Introduction

Comp 333: Concepts of Programming Languages
Fall 2013

Instructor: Professor Schwartz

Concepts of Programming Languages

- History
- Syntax and Semantics
  - Compilers
- Language Constructs
  - Names, Binding, Scoping, Data Types
  - Expressions, Control Structures, Subprograms
- Programming Language Types
  - Imperative
  - Functional
  - Logic
  - Concurrent
  - Object Oriented

Class Discussion

- How many students know more than one programming language?
- What are the advantages of knowing more than one programming language?

Why should we study concepts of programming langauges?

- To improve our ability to
  - Write programs
  - Read programs
  - Debug programs
  - Learn new languages faster
  - Choose most appropriate language for a project
  - Design and Implement a programming language

Programming Language Spectrun

- Imperative Languages
  - C, C++, Fortran, Java
- Functional Languages
  - Lisp, Scheme, ML
- Logic Programming Languages
  - Prolog
- Object-Oriented Languages
  - Java, C++, Smalltalk
- Scripting Languages
  - Javascript, Perl, Python
- Tiobe Programming Community Index

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Chapter 1 5

Factorial Function in Java

```
int factorial ( int n)
{
      int result = 1;
      for( int k = 2; k <= n; k++)
            result = result * k;
      return result;
}
```

(Similar syntax for C and C++ and other imperative languages)

Chapter 1 6

Factorial Function in Scheme

```
(define factorial
  ( lambda (n)
      (if ( <=  n 0)
            1
            ( *  n   (factorial ( - n 1)))
      )
  )
)
```


Factorial Function in Prolog

```
factorial ( X, 1) :-  X=0.

factorial ( X, Result)  :-
      X > 0,
      A is X-1,
      factorial ( A, Z),
      Result is X * Z.
```

(The factorial function does not show off the best features of Prolog!)

Checking List Equality in Prolog

Lists look like [2,5,7,9] or [a,7,9]

Definition:

```
equalList([], []).
equalList( [A|B], [A|C] ) :-  equalList(B,C).
```

Run:

```
> equalList( [1,2,3], [1,X, 3]).
> X = 2
```


What makes a language successful?

- Facilitates writing clear, concise, reliable and maintainable code
- Easy to learn
- Easy to implement (compilers, interpreters)
- Standardization (for portability)
- Good supporting tools (compilers, libraries)
- Economic Issues
  - Free, easy to install compilers and support tools
  - Legacy code makes it expensive to move to a new language (e.g. Cobol)

Why are there so many languages?

- Different program domains
  - Scientific applications (Fortran , C)
  - Business applications (Cobol)
  - Artificial Intelligence (Lisp, Prolog)
  - Systems programming (C)
  - Web programming (Javascript, Perl, PHP)
  - Embedded Systems DOD (ADA)
  - Education (Pascal)
- Complexity of modern software
  - Need for Increased Program Modularity
  - Need for Increased Reliability and Maintainability

First Programming Languages: Assembly Languages

- Symbolic locations and opcodes
- Computation of N = I + J (Pentium 4)

```
FORMULA:   MOV   EAX,I
           ADD   EAX, J
           MOV   N, EAX
I          DD    3     ;reserve 4 bytes
J          DD    4     ;reserve 4 bytes
N          DD    0     ;reserve 4 bytes
```

Programming Language Features Added Over Time

- Variables: x,y,z
- Arithmetic Expressions: z = x + y
- Data types: int, double, string
- Block structure: local scope rules
- Functions and procedures
- Data structures: arrays, records, pointers
- Recursion
- Runtime Exception Handling
- Support for concurrency: threads
- Object -Oriented Language Features
  - classes, objects, inheritance, polymorphism

FORTRAN

- Fortran Mid 1950s
  - Developed by John Backus and his group at IBM
  - Used to perform math computations (formulas)
  - One of the first “high level” languages
  - Continued development Fortran IV, Fortran77,..
- Features
  - Variables, expressions, statements
  - Arrays
  - Iteration and conditional branching
  - Subroutines (independently compiled)
  - FORMAT for input and output

