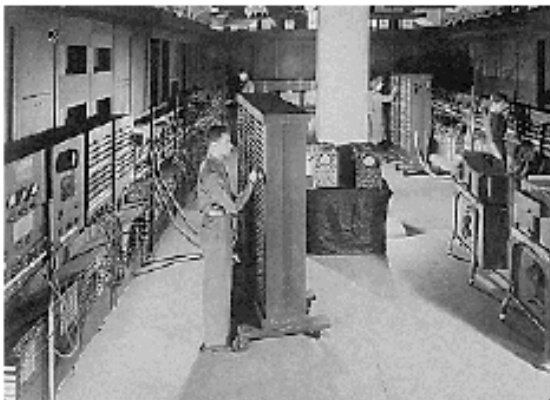
Zuse



Atanasoff



Aiken



# Old Computers



UNIVAC 1108, circa 1970



IBM 704 (Lawrence Livermore National Laboratory)

# Konrad Zuse and Plankalkul



Konrad Zuse began work on Plankalkul (plan calculus), the first algorithmic programming language, with an aim of creating the theoretical preconditions for the formulation of problems of a general nature.

Seven years earlier, Zuse had developed and built the world's first binary digital computer, the Z1. He completed the first fully functional program-controlled electromechanical digital computer, the Z3, in 1941.

Only the Z4, the most sophisticated of his creations, survived World War II.

# The 1940s: Von Neumann and Zuse

- Konrad Zuse (Plankalkul)
  - in Germany - in isolation because of the war
  - defined Plankalkul (program calculus) circa 1945 but never implemented it.
  - Wrote algorithms in the language, including a program to play chess.
  - His work finally published in 1972.
  - Included some advanced data type features such as
    - » Floating point, used twos complement and hidden bits
    - » Arrays
    - » records (that could be nested)

# Plankalkul notation

**A(7) := 5 \* B(6)**

		<b>5</b>	*	<b>B</b>	=>	<b>A</b>	
<b>V</b>				<b>6</b>		<b>7</b>	<b>(subscripts)</b>
<b>S</b>				<b>1.n</b>		<b>1.n</b>	<b>(data types)</b>

# Machine Code (1940's)

- Initial computers were programmed in raw machine code.
- These were entirely numeric.
- What was wrong with using machine code?

Everything!

- Poor readability
- Poor modifiability
- Expression coding was tedious
- Inherit deficiencies of hardware, e.g., no indexing or floating point numbers

# Pseudocodes (1949)

- Short Code or SHORTCODE - John Mauchly, 1949.
- Pseudocode interpreter for math problems, on Eckert and Mauchly's BINAC and later on UNIVAC I and II.
- Possibly the first attempt at a higher level language.
- Expressions were coded, left to right, e.g.:

X0 = sqrt(abs(Y0))

00 X0 03 20 06 Y0

- Some operations:

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

# More Pseudocodes

## Speedcoding; 1953-4

- A pseudocode interpreter for math on IBM 701, IBM 650.
- Developed by John Backus
- Pseudo ops for arithmetic and math functions
- Conditional and unconditional branching
- Autoincrement registers for array access
- Slow but still dominated by slowness of s/w math
- Interpreter left only 700 words left for user program

## Laning and Zierler System - 1953

- Implemented on the MIT Whirlwind computer
- First "algebraic" compiler system
- Subscripted variables, function calls, expression translation
- Never ported to any other machine

