Programming Languages
A few bits of history

A (biased, incomplete, selective) collection of impressions

Hal Perkins
Spring 2011
Some Sources & References

  - Links to proceedings and papers on the course web

- 50 in 50: multimedia presentation by Guy Steele and Richard Gabriel
  - Several versions on the web - links on the course site
  - Best 50 min. lecture about PL you’re likely to see (including this one)

- Wikipedia is pretty good on many of these topics

- Various History of Computing journals, web archives, …
In the beginning...

- **1940’s, 1950’s – assembly language**
  - A step up from programming in octal (base 8)
  - First software libraries – sin, cos, sqrt

- **Each new computer had its own machine/assembler language**
  - Computer architecture (family of computers with a common instruction set) didn’t appear until the IBM 360 series in 1964

- **Had to recode everything when you got a new computer**
1954 FORTRAN – IBM Mathematical FORmula TRANslating System

- **Goal:** Design a translator to convert “scientific” source code into IBM 704 machine code with execution speed comparable to hand-written code.

- **IBM 704:** Hardware floating-point, index registers, ...

- The compiler was the important piece – the language was made up as the project went along
  - Assignment, DO (counting) loops, integer and floating-point values, subscripted variables (up to 3 dimensions but limited forms for subscripts, stored in column-major order), sequential I/O for cards, printing, tapes
  - Many constructs inspired by need to exploit IBM 704 instructions

- **FORTRAN I** released in 1957

- **Subroutines and functions appeared in FORTRAN II in 1958**
  - No recursion until FORTRAN 77
From the first FORTRAN manual

Given an N x N square matrix A, to find those off-diagonal elements which are symmetric and to write them on binary tape.

```
REWIND 3
DO 3 I = 1,N
DO 3 J = 1,N
IF(A(I,J)-A(J,I)) 3,20,3
3    CONTINUE
    END FILE 3
    MORE PROGRAM
20   IF(I-J) 21,3,21
21   WRITE TAPE 3,I,J, A(I,J)
    GO TO 3
```
Impact

- The FORTRAN I and II compilers were the best optimizing compilers until IBM 360’s FORTRAN H in 1968-69
  - Nobody would have taken it seriously if the code hadn’t been fast

- But almost immediately efficiency didn’t matter – the advantages of writing relatively portable code quickly were more important

- FORTRAN compilers appeared for most major systems within a few years
1958 LISP

- List Processing language
- Symbolic computation, not numbers
- S-expressions (lists, recursive data)
- Recursion, conditional expressions, $\lambda$-expressions (functions), closures (FUNARG) – e.g. lexical scoping
- `eval` function that defined the language and served as an interpreter
- Garbage collection to manage storage
- Clean mathematical semantics

- Original implementation on IBM 704 (cf FORTRAN)
- Major application area: Artificial intelligence
“Algol 60 was not only an improvement on its predecessors, but also on nearly all its successors.”

C. A. R. “Tony” Hoare

Revised Report on the Algorithmic Language ALGOL 60

By

J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy,
P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois,
J. H. Wegstein, A. van Wijngaarden, M. Woodger

Edited by

Peter Naur

Dedicated to the memory of William Turanski
ALGOL 60

- Developed in 1958-1960
- Attempt to come up with a common language not tied to a single vendor (e.g., IBM)
- International committee sponsored by ACM
- Primarily a numeric language
- Functions, procedures, assignment, loops, arrays, etc.
- Block structure – compound statements, nested scopes
- Recursive functions and call by value, call-by-name
- But no standardized I/O built in to the language (right idea: put it in library routines, wrong: a standard set never appeared)
- Reference/publication/hardware representations
  - $a \leftarrow b$ vs $a := b$ vs punch cards
- Formal syntax (Backus, based on ideas from linguistics)
Call-by-name & Jensen’s device

the procedure,

real procedure IP(z,y,n,j); value n;
begin real a; a := 0;
  for j := 1 step 1 until n do a := a + z × y;
  IP := a end

when called as IP(A[i], B[i], 70, i) computed the inner product of the vectors A and B, but
when called as IP(C[i,i], D[k,i], 70, i) computed the inner product of the diagonal of C and
the kth row of D. This use of evaluation on the call side of an internal variable of a proce-
dure is called Jensen’s device, named after Jorn Jensen of the Danish Regnecentralen,
who first noted this use of call by name parameters.
ALGOL 60 Implementations & Impact