Fortran IV Fragment of a Program (See handout)

```
        DIMENSION X(52), Y(2,50), LITERL(2)
        DOUBLE PRECISION S1,S2,S3,S4,S5,T,S,B,D,R,E1,E2,BBAR
        WRITE (5,10)
 10     FORMAT('0',1X,'*   *   * LINEAR REGRESSION ANALYSIS  *   *   *',//)
        WRITE (5,20)
 20      FORMAT(1X,'HOW MANY PAIRS TO BE ANALYZED?'$)
        READ (5,*) N
         IF (N.GT.50) GOTO 70
        WRITE (5,30)
 30      FORMAT(//1X,'Enter one pair at a time')
        WRITE (5,40)
 40     FORMAT(1X,'and separate X from Y with a comma.'//)
        WRITE (5,50)
 50     FORMAT(1X,'Enter pair number one : '$)
        READ (5,*) X(1), Y(1,1)
                  DO 60 I=2,N
                  WRITE (5,55) I
 55               FORMAT(1X,'Enter pair number',I3,' : '$)
                 READ (5,*) X(I), Y(1,I)
 60              CONTINUE
        GOTO 90
70      WRITE (5,80)
 80     FORMAT(1X,'At present this program can only handle 50 data pairs.')
        STOP
```


FORTRAN IV Example -- Another Fragment

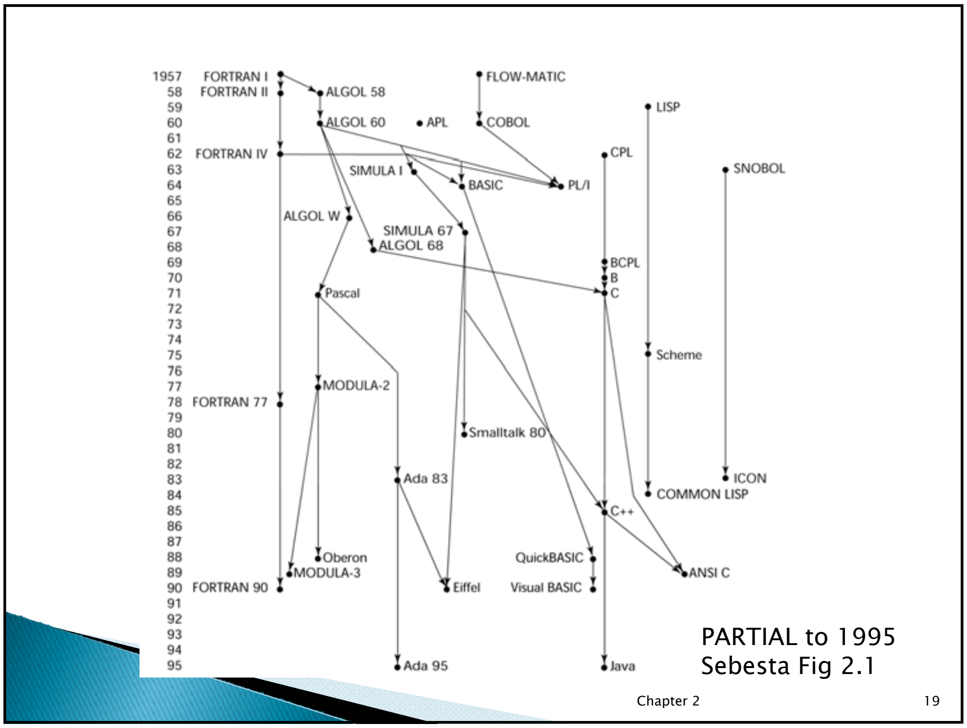
```
200       DO 210 I=1,N
         S1=S1+X(I)
         S2=S2+Y(1,I)
         S3=S3+X(I)*Y(1,I)
         S4=S4+X(I)*X(I)
         S5=S5+Y(1,I)*Y(1,I)
 210        CONTINUE
      T=N*S4-S1*S1
      S=(N*S3-S1*S2)/T
       B=(S4*S2-S1*S3)/T
            DO 220 I=1,N
            Y(2,I)=S*X(I)+B
             D=D+(Y(2,I)-Y(1,I))**2
 220        CONTINUE
      D=D/(N-2)
       E1=DSQRT(D*N/T)
       E2=DSQRT(D/N*(1+S1*S1/T))
       R=(N*S3-S1*S2)/
   2         (DSQRT(ABS(((N*S4-ABS(S1)**2))*(N*S5-ABS(S2)**2))))
       WRITE (5,230)
 230   FORMAT(////,10X,'X-VALUE',20X,'Y-OBS',22X,'Y-CALC')
      WRITE (5,235)
```

