Programming Languages

1. History ...
   Machine Language ->
   Assembly Language ->
   High Level Languages ->
   4th Generation Languages (4GLs)

   Machine Language                  01130
     (DRC)                           01231
                                      10312
                                      05331

   Assembly Language                LDM R1, M30
     (DRC)                          LDM R2, M31
                                      ADD R3, R1, R2
                                      STM R3, M32

   Algol                             A <- B + C
   Pascal                            A := B + C
   BASIC                            Let A = B + C
   FORTRAN                          A = B + C
   C                                 C = B + C;

   Example: A <- B + C
            (Not the same as the algebraic statement   A = B + C    ... rather A gets the sum of B and C   A = A + 1 ... makes no sense)

2. Purpose of developing languages
   • more efficient programming (ML -> Assembly)
   • machine independence (Assembly -> HLL)
   • productivity   (Assembly -> HLL, HLL -> 4GLs)
   • reduction of errors (HLL -> 4glS)
3. **4GLs**
   - SQL ... for database work
   - Stella ... for modelling
   - Powerbuilder ... for database GUIs

4. **Paradigms**
   - Procedural (FORTRAN, COBOL, ALGOL, BASIC, APL, PL1, C, Pascal, Ada)
   - Declarative (Prolog, Stella)
   - Functional (Lisp, Scheme, LOGO)
   - Object Oriented (Simula, Smalltalk, C++, Java)
   - Parallel (Machine Dependent FORTRAN, C*, Sequent-C, Occam, HPF, Parallaxis, Modula-P)

5. **Translation**
   - Computer understands only machine language (in binary ... 0/5 volts)
   - Translation Process
     - Source Code C = A + B
     - Lexical Analysis: Three variables A, B, C; two operators =, +
     - Parsing: Analysing grammatical structure Assignment, Addition
     - Code Generation: Machine language statements (may be assembly language)
     - Object Code

6. **Linking and Loading**
   - Linking ... combine Object Code with any Libraries or other Object Code called
   - Loading ... determination of absolute addresses ... placed in memory ... jump to start, jump back to OS
7. Translation (compiler) versus Interpretation (interpreter)
   • Compiler creates object code all at once
   • Interpreter creates object code one line at a time ... and executes it.
   • Java ... the Java Virtual Machine
     • The virtual machine ... an abstract machine
     • Java compiler creates "machine instructions" for the "Java virtual machine" ... these instructions are called byte code
     • Java interpreter translates byte code to the machine code of the target machine. Therefore Java byte code "runs on any platform" (that has a Java interpreter).

8. Semantics versus Syntax
   • semantics ... clear and unambiguous meaning ... not like ...
     • Mary is blue.
     • Frank is gay.
     • Should I turn left or right? Yes.
   • syntax ... grammatically correct
     • He go to the store.
     • Throw the lamb over the fence some hay.
     • The boy jumped off the bridge with the red hair.

9. Representation of syntax
   • BNF (Backus Naur Form) ... see below ... BNF for Karel
   • syntax (bubble) diagrams
BNF for Karel

BNF is the Backus-Naur Form of describing a programming language

 ::= this symbol is called a production. Each definition in BNF is called a production.

<> words within the angle braces are names of objects

| a symbol which represents alternative choices

[ ] anything within square braces is repeated 0 or more times

program ::= BEGINNING-OF-PROGRAM
          [<new-instruction>]
          BEGINNING-OF-EXECUTION
          <instruction> [<instruction>]
          END-OF-EXECUTION
          END-OF-PROGRAM

<new-instruction> ::= DEFINE-NEW-INSTRUCTION <name> AS
                      BEGIN <instruction> END ;

<instruction> ::= <primitive> | <conditional> | <repeat>

<primitive> ::= move; | turnleft; | pickbeeper; | putbeeper; | turnoff;

<conditional> ::= IF <test> THEN BEGIN <instruction> END
                ELSE BEGIN <instruction> END;
                IF <test> THEN BEGIN <instruction> END;

<repeat> ::= WHILE <test> DO BEGIN <instruction> END;
            ITERATE <integer> TIMES BEGIN <instruction> END;

@test ::= <side>-is-clear | <side>-is-blocked | next-to-a-beeper
          not-next to a beeper | facing-<direction>
          not-facing-<direction> | any-beepers-in-beeper-bag
          no-beepers-in-beeper-bag
10. Procedural Languages
   • Data Structures
     • simple types ... integer, real
     • compound types ... array of integers
   • Assignment Statements
   • Control
     • Sequence
     • Branch
     • Loop
   • I/O ... reads and writes
     • keyboard, screen
     • disk files
     • i/o streams
   • Subprograms (procedures, functions, subroutines)
     • Call versus Declaration
     • Passing of Parameters

