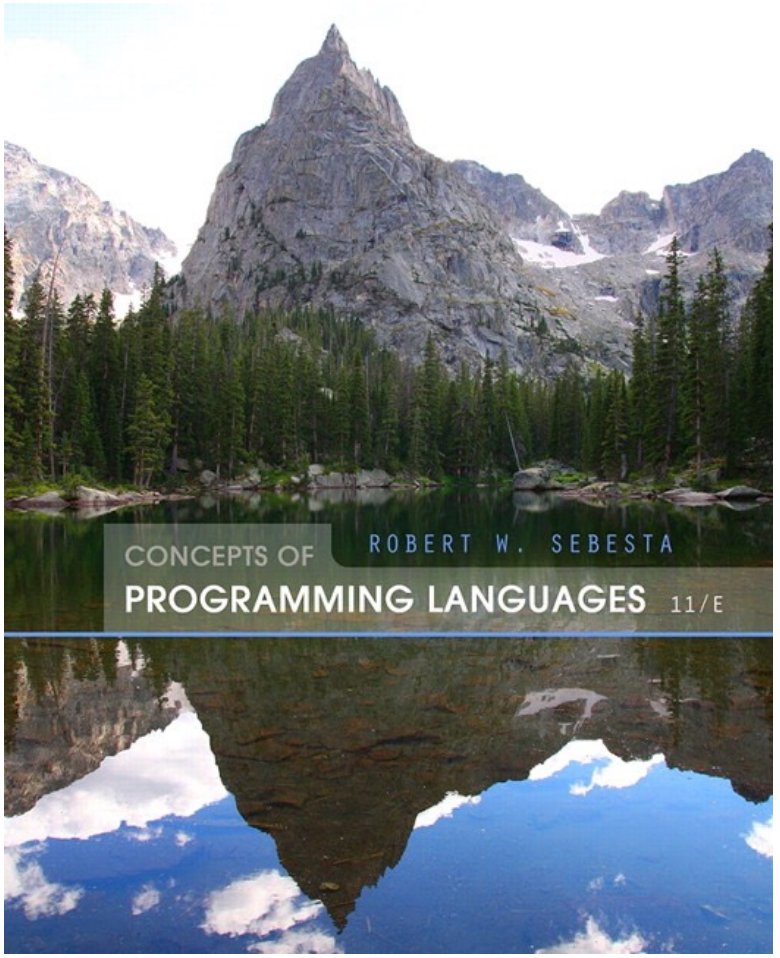
# Chapter 2

## Evolution of the Major Programming Languages (part 2)




ISBN 0-321-49362-1

# 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?

# 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)

# 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

# Early Design Process

- ACM and GAMM met for four (!) days for design (May 27 to June 1, 1958)

- Goals of the language
  - Syntax close to mathematical notation
  - Good for describing algorithms in publications - new
  - Must be mechanically translatable into machine code

# Early Design Process

- Example: Assignment statement:
  - Initially like Plankalkül (not yet published, but some European members were familiar with)
    expression => variable
  - Discussion/arguments: Card punches at time did not include greater than symbol
  - Later changed to Fortran form:
    variable := expression

# ALGOL 58

In many ways descendent from Fortran, but meant to be more general and machine independent

- Concept of type was formalized
- Names could be any length (machine independent)
- 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

- No I/O - “would make it machine dependent”
- Knuth 1964, A proposal for input-output conventions in ALGOL 60: “The ALGOL 60 language as first defined made no explicit reference to input and output processes. Such processes appeared to be quite dependent on the computer used, and so it was difficult to obtain agreement on those matters. As time has passed, a great many ALGOL compilers have come into use, and each compiler has incorporated some input-output facilities.”

# ALGOL 58 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid 1959 (difficult to read; understand; more support for Fortran)

# ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - Block structure (local scope)
  - Two parameter passing methods
  - Subprogram recursion (new for imperative; note LISP in 1959)
  - Stack-dynamic arrays (array size set at time of execution)
  - Still no I/O and no string handling

# ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years

# ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative languages are based on it

# ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative languages are based on it
  - First machine-independent language

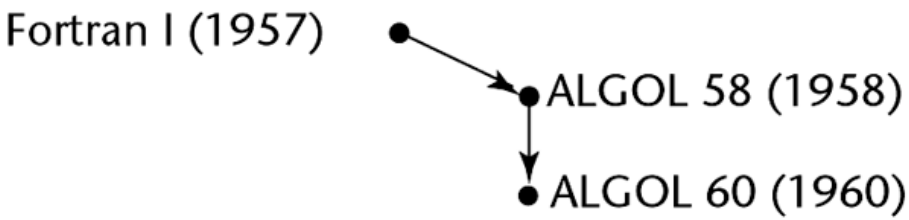
# ALGOL 60 Evaluation

- Successes
  - It was 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; later)

# ALGOL 60 Evaluation

**Figure 2.3**

Genealogy of ALGOL 60




- Most imperative languages are direct or indirect descendants: PL/I, SIMULA 67, C, Pascal, Ada, C++, Java …

# 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--hard to implement
    - Entrenchment of Fortran
    - BNF for formal syntax description - back then seemed strange and complicated; today widely used
    - Lack of support from IBM