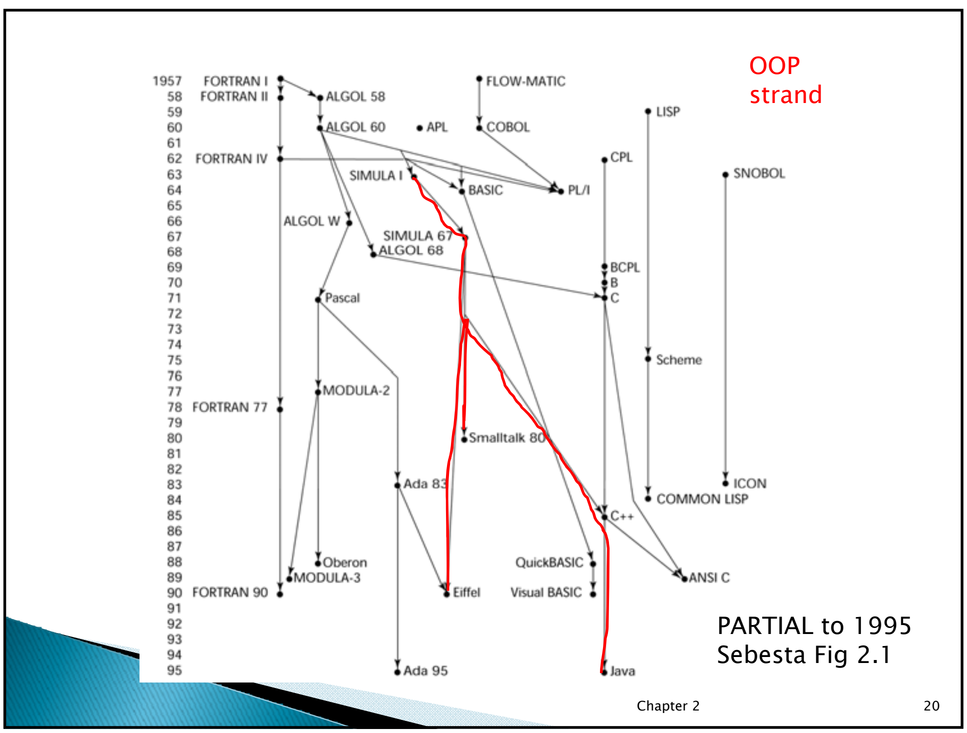
LISP

- Lisp (1959-1960)
  - Developed by John McCarthy at IBM
  - Symbolic processing (eg differentiation)
  - Ancestor of Scheme
- Features
  - Symbolic processing language (eg list processing)
  - Built on lists, atoms, selectors and constructors
  - Dynamically allocated linked lists
  - Garbage Collection
  - Recursion
  - Functions are first class objects

