COS 301

Programming Languages

Evolution of the Major Programming Languages

Topics

- Zuse's Plankalkül
- Minimal Hardware Programming: Pseudocodes
- The IBM 704 and Fortran
- Functional Programming: LISP
- ALGOL 60
- COBOL
- BASIC

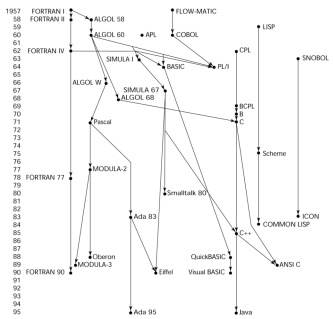
Topics (continued)

- PL/I
- APL and SNOBOL
- SIMULA 67
- Orthogonal Design: ALGOL 68
- Some Early Descendants of the ALGOLs
- Prolog
- Ada

Topics (continued)

- Object-Oriented Programming: Smalltalk
- Combining Imperative ad Object-Oriented Features: C++
- An Imperative-Based Object-Oriented Language: Java
- Scripting Languages
- A C-Based Language for the New Millennium: C#
- Markup/Programming Hybrid Languages

Genealogy of Common Languages



Alternate View

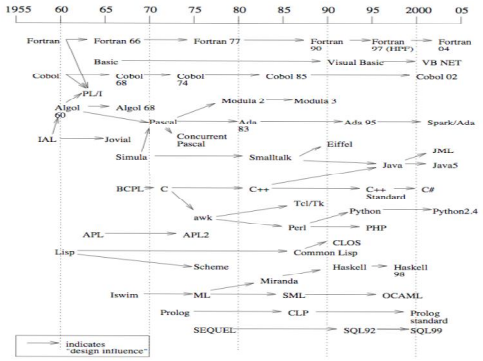


Figure 1.2: A Snapshot of Programming Language History

Zuse's Plankalkül

- Designed in 1945 for computers based on electromechanical relays, but not published until 1972
- Never implemented
- Advanced data structures, based on the bit
  - Two's complement integers, floating point with hidden bit, arrays, records
- Included algorithms for playing chess

Plankalkül Syntax

- 3 lines for a statement:
  - Operation
  - Subscripts
  - Types
- An assignment statement to assign the expression A[4] + 1 to A[5]

```
    |  A + 1 => A
V   |  4        5        (subscripts)
S   |  1.n      1.n      (data types)
```

Minimal Hardware Programming: Pseudocodes

- Late 1940's – early 1950's all programming was done in machine code (NOT assembler)
- What was wrong with using machine code?
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies--no indexing or floating point
  - Absolute addressing

Pseudocodes: Short Code

- Short Code developed by John Mauchly in 1949 for BINAC computers
  - Expressions were coded, left to right, in 12 6-bit bytes
  - Example of operations:

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

- Short codes were interpreted, not translated to machine code
- So X0 = sqrt(abs(Y0)) would be coded as

```
00 X0 03 20 06 Y0
```

Pseudocodes: Speedcoding

- Speedcoding developed by John Backus in 1954 for the IBM 701
- A virtual 3-address calculator
  - Pseudo ops for arithmetic and math functions
  - Conditional and unconditional branching
  - Auto-increment registers for array access
  - Only 700 words left for user program

Pseudocodes: Related Systems

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

IBM 704 and Fortran

- FORTRAN: The IBM Mathematical FORmula TRANslating System
- Computing environment at that time:
  - Machines had small memories, were slow and unreliable
  - Mostly used for scientific computation (number-crunching)
  - No programming tools
  - Overhead of interpretive systems was small compared to simulating floating point ops in software
- Fortran 0: 1954 - not implemented
- Fortran I:1957
  - Designed for the new IBM 704, which had index registers and floating point hardware
  - This led to the idea of compiled programming languages, because there was no place to hide the cost of interpretation (no floating-point software)

