# History of Programming Languages

## Plankalkül

Plankalkül - 1945

Never implemented

Advanced data structures

- floating point, arrays, records

Invariants

# FORTRAN 1

FORTRAN I – 1957 (FORTRAN 0 - 1954 - not implemented)

- 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

# FORTRAN I

Factors effecting design decisions

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

Variable names could have up to six characters

Posttest counting loop (DO)

Formatted i/o

User-defined subprograms

Three-way selection statement (arithmetic IF)

No data typing statements

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 the 704

Code was very fast

Quickly became widely used

# FORTRAN II

FORTRAN II - 1958

- Independent compilation
- Fix the bugs

FORTRAN IV - 1960-62

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

# FORTRAN 77 and 90

FORTRAN 77 – 1978

- Character string handling
- Logical loop control statement
- IF-THEN-ELSE statement

FORTRAN 90 – 1990

- Modules
- Dynamic arrays
- Pointers
- Recursion
- CASE statement
- Parameter type checking

## Fortran 77 example

```
      program circle
      real r, area

c This program reads a real number r and prints
c the area of a circle with radius r.

      write (*,*) 'Give radius r:'
      read  (*,*) r
      area = 3.14159*r*r
      write (*,*) 'Area = ', area

      stop
      end
```

# LISP

LISP – 1959

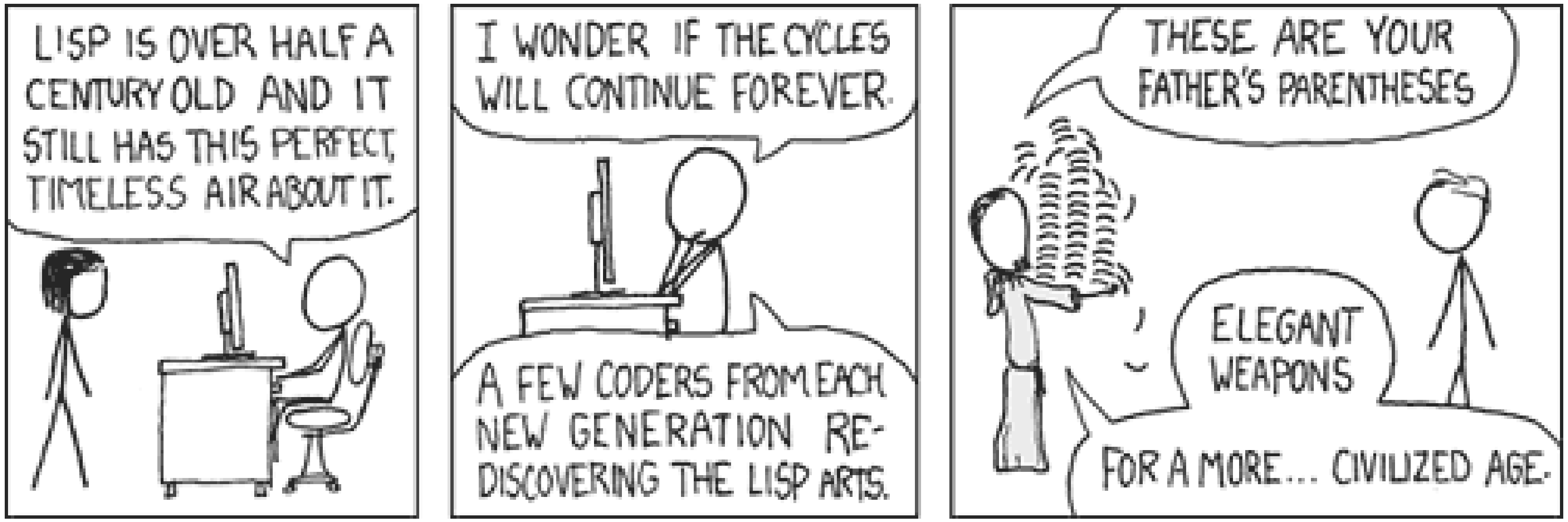
- LISt Processing language
- (Designed at MIT by McCarthy)
- AI research needed a language that:
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)

- Only two data types: atoms and lists
- Syntax is based on lambda calculus
- 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, Miranda, and Haskell are related languages

# LISP examples (actually Scheme)

```
(define (factorial  n)
   (if (= n 0)
      ;true
      1
     ;false
     (* n (factorial (- n  1)))
   )
)
(define (inList searchFor L)
   (cond ((null? L) #f)
         ((equal? searchFor (car L)) #t)
         (else (inList searchFor (cdr L))
         ) ;end else
   ) ;end cond
)
```