- Implementation efforts in Europe and US; available on most major computers (but often University efforts)
- Many standard techniques pioneered/discovered
  - e.g., stack frames for recursive procedures: “Recursive Programming” by E. W. Dijkstra
- “ALGOL 60 is slow” – reputation compared to FORTRAN because of mismatch with (hostile?) computer architectures
  - Can a language (vs an implementation) be said to be “slow” or “fast”?
- Burroughs 5000 – stack machine designed to run ALGOL
  - OS and compilers written in ALGOL
  - But FORTRAN arrays were slow – hardware/software mismatch
- FORTRAN had too much of a lead for ALGOL 60 to displace it. Lack of standard I/O and dialect differences didn’t help.
COBOL 60 Common Business Oriented Language

- Goal: come up with a common language to handle business data processing – sponsored by DoD
- Key technical contribution was attention to data layouts – the original records (struct, each-of, etc.)
  - Particular attention to mapping program data to external storage layout
  - Hierarchical data organization
- Program logic separated from data and environment defs.
- Some hope that English-like statements would make it possible for “end users” to write programs
- Dominant business programming language into the 90s, and your paycheck is probably printed by it today
COBOL 60

SMALL SAMPLE OF ACTUAL COBOL ENVIRONMENT DIVISION:

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
OBJECT-COMPUTER. IBM 705.
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT INPUT-FILE-1
   ASSIGN TO TAPE 5.
   SELECT FILE-2 ASSIGN TAPE 6.

SMALL SAMPLE OF ACTUAL COBOL DATA DIVISION—FILE SECTION:

DATA DIVISION
FILE SECTION
FD INPUT-FILE-1; RECORDING MODE IS F;
BLOCK CONTAINS 5 RECORDS;
LABEL RECORDS ARE STANDARD;
DATA RECORD IS INPUT-RECORD.

SMALL SAMPLE OF ACTUAL COBOL DATA DIVISION—RECORD DESCRIPTION

01 INPUT-RECORD.
   02 NAME CLASS IS ALPHABETIC;
      SIZE IS 40.
   02 ID-NUMBER PICTURE 9 (10).
   02 ADDRESS CLASS IS ALPHANUMERIC;
      SIZE IS 47.
   03 STREET PICTURE 9 (40).
   03 STATE PICTURE A (2).
   03 ZIPCODE PICTURE 9 (5).

SMALL SAMPLE OF ACTUAL COBOL PROCEDURE DIVISION:

OPEN INPUT EMPLOYEE-FILE,
   OUTPUT FILE-1, FILE 2.
SELECTION SECTION.
PARAGRAPH-1. READ EMPLOYEE-FILE
   AT END. GO TO YOU-KNOW-WHERE.
IF FIELD-A EQUALS FIELD-B PERFORM
   COMP ELSE MOVE FIELD-A TO
   FIELD-B.
mid 60s: PL/I – If FORTRAN and COBOL are a good idea, let’s combine them

- Big idea: combine scientific and business computing in one language, just like IBM 360 hardware for both
- Led by IBM and IBM user groups
- Variety of data types for numeric and string processing, bits, COBOL-like string editing, array expressions, records, but...
- Lessons learned about unexpected interactions when language features are combined
- Rudimentary exception handling (ON-conditions)
- Shipped on IBM mainframes, but implemented by other manufacturers and fairly wide use in 60s-70s.
- Primary implementation language for MULTICS (Bell Labs, MIT, GE “information utility” project)
Application Languages: APL

- APL: A Programming Language (Kenneth Iverson, 1961)
- Data objects: arrays and matrices, also significant use in hardware modeling (hardware = arrays/matrices of bits)
- Operations: Individual operations on array elements, but real power was in higher-level operators on arrays like map, fold, reduce, transpose, inner & outer product, etc.
- Elaborate mathematical character set: used a special golf-ball element for IBM typewriters
- Implementation: interpreter; early implementation was APL\360, APL2 followed in 70s, 80s
- Descendants still used in financial community (A+)
Application Languages: SNOBOL

