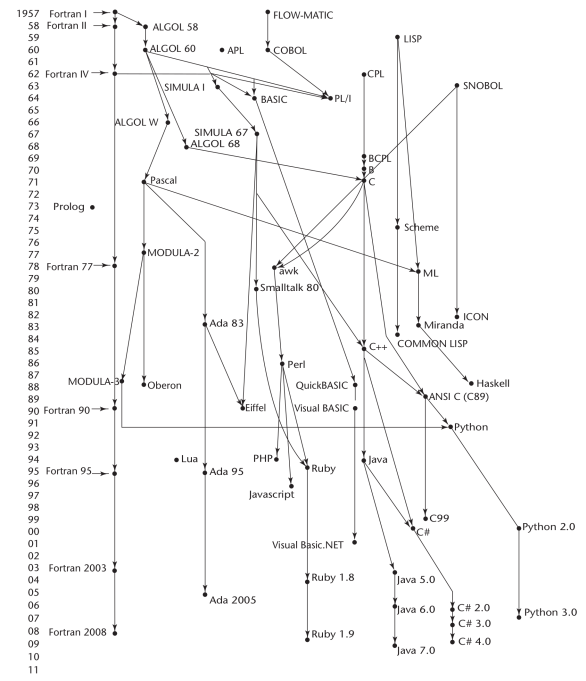
IA010: Principles of Programming Languages

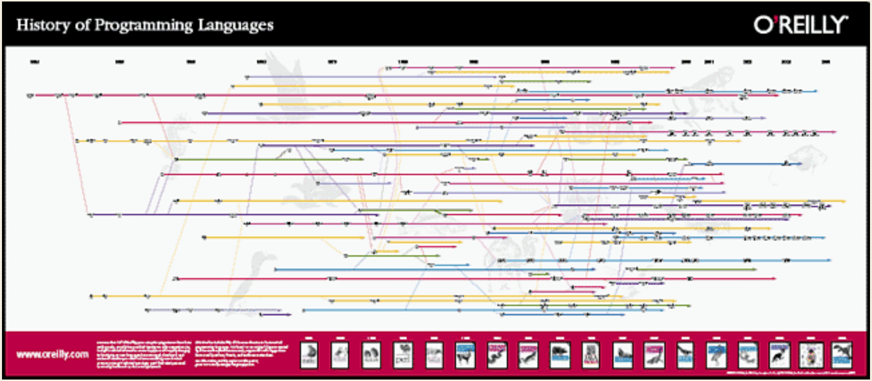
A bit of history

Achim Blumensath

blumens@fi.muni.cz

Faculty of Informatics, Masaryk University, Brno


1957
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
00
01
02
03
04
05
06
07
08
09
10
11
Fortran I
Fortran II
ALGOL 58
FLOW-MATIC
LISP
ALGOL 60
APL
COBOL
Fortran IV
CPL
SNOBOL
SIMULA I
BASIC
PL/I
ALGOL W
SIMULA 67
ALGOL 68
BCPL
B
C
Pascal
Prolog
Scheme
MODULA-2
Fortran 77
awk
ML
Smalltalk 80
ICON
Ada 83
Miranda
COMMON LISP
C++
Perl
MODULA-3
Oberon
QuickBASIC
Haskell
ANSI C (C89)
Eiffel
Visual BASIC
Fortran 90
Python
Lua
PHP
Java
Fortran 95
Ada 95
Ruby
Javascript
C99
C#
Python 2.0
Visual Basic.NET
Fortran 2003
Ruby 1.8
Java 5.0
Ada 2005
Java 6.0
C# 2.0
Python 3.0
C# 3.0
Fortran 2008
Ruby 1.9
C# 4.0
Java 7.0



http://archive.oreilly.com/pub/a/oreilly/news/languageposter_0504.html

Fortran (1957)

The language which started it all

The determined Real Programmer can write FORTRAN programs in any language.

Ed Post

Fortran

- "The IBM Mathematical **FOR**mula **TRAN**slating System"
- motivation: IBM 704 implemented floating-point instructions
- goal: at most 2 times slower than hand-written machine code **(this goal was achieved!)**
- primary use (then and today): numerical computing
- main characteristics:
  - the **first** high level language with arithmetical expressions
  - conditional statements (If) and loops (Do)
  - user-defined functions, but no recursion
  - formatted I/O
  - all memory is allocated at compile time (no stack, no heap) $\Rightarrow$ no support for recursive functions
- **huge success**, changed the way computers are used
- later versions: ..., Fortran 77, Fortran 90, Fortran 2008

```
C AREA OF A TRIANGLE - HERON'S FORMULA
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      INTEGER A,B,C
      READ(5,501) A,B,C
  501 FORMAT(3I5)
      IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) STOP 1
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C))
      WRITE(6,601) A,B,C,AREA
  601 FORMAT(4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= &
             ,F10.2,12HSQUARE UNITS)
      STOP
      END
```

LISP (1958)

Functional programming arrives

The greatest single programming language ever designed.

