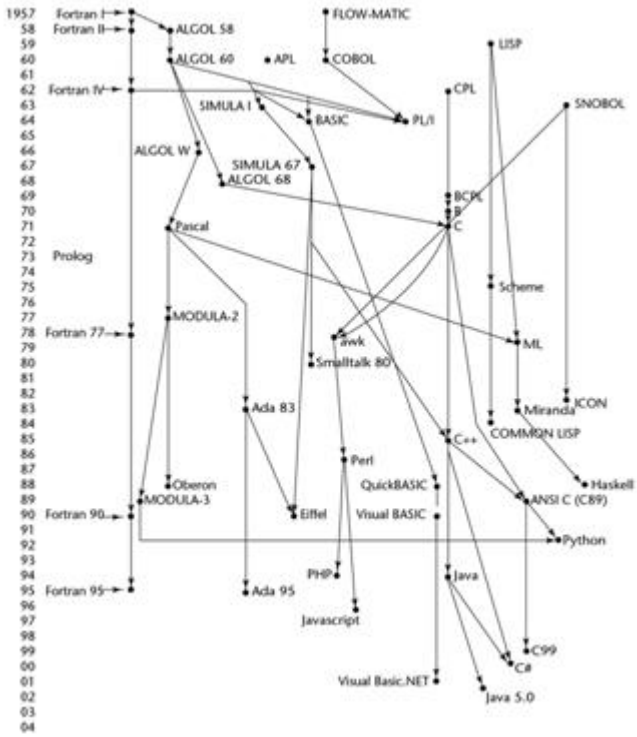
Evolution of the Major Programming Languages – Chapter 2



Zuse's Plankalkul

Historical Background

- Konrad Zuse ("Tsoo-zuh") – German scientist
- 1943 – Ph.D. dissertation proposal – language for expressing expressions
- Language named Plankalkul – means program calculus
- Never implemented
- 1945 – unpublished manuscript defined language along with algorithms for solving a wide variety of problems – published in 1972

Language Overview

- Advanced data structures
    - Single bit
    - Integer types
    - Floating-point numeric type – twos-complement notation
    - Arrays and records
- Included an iterative statement similar to the Ada for
- Included mathematical expressions showing variable relationships

Example of a PlanKalkull assignment statement:

	| A + 1 => A	equivalent to	A[5] = A[4] + 1
V	| 4 5		array subscripts
S	| 1.n 1.n		data types – integer of n bits

Minimal Hardware Programming: Pseudocodes

Machine code programming

- Tedious and error-prone
- Difficult to read – numeric codes specified instructions
- Difficult to modify – add/delete instructions – absolute addressing
- Machine deficiencies – no indexing or floating point arithmetic

Short Code

- 1949 – John Mauchly for the BINAC computer – never published
- 1952 – UNIVAC I – Remington Rand – programming manual
    - Word of memory – 72 bits – 12 six-bit bytes
- Consisted of coded versions of mathematical expressions
- Used a pure interpreter – this was referred to as automatic programming

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

X0 = SQRT(ABS(Y0))	00	X0	03	20	06	Y0

Speedcoding

- 1954 – John Backus for the IBM 701
- Interpreter converted the 701 into a virtual three-address floating-point calculator
- Include pseudoinstructions arithmetic and math operations
- Conditional and unconditional branching
- 700 words of usable memory
- Add instruction – 4.2 milliseconds to execute
- Automatically incrementing address registers – array and matrix access

The UNIVAC “Compiling” System

- 1951 to 1953 – Grace Hopper led team at UNIVAC
- Expanded pseudocode into machine code subprograms

Related Work

- 1950 – David J. Wheeler at Cambridge University
- Developed a method of using blocks of relocatable addresses to partially solve the problem of absolute addressing

The IBM 704 and Fortran

Historical Background

- IBM 704 included both index registers and floating point hardware.
- Fortran is often credited as being the first compiled high-level language – Laning and Zierler system was the first implemented translation system – MIT 1954.