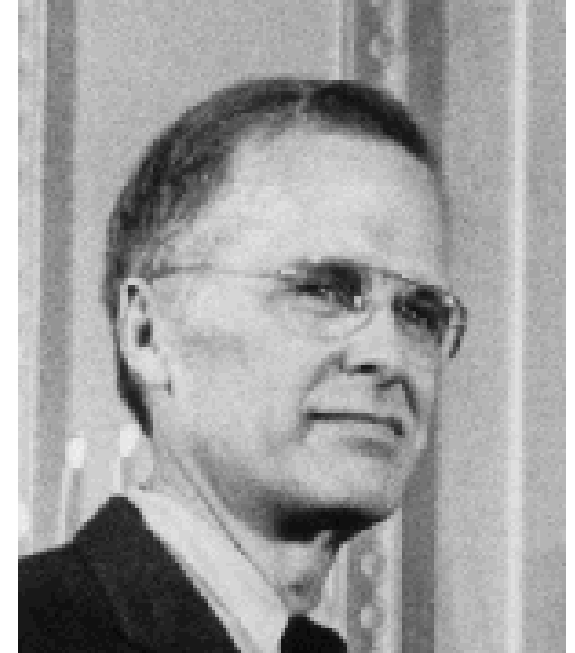


# The 1950s: The First Programming Language

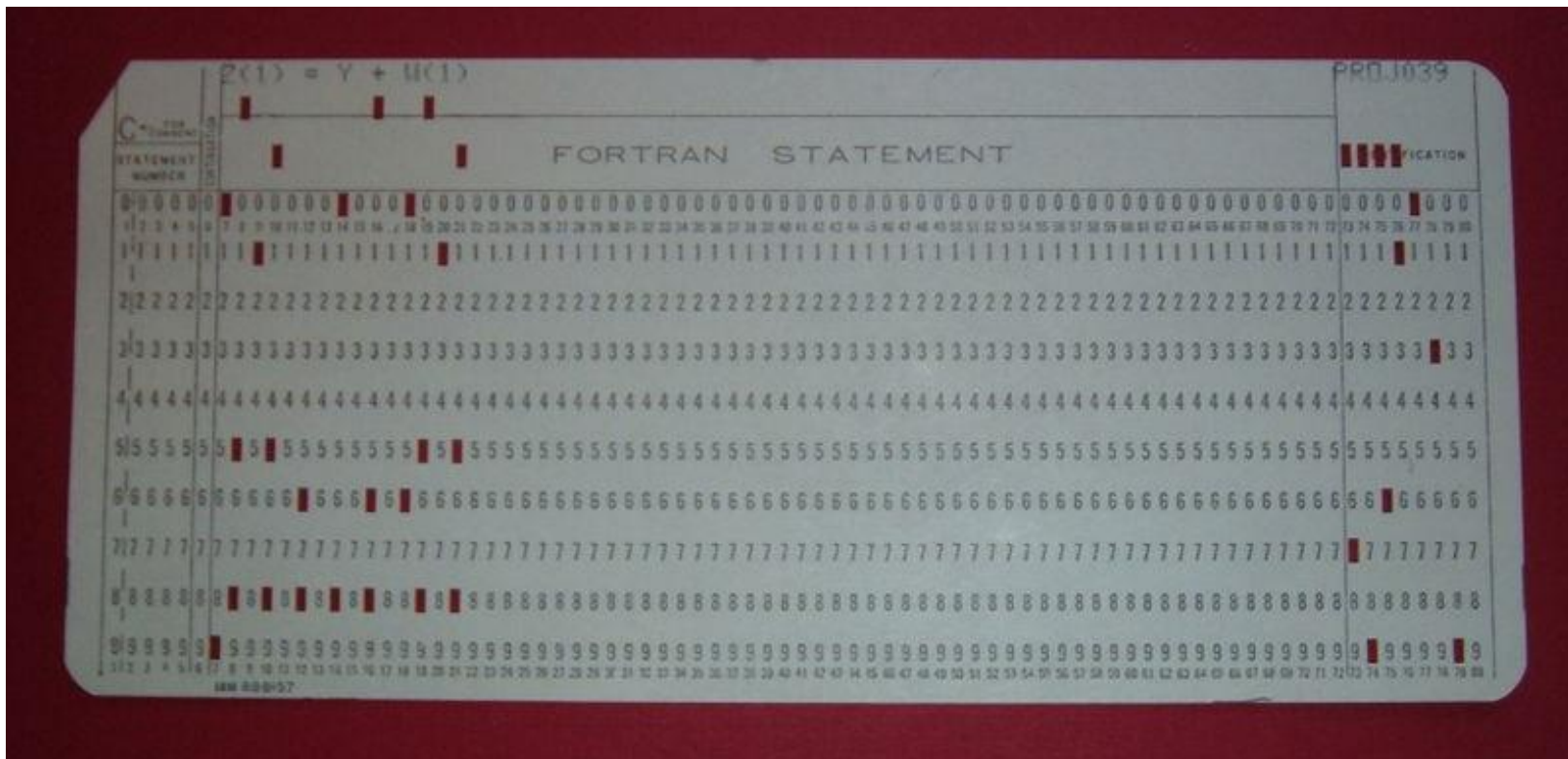
- Pseudocodes: interpreters for assembly language like
- Fortran: the first higher level programming language
- COBOL: the first business oriented language
- Algol: one of the most influential programming languages ever designed
- LISP: the first language to depart from the procedural paradigm
- APL:

# Fortran (1954-57)

- FORMula TRANslator
- Developed at IBM under the guidance of John Backus primarily for scientific programming
- Dramatically changed forever the way computers used
- Has continued to evolve, adding new features & concepts.
  - FORTRAN II, FORTRAN IV, FORTRAN 66, FORTRAN 77, FORTRAN 90
- Always among the most efficient compilers, producing fast code
- Still popular, e.g. for supercomputers



# Punch Card



# Fortran (1954-57)

- Developed by John Backus for the IBM 704 (1955)
- FORMula TRANslator
- For scientific computation
- Optimizing compiler



John Backus

C ← FOR COMMENT		CONTINUATION	FORTRAN STATEMENT	IDENTIFICATION		
STATEMENT NUMBER				72	73	96
1	2	3				
			PROGRAM FOR FINDING THE LARGEST VALUE			
		X	ATTAINED BY A SET OF NUMBERS			
			DIMENSION A(999)			
			FREQUENCY 30(2,1,10), 5(100)			
			READ 1, N, (A(I), 1*1,N)			
1			FORMAT (13/(12F6.2))			
			BIGA = A(1)			
5			DO 20 I = 2,N			
30			IF (BIGA-A(I)) 10,20,20			
10			BIGA = A(I)			
20			CONTINUE			
			PRINT 2, N, BIGA			
2			FORMAT (22H1THE LARGEST OF THESE 13, 12H NUMBERS IS F7.2)			
			STOP 7777			

Example from the IBM 704 FORTRAN Manual

# Fortran 0 and 1

FORTTRAN 0 – 1954 (not implemented)

FORTTRAN I - 1957

Designed for the new IBM 704, which had index registers and floating point hardware

## **Environment of development:**

Computers were small and unreliable

Applications were scientific

No programming methodology or tools

Machine efficiency was most important

## **Impact of environment on design**

- No need for dynamic storage
- Need good array handling and counting loops
- No string handling, decimal arithmetic, or powerful input/output (commercial stuff)