# LISP


LISP IS OVER HALF A CENTURY OLD AND IT STILL HAS THIS PERFECT, TIMELESS AIR ABOUT IT.
I WONDER IF THE CYCLES WILL CONTINUE FOREVER.
A FEW CODERS FROM EACH NEW GENERATION RE-DISCOVERING THE LISP ARTS.
THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

## Algol 58

Goals of the language:

- Close to mathematical notation
- Good for describing algorithms
- Must be translatable to machine code

Language 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

# Algol 60

New Features:

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

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 idefined (BNF)

Failure:

- Never widely used, especially in U.S.
- No i/o and the character set made programs nonportable
- Too flexible--hard to implement
- Entrenchment of FORTRAN
- Lack of formal syntax description
- Lack of support of IBM

# Algol example

```
procedure Absmax(a) Size:(n, m) Result:(y)
  Subscripts:(i, k);
value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a,
  of size n by m,
    is transferred to y, and the subscripts of this
  element to i and k;
begin
    integer p, q;
    y := 0; i := k := 1;
    for p := 1 step 1 until n do
        for q := 1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
```

# COBOL – 1960

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

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
  - Data defined by program requirements, not hardware limitations - PIC S9(5), 9(8)V99, X(20)

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

# COBOL example

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  Iteration-If.
AUTHOR.  Not me :-).

DATA DIVISION.
WORKING-STORAGE SECTION.
01  Num1           PIC 9  VALUE ZEROS.
01  Num2           PIC 9  VALUE ZEROS.
01  Result         PIC 99 VALUE ZEROS.
01  Operator       PIC X  VALUE SPACE.

PROCEDURE DIVISION.
Calculator.
    PERFORM 3 TIMES
       DISPLAY "Enter First Number      : " WITH NO ADVANCING
       ACCEPT Num1
       DISPLAY "Enter Second Number     : " WITH NO ADVANCING
       ACCEPT Num2
       DISPLAY "Enter operator (+ or *) : " WITH NO ADVANCING
       ACCEPT Operator
       IF Operator = "+" THEN
          ADD Num1, Num2 GIVING Result
       END-IF
       IF Operator = "*" THEN
          MULTIPLY Num1 BY Num2 GIVING Result
       END-IF
       DISPLAY "Result is = ", Result
    END-PERFORM.
    STOP RUN.
```

# COBOL example continued

```
PROCEDURE DIVISION.
MAIN-PARA.
OPEN INPUT IN-FILE OUTPUT OUT-FILE.
MOVE HEADING1 TO OUT-REC.
WRITE OUT-REC BEFORE ADVANCING 1 LINE.
MOVE "                         WEEKLY WAGES REPORT"
      TO OUT-REC.
WRITE OUT-REC BEFORE ADVANCING 1 LINE   .
MOVE HEADING1 TO OUT-REC.
WRITE OUT-REC BEFORE ADVANCING 1 LINE.
MOVE "WORKER CODE | HRS. WORKED | WAGE RATE | NET SALARY(RS.) "
 TO OUT-REC.
WRITE OUT-REC BEFORE ADVANCING 1 LINE.

READ IN-FILE AT END MOVE "Y" TO EOF.
PERFORM CALC-PARA UNTIL EOF = "Y".
DISPLAY "THE DETAILS HAVE  BEEN WRITTEN TO FILE Q7OUT.DAT".
CLOSE IN-FILE , OUT-FILE.
STOP RUN.

CALC-PARA.
IF IN-HRS-WORKED  > 42
      COMPUTE OUT-TOTAL-SALARY = 42 * IN-WAGE-RATE
        + ( IN-HRS-WORKED - 42 ) * 2 * IN-WAGE-RATE
ELSE
      COMPUTE OUT-TOTAL-SALARY = IN-HRS-WORKED * IN-WAGE-RATE.
MOVE IN-WORKER-CODE  TO OUT-WORKER-CODE.
MOVE IN-HRS-WORKED TO OUT-HRS-WORKED.
MOVE IN-WAGE-RATE TO OUT-WAGE-RATE.
MOVE OUT-FORMAT TO OUT-REC.
WRITE OUT-REC BEFORE ADVANCING 1 LINE.