Design Process

- 1954 – Fortran 0 – never implemented
- FORmula TRANslating system: FORTRAN
- Environment in which Fortran was developed:
    - Computers were small, slow, and unreliable
    - Computers primary use was scientific computation
    - No existing efficient ways of programming computers
    - Primary goal was machine efficiency

Fortran I – 1957

- Involved 18 worker-years of effort
- Formatted I/O
- Variable names up to 6 characters – Fortran 0 was 2
- User-defined subroutines
- Arithmetic If statement
- Do loop statement
- No data typing
    - I, J, K, L, M, N – first letter of a variable name – implicitly integer
        - Scientists used i, j, and k as subscripts
    - Others – implicitly floating-point
- 300 to 400 lines of code max – poor reliability of the IBM 704

If (arithmetic expression) N1, N2, N3

N1 – statement label – branch to if expression is negative
N2 – statement label – branch to if expression is zero
N3 – statement label – branch to if expression is positive

Do N1 variable = first_value, last_value

N1 was the statement label of the last statement of the body of the loop, and the statement on the line following the Do was the first. Do loop was posttest.

Fortran II – 1958
- Fixed many of the Fortran I bugs
- Independent compilation of subroutines – not available in Fortran I

Fortran IV – 1960-1962
- Standardized as Fortran 66 (ANSI, 1966) – name rarely used
- Explicit variable type declarations
- Logical If statement
- Passing subprograms as parameters to other subprograms

Fortran 77 – (ANSI, 1978a)
- Character string handling
- Logical loop statements
- If-Then-Else (optional Else)

Fortran 90 – (ANSI, 1992)
- Modules
- Dynamic arrays
- Pointers
- Recursion
- Case statement
- Parameter type checking

Fortran 95 – (ANSI, 1997)
- Forall statement
- Removed from Fortran 90: Pause, Assign, assigned Goto, computed Goto, and arithmetic If statements

Evaluation

- Before Fortran 90 – types and storage for all variables are fixed before run time
  - Allows for highly optimizing compilers
  - Simplicity and efficiency over flexibility
  - No recursion and no dynamic data structures
- Dramatically changed forever the way computers are used
  - First widely used high-level language – still one of the most widely used
- Fortran is the *lingua franca* of the computing world. It is the language of the streets in the best sense of the word…

```
!  Fortran 95 Example program
!  Input:       An Integer less than 100 followed by a list of that many Integer values
!  Output:     The number of list values greater than the average of all list values
Implicit one
Integer :: Int_List( 99 )
Integer :: List_Len, Counter, Sum, Average, Result
Result = 0
Sum = 0
Read *, List_Len
If ( ( List_Len > 0 ) .AND. ( List_Len < 100 ) ) Then
!  Read input data into an array and computer its sum
        Do Counter = 1, List_Len
                Read *, Int_List( Counter )
                Sum = Sum + Int_List( Counter )
        End Do
!  Compute the average
        Average = Sum / List_Len
!  Count the values that are greater than the average
        Do Counter = 1, List_Len
                If ( Int_List( Counter ) > Average ) Then
                        Result = Result + 1
                End If
        End Do
!  Print the result
        Print *, 'Number of values > Average is: ', result
Else
        Print *, 'Error – list length is not legal'
End If
End Program Example
```

Functional Programming: LISP

LISP was invented to provide language features for list processing. This need grew out of the first applications in the area of artificial intelligence (AI).

The Beginning of Artificial Intelligence and List Processing

- AI appeared in the mid-1950s
    - Linguists – natural language processing
    - Psychologist – modeling human information storage and retrieval
    - Mathematicians – mechanizing intelligent processes – theorem proving
- A method must be developed to allow computers to process symbolic data in linked lists. At the time, most computation was on numeric data in arrays.

LISP Design Process

- 1958 – MIT AI Project – John McCarthy and Marvin Minsky
- First priority was to produce a system for list processing.
    - LISt Processing language – LISP – "pure LISP"

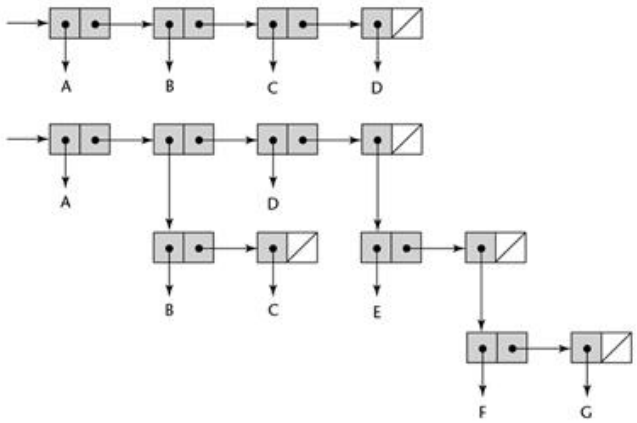
Language Overview

- Pure LISP has two kinds of data structures:
    - Atoms – symbols (form of identifiers) or numeric labels
    - Linked lists