Alan Kay

LISP

- J. McCarthy (MIT)
- "**LIS**t **P**rocessing" (officially)
- "**L**ots of **I**rritating **S**uperfluous **P**arentheses"
- first **functional** language, to work with **lists** (AI)
- syntax based directly on **syntax trees**
- data structures: atoms and lists
- computation: expression evaluation
- based on **recursion** rather than loops
- **dynamic memory management** with **garbage collection**
- modern dialects: Scheme, Common LISP

Later important functional languages

- ML (1978) – syntax without parenthesis, imperative features
- Haskell (1988) – purely functional, lazy evaluation

```lisp
(defun factorial (n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1)))))

(defun -reverse (list)
  (let ((return-value '()))
    (dolist (e list) (push e return-value))
    return-value))

(defun comb (m list fn)
  (labels ((comb1 (l c m)
             (when (>= (length l) m)
               (if (zerop m) (return-from comb1 (funcall fn c)))
               (comb1 (cdr l) c m)
               (comb1 (cdr l) (cons (first l) c) (1- m)))))
    (comb1 list nil m)))

(comb 3 '(0 1 2 3 4 5) #'print)
```

ALGOL 58/ALGOL 60

A huge step forward

Algol 60 is a language so far ahead of its time, that it was not only an improvement on its predecessors, but also on nearly all its successors.

C. A. R. Hoare

ALGOL

- "**ALGO**rithmic **L**anguage"
- **Raison d'être:** a universal, **platform independent** language for scientific application
  - so far no portability of programs!
  - fears of IBM dominance (IBM owned Fortran)
- joint project of ACM (USA) and GAMM (Germany), 1958
- declared goals
  - syntax similar to mathematical notation
  - suitable for describing algorithms in printed publications
  - mechanical translation into machine code
- problem: later abandoned by IBM (in favour of Fortran)

ALGOL 58

ALGOL 58 was heavily influenced by FORTRAN.

Main contributions

- formalized the concept of a data type
- compound statements
- identifiers of unlimited length
- arrays of any dimension (FORTRAN: max. 3)
- lower bounds of arrays can be specified
- nested branching
- the `variable := expression` syntax

ALGOL 60

Main novel features

- syntax for the first time given in the new BNF (Backus-Naur Form) notation

  statement ::= unconditional_statement
  | conditional_statement | for_statement

- block structure
- parameter passing both by value and name
- recursive procedures
- stack dynamic arrays

ALGOL 60

Significance

- for more than 20 years de facto standard for publishing algorithms in print
- the basis for all modern imperative programming languages
- the first language designed to be platform-independent
- the first language with formally described syntax

Drawbacks

- little used in the U.S.
- too flexible/powerful (call-by-name) and difficult to implement
- no platform independent I/O
- BNF (seemed strange and complicated in 1960)

```
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)

The language of Wall Street

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.

E. Dijkstra

Mathematical programs should be written in mathematical notation, data processing programs should be written in English statements.

G. Hopper

COBOL

- CBL – "**C**ommon **B**usiness **L**anguage"
- developed specifically for business applications
- requirements:
  - "as much English as possible" (so managers could read it)
  - ease of use, even at the expense of being less powerful
  - not to be overly restricted by implementation problems
- novel features
  - **hierarchical data structures** (records) and **macros** (`DEFINE`)
  - program code separated into data part and procedural part
  - no functions (and until 1974 no subprograms with parameters)
- its use mandated by the Departmenmt of Defense (DoD)
- **successful:** by the end of 1990s approx. 800 M lines of COBOL code were in use in Manhattan alone
- no influence on other languages (except for PL/I)

```
IDENTIFICATION DIVISION
PROGRAM-ID. SUM-OF-PRICES.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT INP-DATA ASSIGN TO INPUT.
  SELECT RESULT-FILE ASSIGN TO OUTPUT.
DATA DIVISION.
FILE SECTION.
FD INP-DATA LABEL RECORD IS OMITTED.
01 ITEM-PRICE
  02 ITEM PICTURE X(30).
  02 PRICE PICTURE 9999V99.
  02 FILLER PICTURE X(44).
FD RESULT-FILE LABEL RECORD IS OMITTED.
01 RESULT-LINE PICTURE X(132).
...
```

