A brief survey of programming languages: motivation and design/ continued

Lecture 3

Life beyond Fortran (1959), Algol (1960), Lisp 1.5 (1960), Cobol (1961)

"Significant" languages, commercial and/or intellectual
COBOL (1961, 66, .. PL/I),
BASIC (1964) significantly "interactive" at Dartmouth Univ;
(Algol-like languages: PL/I, Pascal, Algol 68, B, C, C++,
Modula, Scheme (1975), Ada, Java, C#),
String-processing languages (snobol I-IV, tcl, Perl),
Functional languages (APL, ML, Haskell, Logo, Lisp),
Visual programming,
Descendents of Fortran, 1959 (I,II, IV, 66, PL/I, 77, 90).
Stack languages (IPL, FORTH, PostScript)
Logic programming (Prolog)
OOP (Simula67, Smalltalk, C++, CLOS/lisp)
Interactive Math (Matlab, Maple, Mathematica, MathCAD)

Even more, Algol, Cobol

There are other languages as well, especially in particular application areas (e.g. controlling machine tools, telescopes, controlling computer jobs, generating reports from databases) but certainly many languages that intend to address a general audience. There is also a journal of the history of programming languages, and a number of survey books.

And there are newsgroups: `comp.lang.*`
There are over 100 language groups, some subdivided further (C++, Java, Lisp))

Are they really different?

If I've missed your favorite programming language, sorry.

There is a tendency for each of us to think that all programming languages must look pretty much like the first programming language learned. E.g. everything is like Basic. or Pascal. or C.

When you learned Scheme, did you change your mind?

So is everything essentially like Java or essentially like Lisp?

Yes and no.. Programming languages that look quite different include snobol, prolog, postscript...

Are they really different?

We will see that the most visible part of a language, its syntax, is almost the first thing removed by a compiler. So much so that after eliminating the "syntactic sugar" many languages are nearly identical.

There are still issues that transcend superficial language differences beyond the syntax:
Variables, memory, scope, the balance between few primitives + extension vs. many built-in features, security, and other notions (much later on in this course).

A closer look at Fortran

If we look at the first Fortran (= Formula Translator) from a modern perspective it seems terribly restricted (IF, DO, GOTO).

But the goals of the Fortran project at IBM were to produce efficient code because it was believed that the only way to succeed versus assembler was to produce fast code.

This was false, but in 1959, who could tell...

A closer look at Fortran

Other restrictions forced by reality of IBM 704 computer: Any compiler had to run on machines that were tiny and slow by our standards.

We can explain the 3-way IF by a machine instruction on IBM 704

IF(a) 1,2,3

A closer look at Fortran

"One of the 704's unusual features was that core storage used signed magnitude, the arithmetic unit used 2's complement, and the index registers used 1's complement. When FORTRAN was implemented on the IBM product that replaced the 704, 7094 etc. series, the 3 way branching if went to the wrong place when testing negative zero.
(It branched negative, as opposed to branching to zero). "

A closer look at Fortran

Statement numbers as targets for GOTOs or branches

i,j,k... n started integer variables. N23 was an integer.
other letters started floats. X43 was a float.
No other declarations.
Easy to make mistakes by misspelling.
One way around this is to require humans to
type everything at least twice (=declarations), as in Java.

A closer look at Fortran

Fortran had weak input and output; not as bad as Algol which had NO input/output.

Fortran was a significant step up from what went before… there were symbolic assemblers and some 'higher level' tools – for example, packages that implemented floating point instructions as macros.

A closer look at COBOL

COBOL (1960) (Common Business Oriented Language) by contrast to Fortran was (and still is) almost ALL input/output. Big features: provided for records of characters and numbers. Written by a committee under government sponsorship, COBOL became an important standard, and is still widely used.

We've never taught much about it here, even though the university payroll may rely on it.

A closer look at Algol (60)

Algol 60 (Algorithmic Language) was a very influential language, a quantum leap over its predecessors (and many of its successors!).

The Algol 60 Report, first used BNF: a formal grammatical presentation of the syntax of the language. Combined with natural language semantic descriptions, the Report was a breakthrough in defining a programming language. (There is an attempt to follow this route in the Java Language Specification.)