Design Process of Fortran

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

Fortran I Overview

- First implemented version of Fortran
  - Names could have up to six characters
  - Post-test counting loop (**DO**)
  - Formatted I/O
  - User-defined subprograms
  - Three-way selection statement (arithmetic **IF**)
  - Control structures based on 704 machine codes
  - No data typing statements. Names beginning with I,J,K,L,M,N were integers; others floating point

Fortran I Overview (continued)

- First implemented version of FORTRAN
  - No separate compilation
  - Compiler released in April 1957, after 18 worker-years of effort
  - Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of 704
  - Code was very fast
  - Quickly became widely used

Fortran II

- Distributed in 1958
  - Independent compilation
  - Fixed the bugs

Fortran IV and Fortran 77

- Evolved during 1960-62
  - Explicit type declarations
  - Logical selection statement
  - Subprogram names could be parameters (consider a generic sort routine)
  - ANSI standard in 1966
- Fortran 77 Became the new standard in 1978
  - Character string handling
  - Logical loop control statement
  - **IF-THEN-ELSE** statement

Fortran 90

- Most significant changes from Fortran 77
  - Modules
  - Dynamic arrays
  - Pointers
  - Recursion
  - **CASE** statement
  - Parameter type checking
- Finally dropped the fixed formatting requirements used with punch cards
- Started using mixed case!

Latest versions of Fortran

- Fortran 95 – relatively minor additions, plus some deletions
- Fortran 2003
  - Added support for OOP (like everybody else…)
  - Parameterized derived types
  - Procedure pointers
  - C language interoperability (changes in object file format)
- Let's look at a couple of examples
  - http://www.99-bottles-of-beer.net/language-fortran-77-760.html

Fortran 66 Spaghetti Code

```
      SUBROUTINE OBACT(TODO)
          INTEGER TODO,DONE,IP,BASE
          COMMON /EG1/N,L,DONE
          PARAMETER (BASE=10)
       13 IF(TODO.EQ.0) GO TO 12
          I=MOD(TODO,BASE)
          TODO=TODO/BASE
          GO
      TO(62,42,43,62,404,45,62,62,62),I
          GO TO 13
       42 CALL COPY
          GO TO 127
       43 CALL MOVE
          GO TO 144
      404 N=-N
       44 CALL DELETE
          GO TO 127
       45 CALL PRINT
          GO TO 144
       62 CALL BADACT(I)
          GO TO 12
      127 L=L+N
      144 DONE=DONE+1
          CALL RESYNC
          GO TO 13
       12 RETURN
          END
```

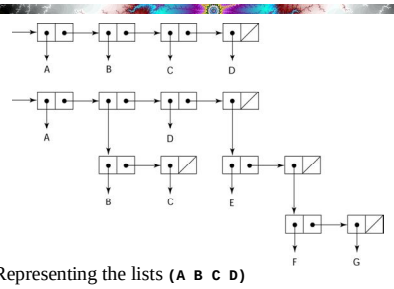
Fortran is different…

- Language before Fortran 90
  - Types and storage of all variables are fixed before run time
    - Speed wins the tradeoff between speed and flexibility
  - No dynamic data structures
  - No recursion – why?
- Dramatically changed forever the way computers are used
- Characterized by Alan Perlis as the *lingua franca* of the computing world

Functional Programming: LISP

- LISt Processing language
  - (or Lots of Irritating Stupid Parentheses)
  - Designed at MIT by McCarthy
- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)
- Only two data types: atoms and lists
- Syntax is based on Church's *lambda calculus*
  - One of several models of computation developed before computers came into existence

Representation of Two LISP Lists



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

LISP …

- Pioneered functional programming
  - Target domain was theorem proving
    - Required recursion and conditional expressions – features not available in FORTRAN
  - No need for variables or assignment
- Still the dominant language for AI
- COMMON LISP and Scheme are contemporary dialects of LISP
- ML, Miranda, and Haskell are related languages

Recursion and Iteration

- Fundamental control structures needed for any language are sequential execution, selection (conditional execution) and repetition (iteration)
- In functional languages iteration accomplished by recursion
- Functional languages are modeled on recursive function theory developed in the 1930's by Alan Turing, Alonzo Church, Kurt Gödel, and others