# Fortran I Features

- Names could have up to six characters
- Post-test counting loop (DO)
- Formatted I/O
- User-defined subprograms
- Three-way selection statement (arithmetic IF)  
IF (ICOUNT-1) 100, 200, 300
- No data typing statements  
variables beginning with i, j, k, l, m or n were integers, all else floating point
- No separate compilation
- Programs larger than 400 lines rarely compiled correctly, mainly due to IBM 704's poor reliability
- Code was very fast
- Quickly became widely used

```

C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
    READ INPUT TAPE 5, 501, IA, IB, IC
501 FORMAT (3I5)
    IF (IA) 777, 777, 701
701 IF (IB) 777, 777, 702
702 IF (IC) 777, 777, 703
703 IF (IA+IB-IC) 777,777,704
704 IF (IA+IC-IB) 777,777,705
705 IF (IB+IC-IA) 777,777,799
777 STOP 1
799 S = FLOATF (IA + IB + IC) / 2.0
    AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
+      (S - FLOATF(IC)))
    WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
601 FORMAT (4H A= ,I5,5H B= ,I5,5H C= ,I5,8H AREA= ,F10.2,
+      13H SQUARE UNITS)
    STOP
    END

```

# Fortran II, IV and 77

## FORTTRAN II - 1958

- Independent compilation
- Fix the bugs

## FORTTRAN IV - 1960-62

- Explicit type declarations
- Logical selection (IF) statement
- Subprogram names could be parameters
- ANSI standard in 1966

## FORTTRAN 77 - 1978

- Character string handling
- Logical loop control (WHILE) statement
- IF-THEN-ELSE statement



# Fortran 77 Bubble Sort

```
SUBROUTINE SSORT (X, IY, N, KFLAG)
  IMPLICIT NONE
```

c

c Example of a Bubble Sort

```
  JMAX=N-1
  DO 200 I=1,N-1
    TEMP=1.E38
    DO 100 J=1,JMAX
      IF(X(J).GT.X(J+1)) GO TO 100
      TEMP=X(J)
      X(J)=X(J+1)
      X(J+1)=TEMP
      ITEMP=IY(J)
      IY(J)=IY(J+1)
      IY(J+1)=ITEMP
100  CONTINUE
      IF(TEMP.EQ.1.E38) GO TO 300
      JMAX=JMAX-1
200  CONTINUE
300  RETURN
  END
```

# Fortran 90 (1990)

Added many features of more modern programming languages, including

- Pointers
- Recursion
- CASE statement
- Parameter type checking
- A collection of array operations, DOTPRODUCT, MATMUL, TRANSPOSE, etc
- dynamic allocations and deallocation of arrays
- a form of records (called derived types)
- Module facility (similar Ada's package)

# FORTRAN 90 Examples

**! EXAMPLE 1**

**!**

```
real,dimension(1000)::a=(/(i,i=1,1000)/)
```

```
real,dimension(1000)::b,c
```

```
b=a
```

```
c=a+b
```

```
print *, c
```

```
end
```

**! EXAMPLE 2**

**!**

```
real,dimension(10,10)::a=(((i+j,i=1,10),j=1,10)/)
```

```
real,dimension(10,10)::b,c
```

```
b=a
```

```
c=a+b
```

```
print *, c
```

```
end
```

# COBOL

- COmmon Business Oriented Language
- Principal mentor: (Rear Admiral Dr.) Grace Murray Hopper (1906-1992)
- *Based on FLOW-MATIC* which had such features as:
  - Names up to 12 characters, with embedded hyphens
  - English names for arithmetic operators
  - Data and code were completely separate
  - Verbs were first word in every statement
- CODASYL committee (Conference on Data Systems Languages) developed a programming language by the name of COBOL



Rear Admiral Grace Murray  
Hopper (1906-1992)

# COBOL



# COBOL

First CODASYL Design 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 computer users
- Must not be biased by current compiler problems

Design committee were all from computer manufacturers and DoD branches

Design Problems: arithmetic expressions? subscripts?  
Fights among manufacturers

# COBOL

## Contributions:

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

## Comments:

- First language required by DoD; would have failed without DoD
- Still the most widely used business applications language

# COBOL Example

- **Description :** This program takes all input records of salesperson data and writes it to an output file reformatted.

```
000100 ID DIVISION.
000200 PROGRAM-ID. SLS02.
000300 FILE-CONTROL.
000400   SELECT SALESPERSON-FILE
000500     ASSIGN TO DISK.
000600   SELECT REPORT-FILE
000700     ASSIGN TO PRINTER.
000800 DATA DIVISION.
000900 FILE SECTION.
001000 FD SALESPERSON-FILE.
001100 01 SALESPERSON-RECORD.
001200   05 FILLER          PIC XX.
001300   05 SP-NUMBER       PIC X(4).
001400   05 SP-NAME        PIC X(18).
001500   05 FILLER          PIC X(21).
001600   05 SP-CURRENT-SALES PIC 9(5)V99.
001700   05 SP-CURRENT-RETURNS PIC 9(4)V99.
001800 FD REPORT-FILE.
001900 01 REPORT-RECORD.
002000   05 FILLER          PIC X(10).
002100   05 RT-NUMBER      PIC X(4).
002200   05 FILLER          PIC X(6).
002300   05 RT-NAME        PIC X(18).
002400   05 FILLER          PIC X(6).
002500   05 RT-CURRENT-SALES PIC ZZ,ZZZ.99.
002600   05 FILLER          PIC X(6).
002700   05 RT-CURRENT-RETURNS PIC Z,ZZZ.99.
002800   05 FILLER          PIC X(65).
```