```
PROCEDURE DIVISION.
START.
  OPEN INPUT INP-DATA AND OUTPUT RESULT-FILE.
READ-DATA.
  READ INP-DATA AT END GO TO PRINT-LINE.
  ADD PRICE TO TOT.
  ADD 1 TO COUNT.
  MOVE PRICE TO PRICE-OUT.
  MOVE ITEM TO ITEM-OUT.
  WRITE RESULT-LINE FROM ITEM-LINE.
  GO TO READ-DATA.
PRINT-LINE.
  MOVE TOT TO SUM-OUT.
  MOVE COUNT TO COUNT-OUT.
  WRITE RESULT-LINE FROM SUM-LINE.
  CLOSE INP-DATA AND RESULT-FILE.
  STOP RUN.
```

BASIC (1964)

The language for (all) students

It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.

E. Dijkstra

BASIC

- J. Kemeny and T. Kurtz (Dartmouth College)
- "**B**eginner's **A**ll-purpose **S**ymbollic **I**nstruction **C**ode"
- design requirements
  - **easy** to learn and use for **non-science students**
  - "pleasant and friendly"
  - provide fast turnaround for homeworks
  - should allow free and private access (use of terminals)
  - user time is more important than computer time
- properties
  - first language used remotely through terminals
  - untyped – no reals or integers, just "numbers"
  - not suited to large programs – poorly structured
  - easy to implement
- widespread – minicomputers, home computers
- resurgence: Visual BASIC, VB.NET

```
10 INPUT "yards?",yd,"feet?",ft, "inches?",in
40 GOSUB 2000: REM print the values
50 PRINT '" = ";
70 GOSUB 1000: REM the adjustment
80 GOSUB 2000: REM print the adjusted values
90 PRINT
100 GOTO 10
1000 REM subroutine to adjust yd, ft, in to the normal form for yards,
        feet and inches
1010 LET in=36*yd+12*ft+in: REM now everything is in inches
1030 LET s=SGN in: LET in=ABS in: REM we work with in positive,
        holding its sign in s
1060 LET ft=INT (in/12): LET in=(in-12*ft)*s: REM now in is ok
1080 LET yd=INT (ft/3)*s: LET ft=ft*s-3*yd: RETURN
2000 REM subroutine to print yd, ft and in
2010 PRINT yd;"yd";ft;"ft";in;"in";: RETURN
```

PL/I (1966)

Everything but the kitchen sink

PL/I – "the fatal disease" – belongs more to the problem set than to the solution set.

E. Dijkstra

PL/I

- "**P**rogramming **L**anguage **One**"
- IBM product, intended to replace FORTRAN, COBOL and LISP
- for both scientific (floating point) and business (decimal) computing
- additionally support for lists and systems programming
- **the recipe:** mix and match the best of ALGOL, FORTRAN and COBOL – and add some new stuff
- new concepts (unfortunately poorly designed)
  - **concurrently** executable subprograms
  - predefined **exceptions**
  - **pointer** data type
- partial success, mainly in 70s

Using PL/I must be like flying a plane with 7,000 buttons, switches and handles to manipulate in the cockpit.

E. Dijkstra

```
/* Read in a line, which contains a string,
/* and then print every subsequent line that contains that string. */

find_strings: procedure options (main);
   declare pattern character (100) varying;
   declare line    character (100) varying;
   declare (line_no, end_file) fixed binary;

   end_file = 0;
   on endfile (sysin) end_file = 1;

   get edit (pattern) (L);
   line_no = 1;
   do while (end_file = 0);
      if index(line, pattern) > 0 then
         put skip list (line_no, line);
      line_no = line_no + 1;
      get edit (line) (L);
   end;