- String processing language developed at Bell Labs in the 60s
- Pattern matching; unusual control structures

```
COUNT = TABLE()
LETTERS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
GETWORD = BREAK(LFTTERS) SPAN(LETTERS) . WORD

READ     LINE = INPUT         :F(PRINT)
NEXTW    LINE    GETWORD =   :F(READ)
            COUNT[WORD] = COUNT[WORD] + 1  : (NEXTW)
PRINT    COUNT = CONVERT(COUNT,"ARRAY")   : F(ERROR)
NEXTI    I = I + 1
END

OUTPUT = COUNT[I,1] ":" COUNT[I,2]    :S(NEXTI)
```

Frame 8. SNOBOL4 word counter.
SIMULA: Object Oriented Programming

- Developed at the Norwegian Computing Center, Oslo, by Nygaard and Dahl
- Goal was a language that could be used for system description and simulation
- Started in 1961, SIMULA I in 1964, SIMULA 67
- Layered objects and classes on top of ALGOL 60 (although not always easy to recognize to modern eyes), virtual functions (dynamic dispatch)
- Quasi-concurrency – activation stack as a graph; coroutines

```
activity car;
begin real V, X₀, T₀;
    real procedure X; X := X₀ + V * (time - T₀);
    procedure update (Vnew); real Vnew;
    begin X₀ := X; T₀ := time; V := Vnew end;

end;
```
ALGOL 68 – A Successor to ALGOL 60

- Done by an international committee with heavy European representation
- Very generalized, “orthogonal”
- Complex definition – 2-level grammar (CFG for static semantics to generate the grammar that generated type-correct programs)
- Some implementations, some influence, particularly in Europe, but never widely used in US

Most important influence may be that it led Wirth to resign from the ALGOL 68 committee and go off in a different direction...
1970s Pascal

- **Influences**
  - Dijkstra’s *Structured Programming*, and programming methodology in general (the “software crisis”). Writing programs that are correct and understandable from first principles.
  - Hoare’s *Notes on Data Structuring*: types as a language concept; fundamental combining operations: records, sequence, recursive data structures (typed pointers)

- **Goal was to produce a small language suitable for teaching and developing real systems**

- **Touchstone language for 20+ years, and dominant teaching language from late 70’s to at least early 90’s**
  - But not perfect: limitations in type system, e.g., array bounds were part of the type, so couldn’t write general matrix multiply; difficult to get at the bits for very low-level programming; “The Program” vs modules
Pascal Implementations

- Initial implementation written in Pascal (several thousand lines), then hand compiled to CDC assembly language
  - Fixed a dozen bugs, then recompiled itself to become self hosting
- Pascal-P portable compiler by 1974, written in Pascal
  - Compiler generated code for a simple stack machine (p-code)
  - Stack machine interpreter supplied in Pascal, but easy to recode in almost anything else
  - Once the interpreter was running, it could be used to run the compiler and modify it to generate native code for the local machine
  - Pascal found on almost every known computer within a couple of years
  - Also found its way onto microcomputers for teaching: UCSD Pascal
- Used in commercial systems: Original Mac OS and software stack written in Pascal (+ core assembly language)
1973 C (ANSI C in 1983)

- Developed at Bell Labs in early 70s, same timeframe as Pascal
- Ancestry is CPL (Strachy, Cambridge) -> BCPL -> B -> C
  - (C is B with byte addressing instead of words)
- Programs are a collection of functions, one of which is “main”
- Unlike Pascal, designed to allow programmer to get close to the hardware, and no attempt to protect programmer from himself (“the programmer knows what he’s doing”)
- Primary implementation language for Unix
  - Therefore became ubiquitous when Unix became ubiquitous on microcomputers and early workstations
Abstract Data Types and Encapsulation

- By the early 70’s modularity emerged as a dominant theme in language design

- Key ideas:
  - Encapsulation / information hiding: systems should be built from modules connected by narrow interfaces; implementation details should be private/hidden
  - Abstract Data Types: Data abstractions consist of both the data structures themselves (linked list, array, whatever) and the operations on them (stack push/pop/top), and these should be packaged together