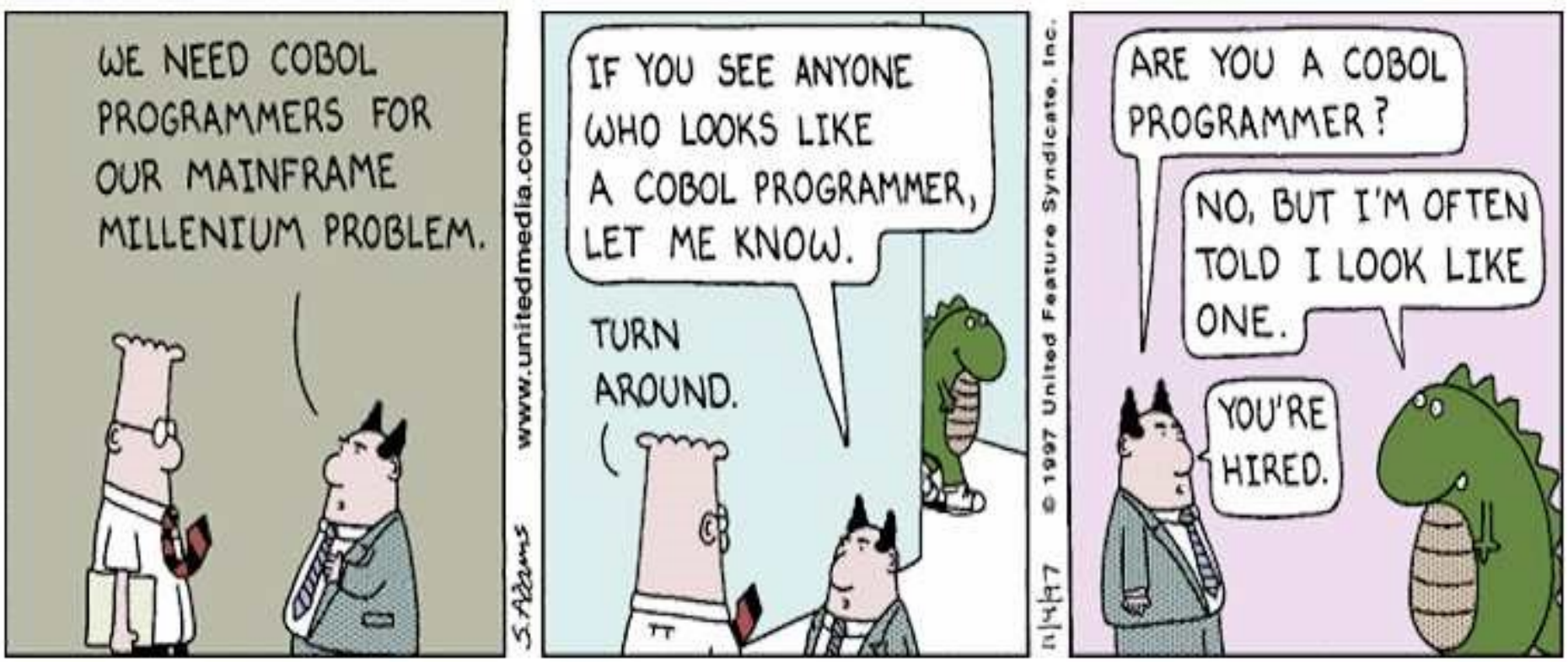
# Example ALGOL code

calculating mean

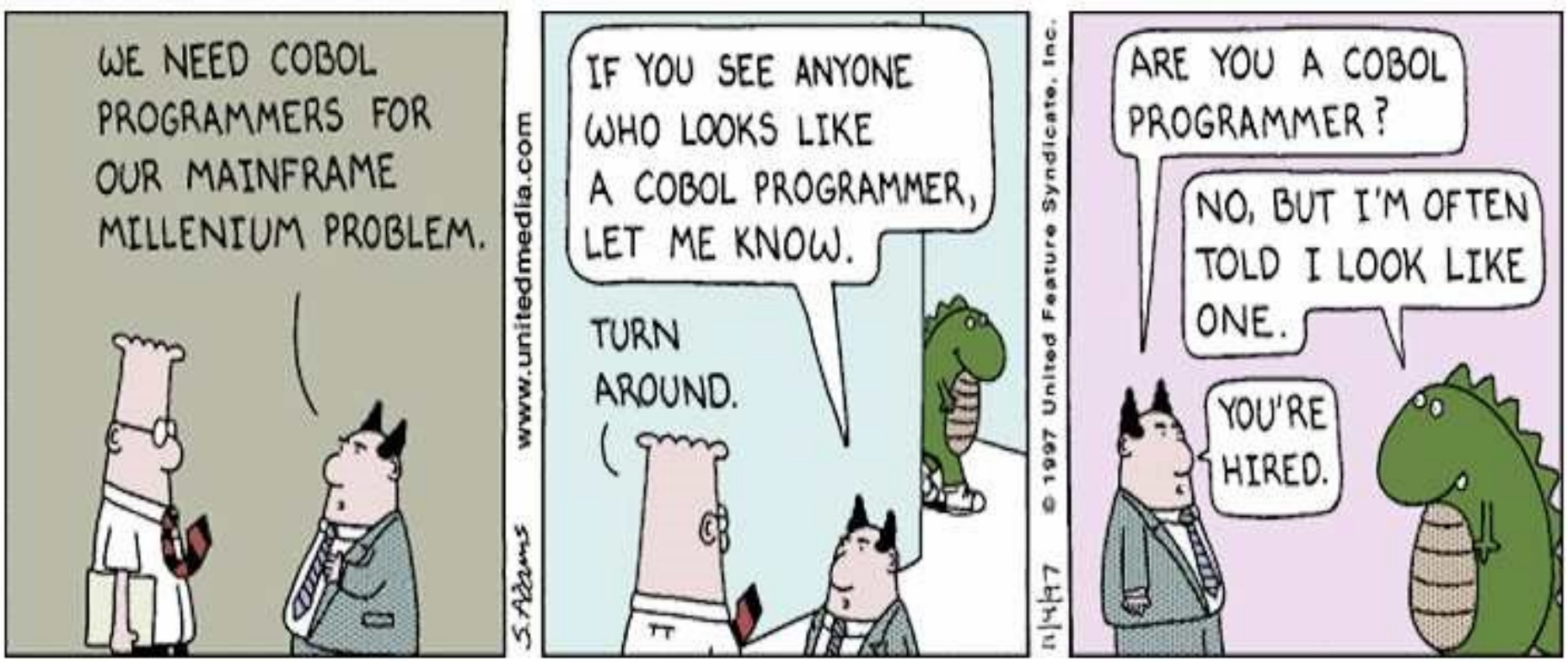
```
// the main program (this is a comment)

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
```

Source: http://www.engin.umd.umich.edu/CIS/course.des/cis400/algol/algol.html, October 2008

# Computerizing Business Records: COBOL



# Computerizing Business Records: COBOL




y2k problem: dates yy to save memory space, versus yyyy; misinterpretation of differences between dates

# Computerizing Business Records: COBOL

- Story a bit opposite to Algol 60…
  - Goal: common language for business applications
  - Has been used for 65 years for business
  - But little effect on design of subsequent languages (only PL/I)

# COBOL Historical Background

- Based on FLOW-MATIC designed at UNIVAC
- Grace Hopper at Univac: “mathematical programs should be written in mathematical notation, data processing programs should be written in English statements”

# COBOL Historical Background

- Based on FLOW-MATIC
- 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

# COBOL Design Process

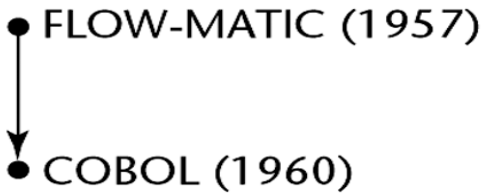
- First Design Meeting (Pentagon) - May 1959
- Design goals
  - Must look like simple English
  - Must be easy to use, even if that means it will be less powerful
  - Must broaden the base of computer users
  - Must not be biased by current compiler problems
- Design committee members were all from computer manufacturers and DoD branches
- Design Problems: arithmetic expressions? subscripts? Fights among manufacturers

# 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 (strong part: ideal for business and accounting reports)

**Figure 2.4**

Genealogy of COBOL



# COBOL: DoD Influence

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

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  ShortestProgram.

