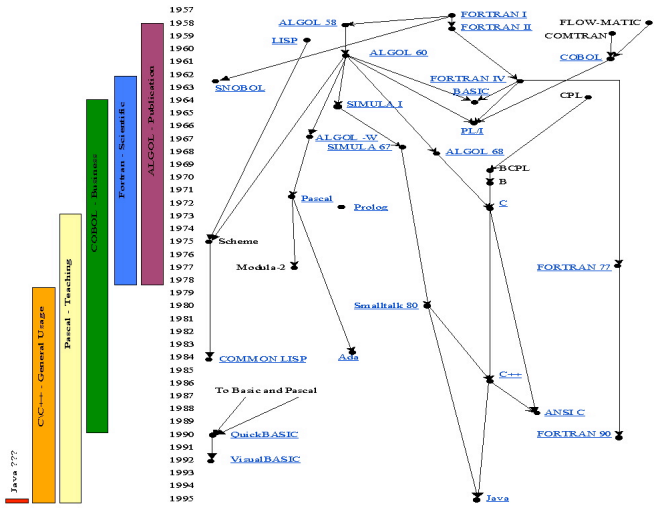
CSCI: 4500/6500 Programming Languages

Origin & Evolution



Programming Paradigm: Imperative

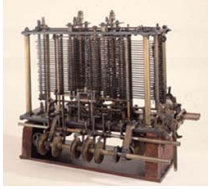
- **Imperative programming: Describes computation in terms of a program state and statements that change the program state.**
  - **Central features are variables, assignment statement and iterations**
  - **sequence of commands for the computer to perform**
  - **FORTRAN, Algol, Pascal, C**
  - **von Neumann**
- **Object Oriented programming: Computer program is composed of a collection of units, or objects that act on each other (instead of a collection of functions. Each objects is capable of receiving a message, processing data and sending messages to other objects**
- **Scripting**

Programming Paradigm: Declarative

- **Not Imperative: Describes what computation should be performed and not how to compute it**
- **Functional or Applicative, Programming: Treats computation as the evaluation of mathematical functions.**
  - **Reactive**
  - **emphasizes the definitions of functions rather than implementations of state machines (idea is to apply functions to given parameters).**
  - **Can be done without assignment statements, and without iteration.**
  - **Advantage: no side-effects**
  - **Scheme, LISP, SM**
- **Logic programming: Defines "what" to be computed, rather than "how" the computation takes place. Example, in Prolog, you supply a database of facts and rules: and perform queries on the database.**
  - **Goal directed: Contraints**

First General Purpose Machine

- **Charles Babbage designed the first computer, the Analytical Engine, starting in 1823 never completed but build 100 years later**



- **A store (memory) holding 1000 numbers**
- **An arithmetical unit**
- **Assembly like language**
- **Loops**
- **Conditionals**
- **3 types of punch cards (similar to ones that described patterns for weaving machines)**
  - **one for arithmetical , constants, one for loads/store**

Ada Lovelace: First Programmer

- **Worked with Babbage, daughter of Lord Byron**
- **Mathematician**
- **Created first program for the Analytical Engine**
  - **Plan for calculating Bernoulli numbers**
- **Language ADA named in her honor**

Zuse's Plankalkül: First High-Level Programming Language

- Formulated a language using predicate logic (Prolog like), boolean algebra and data structures for his general purpose relay computer called the Z4 (which survived the war) around 1942-1945, not published until 1972. First compiler implemented in 2000 5 years after Zuse's death.
- Assignment statements, subroutines, conditional statements, iteration, floating point, hierarchical records, assertions, exceptions handling, goal directed execution, arrays.

Grace Hoper: The First Compiler

- Mathematical PhD Yale 1934
- Wrote first compiler, the "A" compliler for programming
- Co-designer of COBOL (Common Business Oriented Language) the most widely used programming language until recently (1959). 1960 – compiler built. Influenced by Flowmatic.

First Computer Bug?