- Research languages included CLU (Liskov, MIT), Alphard (Wulf, Shaw, CMU)
  - Focus was modules and ADTs, not objects as in Simula
Late 70’s: Mesa (Xerox PARC)

- Modular programming
  - Each module has two or more source files: definition (interface) plus one or more implementation files
- Strong type checking across module boundaries
  - But “unsafe” modules could be used for low-level programming
- Exception handling
- Developed on the Xerox Alto
- Successors included Cedar (added gc among other things)
- Implementation language for Xerox Star – first WYSIWYG workstations (commercial flop, but then there was the Mac...)
- Strong influence on Modula 2, Ada, Java...
1980 - Ada

- DoD sponsored language to replace a cacophony of languages inside DoD with a single, safe language
- Strongly typed, modules (but not objects originally), dynamic storage management, exception handling, generics
- Explicitly addressed concurrency in the language definition
- Focus on compile-time checks to avoid runtime errors

- Reasonably successful in safety-critical and other DoD applications, but expensive compilers, etc. Never became the dominant language for mainstream programming
Modula and Oberon

- Wirth’s successors to Pascal
- Modular programming
- Modula 2 after Wirth spent a sabbatical year at Xerox PARC in 1976, then went home and created his own language and workstation hardware to run it
- Oberon added objects a decade later

- Modula 3 developed by others at DEC SRC late 80’s
  - Lots of PARC people; the “next Mesa”?  
  - Almost became the “next” teaching language, but then the Java stampede happened
Smalltalk

- Developed at Xerox PARC in early 70’s, Alan Kay
  - First version in 1972; significant revision in 1976
- Smalltalk 80 was the widely released version
  - Language + environment, graphics, personal machines, rapid prototyping / exploratory programming, programming for kids; Dynabook vision
  - Lives on as Squeak
  - Still used in the financial community for fast prototyping and modeling

- Concepts
  - Everything is an object
  - Objects are instances of classes
  - Computation is objects sending messages to each other

- Build a system that had the right abstractions; the hardware will eventually catch up

- Implementation: Smalltalk virtual machine – byte code interpreter
- Research implementation at Berkeley on early Sun workstations
  - Generational GC (Ungar) among other things
1987-95: Self

- David Ungar and Randall Smith at Xerox PARC
- Question: If an object-oriented system is all about objects sending messages to each other, why do you need classes?
- Self is all about objects and messages
  - Interactive environment like Smalltalk
  - With no classes, create new objects by cloning existing ones
- Implementation technology: To get adequate efficiency implementation needs to discover commonalities between objects, inline function calls aggressively, dynamic caches, ...
  - Key ideas behind today’s Javascript compiler arms race come from the Self papers from 20 years ago
  - Code from Craig Chambers’ PhD thesis under Ungar is said to be recognizable in Java’s current Hotspot virtual machine
1980s – C++

- Developed by Stroustrup at Bell Labs
- Initial goal was to build something as expressive as Simula for simulations, but with the runtime efficiency of C
- First implementation was as a set of C preprocessor macros(!)
  - “C with Classes”
- Quickly turned into a real programming language with C as its (almost completely unmodified) core
- Huge language – many pragmatic decisions, lots of things that make PL types queasy
- If you read the papers, the big-picture design and vision have been fairly consistent for 20+ years
1995 - Java

- Early 90s: Sun decides it wants to sell more SPARC chips by selling embedded systems development kits
  - But need a software development environment to do that
- Considered Smalltalk(!) (too expensive), C++ (too complex)
- Designed Oak language instead – subset of C++ heavily influenced by Smalltalk, Mesa, others
- Then two non-technical influences: internet, Microsoft
  - Internet as a “platform” alternative to Windows/msft domination
  - Pointy-headed bosses stampede: Java, Java, Java; web, web, web
- Trademark search: Oak can’t be used – so it’s renamed Java
- Chaos ensues: Java everywhere, interns everywhere to implement much larger libraries, etc.
Java technically

- Safe, strong typing, attempts to have no semantic loopholes
  - Generics added in Java 1.5, 2004

- Concurrency and garbage collection baked in

- Portable: compiler target is a byte code machine (.class files)
  - Compiler output can be interpreted directly (original JVM and current Hotspot), or compiled to native code (Hotspot)
  - .class files contain symbolic information about compiled classes, not just executable byte codes