PROCEDURE DIVISION.
DisplayPrompt.
    DISPLAY "I did it".
    STOP RUN.
```

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.
* Uses the ACCEPT and DISPLAY verbs to accept a student record
* from the user and display some of the fields.  Also shows how
* the ACCEPT may be used to get the system date and time.

* The YYYYMMDD in "ACCEPT  CurrentDate FROM DATE YYYYMMDD."
* is a format command that ensures that the date contains a
* 4 digit year.  If not used, the year supplied by the system will
* only contain two digits which may cause a problem in the year 2000.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId       PIC 9(7).
   02  StudentName.
       03 Surname      PIC X(8).
       03 Initials     PIC XX.
   02  CourseCode      PIC X(4).
   02  Gender          PIC X.

* YYMMDD
01 CurrentDate.
   02  CurrentYear     PIC 9(4).
   02  CurrentMonth    PIC 99.
   02  CurrentDay      PIC 99.

* YYDDD
01 DayOfYear.
   02  FILLER          PIC 9(4).
   02  YearDay         PIC 9(3).


* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurrentHour     PIC 99.
   02  CurrentMinute   PIC 99.
   02  FILLER          PIC 9(4).


PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details using template below".
    DISPLAY "Enter - ID,Surname,Initials,CourseCode,Gender"
    DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE YYYYMMDD.
    ACCEPT  DayOfYear FROM DAY YYYYDDD.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is ", Initials SPACE Surname.
    DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurrentHour ":" CurrentMinute.
    STOP RUN.
```

Source: http://www.csis.ul.ie/COBOL/examples/default.htm#SimplePrograms, University of Limerick, Ireland

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  ShortestProgram.

PROCEDURE DIVISION.
DisplayPrompt.
    DISPLAY "I did it".
    STOP RUN.
```

# Shortest program

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.
* Uses the ACCEPT and DISPLAY verbs to accept a student record
* from the user and display some of the fields.  Also shows how
* the ACCEPT may be used to get the system date and time.

* The YYYYMMDD in "ACCEPT  CurrentDate FROM DATE YYYYMMDD."
* is a format command that ensures that the date contains a
* 4 digit year.  If not used, the year supplied by the system will
* only contain two digits which may cause a problem in the year 2000.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId       PIC 9(7).
   02  StudentName.
       03 Surname      PIC X(8).
       03 Initials     PIC XX.
   02  CourseCode      PIC X(4).
   02  Gender          PIC X.

* YYMMDD
01 CurrentDate.
   02  CurrentYear     PIC 9(4).
   02  CurrentMonth    PIC 99.
   02  CurrentDay      PIC 99.

* YYDDD
01 DayOfYear.
   02  FILLER          PIC 9(4).
   02  YearDay         PIC 9(3).


* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurrentHour     PIC 99.
   02  CurrentMinute   PIC 99.
   02  FILLER          PIC 9(4).


PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details using template below".
    DISPLAY "Enter - ID,Surname,Initials,CourseCode,Gender"
    DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE YYYYMMDD.
    ACCEPT  DayOfYear FROM DAY YYYYDDD.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is ", Initials SPACE Surname.
    DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurrentHour ":" CurrentMinute.
    STOP RUN.
```

Source: http://www.csis.ul.ie/COBOL/examples/default.htm#SimplePrograms, University of Limerick, Ireland

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  ShortestProgram.

PROCEDURE DIVISION.
DisplayPrompt.
    DISPLAY "I did it".
    STOP RUN.
```

Separate data and procedure division

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.
* Uses the ACCEPT and DISPLAY verbs to accept a student record
* from the user and display some of the fields.  Also shows how
* the ACCEPT may be used to get the system date and time.

* The YYYYMMDD in "ACCEPT  CurrentDate FROM DATE YYYYMMDD."
* is a format command that ensures that the date contains a
* 4 digit year.  If not used, the year supplied by the system will
* only contain two digits which may cause a problem in the year 2000.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId       PIC 9(7).
   02  StudentName.
       03 Surname      PIC X(8).
       03 Initials     PIC XX.
   02  CourseCode      PIC X(4).
   02  Gender          PIC X.

* YYMMDD
01 CurrentDate.
   02  CurrentYear     PIC 9(4).
   02  CurrentMonth    PIC 99.
   02  CurrentDay      PIC 99.

* YYDDD
01 DayOfYear.
   02  FILLER          PIC 9(4).
   02  YearDay         PIC 9(3).


* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurrentHour     PIC 99.
   02  CurrentMinute   PIC 99.
   02  FILLER          PIC 9(4).


PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details using template below".
    DISPLAY "Enter - ID,Surname,Initials,CourseCode,Gender"
    DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE YYYYMMDD.
    ACCEPT  DayOfYear FROM DAY YYYYDDD.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is ", Initials SPACE Surname.
    DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurrentHour ":" CurrentMinute.
    STOP RUN.
```

Source: http://www.csis.ul.ie/COBOL/examples/default.htm#SimplePrograms, University of Limerick, Ireland

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  ShortestProgram.

PROCEDURE DIVISION.
DisplayPrompt.
    DISPLAY "I did it".
    STOP RUN.
```

Instead of data types, "Declaration by example" of picture of storage required for data

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.
* Uses the ACCEPT and DISPLAY verbs to accept a student record
* from the user and display some of the fields.  Also shows how
* the ACCEPT may be used to get the system date and time.

* The YYYYMMDD in "ACCEPT  CurrentDate FROM DATE YYYYMMDD."
* is a format command that ensures that the date contains a
* 4 digit year.  If not used, the year supplied by the system will
* only contain two digits which may cause a problem in the year 2000.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId       PIC 9(7).
   02  StudentName.
       03 Surname      PIC X(8).
       03 Initials     PIC XX.
   02  CourseCode      PIC X(4).
   02  Gender          PIC X.

* YYMMDD
01 CurrentDate.
   02  CurrentYear     PIC 9(4).
   02  CurrentMonth    PIC 99.
   02  CurrentDay      PIC 99.

* YYDDD
01 DayOfYear.
   02  FILLER          PIC 9(4).
   02  YearDay         PIC 9(3).


* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurrentHour     PIC 99.
   02  CurrentMinute   PIC 99.
   02  FILLER          PIC 9(4).


PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details using template below".
    DISPLAY "Enter - ID,Surname,Initials,CourseCode,Gender"
    DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE YYYYMMDD.
    ACCEPT  DayOfYear FROM DAY YYYYDDD.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is ", Initials SPACE Surname.
    DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurrentHour ":" CurrentMinute.
    STOP RUN.
```

Template or picture of the storage:
9(7) = 7 character digit
X(8) = 8 character alphabetic

Source: http://www.csis.ul.ie/COBOL/examples/default.htm#SimplePrograms, University of Limerick, Ireland

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  ShortestProgram.

PROCEDURE DIVISION.
DisplayPrompt.
    DISPLAY "I did it".
    STOP RUN.
```

# Note date warning

```
      $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  Michael Coughlan.
* Uses the ACCEPT and DISPLAY verbs to accept a student record
* from the user and display some of the fields.  Also shows how
* the ACCEPT may be used to get the system date and time.