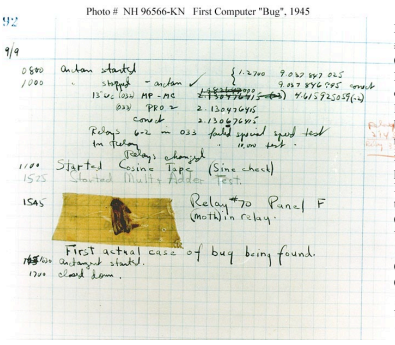


Photo #: NH 96566-KN (Color)

The First "Computer Bug"

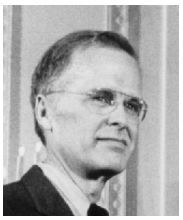
Moth found trapped between points at Relay # 70, Panel F, of the Mark II Aiken Relay Calculator while it was being tested at Harvard University, 9 September 1945. The operators affixed the moth to the computer log, with the entry: "First actual case of bug being found". They put out the word that they had "debugged" the machine, thus introducing the term "debugging a computer program".
In 1988, the log, with the moth still taped by the entry, was in the Naval Surface Warfare Center Computer Museum at Dahlgren, Virginia.

Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988.

U.S. Naval Historical Center Photograph.

http://www.history.navy.mil/photos/images/h96000/h96566kc.htm

First Major Language: FORTRAN



- Developed by John Backus for IBM 704 (1955)
- Scientific Computing
- Names could have up to six characters
- Post-test counting loop (`DO`)
- Formatted I/O
- User-defined subprograms
- Three-way selection statement (arithmetic `IF`)
- No data typing statements

Fortran II

- Distributed in 1958
  - » Independent compilation
  - » Fixed the bugs

Fortran IV

- Evolved during 1960-62
  - » Explicit type declarations
  - » Logical selection statement
  - » Subprogram names could be parameters
  - » 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

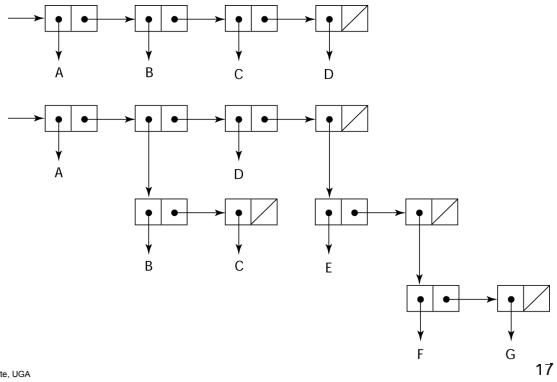
Fortran Evaluation

- Highly optimizing compilers (all versions before 90)
  - » Types and storage of all variables are fixed before run time
- Dramatically changed forever the way computers are used

Functional Programming: LISP

- LISt Processing language (2nd oldest high level programming language – FORTRAIN is oldest)
  - » Designed at MIT by McCarthy (1958)
- 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 Alonzo Church's *lambda calculus*

Representation of Two LISP Lists



LISP Evaluation

- 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

Scheme

- Developed at MIT in mid 1970s
- Small
- Extensive use of static scoping
- Functions as first-class entities
- Simple syntax (and small size) make it ideal for educational applications

COMMON LISP

- An effort to combine features of several dialects of LISP into a single language
- Large, complex

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

Early Design Process

- ACM and GAMM met for four days for design (May 27 to June 1, 1958)
- Goals of the language
  - » Close to mathematical notation
  - » Good for describing algorithms
  - » Must be translatable to machine code

ALGOL 58

- 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)
- Although IBM was initially enthusiastic, all support was dropped by mid 1959

ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - » Block structure (local scope)
  - » Two parameter passing methods
  - » Subprogram recursion
  - » Stack-dynamic arrays
  - » Still no I/O and no string handling

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)

ALGOL 60 Evaluation (continued)

- Failure
  - » Never widely used, especially in U.S.
  - » Reasons
    - – Lack of I/O and the character set made programs non-portable
    - – Too flexible--hard to implement
    - – Entrenchment of Fortran
    - – Formal syntax description
    - – Lack of support from IBM