READ IN-FILE AT END MOVE "Y" TO EOF.
```

# BASIC

Designed by Kemeny & Kurtz at Dartmouth (1964)

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

# Basic example

```
5 REM EXAMPLE BASIC PROGRAM
10 INPUT "What is your name: ", U$
20 PRINT "Hello "; U$
30 INPUT "How many stars do you want: ", N
40 S$ = ""
50 FOR I = 1 TO N
60 S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Do you want more stars? ", A$
100 IF LEN(A$) = 0 THEN GOTO 90
110 A$ = LEFT$(A$, 1)
120 IF A$ = "Y" OR A$ = "y" THEN GOTO 30
130 PRINT "Goodbye "; U$
140 END
```

# 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, nothing really new

Widely used language for teaching programming in colleges (but use is shrinking)

## Pascal example

```
program factorial(input,output);
const
  number = 5;
var
  result: integer;
function fact(n: integer): integer; (* demonstrate that a *)
var                                 (* function name is how a *)
    i, answer: integer;             (* function returns *)
begin                               (* its value *)
   fact := 1;
   answer := 1;
   if n > 1 then
      for i := 2 to n do
          answer := answer * i;
      fact := answer;
end;
begin                (* main program *)
   result := fact(number);
   writeln("Factorial of ", number, " is ", result);
end.
```

# C

Designed for systems programming (at Bell Labs by Dennis Richie) - 1972

Evolved primarily from B (which derived from BCPL – both were typeless), but also ALGOL 68

Powerful set of operators, but poor type checking

Initially spread through UNIX

# Contributors to C’s success

Highly portable

- Data types are close to concrete data types
- Operations easily implemented by computers or the day
- “Close to the machine”

Connection/integration with UNIX

Standard library concept

Powerful set of operators

Access to the hardware

## C example

```c
#include <stdio.h>

int max(int num1, int num2); /* forward declaration */

int main () {
   int a = 100, b = 200, ret;

   ret = max(a, b);
   printf( "Max value is : %d\n", ret );
   return 0;
}

int max(int num1, int num2) {
   int result;
   if (num1 > num2)
      result = num1;
   else
      result = num2;
   return result;
}
```

## Prolog

Developed at the University of Aix-Marseille, by Comerauer and Roussel, with some help from Kowalski at the University of Edinburgh – 1972

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

# Prolog example

```
start:- radius(R),
        circ(R,C),write('Circumference is '),write(C),nl,
        area(R,A),write('Area is '),write(A).

radius(R):- write('Radius= '),read(R).

circ(R,C):- C is 2*3.14*R.

area(R,A):- A is 3.14*R*R.
```

Running the program:

```
?- start.
Radius= |: 1.
Circumference is 6.28
Area is 3.14
```

## Prolog example

```
has(jack,apples).
has(ann,plums).
has(dan,money).
fruit(apples).
fruit(plums).

?- has(jack,X).                /* what does Jack have? */
X = apples
?- has(jack,_).                /* does Jack have something? */
yes
?- has(X,apples),has(Y,plums). /* who has apples and who has plums? */
X = jack
Y = ann
?- has(X,apples),has(X,plums). /* anyone have apples and plums? */
no
?- has(dan,X),fruit(X).        /* does Dan have fruit? */
no
?- has(X,Y),not fruit(Y).      /* does someone have something else? */
X = dan
Y = money
```

# Ada

Ada - 1983 (began in mid-1970s)

Huge design effort, involving hundreds of people, much money, and about eight years

Contributions:

- Packages - support for data abstraction
- Exception handling – elaborate
- Generic program units
- Concurrency - through the tasking model

Comments:

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

## Ada 95

Ada 95 (began in 1988)

Support for OOP through type derivation

Better control mechanisms for shared data (new concurrency features)

More flexible libraries

## Smalltalk

Smalltalk - 1972-1980

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 type binding)

Pioneered the graphical user interface

# Smalltalk example

```
"Prints the integers between 1 and 10, and a string
  stating whether each is even or odd"

1 to: 10 do: [:n |
  n even
      ifTrue: [
            Transcript show: n; show: ' is even'; cr ]
      ifFalse: [
            Transcript show: n; show: ' is odd'; cr ]
]

" create an array "
x := Array new: 4.

" add up all the values from the array"
sum := 0. 1 to: (x size)
  do: [:a | sum := sum + (x at: a)].
```

# C++

C++ - 1985
- Developed at Bell Labs by Stroustrup
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67, were 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

# Java

Java (1995)
- Developed at Sun in the early 1990s
- Based on C++
- Significantly simplified
- Supports only OOP
- Has references, but not pointers
- Includes support for applets and a form of concurrency