* The YYYYMMDD in "ACCEPT  CurrentDate FROM DATE YYYYMMDD."
* is a format command that ensures that the date contains a
* 4 digit year.  If not used, the year supplied by the system will
* only contain two digits which may cause a problem in the year 2000.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId       PIC 9(7).
   02  StudentName.
       03 Surname      PIC X(8).
       03 Initials     PIC XX.
   02  CourseCode      PIC X(4).
   02  Gender          PIC X.

* YYMMDD
01 CurrentDate.
   02  CurrentYear     PIC 9(4).
   02  CurrentMonth    PIC 99.
   02  CurrentDay      PIC 99.

* YYDDD
01 DayOfYear.
   02  FILLER          PIC 9(4).
   02  YearDay         PIC 9(3).


* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurrentHour     PIC 99.
   02  CurrentMinute   PIC 99.
   02  FILLER          PIC 9(4).


PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details using template below".
    DISPLAY "Enter - ID,Surname,Initials,CourseCode,Gender"
    DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE YYYYMMDD.
    ACCEPT  DayOfYear FROM DAY YYYYDDD.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is ", Initials SPACE Surname.
    DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurrentHour ":" CurrentMinute.
    STOP RUN.
```

Source: http://www.csis.ul.ie/COBOL/examples/default.htm#SimplePrograms, University of Limerick, Ireland

# BASIC (1971)

- BASIC meaning?

# BASIC (1971)

- BASIC meaning?
  Beginners All purpose Symbolic Instruction code

# BASIC (1971)

- BASIC meaning?
  Beginners All purpose Symbolic Instruction code

- Initially only 14 different type of statements and a single data type (floating point)!

# BASIC (1971)

- 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 - new concept…

# BASIC (1971)

- 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 - new concept…
- Current popular dialect:  Visual BASIC

# The Beginning of Timesharing: BASIC

- Designed by Kemeny & Kurtz at Dartmouth
- First widely used language with time sharing. Terminals connected to remote computers (before punch cards or tapes)


http://gt.ibnlive.in.com.akadns.net/photogallery/15806-13.html

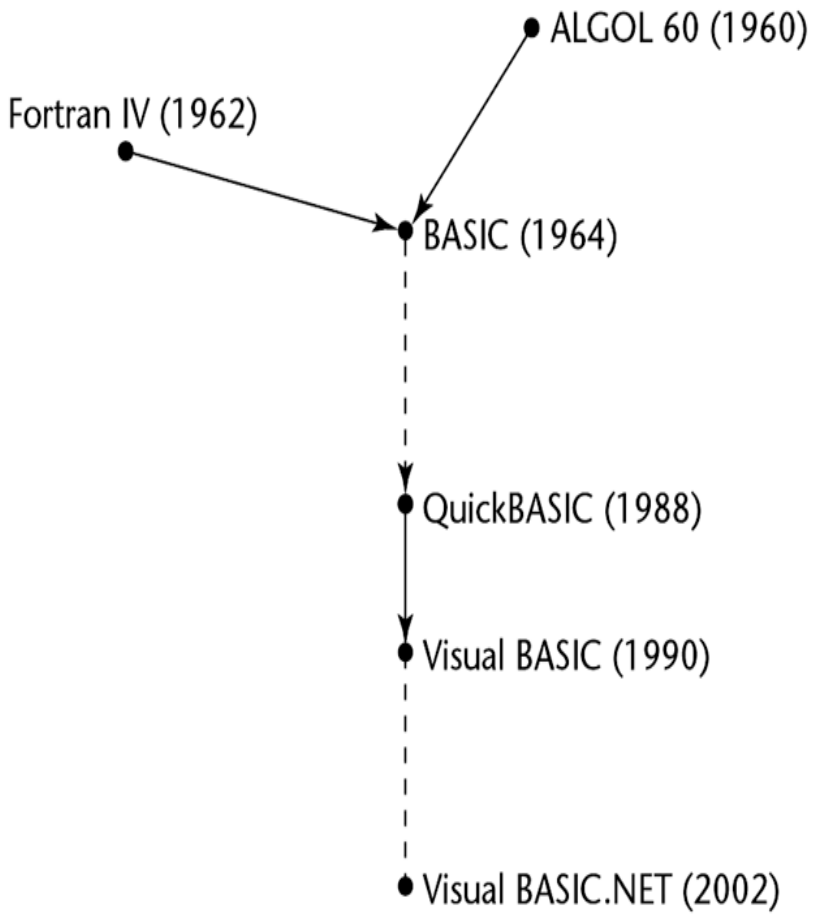
# BASIC Evaluation

- First widely used language that used terminals
- Design largely from FORTRAN, some from ALGOL
- Criticized for poor structure of programs
- Readability and reliability
- Resurgence by Visual Basic in 1990
  - GUI
  - VB .NET

# BASIC Evaluation

**Figure 2.5**

Genealogy of BASIC



```
CLS
x = INT(RND * 10) + 1
PRINT "I am thinking of a number between 1 and 10, can you guess it?"
PRINT "You have 3 chances"
INPUT "What is it, Chance 1"; i%
IF i% = x GOTO win
IF i% x THEN PRINT "Guess lower!"
INPUT "Chance 2"; i%
IF i% = x GOTO win
IF i% x THEN PRINT "Guess lower!"
INPUT "Chance 3, last chance"; i%
IF i% = x GOTO win
IF i% x THEN GOTO loose
win: CLS
COLOR 9
PRINT "Congratulations!!! You guessed right"
END