11. Object Oriented Languages (OOP)
   • Classes ... a general category of a "thing" that has a data and methods
   • Objects ... specific instances of a class
     • Class fruit ... Object apple
• Objects send messages to other objects to perform certain operations
• Usually one object can change another object only by sending it a message to change itself. This protects an object's data.
• OOP is thought to be
  • easier to design algorithms
  • have reusable code
  • be more modularizable
• OOP is the basis of a type of database
12. Programming in Logic ... Prolog
   • a declarative language
   • the programmer must state the problem ... not how to solve it
   • the programmer must state the problem **VERY CLEARLY**
   • Prolog has
     • Domains ... define the terms/symbols/words used in the program
     • Predicates ... relationships between things
     • Clauses ... facts and rules
     • Goals ... question to be asked

   • Example 1

   domains
   person = symbol

   predicates
   male (person)
   female (person)
   sister(person,person)
   parents (person, person, person)

   clauses
   male (chris) .
   male (paul) .
   female (colette) .
   female (becky) .
   parents (becky, paul, colette) .
   parents (chris, paul, colette) .
   sister(X,Y) if female (X) and
    parents(X,F,M) and
    parents (Y, F, M) .

Example 2 ... next page
domains
  person = symbol
predicates
  male(person)
  female(person)
  father(person,person)
  mother(person,person)
  parent(person,person)
  sister(person,person)
  brother(person,person)
  uncle(person,person)
  grandfather(person,person)

clauses
  male(alan).
  male(charles).
  male(bob).
  male(ivan).
  female(beverly).
  female(fay).
  female(marilyn).
  female(sally).
  mother(marilyn,beverly).
  mother(alan,sally).
  father(alan,bob)
  father(beverly,charles).
  father(fay,bob).
  father(marilyn,alan).
  parent(X,Y) if mother(X,Y).
  parent(X,Y) if father(X,Y).

  brother(X,Y) if /* The brother of X is Y if */
               male(Y) and /* Y is a male and */
               parent(X,P) and /* the parent of X is P and */
               parent(Y,P) and /* the parent of Y is P and */
               X <> Y.    /* X and Y are not the same */

  sister(X,Y) if /* The sister of X is Y if */
               female(Y) and /* Y is female and */
               parent(X,P) and /* the parent of X is P and */
               parent(Y,P) and /* the parent of Y is P and */
               X <> Y.    /* X and Y are not the same */

  uncle(X,U) if /* The uncle of X is U if */
               mother(X,P) and /* the mother of X is P and */
               brother(P,U). /* the brother of U is u. */

  uncle(X,U) if /* The uncle of X is U if */
               father(X,P) and /* the father of X is P and */
               brother(P,U). /* the brother of P is U */

  grandfather(X,G) if /* The grandfather of X is G */
                    father(P,G) and /* if the father of P is G */
                    mother(X,P).  /* and the mother of X is P. */

  grandfather(X,G) if /* The grandfather of X is G */
                    father(X,P) and /* if the father of P is G */
                    father(P,G). /* the father of P is G */
1. Simulation Programming with **Stella**
   - Icons represent processes
   - The “system is described”
   - Stella generates a solution
     - Applies the appropriate numerical method
     - Calculates the solution
     - Makes graphs automatically
     - Generates tables of data

\[
\frac{dP}{dt} = aP - bP
\]
Prolog version of Towers of Hanoi ... an experiment

/* Towers of Hanoi */

domains
/* none needed */

predicates
  hanoi(integer, symbol, symbol, symbol)
  move_a_ring(symbol, symbol)

clauses
  hanoi (1,A,_,-C) if move_a_ring(A,C) and !.
  hanoi(N,A,B,C) if N1 = N-1 and hanoi(N1,A,C,B)
  and move_a_ring(A,C) and hanoi(N1,B,A,C).
  move_a_ring(Post1,Post2) if
    write("\nMove a disk from ",Post1, " to ", Post2,"\n").

Execute the program ...

There is too much output to see in the goals for n \geq 4, therefore it will be written to a file. Prolog has this capability.
/* Towers of Hanoi written to a file */

domains
    file = mdrive

predicates
    hanoi(integer, symbol, symbol, symbol)
    hanoi1(integer, symbol, symbol, symbol)
    move_a_ring(symbol, symbol)

clauses
    hanoi(N,A,B,C) if disk("M:\temp1")
        and openwrite(mdrive,"data.txt")
        and writedevice(mdrive)
        and hanoi1(N,A,B,C)
        and write("\n")
        and flush(mdrive)
        and closefile(mdrive)
        and writedevice(screen).

    hanoi1(1,A,_,C) if move_a_ring(A,C) and !.

    hanoi1(N,A,B,C) if N1 = N-1 and hanoi1(N1,A,C,B)
        and move_a_ring(A,C) and hanoi1(N1,B,A,C).

    move_a_ring(Post1,Post2) if
        write("\nMove a disk from ", Post1, " to ", Post2).

Collect data on the number of moves for 1 <= N <= 64

How long would it take to do Hanoi(64) if you could do one move per second?