# COBOL Example

```
002900 WORKING-STORAGE SECTION.  
003000 01 WS-EOF-FLAG      PIC X.  
003100*  
003200 PROCEDURE DIVISION.  
003300*  
003400 MAIN-ROUTINE.  
003500  OPEN INPUT SALESPERSON-FILE  
003600      OUTPUT REPORT-FILE  
003700  MOVE "N" TO WS-EOF-FLAG  
003800  READ SALESPERSON-FILE  
003900      AT END MOVE "Y" TO WS-EOF-FLAG  
004000  END-READ  
004100*  
004200 PERFORM UNTIL WS-EOF-FLAG IS EQUAL TO "Y"  
004300  MOVE SPACES TO REPORT-RECORD  
004400  MOVE SP-NUMBER TO RT-NUMBER  
004500  MOVE SP-NAME TO RT-NAME  
004600  MOVE SP-CURRENT-SALES TO RT-CURRENT-SALES  
004700  MOVE SP-CURRENT-RETURNS TO RT-CURRENT-RETURNS  
004800  WRITE REPORT-RECORD  
004900  READ SALESPERSON-FILE  
005000      AT END MOVE "Y" TO WS-EOF-FLAG  
005100  END-READ  
005200 END-PERFORM  
005300*  
005400 CLOSE SALESPERSON-FILE, REPORT-FILE  
005500 STOP RUN.
```

---

## Sample Run

0005	BENNETT ROBERT	1,600.35	12.50
0016	LOCK ANDREW S	357.72	79.85
0080	PARKER JAMES E	18,200.00	165.00

# BASIC (1964)

- Beginner's All purpose Symbolic Instruction Code
- Designed by Kemeny & Kurtz at Dartmouth for the GE 225 with the goals:
  - Easy to learn and use for non-science students and as a path to Fortran and Algol
  - Must be "pleasant and friendly"
  - Fast turnaround for homework
  - Free and private access
  - User time is more important than computer time
- Well-suited for implementation on first PCs, e.g., Gates and Allen's 4K Basic interpreter for the MITS Altair personal computer (circa 1975)
- Current popular dialects: Visual BASIC

# BASIC Example

- Description : This program performs basic arithmetic operations.

## Source Code

```
10 INPUT "ENTER TWO NUMBERS SEPARATED BY A COMMA:"  
20 LET S = N1 + N2  
30 LET D = N1 - N2  
40 LET P = N1 * N2  
50 LET Q = N1 / N2  
60 PRINT "THE SUM IS ", S  
70 PRINT "THE DIFFERENCE IS ", D  
80 PRINT "THE PRODUCT IS ", P  
90 PRINT "THE QUOTIENT IS ", Q  
100 END
```

---

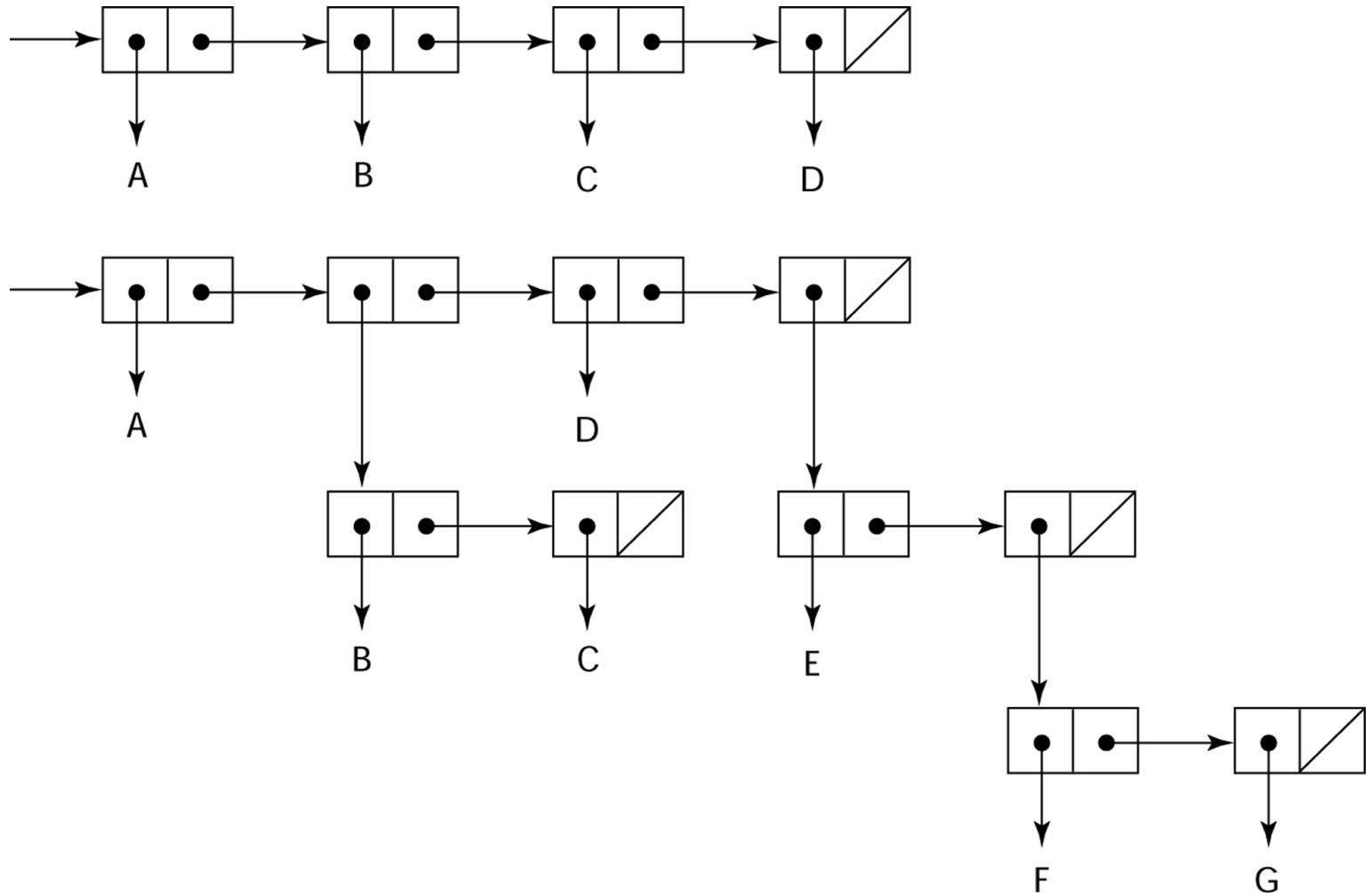
## Sample Run

```
ENTER TWO NUMBERS SEPARATED BY A COMMA:  
4 2  
THE SUM IS 6  
THE DIFFERENCE IS 2  
THE PRODUCT IS 8  
THE QUOTIENT IS 2
```