lose: CLS
PRINT "You lost!"
PRINT
PRINT "The correct number was "; x
```

# 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
    - SHARE user group
  - Business computing
    - IBM 1401, 7080 computers
    - COBOL
    - GUIDE user group

# 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
    - SHARE user group
  - Business computing
    - IBM 1401, 7080 computers
    - COBOL
    - GUIDE user group

Goal: Large-scale attempt at language that can be used for variety of problems

# PL/I: Design Process

- Designed in five months by the 3 x 3 Committee
  - Three members from IBM, three members from SHARE
- Initially called NPL (New Programming Language)
- Name changed to PL/I in 1965 (to avoid confusion with “National Physical Laboratory” in England)

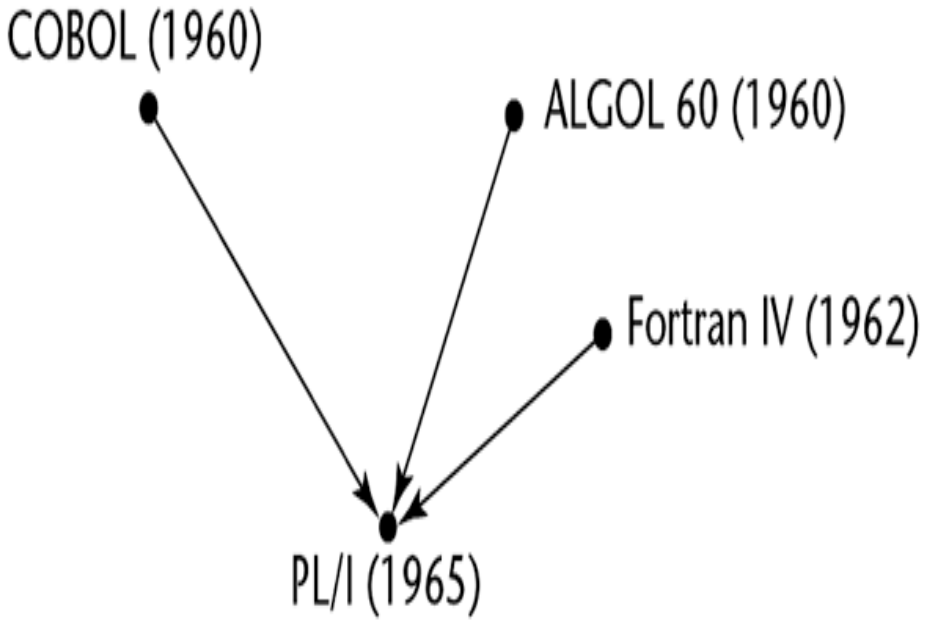
# PL/I: Language overview

- Included best parts of
  - ALGOL 60: recursion and block structure
  - Fortran IV: separate compilation, communication through global data
  - COBOL 60: data structures, i/o, report generation

# PL/I: Evaluation

**Figure 2.6**

Genealogy of PL/I



# PL/I: Evaluation

- PL/I contributions
  - First concurrently executing subprograms
  - First exception handling
  - Switch-selectable recursion (could turn off recursion for efficiency)
  - First pointer data type
  - First array cross sections (e.g., referencing third row of matrix)
- Concerns
  - Many new features were poorly designed
  - Too large and too complex
- No longer used today

```
SHELL:   PROCEDURE OPTIONS (MAIN);
         DECLARE
           ARRAY(50) FIXED BIN(15),
           (K,N) FIXED BIN(15);

         GET LIST(N);
         GET EDIT((ARRAY(K) DO K = 1 TO N));
         PUT EDIT((ARRAY(K) DO K = 1 TO N));
         CALL BUBBLE(ARRAY,N);

END BUBBLE;

BUBBLE:   PROCEDURE(ARRAY,N); /* BUBBLE SORT*/
          DECLARE (I,J) FIXED BIN(15);
          DECLARE S BIT(1);         /* SWITCH */
          DECLARE Y FIXED BIN(15); /* TEMPO */
          DO I = N-1 BY -1 TO 1;
            S = '1'B;
            DO J = 1 TO I;
              IF X(J)>X(J+1) THEN DO;
                S = '0'B;
                Y = X(J);
                X(J) = X(J+1);
                X(J+1) = Y;
                END;
              END;
            IF S THEN RETURN;
            END;
          RETURN;
          END SRT;
```

What is this program doing?

Source: http://www.engin.umd.umich.edu/CIS/course.des/cis400/pl1/pl1bubble.html verified on Sep 9, 2008

# Two Early Dynamic Languages: APL and SNOBOL

- Characterized by dynamic typing and dynamic storage allocation

# Two Early Dynamic Languages: APL and SNOBOL

- Characterized by dynamic typing and dynamic storage allocation

  What is dynamic typing?

# Two Early Dynamic Languages: APL and SNOBOL

- Characterized by dynamic typing and dynamic storage allocation

  What is dynamic typing?
  Examples today: Python; Javascript; variable gets its type when assigned value at run time:
  a=10; a=5.5; more later…

# 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
- Considered “throw away” language: write quickly, then discard for readability and maintaining hard
- Initially used on IBM printing terminals that had optional print balls with odd character set
- Language still in use, though not widely; minimal changes

# APL: A Programming Language

- Readability: Example APL code and special keyboard for computing matrix determinant: (http://www.computerhistory.org/atchm/the-apl-programming-language-source-code/)


```
      ∇DET[⎕]∇
     ∇ Z←DET A;B;P;I
[1]    I←⎕IO
[2]    Z←1
[3]   L:P←(|A[;I])⍳⌈/|A[;I]
[4]    →(P=I)/LL
[5]    A[I,P;]←A[P,I;]
[6]    Z←-Z
[7]   LL:Z←Z×B←A[I;I]
[8]    →(0 1 ∨.=Z,1↑⍴A)/0
[9]    A←1 1 ↓A-(A[;I]÷B)∘.×A[I;]
[10]   →L
[11]  ⍝EVALUATES A DETERMINANT
      ∇
```

# 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

https://motherboard.vice.com/en_us/article/78x5ba/this-70-year-old-programmer-is-preserving-an-ancient-coding-language-on-github

# 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)

Why slower?

# 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)

Why slower? Dynamically typed…

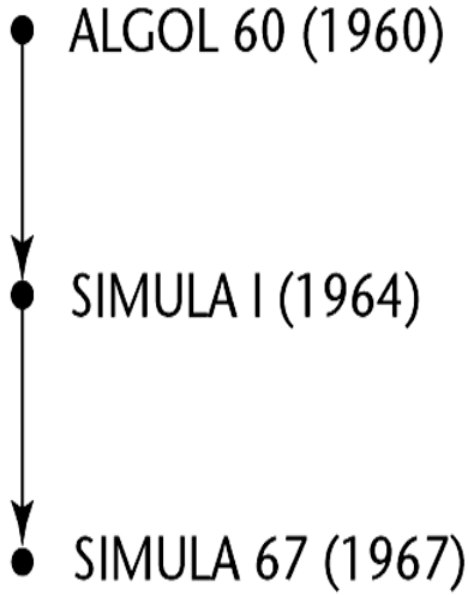
# The Beginning of Data Abstraction: SIMULA 67

- Designed primarily for system simulation in Norway by Nygaard and Dahl
- Did not achieve widespread use
- Based on ALGOL 60 (and initial SIMULA I)
- Primary Contributions
  - Coroutines - a kind of subprogram (allow restart where left off for simulations)
  - Classes, objects, and inheritance

# The Beginning of Data Abstraction: SIMULA 67

**Figure 2.7**

Genealogy of SIMULA 67



# Orthogonal Design: ALGOL 68

- From the continued development of ALGOL 60
- Design is based on the concept of orthogonality
  - A few basic concepts, plus a few combining mechanisms (eg, combining few primitive types and structures for user defined data structures)

# Orthogonal Design: ALGOL 68

- Source of several new ideas (even though the language itself never achieved widespread use)
- Had strong influence on subsequent languages, especially Pascal, C, and Ada
- Popularity reduced due to complicated grammar used to describe language (mistake of Algol 60)

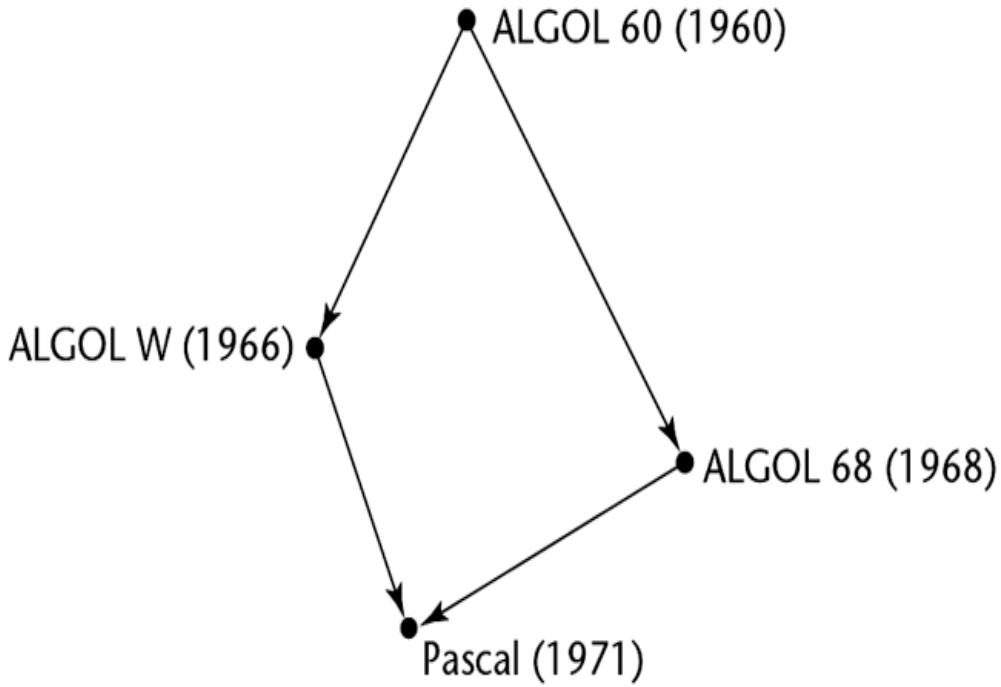
# Pascal - 1971

- 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

# Pascal - 1971

**Figure 2.9**

Genealogy of Pascal