- Just-in-time compilers (JIT): monitor code as it runs, identify frequently executed code, then compile on the fly into native code; backpatch interpreted code to jump to compiled code
  - JIT compiler has all the information available to typical optimizing compilers (from .class files) and performs standard optimizations
  - Performance comparable to C/C++ these days for many things
C# / Common Language Runtime

- **Background - Java**
  - Microsoft had one of the best Java 1.0/1.1 environments; started adding “extensions” to standard libraries to make code tie better to Windows
  - Sun sues Microsoft for violating “pure Java” contract; Microsoft loses, never able to get license for Java 1.2 (new collection classes) and later

- **Background - DLL Hell**
  - Problems with incompatible versions of dynamically linked libraries trying to coexist on the same system for different programs

- **Technical (& business) solution: Common Language Runtime and Java-like language C#, with Windows extensions**
  - CLR incorporated ideas from a wide selection of the PL community
  - Extensions allow for unsafe modules, mixing managed code with older code that uses old abstractions/runtime structures (COM, DCOM)
  - Microsoft Intermediate Language (MSIL) is a lot like Java bytecodes
    - One key difference: always compiled to native code before execution
Meanwhile, in the Land of LISP...

- **LISP was the dominant language in the AI community throughout the 60’s and 70’s**
- **By the mid 60’s dialects started to proliferate:**
  - MacLisp (MIT)
  - BBN-LISP
  - Interlisp (Xerox PARC)
  - Various LISP machines (special-purpose machines)
  - Franz Lisp (Berkeley Unix)
  - Others...
- **1975: Scheme (MIT, Sussman & Steele; Steele’s MS thesis)**
- **1984: Common LISP – DoD ARPA attempt to mandate a common dialect (so groups they funded could share code)**
  - Much petty behavior, hurt feelings, and rivalries along the way
Functional Programming – ML family

- ML developed in early 1970s at Edinburgh (Milner & others)
- Original use as a language for writing proof tactics for automatic theorem proving systems
- Major research results in type inference and type systems (Hindley-Milner algorithm), polymorphism

Modern dialects

- SML (Standard ML) 1990, 1997
- OCaml (INRIA, France) 1996
- F# (Microsoft, standard part of Visual Studio 2010)
Functional Programming – Haskell

- Also a strong, statically typed functional language
- Originally defined in late 80’s, first release in 1990, core group at Glasgow
- Key difference: lazy evaluation is the norm
- Many contributions to type theory and language design
- Haskell draws a careful distinction between the purely functional part and impure code; theory of Monads to deal with I/O and other side effects in a functional system
- Now mostly hosted at Microsoft Research, Cambridge (England)
  - Right down the hall from the F# folks
Functional Programming redux

- First-class functions, polymorphic types, immutable data, type theory
- These have been around for 30+ years, but are starting to show up in all sorts of interesting places
  - Databases (Microsoft LINQ)
  - Big data & concurrency (Google MapReduce, open source Hadoop)
  - Mainstream languages (lambdas and closures in recent Java, C#)
  - Parallel programming (multicore)
  - Software tools for analyzing bugs, safety, more...
  - Next?
Of things not covered

- Basic
- “Visual programming” languages
- Languages for beginners / non-programmers: Logo, Processing (artists as well as beginners), Alice
- Constraint and logic languages (prolog, clpr, excel(!))
- Objective C (C meets Smalltalk, the “other” object-oriented extension to C; used in NeXt/Apple systems, your iGadget)
- Scripting languages (Perl, Python, Ruby, ...)
  - Ruby is the most interesting of this bunch, combining scripting with Smalltalk semantics and other PL ideas
- Javascript
- Many more...
Language Futures

- (Editorial opinion) The Java stampede knocked the wind out of new programming language development for a decade

- New ideas have started to get traction in the last few years
  - Languages built on top of JVM (Clojure, Groovy, Python and Ruby implementations)
  - New languages that combine functional and object-oriented programming in interesting ways: Scala is a high-profile example

- Programming now is more about plugging components together than in the old days, where hard-core CS was essential

- What language do you think you’ll be using in 10 years?
- What ideas will you contribute?