Call by value and call by name.

NO Input/Output.

Recursion (it was not in Fortran).

A closer look at Algol (60)

There were many attempts to clean up the trouble spots and include extra features like I/O. The most commonly used was probably Pascal (1968).

Another re-design, Algol 68 had limited appeal (too complex) and never caught on.

Pascal/(Pascal named for Blaise Pascal, French mathematician) By Niklaus Wirth … rode a wave interest in "Structured Programming (Dijkstra's Goto considered harmful, 1968) http://www.computerhistory.org/timeline/1968/dijkstra_goto.page

Wirth/Pascal pushed case /while/ repeat-until/ functions as primary control structures.

Scheme is in some ways like Algol in Lisp syntax.

A closer look at Lisp

Lisp (List Processor) was designed in part around symbolic manipulation, in particular encoding arithmetic tree expressions. Weakly related to Weakly related to IPL-V, COMIT.

A motivating test was to program a differentiation procedure.
(nice features: using functions,e.g. with MAP, recursion, interaction). Original lisp 1.0 or 1.5 (1959) was 'even uglier'.
The inspiration of lambda-calculus provided some notation. Garbage collection was invented first for Lisp.

The language developed into many threads from which two competing approaches eventually emerged: Scheme vs. Common Lisp

Some other spots along the timeline

1969:UNIX released
1972 PONG and Atari Corp

1979 Visicalc predecessor of Excel

1985 C++ made OO more respectable. Origins in Simula (1967), widely used in "real" Lisp, but who is counting..

1987 Hypercard for Apple Mac
1990 Windows 3.0

Visual programming

What is the best programming language?

Not a well-formed question.

What is the best transportation? An electric car, mini-van, a greyhound bus, a Boeing 747, an F-16 fighter plane, a container ship, a kayak?

"What is your favorite language?"
(You might object: for what?)

Possible objectives for programming language

speed of compiled code (depends on implementation, but…)
coverage – e.g. does complex double floating-point..
applicability – web? database?
portability or availability on machine X.
Availability of many programmers who know the language.
EASE OF PROGRAMMING CORRECT PROGRAMMING LANGUAGE IMPLEMENTATIONS ☺

Other characterizations

Syntactic structure:
Nature of tokens, numbers, statements, key words. (most visible, least important)

Approach to generality
extensible (Scheme) vs. inclusive (PL/I, Java+libraries)

Data semantics:
Compound objects (vectors, arrays, lists)
Declarations of types (of data) associated with names

Other characterizations

Execution semantics: e.g. backtracking?

Subroutines, libraries

parallelism (threads, exceptions)

input/output complexity

access to machine ops (assembler)

Other characterizations

Ability to make a program formally correspond to a specification (possibility of proving a program correct!)

Extensibility of types, parameterized types, classes, inheritance.

Other characterizations

Can we extract out of this some systematic evaluation criteria?

Characteristic	Effect on 3 criteria		
	readability	writability	reliability
simplicity	x	x	x
types	x	?	x
type checking			x
"good syntax"	x	x	x
abstraction		x	x
expressivity		x	x
exception handling			x

Studying history of PL can contribute to ..

(a) better understanding of the relationships between languages/compilers/programs

(b) understanding concepts and features common to existing and future programming languages (learn them more easily)

Example programs in a few languages

Selected just to remind you that not all languages are just like the ones you already know.

Example programs FORTH

:WASHER WASH SPIN RINSE SPIN; <return>

defines the program WASHER which calls each of those other programs.

15 SPACES <return>

makes the computer print 15 spaces.

3 4 + <return>

makes the computer leave 7 on top of a stack

3 4 + . <return>

makes the computer leave 7 on top of a stack, then print it.

SIMILAR to: Postscript. HP "reverse polish" calculators.

Example programs Logic (e.g. prolog)

definition of append. Various ways of 'reading' the program. Here's one way.

append([ ],y,y)

It is true that appending [ ] and y gives you y.

append(h|x,y,h|z) if append(x,y,z)

It is true that appending the string h|x, (like (cons h x) in lisp), to the string y gives you the string h|z, if it is true that appending x and y gives z.