# LISP (1959)

- LISt Processing language (Designed at MIT by McCarthy)
- *AI research needed a language that:*
  - Process data in lists (rather than arrays)
  - Handles symbolic computation (rather than numeric)
- One universal, recursive data type: the s-expression
  - An s-expression is either an atom or a list of zero or more s-expressions
- Syntax is based on the lambda calculus
- *Pioneered functional programming*
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Status
  - Still the dominant language for AI
  - COMMON LISP and Scheme are contemporary dialects
  - ML, Miranda, and Haskell are related languages

# Representation of Two LISP Lists

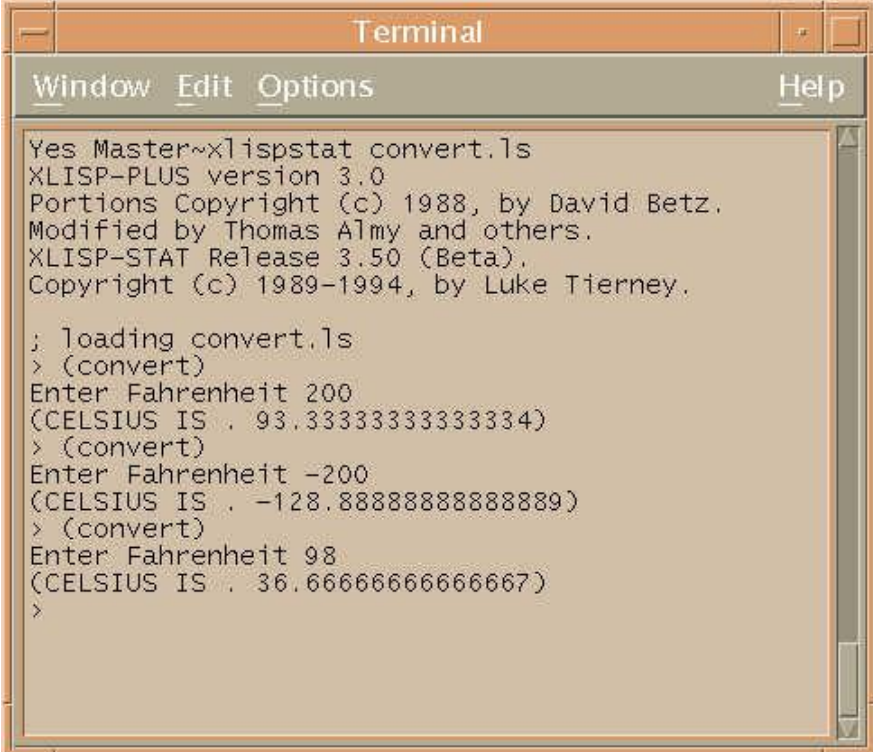


# LISP Example

## Source Code

;;; This function, given a specific degree in Fahrenheit,  
;;; presents the user with equivalent Celsius degree.

```
(defun convert ()  
  (format t "Enter Fahrenheit ")  
  (LET (fahr)  
    (SETQ fahr (read fahr))  
    (APPEND '(celsius is) (*(- fahr 32)/( 5 9)) )  
  )  
)
```



The image shows a terminal window titled "Terminal" with a menu bar containing "Window", "Edit", "Options", and "Help". The terminal output is as follows:

```
Yes Master~xlispstat convert.l  
XLISP-PLUS version 3.0  
Portions Copyright (c) 1988, by David Betz.  
Modified by Thomas Almy and others.  
XLISP-STAT Release 3.50 (Beta).  
Copyright (c) 1989-1994, by Luke Tierney.  
  
; loading convert.l  
> (convert)  
Enter Fahrenheit 200  
(CELSIUS IS . 93.33333333333334)  
> (convert)  
Enter Fahrenheit -200  
(CELSIUS IS . -128.88888888888889)  
> (convert)  
Enter Fahrenheit 98  
(CELSIUS IS . 36.66666666666667)  
>
```

# Algol

*Environment of development:*

1. FORTRAN had (barely) arrived for IBM 70x
2. Many other languages were being developed, all for specific machines
3. No portable language; all were machine-dependent
4. No universal language for communicating algorithms

ACM and GAMM met for four days for design

- *Goals of the language:*

1. Close to mathematical notation
2. Good for describing algorithms
3. Must be translatable to machine code

# Algol 58 Features

- Concept of type was formalized
- Names could have 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