Scheme

- A descendant of LISP developed at MIT in the mid 1970s
- Small language
- Extensive use of static scoping
- Functions as first-class entities
- Simple syntax (and small size) make it well suited for educational applications

COMMON LISP

- An effort to combine features of several dialects of LISP (including Scheme) into a single language
- A large, complex language
  - Both static and dynamic scoping
  - Data types include records, arrays, complex numbers and character strings
  - Packages facilitate abstract data types

ML and other functional languages

- ML is a functional language with support for imperative programming
  - Robin Milner, Edinburgh, 1970's
  - Does not use the parenthesized syntax of LISP
  - Has static typing
- Descendants include Miranda and Haskell
- Haskell uses lazy evaluation (delays evaluation of an expression until the result is needed)
  - Provides interesting capabilities such as computation with infinite data structures

A Step Toward Expressiveness: ALGOL

- ALGOL development environment
  - FORTRAN had (barely) arrived for IBM 70x
  - Many other languages were being developed, all for specific machines
  - No portable languages; all were machine-dependent
  - No universal language for communicating algorithms
- ALGOL 60 was the result of efforts to design a universal language for scientific applications

Early Design Process

- ACM and GAMM met for four days for design (May 27 to June 1, 1958)
- Goals of the language
  - Syntax should be close to standard mathematical notation
  - Should be possible to use the language to describe algorithms in publications
  - Must be translatable to machine code

ALGOL 58

- Derived a lot from FORTRAN
- Concept of type was formalized
- Names could be any length
- Arrays could have any number of subscripts
- Parameters were separated by mode (in & out)
- Subscripts were placed in brackets
- Compound statements (**`begin ... end`**)
- Semicolon as a statement separator
- Assignment operator was :=
- **`if`** had an **`else-if`** clause
- No I/O - "would make it machine dependent"

ALGOL 58 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
  - Jule's Own Version of the International Algorithmic Language(JOVIAL) was the official scientific language of the US Air Force until 1984
- Although IBM was initially enthusiastic, all support was dropped by mid 1959

ALGOL 60

- Modified ALGOL 58 at 6-day meeting in Paris
- One of the most significant developments was the introduction of Backus-Naur Form (BNF) to describe syntax
- New features
  - Block structure (local scope)
  - Two parameter passing methods (by value and by name)
  - Subprogram recursion
  - Stack-dynamic arrays (variables hold index limits)
  - Still no I/O and no string handling

ALGOL 60 Successes

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative languages owe something to Algol 60
    - Direct and indirect descendants include PL/I, Simula 97, Algol 68, C, Pascal, Ada, C++ and Java
  - First language designed to be machine-independent
  - First language whose syntax was formally defined (BNF)
  - Block structure and recursive subprogram calls led to the adoption of hardware-stack machines

ALGOL 60 Failures

- Failure
  - Never widely used, especially in U.S.
  - Reasons
    - Lack of I/O and the character set made programs non-portable
    - Too flexible – some features were hard to implement and hard to understand
    - Entrenchment of Fortran
    - BNF was considered strange and difficult to understand
    - Lack of support from IBM

Algol 60 Example

```
// the main program, calculate the mean of
// some numbers
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
```

COBOL (Common Business Oriented Language)

- Algol was almost never actually used, but had a huge impact on subsequent language development
- COBOL is one of the most widely used languages in the world but has had almost no influence on subsequent language development
- COBOL met its design requirements very well

COBOL

- Late 1950's
  - UNIVAC used FLOW-MATIC (proprietary)
  - The USAF was beginning to use AIMACO (a FLOW_MATIC variant)
  - IBM was developing COMTRAN
- Grace Hopper 1953:
  - "Mathematical programs should be written in mathematical notation; data processing programs should be written in English statements."