```
program temperature(output) ;

{ Program to convert temperatures from
 Fahrenheit to Celsius. }

const
    MIN = 32 ;
    MAX = 50 ;
    CONVERT = 5 / 9 ;

var
    fahren: integer ;
    celsius: real ;

begin
    writeln('Fahrenheit     Celsius') ;
    writeln('-----------    -------') ;
    for fahren := MIN to MAX do begin
        celsius := CONVERT * (fahren - 32) ;
        writeln(fahren: 5, celsius: 18: 2) ;
 end ;
end.
```

Source: http://www.informatik.uni-hamburg.de/RZ/software/SUNWspro/pascal/user_guide/using_pascalug.doc.html, verified on Sep 11, 2008l

# C - 1972

- Designed originally for systems programming (at Bell Labs by Dennis Richie)
- Evolved primarily from BCLP, B (untyped), but also ALGOL 68 (for, switch, pointers)
- Powerful set of operators, but poor type checking
- Initially spread through UNIX (free; widespread use)
- Many areas of application

# C - 1972

- Standard for a long time: Kernaghan and Ritchie 1978 book
- 1989 ANSI standard

```
#include <stdio.h>

inline float convert(float f)  {
    return ((5.0/9.0) * (f - 32));
}

int main()  {
    float f;
    for(f = -40; f <= 220; f += 10) {
      printf("%f degrees fahrenheit = %f degrees celsius.\n", f, convert(f));
    }
}
```

# Programming Based on Logic: Prolog

- Developed, by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski ( University of Edinburgh) early 1970s
- 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
- Highly inefficient, small application areas

```
mother(joanne,jake).
father(vern,joanne).


grandparent(X,Z) :=
  parent(X,Y),
  parent(Y,Z).
```

```
Query: father(bob,darcie).
```

# History's Largest Design Effort: Ada

- Huge design effort, involving hundreds of people, much money, and about eight years
  - Strawman requirements (April 1975)
  - Woodman requirements (August 1975)
  - Tinman requirements (1976)
  - Ironman equipments (1977)
  - Steelman requirements (1978)
- Named Ada after Augusta Ada Byron (1815-1841), the first programmer
- Developed by DoD

# Ada Evaluation

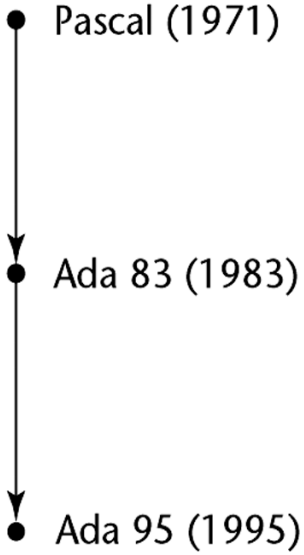
- Competitive design
- Included all that was then known about software engineering and language design
- First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed
- Too large and complex

# Ada 95

- Ada 95 (began in 1988), included among other features OOP and more flexible libraries
- Popularity suffered because the DoD no longer requires its use but also because of popularity of C++

**Figure 2.11**

Genealogy of Ada