Computerizing Business Records: COBOL

- Environment of development
  - » UNIVAC was beginning to use FLOW-MATIC
  - » USAF was beginning to use AIMACO
  - » IBM was developing COMTRAN

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
  - » Verbs were first word in every statement

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

COBOL: DoD Influence

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

The Beginning of Timesharing: 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

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
  - 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: Design Process

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

PL/I: Evaluation

- PL/I contributions
  - First unit-level concurrency
  - First exception handling
  - Switch-selectable recursion
  - First pointer data type
  - First array cross sections
- Concerns
  - Many new features were poorly designed
  - Too large and too complex

Two Early Dynamic Languages: APL and SNOBOL

- Characterized by dynamic typing and dynamic storage allocation
- Variables 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
- Still in use; minimal changes

SNOBOL

- Designed as a string manipulation language at Bell Labs by Farber, Griswold, and Polensky
- Powerful operators for string pattern matching
- Slower than alternative languages (and thus no longer used for writing editors)
- Stilled used for certain text processing tasks

The Beginning of Data Abstraction: SIMULA 67

- Designed primarily for system simulation in Norway by Nygaard and Dahl
- Based on ALGOL 60 and SIMULA I
- Primary Contributions
  - Co-routines - a kind of subprogram
  - Implemented in a structure called a class
  - Classes are the basis for data abstraction
  - Classes are structures that include both local data and functionality

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)
- Design is based on the concept of orthogonality
  - A few principle concepts, few combining mechanisms

ALGOL 68 Evaluation

- Contributions
  - » User-defined data structures
  - » Reference types
  - » Dynamic arrays (called flex arrays)
- Comments
  - » Less usage than ALGOL 60
  - » Had strong influence on subsequent languages, especially Pascal, C, and Ada

Early Descendants of ALGOLs

- ALGOL languages impacted all imperative languages
  - » Pascal
  - » C
  - » Modula/Modula 2
  - » Ada
  - » Oberon
  - » C++/Java
  - » Perl (to some extent)

Pascal - 1971

- Developed by Wirth (a member of the ALGOL 68 committee)
- Designed for teaching structured programming
- Small, simple, nothing really new
- Largest impact on teaching programming
  - » From mid-1970s until the late 1990s, it was the most widely used language for teaching programming

C - 1972

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

Perl

- Related to ALGOL only through C
- A scripting language
  - » A *script* (file) contains instructions to be executed
  - » Other examples: sh, awk, tcl/tk
- Developed by Larry Wall
- Perl variables are statically typed and implicitly declared
  - » Three distinctive namespaces, denoted by the first character of a variable's name
- Powerful but somewhat dangerous
- Widely used as a general purpose language

Programming Based on Logic: Prolog

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

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, known as being the first programmer

Ada Evaluation

- 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
- 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 type binding)
- Pioneered the graphical user interface design
- Promoted OOP

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
- ANSI standard approved in November 1997
- Microsoft's version (released with .NET in 2002): Managed C++
  - » delegates, interfaces, no multiple inheritance

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
  - » More elegant and safer than C++

An Imperative-Based Object-Oriented Language: Java

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

Java Evaluation

- Eliminated unsafe features of C++
- Concurrency features
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Widely used for WWW pages
- Use for other areas increased faster than any other language
- Most recent version, 5.0, released in 2004

Scripting Languages for the Web

- JavaScript
  - A joint venture of Netscape and Sun Microsystems
  - Used in Web programming (client side) to create dynamic HTML documents
  - Related to Java only through similar syntax
- PHP
  - PHP: Hypertext Preprocessor
  - Used for Web applications (server side); produces HTML code as output
- Python
  - An OO interpreted scripting language
  - Type checked but dynamically typed
  - Supports CGI and form processing

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

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

Markup/Programming Hybrid Languages

- 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; servlet's output is displayed by the browser