Before COBOL:

- 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 (DEFINE) in a high-level language
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with hyphens
  - Separate data division
- Weaknesses
  - Lack of functions
  - Prior to 1974, no parameters for subprogram calls

COBOL: DoD Influence

- First language required by DoD
  - would have failed without DoD due to poor compilers
- Still the most widely used business applications language
- E. Dijkstra on COBOL
  - "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense."

COBOL Example 1: Multiplication

```
 $ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  Multiplier.
AUTHOR.  Michael Coughlan.
* Example program using ACCEPT, DISPLAY and MULTIPLY to
* get two single digit numbers from the user and multiply them together

DATA DIVISION.

WORKING-STORAGE SECTION.
01  Num1                                PIC 9  VALUE ZEROS.
01  Num2                                PIC 9  VALUE ZEROS.
01  Result                              PIC 99 VALUE ZEROS.

PROCEDURE DIVISION.
    DISPLAY "Enter first number  (1 digit) : " WITH NO ADVANCING.
    ACCEPT Num1.
    DISPLAY "Enter second number (1 digit) : " WITH NO ADVANCING.
    ACCEPT Num2.
    MULTIPLY Num1 BY Num2 GIVING Result.
    DISPLAY "Result is = ", Result.
    STOP RUN.
```

Example 2: Count student records from file

```
  $ SET SOURCEFORMAT "FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID.  StudentNumbersReport .
AUTHOR. Michael Coughlan.

*INPUT      The student record file Students.Dat  Records in this file
*           are sequenced on ascending Student Number.
*OUTPUT     Shows the number of student records in the file and the
*           number of records for males and females.
*PROCESSING For each record read;
*               Adds one to the TotalStudents count
*               IF the Gender is Male   adds one to TotalMales
*               IF the Gender is Female adds one to TotalFemales
*           At end of file writes the results to the report file.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT StudentFile ASSIGN TO "STUDENTS.DAT"
                ORGANIZATION IS LINE SEQUENTIAL.
    SELECT ReportFile ASSIGN TO "STUDENTS.RPT"
                ORGANIZATION IS LINE SEQUENTIAL.
```

Example 2: Count student records from file

```
  DATA DIVISION.
FILE SECTION.
FD  StudentFile.
01  StudentDetails.
    88  EndOfStudentFile VALUE HIGH-VALUES.
    02  StudentId        PIC 9(7).
    02  StudentName.
        03 Surname       PIC X(8).
        03 Initials      PIC XX.
    02  DateOfBirth.
        03 YOBirth       PIC 9(4).
        03 MOBirth       PIC 9(2).
        03 DOBirth       PIC 9(2).
   02  CourseCode        PIC X(4).
   02  Gender            PIC X.
       88 Male           VALUE "M", "m".

FD  ReportFile.
01  PrintLine            PIC X(40).
```

Example 2: Count student records from file

```
  WORKING-STORAGE SECTION.
01  HeadingLine           PIC X(21) VALUE "  Record Count Report".

01  StudentTotalLine.
    02  FILLER            PIC X(17) VALUE "Total Students = ".
    02  PrnStudentCount   PIC Z,ZZ9.

01  MaleTotalLine.
    02  FILLER            PIC X(17) VALUE "Total Males    = ".
    02  PrnMaleCount      PIC Z,ZZ9.

01  FemaleTotalLine.
    02  FILLER            PIC X(17) VALUE "Total Females  = ".
    02  PrnFemaleCount    PIC Z,ZZ9.

01  WorkTotals.
    02  StudentCount      PIC 9(4) VALUE ZERO.
    02  MaleCount         PIC 9(4) VALUE ZERO.
    02  FemaleCount       PIC 9(4) VALUE ZERO.
```

Example 2: Count student records from file

```
  PROCEDURE DIVISION.
Begin.
    OPEN INPUT StudentFile
    OPEN OUTPUT ReportFile
    READ StudentFile
      AT END SET EndOfStudentFile TO TRUE
    END-READ
    PERFORM UNTIL EndOfStudentFile
       ADD 1 TO StudentCount
       IF Male  ADD 1 TO MaleCount
         ELSE   ADD 1 TO FemaleCount
       END-IF
       READ StudentFile
         AT END SET EndOfStudentFile TO TRUE
       END-READ
    END-PERFORM
    PERFORM PrintReportLines
    CLOSE StudentFile, ReportFile
    STOP RUN.
```