Simple list of atoms – list of four elements: (A B C D)

Nested list structure – list of four elements: (A (B C) D (E (F G)))

atom A – sublist (B C) – atom D – sublist (E (F G))



Processes in Function Programming

- All computations are accomplished by applying functions to arguments.
- Iterative processes are specified with recursive function calls.

The Syntax of LISP

- Syntax is a model of simplicity.
- Program code and data have exactly the same form: (A B C D)
    - Data – list of four elements
    - Code – function named A with three parameters B, C, and D.

Evaluation

- LISP is the most widely used language for AI
- LISP pioneered functional programming
    - No need for variables or assignment
    - Control provided through recursion and conditional expressions
- COMMON LISP – standard version – resolve portability issue

```
;  LISP Example function
;  LISP predicate function that takes two lists as arguments and returns True
;  if the two lists are equal, and NIL (False) otherwise
        ( DEFUN equal_lists ( lis1 lis2 )
                ( COND
                        ( ( ATOM lis1 ) ( EQ lis1 lis2 ) )
                        ( ( ATOM lis2 ) NIL )
                        ( ( equal_lists ( CAR lis1 ) ( CAR lis2 ) )
                                ( equal_lists ( CDR lis1 ) ( CDR lis2 ) ) )
                        ( T NIL )
                )
)
```

Two Descendents of LISP

Two dialects of LSIP are now commonly used, Scheme and COMMON LISP.

Scheme – mid-1970s
- MIT
- Small language, simple syntax and semantics
    - Good for educational applications
- Exclusive use of static scoping
- Functions as first-class entities can be:
    - Values of expressions, elements of lists, assigned to variables, passed as parameters, and returned as values

COMMON LISP – 1996
- Effort to combine the features of several dialects of LISP
- Large and complex language

Related Languages
- ML – MetaLanguage – University of Edinburgh
- Miranda – University of Kent – based on ML, SASL, and KRC
- Haskell – based on Miranda

The First Step Toward Sophistication: ALGOL 60

ALGOL 60 has had a great influence on subsequent languages and is therefore of central importance in any historical study of languages.

Historical Background
- 1957 – proposal to design a universal language submitted to Association for Computing Machinery (ACM)
    - SHARE – IBM scientific user group
    - USE – UNIVAC scientific user group
- 1958 – GAMM – German Society for Applied Mathematics and Mechanics
    - Joined effort with ACM

Early Design Process – 1958 – Zurich
- Syntax close to standard mathematical notation – readable – scientific
- Language used for algorithm description in publications
- Programs translatable into machine language

ALGOL 58

- Zurich meeting name: International Algorithmic Language – IAL
- Name changed to ALGOrithmic Language – ALGOL
- Formalized data type – non floating-point explicitly declared
- Compound statement
- Identifier names any length
- Any number of array dimensions – lower bound can be specified
- Nested selection statements
- Assignment statement: variable := expression

Reception of the ALGOL 58 Report

- Not meant to be a finished product – MAD, NELIAC, JOVIAL implemented
- 1959 – IBM abandoned ALGOL – stayed with Fortran

ALGOL 60 Design Process

- Debate ALGOL 58 modification suggestions – Paris

ALGOL 60 Overview

- Block structure – localization – allows for new data environments, or scopes
- Two argument passing methods: pass by value and pass by name
- Recursive procedures – new for imperative languages
- Stack-dynamic arrays – subscript ranges are variable specified – array storage allocated when declaration statement is executed
- Formatted I/O rejected – machine dependent
- 1962 – third meeting in Rome to discuss ambiguities and other problems