## **Comments:**

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid-1959



# Algol 60

Modified ALGOL 58 at 6-day meeting in Paris adding such new features as:

- Block structure (local scope)
- Two parameter passing methods
- Subprogram recursion
- Stack-dynamic arrays
- Still no I/O and no string handling

*Successes:*

- The standard way to publish algorithms for over 20 years
- All subsequent imperative languages are based on it
- First machine-independent language
- First language whose syntax was formally defined (BNF)

# Algol 60 (1960)

*Failure:* Never widely used, especially in U.S., mostly because

1. No I/O and the character set made programs nonportable
2. Too flexible--hard to implement
3. Entrenchment of FORTRAN
4. Formal syntax description
5. Lack of support by IBM

# Sample ALGOL

- **Description** : This program computes the mean (average) of the absolute value of an array. Block structures, a dynamic array, and iterative statements are featured in this program. The bold type print represent keywords.

```
// the main program (this is a comment)
```

```
begin
```

```
integer N;
```

```
Read Int(N);
```

```
begin
```

```
real array Data[1:N];
```

```
real sum, avg;
```

```
integer i;
```

```
sum:=0;
```

```
for i:=1 step 1 until N do
```

```
begin real val;
```

```
Read Real(val);
```

```
Data[i]:=if val<0 then -val else val
```

```
end;
```

```
for i:=1 step 1 until N do
```

```
sum:=sum + Data[i];
```

```
avg:=sum/N;
```

```
Print Real(avg)
```

```
end
```

```
end
```

# APL

- A Programming Language
- Designed by K.Iverson at Harvard in late 1950's
- A language for programming mathematical computations
  - especially those using matrices
- Functional style and many whole array operations
- Drawback is requirement of special keyboard

# The 1960s: An Explosion in Programming Languages

- The development of hundreds of programming languages
- PL/I designed in 1963-4
  - supposed to be all purpose
  - combined features of FORTRAN, COBOL and Algol 60 and more!
  - translators were slow, huge and unreliable
  - some say it was ahead of its time.....
- Algol 68
- SNOBOL
- Simula
- BASIC

# PL/I

- 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
- IBM's goal: develop a single computer (IBM 360) and a single programming language (PL/I) that would be good for scientific and business applications.
- Eventually grew to include virtually every idea in current practical programming languages.

# PL/I

PL/I contributions:

1. First unit-level concurrency
2. First exception handling
3. Switch-selectable recursion
4. First pointer data type
5. First array cross sections

Comments:

- Many new features were poorly designed
- Too large and too complex
- Was (and still is) actually used for both scientific and business applications
- Subsets (e.g. PL/C) developed which were more manageable

# Simula (1962-67)

- Designed and built by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Centre (NCC) in Oslo between 1962 and 1967
- Originally designed and implemented as a language for discrete event simulation
- Based on ALGOL 60

## *Primary Contributions:*

- Coroutines - a kind of subprogram
- Classes (data plus methods) and objects
- Inheritance
- Dynamic binding

=> Introduced the basic ideas that developed into object-oriented programming.



# Algol 68

From the continued development of ALGOL 60, but it is not a superset of that language

- Design is based on the concept of orthogonality
- *Contributions:*
  - User-defined data structures
  - Reference types
  - Dynamic arrays (called flex arrays)
- *Comments:*
  - Had even less usage than ALGOL 60
  - Had strong influence on subsequent languages, especially Pascal, C, and Ada

# The 1970s: Simplicity, Abstraction, Study

- Algol-W - Nicklaus Wirth and C.A.R.Hoare
  - reaction against 1960s
  - simplicity
- Pascal
  - small, simple, efficient structures
  - for teaching program
- C - 1972 - Dennis Ritchie
  - aims for simplicity by reducing restrictions of the type system
  - allows access to underlying system
  - interface with O/S - UNIX

# Pascal (1971)

- Designed by Wirth, who quit the ALGOL 68 committee (didn't like the direction of that work)
- Designed for teaching structured programming
- Small, simple
- Introduces some modest improvements, such as the case statement
- Was widely used for teaching programming ~ 1980-1995.

# Pascal Sample

```
(*****  
* A simple bubble sort program. Reads integers, one per line, and prints *  
* them out in sorted order. Blows up if there are more than 49. *  
*****)  
PROGRAM Sort(input, output);  
  CONST  
    (* Max array size. *)  
    MaxEls = 50;  
  TYPE  
    (* Type of the element array. *)  
    IntArrType = ARRAY [1..MaxEls] OF Integer;  
  
  VAR  
    (* Indexes, exchange temp, array size. *)  
    i, j, tmp, size: integer;  
  
    (* Array of ints *)  
    arr: IntArrType;  
  
  (* Read in the integers. *)  
  PROCEDURE ReadArr(VAR size: Integer; VAR a: IntArrType);  
  BEGIN  
    size := 1;  
    WHILE NOT eof DO BEGIN  
      readln(a[size]);  
      IF NOT eof THEN  
        size := size + 1  
    END  
  END;  
  
  BEGIN  
    (* Read *)  
    ReadArr(size, arr);  
  
    (* Sort using bubble sort. *)  
    FOR i := size - 1 DOWNTO 1 DO  
      FOR j := 1 TO i DO  
        IF arr[j] > arr[j + 1] THEN BEGIN  
          tmp := arr[j];  
          arr[j] := arr[j + 1];  
          arr[j + 1] := tmp;  
        END;  
  
        (* Print. *)  
        FOR i := 1 TO size DO  
          writeln(arr[i])  
        END.  
  END;
```

# C (1972-)

- Designed for systems programming at Bell Labs by Dennis Ritchie and colleagues.
- Evolved primarily from B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX and the availability of high quality, free compilers, especially gcc.