Example 2: Count student records from file

```
PrintReportLines.
  MOVE StudentCount TO PrnStudentCount
  MOVE MaleCount    TO PrnMaleCount
  MOVE FemaleCount  TO PrnFemaleCount

  WRITE PrintLine FROM HeadingLine
          AFTER ADVANCING PAGE
  WRITE PrintLine FROM StudentTotalLine
          AFTER ADVANCING 2 LINES
  WRITE PrintLine FROM MaleTotalLine
          AFTER ADVANCING 2 LINES
  WRITE PrintLine FROM FemaleTotalLine
          AFTER ADVANCING 2 LINES.
```

BASIC

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

BASIC

- Like COBOL, widely used but gets little respect
  - "The Rodney Dangerfield of computer languages"
- Designed by John Kemeny & Thomas Kurtz at Dartmouth for liberal arts students
- 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 was designed for interactive terminals on a time-sharing system

BASIC

- Based on FORTRAN
- Many different versions came into existence; 1978 ANSI standard was minimal
- Digital used a version of BASIC to write part of the operating system for the PDP-11

E. Dijkstra on BASIC

- 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. –

Unstructured Programming

- Djikstra's comment referred to code like this:

```
10 IF X = 42 GOTO 40
20 X = X + 1
30 GOTO 10
40 PRINT "X is finally 42!"
```

Modern BASIC

- Most hobby computers from the 1970s were equipped with tiny BASIC interpreters
- MS-DOS include BASICA and later QBASIC
- With Windows, Microsoft started developing Visual Basic
  - Even the oldest VB versions are object-oriented languages with classes, inheritance, etc.
  - Visual Studio 6 (1998) was the most popular version
  - VBScript was (and still is) used for web development (Classic ASP) while VBA was (and still is) used to automate Office applications

VB.NET

- VB 7 released 2002 with .NET broke compatibility with earlier versions
- Can be used for anything from console applications to web development
- Virtually the same capabilities as C#
- With Visual Studio 2008 VB acquired capabilities such as support for lambda expressions, support for anonymous types, type inferencing, etc.

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

PL/I: Background

- By 1963
  - Scientific users began to need more elaborate I/O, like COBOL had; business users began to need floating point and arrays for MIS
  - It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly
- The obvious solution
  - Build a new computer to do both kinds of applications
  - Design a new language to do both kinds of applications
  - PL/I could replace COBOL, FORTRAN, LISP and assembler

PL/I: Design Process

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

PL/I Overview

- Famous for including everything possible in a programming language, even the kitchen sink
- PL/I contributions
  - Programs could create concurrently executing subprograms
  - First exception handling in a programming language
  - Recursion allowed, but could disabled for efficient function calls
  - Pointer data type
  - Array cross sections
- Concerns
  - Many new features were poorly designed
  - Too large and too complex

PL/I …

- A partial success, but widely criticized for slow compilers, difficult to use features, partial implementations, and buggy compilers
- Widely used in the 1970's on mainframes, usage continued until the 1990's with some PC implementations
- Virtually dead now

Early Dynamic Languages: APL and SNOBOL

- Characterized by dynamic typing and dynamic storage allocation
- "Variables are untyped" should be read as "Variable declarations are untyped"
  - A variable acquires a type when it is assigned a value
- Storage is allocated to a variable when it is assigned a value

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 because of the use of single special characters for complex operations
  - A "write-only" language
- Still in use after 45 years; minimal changes

Classic APL Example

- Find all prime numbers <= n

```
PRIMES : (~R∊R∘.×R)/R←1↓⍳R
```

SNOBOL (String Oriented Symbolic Language)