```
1  -- main.adb:  main program for approximate string matching
2
3  with
4    Ada.Command_Line,      -- Access to external execution env (Ada95 A.15)
5    Ada.Text_IO,           -- Usual string oriented IO package
6    Approx;                -- User defined function
7  use Ada;
8
9  procedure main is
10    K : constant Natural := Integer'Value (Command_Line.Argument(1));
11    M : constant Boolean := Approx (K,
12       Command_Line.Argument (2), Command_Line.Argument (3));
13 begin
14    if M then
15       Text_IO.Put_Line ("Match.");
16    else
17       Text_IO.Put_Line ("No match.");
18    end if;
19 end main;
```

Source: http://www.cs.fit.edu/~ryan/ada/programs/access/main-adb.html, verified on Sep 11, 2008

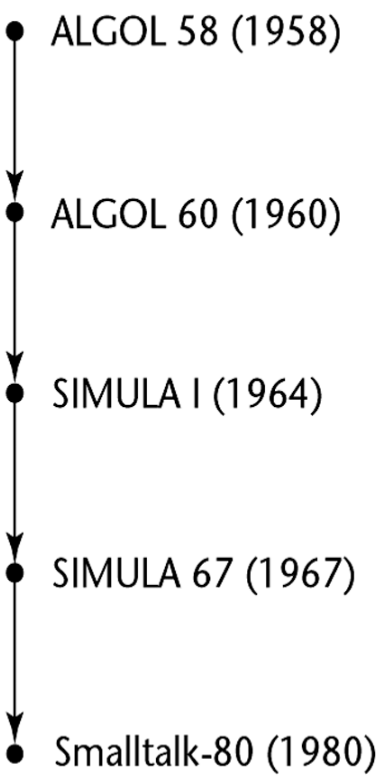
# Object-Oriented Programming: Smalltalk (1980)

- Developed at Xerox PARC, initially by Alan Kay (PhD late 1960s), later by Adele Goldberg
- First full implementation of an object-oriented language
- populated by objects; all computing by sending a message to an object to invoke method
- Pioneered the graphical user interface design
- Promoted OOP

# Object-Oriented Programming: Smalltalk

**Figure 2.12**

Genealogy of Smalltalk



# Object-Oriented Programming: Smalltalk

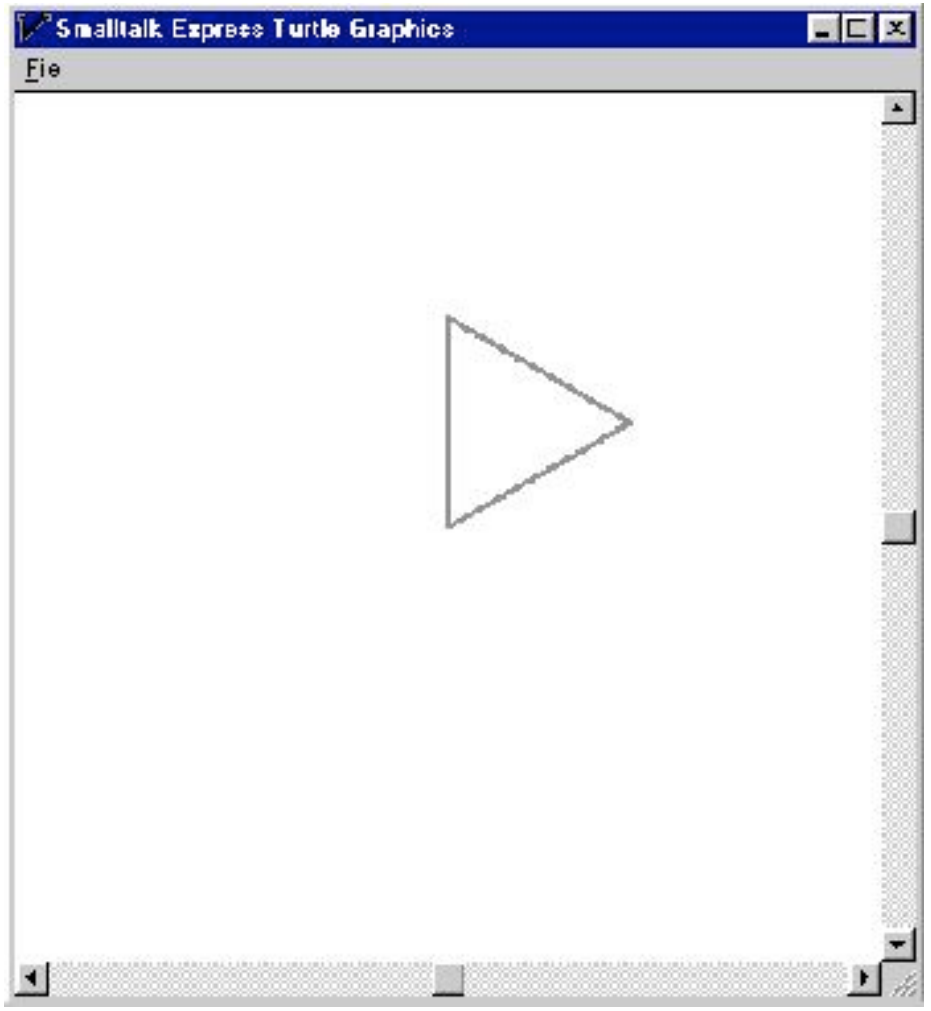
- Developed at Xerox PARC, initially by Alan Kay (PhD late 1960s), later by Adele Goldberg (developed until Smalltalk-80)
- First full implementation of an object-oriented language
- all objects; all computing by sending a message to an object to invoke one of its methods
- Pioneered the graphical user interface design
- Promoted OOP

# Example Smalltalk class definition

```
"The following is a class definition, instantiations of which can draw equilateral polygons of any
number of sides"
class name
superclass
instance variable names
numSides
sideLength
"Class methods"

  "Create an instance"
  new
^ super new getPen

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

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

# Graphics with Smalltalk

```
Window turtleWindow: 'Turtle Graphics'.
     Turtle
          defaultNib: 2;
          foreColor: ClrDarkgray;
          home;
          go: 100;
          turn: 120;
          go: 100;
          turn: 120;
          go: 100;
          turn: 120
```



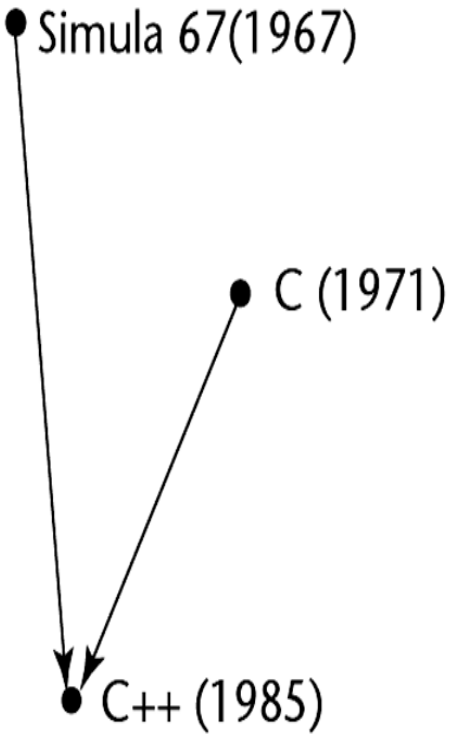
# Combining Imperative and Object-Oriented Programming: C++

- Developed at Bell Labs by Stroustrup in 1980
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67
- Provides exception handling
- A large and complex language, in part because it supports both procedural and OO programming
- Rapidly grew in popularity, along with OOP
- good and inexpensive compilers; backwards compatible with C
- ANSI standard approved in November 97
- Microsoft's version (released with .NET in 2002): Managed C++

# Combining Imperative and Object-Oriented Programming: C++

**Figure 2.13**

The ancestry of C++



# Objective C

- Another hybrid with both imperative and OOP
- Initially consisted of C, plus classes and message passing like SmallTalk
- Steve Jobs founded NeXT computer systems used; then Apple bought NeXT and Objective C
- Language of MAC OS X, iphone software, which increased popularity
- Now Swift (safer, e.g. removes unsafe pointer management)

# An Imperative-Based Object-Oriented Language: Java

- Developed at Sun in the early 1990s
  - What applications originally developed for?

