Programming Languages History

CSE 413, Autumn 2007

11-02-2007

Topics

- Big Picture
- Where Scheme, C, C++, Java fit in
- Fortran
- Algol 60

Donald Knuth:

» *Programming is the art of telling another human being what one wants the computer to do.*

Programming Domains

- Scientific Applications:
  - » Using The Computer As A Large Calculator
  - » FORTRAN, Mathematica
- Business Applications:
  - » Data Processing And Business Procedures
  - » COBOL, Some PL/I, Spreadsheets
- Systems Programming:
  - » Building Operating Systems And Utilities
  - » C, C++

Programming Domains (2)

- Parallel Programming:
  - » Parallel And Distributed Systems
  - » Ada, CSP, Modula
- Artificial Intelligence:
  - » uses symbolic rather than numeric computations
  - » lists as main data structure, flexibility (code = data)
  - » Lisp 1959, Prolog 1970s
- Scripting Languages:
  - » A list of commands to be executed
  - » UNIX shell programming, awk, tcl, perl

Programming Domains (3)

- Education:
  - » Languages Designed Just To Facilitate Teaching
  - » Pascal, BASIC, Logo

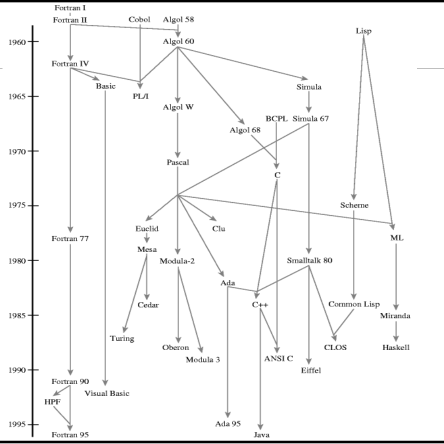
C History

- Developed in early 1970s
- Purpose: implementing the Unix operating system
- *The C Programming Language*, aka `K&R‘, for its authors Brian Kernighan & Dennis Ritchie, is canonical reference (1978)
- Evolution: ANSI, C99

Lisp & Scheme History

Lisp:
- McCarthy (1958/1960) for symbolic processing
- Based on Chruch’s Lambda Calculus
- 2 main Dialects: Scheme & Common Lisp
- Aside: Assembly language macros for the IBM 704 : car (Contents of Address Register) and cdr (Contents of Decrement Register)

Scheme:
- Developed by Guy Steele and Gerald Sussman in the 1970s
- Smaller & cleaner than Lisp
- Static scoping, tail recursion implemented efficiently



Fortran

Before High Level Languages

- Early computers (40’s) programmed directly using numeric codes
- Then move to assembly language (requires an assembler)
- Assembly language – slightly more friendly version of machine code.

FORTRAN

- **"Formula TRANslating system":**
- **original proposal in 1954, John Backus, IBM 704**
- **based on assembly language**
- ***language design* was secondary to task of *writing a compiler***
- **much skepticism about higher-level languages. Fortran had to be efficient to be accepted.**
- **Fortran was enormously successful; still in widespread use. (Fortran77, Fortran 90)**

Quotes from John Backus:

“As far as we were aware, we simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler which could produce efficient programs." [hopl-1 1978]

The project was to design a language for the IBM 704, NOT a general language:

"We certainly had no idea that languages almost identical to the one we were working on would be used for more than one IBM computer, not to mention those of other manufacturers."

13

"Unfortunately we were hopelessly optimistic in 1954 about the problems of debugging Fortran programs (thus we find on page 2 of the Report: "Since Fortran should virtually eliminate coding and debugging...") and hence syntactic error checking facilities were weak."

14

"In our naive unawareness of language design problems of course we knew nothing of many issues which were later thought to be important, e.g. block structure, conditional expressions, type declarations – it seemed to us that once one had the notions of the assignment statement, the subscripted variable, and the DO statement in hand, then the remaining problems of language design were trivial: either their solution was thrust upon one by the need to provide some machine facility such as reading input, or by some programming task which could not be done with existing structures."

15

FORTRAN "Features"

- 80 columns only
- Implicit variable declaration
- No reserved words
- Weak typing
- Lots of GOTOs
- Early: no recursion, subroutines came late.
- Independently compiled subroutines was a win.
- No records or heterogeneous data types.
- Common and equivalence for sharing data – not a win.

16

Control Statements in Fortran

- Based on IBM 704 branch instructions
  - » If statements
  - » Go to statements
- IF Statement:

  IF (expression) n1, n2, n3

17

Fortran GOTO Statement

Unconditional GOTO:

GOTO 31

Computed GOTO:

GOTO (30,31,32,33), I

Assigned GOTO:

ASSIGN 20 TO N

…

GOTO N, (20,30,40) ***these are just comments!***

18

Examples

Assign 20 to N

….

GOTO (20,30, 40,50), N

I = 3

…

GOTO I, (20, 30, 40,50)

19

DO LOOPS

```
      DO 100 I=1, N
      A(I) = A(I)*2
100   CONTINUE
```

20

Parameter Passing

- Pass by Reference for efficiency

```
SUBROUTINE DIST(D,X,Y)
D = X – Y
IF (D .LT. 0) D= -D
RETURN
END

CALL DIST(DIST1, X1, Y1)
```

21

Example

```
SUBROUTINE SWITCH (N)
N = 3
RETURN
END

CALL SWITCH(I)
```

22

Subroutines

- Implemented using activation records
- No recursion initially
- So only need one activation record per subprogram.