- Designed as a string manipulation language at Bell Labs by Farber, Griswold, and Polensky in 1964
- Powerful operators for string pattern matching
  - Strings created at runtime can be treated as programs and executed.
  - SNOBOL patterns are equivalent to context-free grammers and thus "more powerful" than regular expressions used in perl, javascript, awk etc.
- Uses a backtracking algorithm for pattern matching similar to Prolog execution

SNOBOL

- Still used for some text-processing tasks
- See http://www.snobol4.org
- SPITBOL (A Speedy Implementation of SNOBOL) was released under GNU license in 2009
- See http://code.google.com/p/spitbol/

SIMULA 67

- Designed primarily for system simulation in Norway by Kristen Nygaard and Ole-Johan Dahl
- Based on ALGOL 60 and SIMULA I
- Primary Contributions
  - Coroutines - a kind of subprogram (minor)
  - Classes, objects, and inheritance (MAJOR)
- The main ancestor of Smalltalk, the origin of OOP

Orthogonal Design: ALGOL 68

- From the continued development of ALGOL 60 but not a superset of that language
- Source of several new ideas (even though the language itself never achieved widespread use)
  - Introduced user-defined data types
  - Dynamic arrays
  - Reference types
- Design is based on the concept of orthogonality
  - A few basic concepts, plus a few combining mechanisms

ALGOL 68 ...

- Less usage than ALGOL 60
- Had a very strong influence on subsequent languages, especially Pascal, C, and Ada
- The English language used for to describe ALGOL was a problem

  "*The coercion is called deproceduring. This can be employed in any soft, and therefore any weak, meek, firm or strong syntactic position.*"

  From ALGOL 68: a first and second course.

  Andrew D. McGettrick

Early Algol Descendants

- Two very important ones: Pascal and C

Pascal - 1971

- Developed by Niklaus 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
- Emphasis on reliable programming: type-safety, index bounds check, etc.

Pascal

- Lacked a few features necessary for real-world programming such as separate compilation
- Non-standard dialects were developed
- In particular Turbo Pascal from Borland; for the IBM PC
  - 35kb of code written in assembler included complete editor, compiler and debugger
- While Pascal is rarely used any more its object-oriented descendant Delphi is still in wide use

C - 1972

- Designed for systems programming (at Bell Labs by Dennis Richie)
- Evolved primarily from BCLP, B, but also ALGOL 68
  - BCLP and B are not typed languages
  - All data were considered to be machine words
- Powerful set of operators, but poor type checking
- Initially spread through UNIX, many areas of application

C

- The only "standard" for the language until 1989 ANSI description was the classic "C Programming Language" by Kernighan and Ritchie
- 2nd edition of K&R came out after 1989 ANSI C
- C has weak type checking
  - No boolean types, ints are used
  - No built in character string support (types char and unsigned char are 8-bit numbers)
  - This code compiles but what does it do?

```
int x = 5;
char y[] = "37";
char* z = x + y;
```

Programming Based on Logic: Prolog

- Developed, by Alain Comerauer and Phillipe Roussel (University of Aix-Marseille), with help from Robert Kowalski (University of Edinburgh)
  - Based on predicate logic and proof system called resolution
  - 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

Prolog Programs (databases

- Consist of two components: facts and rules. Ex:

```
speaks(allen, russian).
speaks(bob, english).
speaks(mary, russian).
speaks(mary, english).
talkswith(P1,P2) :-
  speaks(P1,L),speaks(P2,L), P1\= P2.
```

Queries

```
?- speaks(Who, russian).
```

- asks for an instantiation of the variable Who for which the query speaks(Who, russian) succeeds.
- To answer the query Prolog considers every fact and rule whose head is speaks. (If more than one, consider them in order.)

```
Who = allen ;
Who = mary ;
No
```

History's Largest Design Effort: Ada

- Named Ada after Augusta Ada Byron, the first programmer
- Department of Defense (DoD) had about 450 different programming languages in use by 1974
- Over half of the applications were in embedded systems
  - Many used assembly language for specialized processors
  - No high-level languages were suitable
  - Very little code reuse was taking place
  - No general software development tools

Ada

- High Order Language Working Group (HOLWG) produced requirement documents
- A huge design effort, involving hundreds of people, much money, and about eight years
  - Strawman requirements (April 1975)
  - Woodenman requirements (August 1975)
  - Tinman requirements (1976)
  - Ironman requirments (1977)
  - Steelman requirements (1978)
- By 1979 hundreds of proposals were narrowed to four, all based on Pascal

Ada ...