# An Imperative-Based Object-Oriented Language: Java

- Developed at Sun in the early 1990s
  - What applications originally developed for?
    toasters, microwave ovens, interactive TV systems

# An Imperative-Based Object-Oriented Language: Java

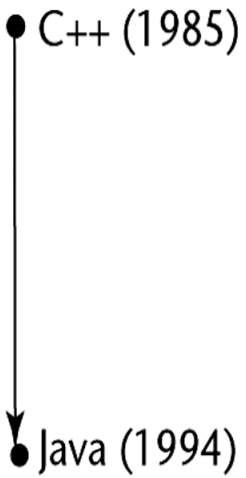
- Developed at Sun in the early 1990s
  - C and C++ were not satisfactory for embedded electronic devices
  - Reliability was a primary goal
- Based on C++
  - Significantly simplified (e.g., does not include `union`, pointer arithmetic, and half of the assignment coercions of C ++)
  - Supports OOP
  - Has references, but not pointers
  - Includes support for applets and a form of concurrency

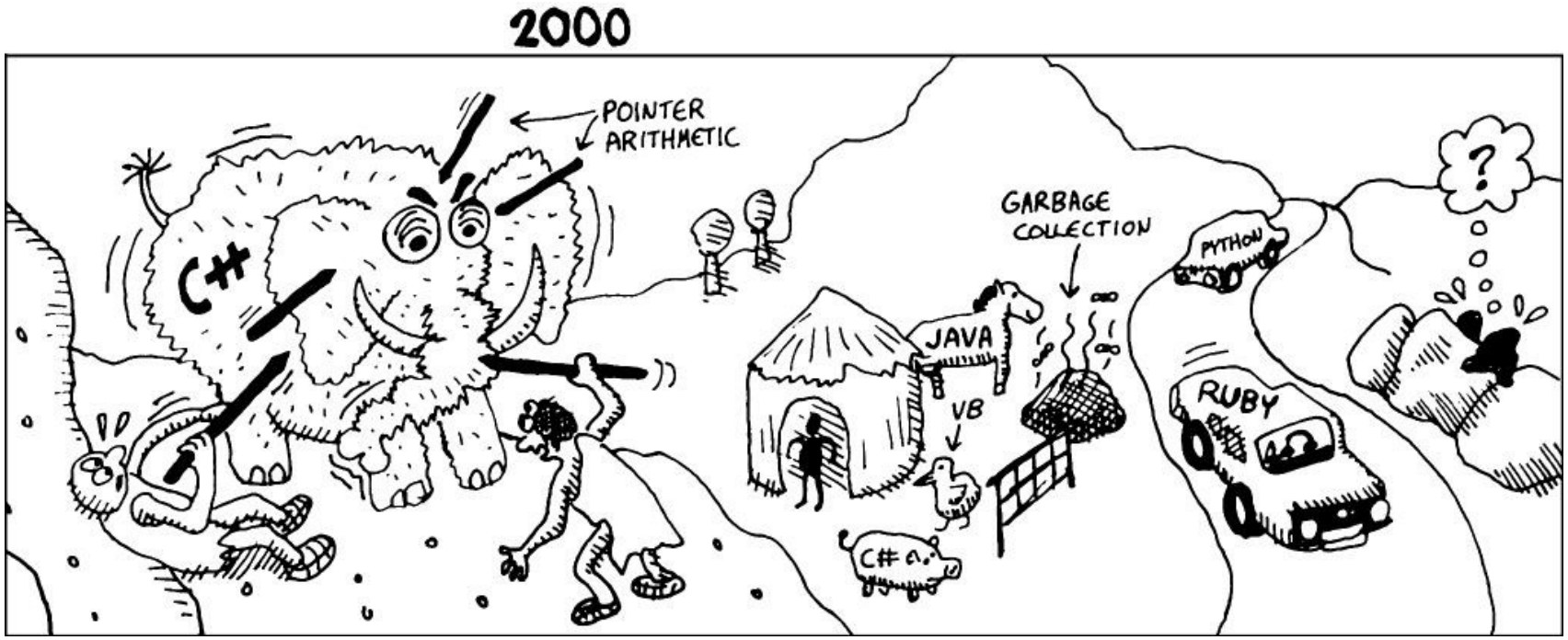
# Java Evaluation

- Eliminated many unsafe features of C++
- Supports concurrency
- Libraries for applets, GUIs, database access
- Widely used for Web programming
- Use increased faster than any previous language
- Most recent version, Java SE8, appeared in 2014

**Figure 2.14**

The ancestry of Java






Source: https://onionesquereality.wordpress.com/tag/lisp/
referencing “Land of Lisp: Learn to Program in List, One Game at a time” by M. D. Conrad Barski.

# Scripting Languages for the Web

- JavaScript (mid 1990s)
  - –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 to provide tool for tracking visitors to his website
  - –A server-side HTML-embedded scripting language, often used for form processing and database access through the Web

Example Javascript

From Burt Rosenberg:

http://www.cs.miami.edu/home/burt/learning/Csc517.101/workbook/partition.html

# Scripting Languages (and beyond)

- Python
  - –Named after?
  - –An OOP interpreted scripting language
  - –Type checked but dynamically typed
  - –Used initially for CGI (Common Gateway Interface) programming; set of standards that define how information is exchanged between the web server and a custom script.
  - –More recently prominent in other areas, such as machine learning and scientific computing

# Scripting Languages (and beyond)

- Python
  - –Named after? Monty Python
  - –An OOP interpreted scripting language
  - –Type checked but dynamically typed
  - –Used initially for CGI (Common Gateway Interface) programming; set of standards that define how information is exchanged between the web server and a custom script.
  - –More recently prominent in other areas, such as machine learning and scientific computing

# Python example

- Partial example from ipython notebook (Luis Gonzalo Sanchez Giraldo)

```
import numpy as np
import matplotlib.pyplot as plt

# instantiate an empty network
my_net = DNN.Net()
# add layers to my_net in a bottom up fashion
my_net.addLayer(n_in=2, n_out=4, activation='relu')
my_net.addLayer(n_out=1, activation='sigmoid')

solver_params = {'lr_rate': 0.01,
                 'momentum': 0.9,  \
                 'solver': 'sgd'}
my_solver = DNN.Solver(solver_params)

def addLayer(self, n_in=None, n_out=None, activation=None):
    assert n_in is not None or self.n_layer > 0, "n_in must be specified for
    input layer"
    assert n_out is not None, "n_out must be specified"
    assert activation is not None, "activation must be specified"

    self.layers += [Layer(n_in, n_out, activation)]
    self.n_layer += 1
```

# Scripting Languages for the Web

- Other scripting languages: Perl, Lua
- Ruby (1996); popularized by Ruby on rails (2004)
- 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
  - Purely interpreted

# A C-Based Language for the New Millennium: C#

- Part of the .NET development platform (2000)
- Based on C++ , Java, and Delphi
- Provides a language for component-based software development; can easily combine components from variety of languages
- 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
- All compiled into same intermediate form

# Summary

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