ALGOL 60 Evaluation

- Successes:
    - Standard way to publish algorithms for over 20 years
    - All newer imperative languages based on it
    - First machine-independent language
    - First language with formal syntax definition – Backus-Naur Form – BNF
- Failures:
    - Not widely used – especially US
    - Too flexible – understanding difficult – implementation inefficient
    - Lack of I/O statements – portability issues
    - Entrenchment of Fortran and lack of support from IBM

```
comment   ALGOL 60 Example Program
  Input:    An Integer less than 100 followed by a list of that many Integer values
  Output:   The number of list values greater than the average of all list values;
begin
        integer array intlist[ 1:99 ];
        integer listlen, counter, sum, average, result;
        result := 0;
        sum := 0;
        readint( listlen );
        if ( listlen > 0 ) ^ ( listlen < 100 ) then
                begin
comment   Read input data into an array and computer its sum;
                        for counter := 1 step 1 until listlen do
                                begin
                                        readint( intlist[ counter ] );
                                        sum := sum + intlist[ counter ]
                                end;
comment   Compute the average;
                        average := sum / listlen;
comment   Count the values that are greater than the average;
                        for counter := 1 step 1 until listlen do
                                begin
                                        if ( intlist[ counter ] > average ) then
                                                result := result + 1;
                                end;
comment   Print the result;
                        printstring( “Number of values > Average is: ” );
                        printint( result )
                end;
        else
                printstring( “Error – list length is not legal” );
end
```

Computerizing Business Records: COBOL

- COBOL has had little influence on the design of subsequent programming languages other than PL/I.
    - Very little effort has gone into developing new languages for business.
- May still be the most widely used language.

Historical Background
- UNIVAC – using FLOW-MATIC
- US Air Force – using AIMACO
- IBM – using COMTRAN – COMmercial TRANslator

FLOW-MATIC
- 1953 – Grace Hopper – Remington-Rand UNIVAC
    - Mathematical programs should be written in mathematical notation
    - Data processing programs should be written in English statements

COBOL Design Process
- 1959 – meeting sponsored by the Department of Defense – Pentagon
- Originally named CBL – Common Business Language
- Use English as much as possible – allow managers to read programs
- Easy to use – even if less powerful
- Design should not be overly restricted by implementation problems
- Data and code reside in different parts of programs
- 1960 – initial language specification COBOL 60 – revised 1961 and 1962
- ANSI standardized 1968 – again in 1974, 1985, and 2002