# Other descendants of ALGOL

- **Modula-2** (mid-1970s by Niklaus Wirth at ETH)
  - Pascal plus modules and some low-level features designed for systems programming
- **Modula-3** (late 1980s at Digital & Olivetti)
  - Modula-2 plus classes, exception handling, garbage collection, and concurrency
- **Oberon** (late 1980s by Wirth at ETH)
  - Adds support for OOP to Modula-2
  - Many Modula-2 features were deleted (e.g., for statement, enumeration types, with statement, non-integer array indices)

# The 1980s: Consolidation and New Paradigms

- Ada
  - US Department of Defence
  - European team lead by Jean Ichbiah. (Sam Lomonaco was also on the ADA team :-)
- Functional programming
  - Scheme, ML, Haskell
- Logic programming
  - Prolog
- Object-oriented programming
  - Smalltalk, C++, Eiffel

# Ada

- In study done in 73-74 it was determined that the DoD was spending \$3B annually on software, over half on embedded computer systems.
- The Higher Order Language Working Group was formed and initial language requirements compiled and refined in 75-76 and existing languages evaluated.
- In 1977, it was concluded that none were suitable, though Pascal, ALGOL 68 or PL/I would be a good starting point.
- Language DoD-1 was developed through a series of competitive contracts.



# Ada

- Renamed Ada in May 1979.
- Reference manual, Mil. Std. 1815 approved 10 December 1980. (Ada Bryon was born 10/12/1815)
- “mandated” for use in DoD work during late 80’s and early 90’s.
- Ada95, a joint ISO and ANSI standard, accepted in February 1995 and included many new features.
- The Ada Joint Program Office (AJPO) closed 1 October 1998 (Same day as ISO/IEC 14882:1998 (C++) published!)

# Ada

## *Contributions:*

1. Packages - support for data abstraction
2. Exception handling - elaborate
3. Generic program units
4. 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
- Very difficult to mandate programming technology

# Logic Programming: Prolog

- Developed at the University of Aix Marseille, by Comerauer and Roussel, with some help from Kowalski at the University of Edinburgh
- Based on formal logic
- Non-procedural
- Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries

```
sibling(X, Y)    :- parent_child(Z, X), parent_child(Z, Y).  
parent_child(X, Y) :- father_child(X, Y).  
parent_child(X, Y) :- mother_child(X, Y).  
mother_child(trude, sally).  
father_child(tom, sally).  
father_child(tom, erica).  
father_child(mike, tom).
```

# Functional Programming

- **Common Lisp:** consolidation of LISP dialects spurred practical use, as did the development of Lisp Machines.
- **Scheme:** a simple and pure LISP like language used for teaching programming.
- **Logo:** Used for teaching young children how to program.
- **ML:** (Meta Language) a strongly-typed functional language first developed by Robin Milner in the 70's
- **Haskell:** polymorphically typed, lazy, purely functional language.

# Smalltalk (1972-80)

- Developed at Xerox PARC by Alan Kay and colleagues (esp. Adele Goldberg) inspired by Simula 67
- First compilation in 1972 was written on a bet to come up with "the most powerful language in the world" in "a single page of code".
- In 1980, Smalltalk 80, a uniformly object-oriented programming environment became available as the first commercial release of the Smalltalk language
- Pioneered the graphical user interface everyone now uses
- Industrial use continues to the present day

# Combining Imperative and Object-Oriented Programming: C++ (1985)

- Developed at Bell Labs by Stroustrup
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67, added to C
- Also has exception handling
- 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

# Related OOP Languages

- Objective-C (designed by Brad Cox – early 1980s)
  - C plus support for OOP based on Smalltalk
  - Uses Smalltalk’s method calling syntax
  - Used by Apple for systems programs
- Delphi (Borland)
  - Pascal plus features to support OOP
  - More elegant and safer than C++
- Go (designed at Google - 2009)
  - Loosely based on C, but also quite different
  - Does not support traditional OOP

# An Imperative-Based Object-Oriented Language: Java

- Developed at Sun in the early 1990s with original goal of a language for embedded computers
  - C and C++ were not satisfactory for embedded electronic devices
- Principals: Bill Joy, James Gosling, Mike Sheradin, Patrick Naughton
- Original name, Oak, changed for copyright reasons
- Based on C++
  - Significantly simplified (does not include **struct**, **union**, **enum**, pointer arithmetic, and half of the assignment coercions of C++)
  - Supports *only* OOP
  - Has references, but not pointers
  - Includes support for applets and a form of concurrency



# Java

- 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



# 1990's: the Internet and Web

During the 90's, Object-oriented languages (mostly C++) became widely used in practical applications

The Internet and Web drove several phenomena:

- Adding concurrency and threads to existing languages
- Increased use of scripting languages such as Perl and Tcl/Tk
- Java as a new programming language

# Scripting Languages for the Web

- 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

# 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
- Based on C++ and Java
- Provides a language for component-based software development
- All .NET languages (C#, Visual BASIC.NET, Managed C++, J#.NET, and Jscript.NET) use Common Type System (CTS), which provides a common class library
- Likely to become widely used

# Markup/Programming Hybrid Languages

- XML, XSLT
  - eXtensible Markup Language (XML): a metamarkup language
  - eXtensible Stylesheet Language Transformation (XSLT) transforms XML documents for display
  - Programming constructs (e.g., looping)
- JSP
  - Java Server Pages: a collection of technologies to support dynamic Web documents
  - servlet: a Java program that resides on a Web server; servlet's output is displayed by the browser