99 Bottles of Beer in 877 different programming languages (1994)

```
#include <stdio.h>  /* C version */

int main(void)
{
int b;

for( b = 99; b >= 0; b-- ) {
      switch (b) {
            case 0:
               printf("No more bottles of beer on the wall, no more bottles of beer.\n");
              printf("Go to the store and buy some more, 99 bottles of beer on the wall.\n");
              break;
            case 1:
              printf("1 bottle of beer on the wall, 1 bottle of beer.\n");
              printf("Take one down and pass it around, no more bottles of beer on the wall
\n");
              break;
            default:
              printf("%d bottles of beer on the wall, %d bottles of beer.\n", b, b);
              printf("Take one down and pass it around, %d %s of beer on the wall.\n"
              ,b - 1 ,((b - 1) > 1)? "bottles" : "bottle");
              break;
              }
}
return 0;
}
```

```
10 REM BASIC Version of 99 Bottles of beer
20 FOR X=100 TO 1 STEP -1
30 PRINT X;"Bottle(s) of beer on the wall,";X;"bottle(s) of beer"
40 PRINT "Take one down and pass it around,"
50 PRINT X-1;"bottle(s) of beer on the wall"
60 NEXT
```

99 Bottles of Beer in 877 different programming languages (1994)

```
;;; Tim Goodwin (tim@pipex.net)    Scheme
(define bottles
   (lambda (n)
      (cond ((= n 0) (display "No more bottles"))
            ((= n 1) (display "One bottle"))
            (else (display n) (display " bottles")))
      (display " of beer")))

(define beer
   (lambda (n)
     (if (> n 0)
         (begin
           (bottles n) (display " on the wall") (newline)
           (bottles n) (newline)
           (display "Take one down, pass it around") (newline)
           (bottles (- n 1)) (display " on the wall") (newline)
           (newline)
           (beer (- n 1))))))
(beer 99)
```


99 Bottles of Beer in 877 different programming languages (1994)

```
#!/usr/local/bin/python
#   python version of 99 bottles of beer, compact edition
#   by Fredrik Lundh (fredrik_lundh@ivab.se)

def bottle(n):
    try:
        return { 0: "no more bottles",
                 1: "1 bottle"} [n] + " of beer"
    except KeyError: return "%d bottles of beer" % n

for i in range(99, 0, -1):
    b1, b0 = bottle(i), bottle(i-1)
    print "%(b1)s on the wall, %(b1)s,\n"\
          "take one down, pass it around,\n"
          "%(b0)s on the wall." % locals()
```

```
#!/usr/bin/awk -f
#   awk version of 99 bottles of beer
#   by Whitey (whitey@netcom.com) - 06/05/95

BEGIN {
   for(i = 99; i > 0; i--) {
      print s = bottle(i), "on the wall,", s ","
      print "take one down, pass it around,"
      print bottle(i - 1), "on the wall."
   }
}

function bottle(n) {
   return sprintf("%s bottle%s of beer", n ? n : "no
more", n - 1 ? "s" : "")
}
```


99 Bottles of Beer in 877 different programming languages (1994)

```
% 99 bottles of beer.    Prolog
% Remko Troncon <spike@kotnet.org>

bottles :-
    bottles(99).

bottles(1) :-
    write('1 bottle of beer on the wall, 1 bottle of beer,'), nl,
    write('Take one down, and pass it around,'), nl,
    write('Now they are alle gone.'), nl.
bottles(X) :-
    X > 1,
    write(X), write(' bottles of beer on the wall,'), nl,
    write(X), write(' bottles
    write('Take one down and p
    NX is X - 1,
    write(NX), write(' bottles
    bottles(NX).
```

```
"Programmer: patrick m. ryan - Smalltalk
 pryan@access.digex.net"http://www.access.digex.net/~pryan

99 to: 1 by: -1 do: [ :i |
     i print. ' bottles of beer on the wall, ' print.
     i print. ' bottles of beer. ' print.
     'take one down, pass it around, ' print.
     (i-1) print. ' bottles of beer on the wall, ' print.
     ]
```