There are also interpretations for proof or search.

Example programs Fortran

```
      DO 10 I=1,N
       IF F(I) 10,12,10
10    CONTINUE
      GOTO 13
12    WRITE(6)I
13
C     print out the first value of i<= where f[i]=0.
...
      END
```

Similar to Fortran II, 66, 77, PL/I, BASIC

Example programs LISP

```
;; some variants of a simple program, all legal Common Lisp.
;; some people like programs with lots of keywords and not so
;; many parentheses.

(defun power(x n) ;recursive, but also slow
   (if (= n 0) 1 (* x (power x (1- n)))))

(defun power(x n)                          ;still recursive, but faster
  (labels
     ((square (x)(* x x)))
        (cond((= n 0) 1)
             ((evenp n)(square (power x (/ n 2))))
             (t (* x (power x (- n 1)))))))
```

Example programs LISP

```
;; some  more variants of a simple program,
;; all legal Common Lisp

(defun power (x n) ;; iterative
  "compute x^n, integer n>=0"
  (loop with result = 1
        for exp = n then  (floor exp 2)
        for sqr = x then (* sqr sqr)
        until (zerop exp)
        when (oddp exp) do
        (setf result (* result sqr))
      finally (return result)))

(defun power (x n)                          ; compute x^n
  (let ((res 1))
    (do ((exp n (floor exp 2))
                 (sqr x (* sqr sqr)))
                ((zerop exp) res)  ;test, return-value
      (if (oddp exp)(setf res (* res sqr))))))
```

Example programs LISP

```
;;more variants of a simple program, all legal Common Lisp


(defun power (x n)
  "compute x^n"
  (assert (integerp n)(n)"Power: The exponent ~s should be an integer" n)
  (assert (>= n 0)    (n)"Power: The exponent ~s should be non-negative" n)
  (assert (numberp x) (x)"Power: The base ~s should be a number" x)

  (do ((res 1 (if (oddp exp)(* res sqr) res))
       (exp n (floor exp 2))
       (sqr x (* sqr sqr)))
      ((zerop exp) res) ))
```

Example programs LISP

```
;; yet more ...all legal Common Lisp

(defun power (x n)
  "compute x^n"
  (declare (optimize (speed 3)(safety 0)(debug 0))
           (fixnum x n))
  (do ((res 1 (if (oddp exp)(* res sqr) res))
       (exp n (floor exp 2))
       (sqr x (* sqr sqr)))
      ((zerop exp) res)
    (declare (fixnum res exp sqr)))) ;; 120 bytes
```

More example programs LISP

```
;; A bunch of programs to make a list (1 2 3 ….. N)
(defun L2(n) (L2a 1 n))

(defun L2a(s e)                                    ;s=start, e=end
  (if (<= s e)
      (cons s (L2a (1+ s) e))))

;; a strange version using map, and one using dotimes…
(defun L3 (n) (addem (make-sequence 'list n :initial-element 1)))
(defun addem(r)  (if r (cons (car r) (addem (map 'list #'1+ (cdr r))))))

(defun L4(n) (let ((s nil)) (dotimes (i 10 (reverse s)) (push (1+ i) s)))

;;  A very Un-lispy version using "Loop"

(defun L1 (n)  (loop for i from 1 to n collect i))
```

Why the external LISP is almost arbitrary

;;;;;;;;;;Loop is defined as a “macro” expanded before execution.

(macroexpand '(loop for i from 1 upto 10 do (print i)))

==> ... something like this...

```
(let ((i 1))
  (declare (type real i))          ;; otherwise (> i 10) meaningless
  (block nil
    (tagbody
     next-loop (print i)           ;; next-loop is a LABEL
           (setf i (1+ i))
           (when (> i 10) (go end-loop))
           (go next-loop)
     end-loop)))                   ;; end-loop is also a LABEL
```

Summary

Many languages have been designed and implemented. Some of them are interesting, some are meritorious for various reasons. There is not one "best" language for everything.

Prof. Fateman's opinion: if you are prototyping a language implementation, Lisp helps a lot.