end find_strings;
```

SIMULA 67

The origins of data abstraction

- Nygard & Dahl (Norway)
- language for systems simulation
- ALGOL 60 descendant
- never widely used

Important novel concepts

- **coroutines**
  - special kind of subprograms
  - allows to interrupt (and later resume) subprogram execution
  - useful for running simulations
- the **class** construct
  - the concept of data abstraction
  - laid the foundations for OOP

ALGOL descendants

ALGOL 68

- significantly different from ALGOL 60
- the most important innovation: **orthogonality**
  - user-defined types
  - few primitive types
  - which can be combined using few combining mechanisms

ALGOL descendants

ALGOL 68

- significantly different from ALGOL 60
- the most important innovation: **orthogonality**
  - user-defined types
  - few primitive types
  - which can be combined using few combining mechanisms

Pascal (N. Wirth, 1971)

- designed as a teaching language
- remarkable combination of simplicity and expressivity
- lacks some features essential to specific applications
- relatively safe (compared to C and FORTRAN)
- the case statement (ALGOL-W – Wirth and Hoare)
- very easy to implement on a new system

Pascal bootstrap

Wirth's Pascal distribution

- A Pascal compiler, written in Pascal, that would generate output in P-code, a stack-based language similar to the byte code of modern Java compilers.
- The same compiler, already translated into P-code.
- A P-code interpreter, written in Pascal.

To get Pascal running on a new machine …

- the only thing needed is to translate the P-code interpreter (by hand) into some locally available language.

Forth (1970)

- developed by Charles H. Moore
- **stack based**
- maximal simplicity
- very small implementation suitable for embedded systems (e.g. printers, spacecraft, microcontrollers)
- descendants: PostScript, RPL, Rebol

```
2 5 * 7 + .

: fac recursive
dup 1 > if
  dup 1 - fac *
else
  drop 1
endif ;
```

C (1972)

not new, but successful

- D. Ritchie, AT&T Bell Labs
- closely tied to the development of Unix
- descended from ALGOL 68
- **little contribution** to development of PLs
- **systems programming** language
- very limited typechecking
- the **“there is no C”** problem:
  - for years, Kernighan and Ritchie (1978) was the only reference
  - the first standard: ANSI C89 (later C99, C11)
  - many vendor-specific extensions
- **huge plus:** widely available compiler (part of Unix)

ML (1973)

the language for mathematicians

- "**M**eta**L**anguage"
- R. Milner (Edinburgh)
- functional language with side-effects (like LISP)
- rigorous semantics, mathematically clean design
- the first compiler to be **proven correct**
- important language features:
  - powerful **static type system** with **type inference**
  - powerful **module system**
- descendants: Standard ML, OCaml, Haskell, F#

Prolog (1972)

what, not how

- "**Pro**gramming **log**ic"
- A. Colmerauer (Marseille)
- **declarative:** describe what the result should be, not how to compute it
- proven useful mainly for advanced databases and AI

```
append([], L, L).
append([H|T], L, [H|R]) :- append(T, L, R).

rev([], []).
rev([H|T], R) :- rev(T, RevT), append(RevT, [H], R).
```

Ada (1975/1983)

Expensive and safe

When Roman engineers built a bridge, they had to stand under it while the first legion marched across. If programmers today worked under similar ground rules, they might well find themselves getting much more interested in Ada!

Robert Dewar

Ada

Augusta Ada Byron (1815–1852) – Countess of Lovelace, a friend of Charles Babbage, the **first programmer**

- developed for the DoD (MIL STD 1815)
- standardised high-level language for **embedded systems**
- main contributions:
  - packages (encapsulation)
  - exception handling
  - generics (generic units)
  - concurrent execution, synchronization
- criticism: too large and complex
- problems:
  - compiler (un)availability
  - no support for inheritance and polymorphism (until Ada95)
- overtaken by C++
- **used in:** civil and military avionics, air traffic control, rail transportation …

Smalltalk (1980)

Object-oriented programming arrives

- A. Kay (Xerox)
- foresight: future availability of powerful desktop computers
- invention of the modern **graphical user interface**
- **nothing but objects**
- **computation**: sending messages to objects
- developed and extended the concepts from SIMULA
- the **first** mature **object-oriented language**
- unusual syntax (later adopted by Objective-C)

```
"The following is a class definition, instantiations
of which can draw equilateral polygons of any number of sides"
class name                Polygon
superclass                Object
instance variable names   ourPen
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
```

C++ (1984)

OOP enters the mainstream

- B. Stroustrup, AT&T Bell Labs
- **combines:**
  imperative programming (C) and OOP (Smalltalk)
- **design goals:**
  - compatible with C
  - almost as fast as C
- large and complicated language (generics, exceptions, …)
- less safe than Ada or Java
- **roaring success:**
  - good and easily available compilers
  - backward compatibility with C
  - for years the only available OO language suitable to large projects
  - constant development: C++89, C++03, C++11, C++14, C++17

Java (1995)

the better C++

- James Gosling (Sun Microsystems)
- design goals
  - reliability
  - platform independence, portability (JVM)
  - suitable for web programming
- **cleaned up** version of C++
  - no pointers (but everything is a reference)
  - strictly object-oriented
  - only single inheritance (but multiple interfaces)
  - fewer implicit type conversions
- automatic garbage collection
- support for concurrency/synchronization
- Java7: generics, enumeration class, iteration constructs