Evaluation
- DEFINE verb – first high-level language macro
- Hierarchical data structures
- Identifiers – up to 30 characters with hyphens
- Separate data division – detailed variable and file records definitions
- Detailed printer output – ideal for accounting reports
- pre 1974 versions did not support functions
- Mandated use by the Department of Defense – language may not have survived without the mandate

```
IDENTIFCATION DIVISION.
PROGRAM-ID. PROCEUDRE_REORDER-LISTING.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DEC-VAX.
OBJECT-COMPUTER. DEC-VAX.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT  BAL-FWS-FILE  ASSIGN TO READER.
        SELECT  REORDER-LISTING  ASSIGN TO LOCAL-PRINTER.

DATA DIVISION.
FILE SECTION.
FD      BAL-FWD-FILE
        LABEL RECORDS ARE STANDARD
        RECORD CONTAINS 80 CHARACTERS.

01      BAL-FWD-CARD.
        02  BAL-ITEM-NO             PICTURE  IS  9(5).
        02  BAL-ITEM-DESC           PICTURE  IS  X(20).
        02  FILLER                  PICTURE  IS  X(5).
        02  BAL-UNIT-PRICE          PICTURE  IS  999V99.
        02  BAL-REORDER-POINT       PICTURE  IS  9(5).
        02  BAL-ON-HAND             PICTURE  IS  9(5).
        02  BAL-ON-ORDER            PICTURE  IS  9(5).
        02  FILLER                  PICTURE  IS  X(30).

FD      REORDER-LISTING
        LABEL RECORDS ARE STANDARD
        RECORD CONTAINS 132 CHARACTERS.
01      REORDER-LINE.
        02  RL-ITEM-NO              PICTURE  IS  Z(5).
        02  FILLER                  PICTURE  IS  X(5).
        02  RL-ITEM-DESC            PICTURE  IS  X(20).
        02  FILLER                  PICTURE  IS  X(5).
        02  RL-UNIT-PRICE           PICTURE  IS  ZZZ.99.
        02  FILLER                  PICTURE  IS  X(5).
        02  RL-AVAILABLE-STOCK      PICTURE  IS  Z(5).
        02  FILLER                  PICTURE  IS  X(5).
        02  RL-REORDER-POINT        PICTURE  IS  X(71).
```

```
WORKING-STORAGE SECTION.
01      SWITCHES.
        02  CARD-EOF-SWITCH            PICTURE  IS  X.
01      WORK-FIELDS.
        02  AVAILABLE-STOCK            PICTURE  ISS 9( 5 ).

PROCEDURE DIVISION.
000-PRODUCE-REORDER-LISTING.
        OPEN INPUT BAL-FWD-FILE.
        OPEN OUTPUT REORDER-LISTING.
        MOVE “N” TO CARD-EOF-SWITCH.
        PERFORM 100-PRODUCE-REORDER-LINE
                UNTIL CARD-EOF-SWITCH IN EQUAL TO “Y”.
        CLOSE BAL-FWD-FILE.
        CLOSE REORDER-LISTING.
        STOP RUN.

100-PRODUCE-REORDER-LINE.
        PERFORM 110-READ-INVENTORY-RECORD.
        IF CARD-EOF-SWITCH IS NOT EQUAL TO “Y”
                PERFORM 120-CALCULATE-AVAILABLE-STOCK
                IF AVIALABLE-STOCK IS LESS THAN BAL-REORDER-POINT
                        PERFORM 130-PRINT-REORDER-LINE.

110-READ-INVENTORY-RECORD.
        READ BAL-FWD-FILE RECORD
                AT END
                        MOVE “Y” TO CARD-EOF-SWITCH.

120-CALCULATE-AVAILABLE-STOCK.
        ADD BAL-ON-HAND BAL-ON-ORDER
                GIVING AVAILABLE-STOCK.

130-PRINT-REORDER-LINE.
        MOVE  SPACE                 TO  REORDER-LINE.
        MOVE  BAL-ITEM-NO           TO  RL-ITEM-NO.
        MOVE  BAL-ITEM-DESC         TO  RL-ITEM-DESC.
        MOVE  BAL-UNIT-PRICE        TO  RL-UNIT-PRICE.
        MOVE  AVAILABLE-STOCK       TO  RL-REORDER-STOCK.
        MOVE  BAL-REORDER-POINT     TO  RL-REORDER-POINT.
        WRITE  REORDER-LINE.
```

The Beginning of Timesharing: BASIC

- Beginner's All-purpose Symbolic Instruction Code
- Very popular on microcomputers – late 1970s and early 1980s
    - Easy for beginners to learn
    - Can be implemented on computers with very small memory
- As microcomputers grew the interest in BASIC waned
- Strong resurgence with the appearance of Visual Basic – Microsoft 1991

Design Process

- John Kemeny and Thomas Kurtz – Dartmouth College
- Designed for liberal arts students
- Use terminals for computer access
- Goals:
    - Easy for nonscience students to learn and use
    - Pleasant and friendly
    - Provide fast turnaround for homework
    - Allow free and private access
    - User time more important than computer time

Language Overview

- Original version – small only 14 different statements – not interactive
- Programs – typed in, compiled, and run in a sort of batch mode
- One data type: floating-point – referred to as number

Evaluation

- First widely used language used through terminals
    - Punch cards or punch tape were used prior to this
- Early versions not intended for serious programmers
- Visual Basic .NET is object-oriented and Visual Basic is not.

```
REM  BASIC Example program
REM  Input:    An Integer less than 100
REM              followed by a list of that many Integer values
REM  Output:  The number of list values greater than the average of all list values
        DIM intlist( 99 )
        result = 0
        sum = 0
        INPUT listlen
        IF listlen > 0 AND listlen < 100 THEN
REM  Read input data into an array and computer its sum
                FOR counter = 1 TO listlen
                        INPUT intlist( counter )
                        sum = sum + intlist( counter )
                NEXT counter
REM  Compute the average
                average = sum / listlen
REM  Count the values that are greater than the average
                FOR counter = 1 TO listlen
                        IF intlist( counter ) > average THEN
                                result = result + 1
                NEXT counter
REM  Print the result
                PRINT "Number of values > Average is: "; result
        ELSE
                PRINT "Error – list length is not legal"
        END IF
END
```