- Contributions
  - Packages - support for data abstraction by encapsulating data types, objects and procedures
  - Elaborate exception handling model
  - Generic program units allowed algorithms to be implemented without specifying a data type
  - Concurrency - through the rendezvous mechanism
- 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)
  - Ada packages were very similar to classes but did not allow components to be added to a base class. Added support for OOP through type derivation
  - Added support for OOP with runtime subprogram dispatching
  - Better control mechanisms for shared data
  - New concurrency features
  - More flexible libraries
- Popularity suffered because the DoD no longer requires its use but also because of popularity of C++

Object-Oriented Programming: Smalltalk

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic binding)
- Alan Kay foresaw the development of the desktop PC and the usage of computers by non-programmers
- Pioneered the graphical user interface design based on a desktop model (adopted by Apple after a visit by Steve Jobs)

Smalltalk Overview

- Smalltalk is a very small and simple language
  - Lacks conventional control structures because those are implemented by objects
  - A great deal of Smalltalk is defined in Smalltalk
- The Smalltalk world is populated by objects
  - booleans, numbers, strings but also large complex systems (ex. Class BitBlt used for drawing bitmaps)
- Objects pass messages to other objects

Smalltalk Example

```
' Smalltalk class to constraint a 2D point to a fixed grid
Point subclass: #GriddedPoint
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
!GriddedPoint methodsFor: 'accessing'!

x: xInteger
  "Set the x coordinate gridded to 10 (using rounding,
  alternatively I could use truncating)."
        ^ super x: (xInteger roundTo: 10)

y: yInteger
  "Set the y coordinate gridded to 10 (using rounding,
  alternatively I could use truncating)."
        ^ super y: (yInteger roundTo: 10)

!GriddedPoint methodsFor: 'private'!
setX: xPoint setY: yPoint
  "Initialize the instance variables rounding to 10."
   ^ super setX: (xPoint roundTo: 10) setY: (yPoint roundTo: 10)
```

Combining Imperative and Object-Oriented Programming: C++

- Developed at Bell Labs by Bjarne Stroustrup in 1980
  - Evolved from C and SIMULA 67
  - Facilities for object-oriented programming, taken partially from SIMULA 67
  - Designed to provide classes and inheritance without performance penaly
  - 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
- ANSI standard approved in November 1997
- Microsoft's version (released with .NET in 2002): Managed C++
  - delegates, interfaces, no multiple inheritance

What is Object-Oriented?

- Alan Kay coined the term "object oriented"
  - "…and I can tell you I did not have C++ in mind."
- Combining object oriented constructions with a low-level language like C can produce some strange constructs:

  "If you think C++ is not overly complicated, just what is a protected abstract virtual base pure virtual private destructor, and when was the last time you needed one?" -- Tom Cargil, C++ Journal.

Related OOP Languages

- Eiffel (designed by Bertrand Meyer - 1992)
  - Not directly derived from any other language
  - Smaller and simpler than C++, but still has most of the power
  - Lacked popularity of C++ because many C++ enthusiasts were already C programmers
- Delphi (Borland)
  - Pascal plus features to support OOP
  - Smaller, more elegant and safer than C++

Java

- Developed at Sun in the early 1990s
  - C and C++ were deemed un satisfactory for embedded electronic devices
  - Unsafe, unreliable and not object-oriented
- Based on C++, Significantly simplified from C++
  - Does not allow pointer arithmetic
  - Only allows safe "widening" type coercions, e.g., int-> float is OK, float -> int is not
  - does not include **`struct, union, enum (Why not?)`**
  - Supports *only* OOP
  - Has references, but not pointers
  - Includes support for applets and a form of concurrency
  - Automated memory management
  - Does not support multiple inheritance