# Programming Paradigms

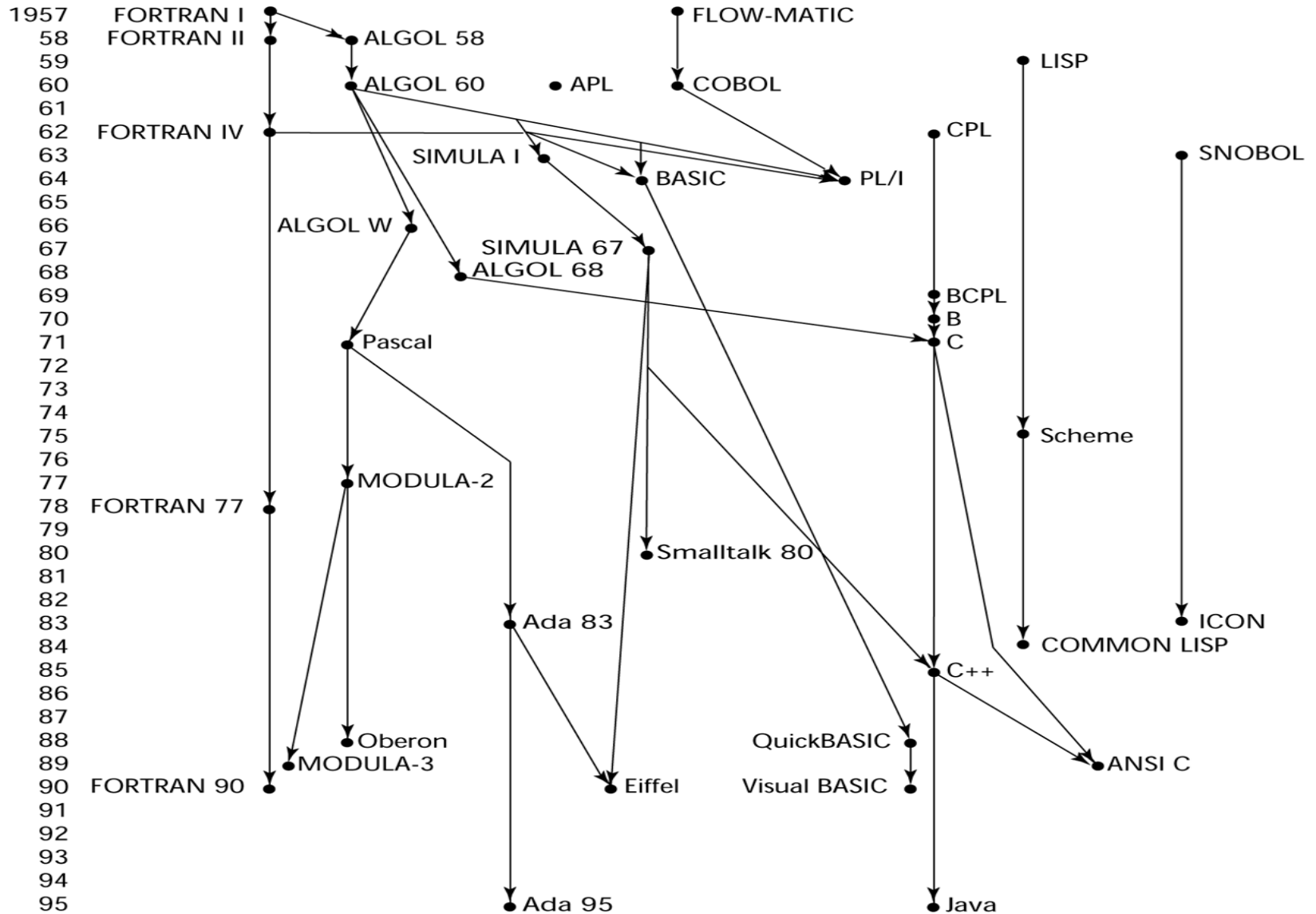
- Imperative
- Object Oriented
- Functional
- Logic
- Parallel/Concurrent
- Distributed
- Constraint (declarative based on specifying constraints)
- Data flow (model as a directed graph of the data flowing between operations)
- Aspect Oriented (separation of cross-cutting concerns, built on top of OO, is it a new paradigm?)
- Generic (sometimes considered as a new paradigm)



# Multiparadigm languages

- 1 paradigm:
  - Imperative: C, Pascal, Basic
- 2 paradigms
  - Imperative + OO: C++
  - Functional + imperative: scheme
- 3 paradigms
  - Functional + imperative + OO: Perl, Python, Tcl
  - Imperative + concurrent + OO: Java, C#
- 4 paradigms
  - Functional + imperative + concurrent + OO: Ruby
  - Functional + imperative + logic + OO: Leda
- 8 paradigms
  - Concurrent + constraint + dataflow + distributed + functional + imperative + logic + OO: Mozart

# Genealogy of Common Languages



# The future

- In the 60's, the dream was a single all-purpose language (e.g., PL/I, Algol)
- The 70s and 80s dream expressed by Winograd (1979)

“Just as high-level languages allow the programmer to escape the intricacies of the machine, higher level programming systems can provide for manipulating complex systems. We need to shift away from algorithms and towards the description of the properties of the packages that we build. Programming systems will be declarative not imperative”

- Will that dream be realised?
- Programming is not yet obsolete