23

Common Blocks

- Named or un-named
- A way of sharing data between subroutines
- Equivalence allowed accessing the same memory locations through different names.
- A way to “re-use” storage locations.

24

Exercise

- Implement an if then else statement

Algol 60

Pre-History of Algol 60

- by mid 50's there were starting to be a proliferation of programming languages/assembly languages/pseudo codes.
- Interest in a universal programming language.
- European and American groups got together in may and june of 1958 in Zurich. --> result was algol 58.
- In 8 days, 8 members basically wrote the language.

Algol Goals

- new language should be as close as possible to standard math notation and readable with little further explanation.
- should be possible to use it for the description of "computing processes" in **publication**.
- should be mechanically translatable into machine programs.

Interesting History

- FORMAL GRAMMAR was used for syntactic description. BNF. IMPORTANT
- LANG was DESIGNED by COMMITTEE. 13 members met for 6 days in 1960.
- Report is a "paradigm of brevity and clarity"
  - » Most languages today require 1000's of pages
  - » BREVITY AND CLARITY contributed to the reputation as a simple, elegant language.

The meetings were exhausting, interminable, and exhilarating. ... Progress was steady and the output, ALGOL 60, was more racehorse than camel. This language proved to be an object of stunning beauty. It was sufficiently perfect and complete so that ensuing implementations were able to append necessities, such as input-output, in the style of ALGOL 60 but their addition propagated no significant changes in the body of the original language.

-- Alan Perlis, "The American Side of the Development of Algol," *The History of Programming Languages*

3 Language Levels

- Reference Language
  - » For use by committee, described in report and used in official Algol publications
- Publication Language
  - » Variation on reference language for publication of algorithms, could have Greek letters
- Hardware Representations
  - » Condensed languages for machine input

Three levels of language

"After two days of probing attitudes and suggestions, the meeting came to a complete deadlock with one European member pounding on the table and declaring: "No! I will never use a period for a decimal point". Naturally the Americans considered the use of comma as decimal point to be beneath ridicule. That evening Wegstein visited the opposing camps and proposed defining the three levels of language."

Alan Perlis, *The American Side of the Development of Algol*, ACM SIGPLAN Noticies, August 1978.

Algol Language Features

- Block Structure
- Explicit type declaration for variables
- Scope rules for local variables
- Dynamic lifetimes for variables
- Nested if-then-else expressions and stmts
- Call by value and call by name
- Recursive subroutines
- Arrays with dynamic bounds

Syntax and Style

- Free format
  - Indentation style
- Algol defined the style of most successors
  - Hierarchical structure
  - Nesting of environments and control structures
- Identifiers could be > 6 characters

Variables and Types

- Data types: integer, real, boolean
- No implicit declarations
- No double precision types
- No complex number types
- Arrays: > 3 dimensions, dynamic bounds, can start at something other than 0/1

I/O

- No I/O

Binding Time

- BINDING of names to locations is done ON ENTRY TO A BLOCK, not at compile time (as in Fortran)
- Stack is central run-time data structure

37

Blocks

- Can use a block of statements anywhere a single statement is needed.

```
begin
 declarations;
 statements
end
```

38

Blocks support structured programming

- Algol 60

```
if x=3 then
   begin
      y:=9; k:=10
    end;
```

- Fortran

```
      IF (X .NEQ. 3) GOTO 100
      Y=9
      K=10
100 ...
```

39

Blocks allow nested scopes

```
begin
 integer x;

 procedure squid;
   begin
     integer x;
     ...
     end;
end;
```

40

Blocks for efficient storage management

```
begin
  ...
  begin
    real array x[1:1000];
    ...
  end;
  ...
  begin
    real array y[1:2000];
    ...
  end;
 end;
```

41

Control Structures

- goto
- if-then-else
- for loop
- switch

42

Call by value vs. call by name

- Call by Name is default
- **call by name:** re-evaluate the actual parameter on every use.
  - » For actual parameters that are **simple variables**, this is the same as call by reference.
  - » For actual parameters that are **expressions**, the expression is re-evaluated on each access.

43

Call by value vs. call by name

```
begin
  integer n;
  procedure p(k: integer);
    begin
    print(k);
    n := n+1;
    print(k);
    end;
  n := 0;
  p(n+10);
  end;
```

44

Fortran example

```
SUBROUTINE S(EL, K)
K = 2
EL = 0
RETURN
END
A(1) = A(2) = 1
I=1
CALL S (A(I),I))
```

45

Algol 60 Example

```
procedure S (el, k);
   integer el, k;
   begin
        k := 2;
        el := 0
   end;
A[1] := A[2] := 1;
i := 1;
S (A[i], i);
```

46

What happens here?

```
begin
  integer n;
  procedure p(k: integer);
     begin
     print(n);
     end;
  n := 5;
  p(n/0);
  end;
```

47

The life of Algol 60

- Didn’t achieve widespread use
- Extremely important in the history of PLs
- Successors: Pascal, Modula-2, Ada, many more.
- Produced important work on lexical analysis, parsing, compilation techniques for block-structured languages.

48