Java Evaluation

- Eliminated many unsafe features of C++ (at the expense of verbosity)
- Supports concurrency
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Use increased faster than almost any previous language
- Java 6 was released in 2006 with significant runtime performance enhancement

Scripting Languages

- Scripting languages evolved as file and computer management tools for computer operators
- A list of commands (a script) is interpreted by a shell
  - First language was sh (Bourne shell) on Unix
  - Many others have evolved: ksh (Korn shell); csh and tcsh (C shell) etc.
  - awk is a programming language for manipulation of text data
- Then scripting language became popular for server-side web processing

Perl

- Perl
  - Designed by Larry Wall—first released in 1987– a combination of sh and awk
  - Variables are statically typed but implicitly declared
  - Three distinctive namespaces, denoted by the first character of a variable's name
  - Has a large number of implicit variables with names like $_, @_, $$
  - Powerful, but somewhat dangerous because of freedom with type coercions
  - Gained widespread use for UNIX administration, then CGI programming on the Web
  - Now extensively used in computational biology and bioinformatics

JavaScript

- Related to Java only through similar syntax
- 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 with processing on the client side rather than the server side
- Purely interpreted by the browser
- Ancestor of ActionScript (Flash programming language)

JavaScript

- JS is relatively low level and subject to browser incompatibilities but is now supplemented with very high level standard libraries such as jQuery and Prototype
- AJAX (Asynchronous Javascript and XML) technology has become very popular over the last 3-4 years
  - Complexity of Javascript apps has grown significantly
- Google's development of the Chrome browser has had significant impact on the maturity of Javascript

PHP

- PHP: Now called Hypertext Preprocessor, designed by Rasmus Lerdorf
  - Original name was "Personal Home Page"
- A purely interpreted server-side HTML-embedded scripting language, often used for form processing and database access through the Web
- Features dynamic strings, associative dynamic arrays, free use of type coercions
- Support for OOP added with second release
- Extensive support for form processing and access to back-end databases
- Maintained as open source product. Huge libraries of code.

Python

- A multiparadigm interpreted scripting language
  - Guido van Rossum
  - Named after Monty Python
- Type checked but dynamically typed
- Basic data types are lists, tuples, and hashes (associative arrays)
- Unique syntax designed for readability
- Designed as an extensible language

The Zen of Python

- http://www.python.org/dev/peps/pep-0020/

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-and preferably only one-obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Lua

- Functional and imperative paradigms but limited support for OO as with Javascript
- Functions are first-class values
- Designed to be extensible
- Only 21 reserved words
- Small number of atomic data types (Booleans, Numbers, Strings)
- One complex data structure called a "table" similar to associative arrays or hashes
- Interesting oddments: Lua is used in both Angry Birds and in the Flame cyber weapon

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
- Has a number of Perl features including implicit variables
- Extensible like Python
- First language designed in Japan to be widely adopted in US

C#

- Part of the .NET development platform (2000)
- Based on C++ , Java, and Delphi
  - A few improvements over C++
- Provides a language for component-based software development
- All .NET languages use Common Type System (CTS), which provides a common class library
- Compiled to byte code for the Common Language Runtime (CLR)

Markup/Programming Hybrid Languages

- XSLT
  - eXtensible Markup Language (XML): a metamarkup language
  - eXtensible Stylesheet Language Transformation (XSTL) transforms XML documents for display
  - Programming constructs (e.g., looping)
- JSP
  - Java Server Pages: a collection of technologies to support dynamic Web documents
  - servlet: a Java program that resides on a Web server and is enacted when called by a requested HTML document; a servlet's output is displayed by the browser
  - JSTL includes programming constructs in the form of HTML elements
- ASP and ASP.NET
  - Active Server Pages
  - Similar to JSP. .NET elements look like HTML but are interpreteted server side and rendered in HTML
  - Any .NET language can be used for programming