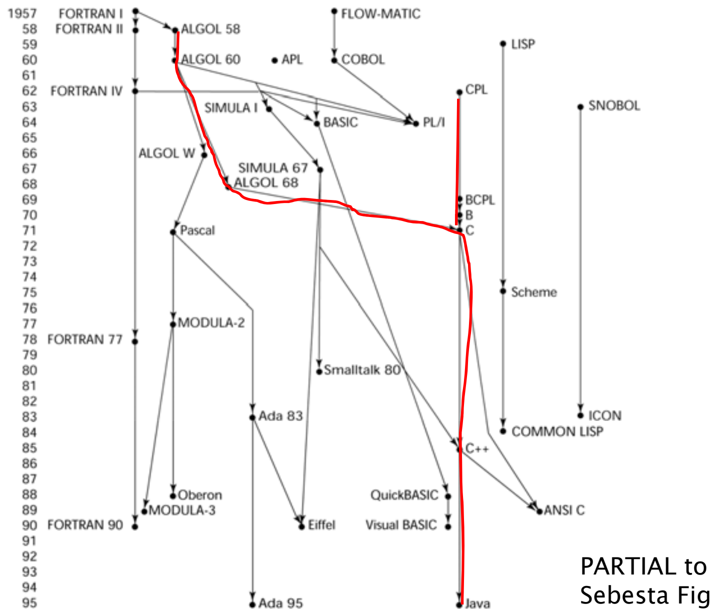
C Programming Language

- C (1972)
  - Designed by Dennis Ritchie at Bell Labs
  - Ancestor of Java, C++
- Features
  - Language for systems programming
  - C compiler was part of the UNIX operating system
  - Used in many application areas
  - Official (ANSI) description of C (1989)


1957 FORTRAN I
58 FORTRAN II
ALGOL 58
FLOW-MATIC
LISP
ALGOL 60
APL
COBOL
62 FORTRAN IV
CPL
SNOBOL
SIMULA I
BASIC
PL/I
ALGOL W
SIMULA 67
ALGOL 68
BCPL
B
C
Pascal
Scheme
MODULA-2
78 FORTRAN 77
Smalltalk 80
Ada 83
ICON
COMMON LISP
C++
Oberon
QuickBASIC
MODULA-3
ANSI C
90 FORTRAN 90
Eiffel
Visual BASIC
Ada 95
Java
PARTIAL to 1995
Sebesta Fig 2.1
Chapter 2
19

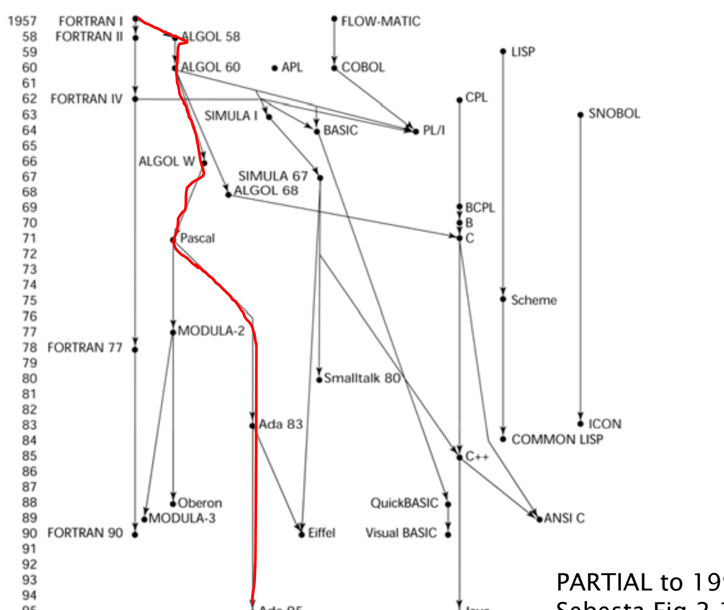

OOP
strand
1957 FORTRAN I
58 FORTRAN II
ALGOL 58
FLOW-MATIC
LISP
ALGOL 60
APL
COBOL
62 FORTRAN IV
CPL
SNOBOL
SIMULA I
BASIC
PL/I
ALGOL W
SIMULA 67
ALGOL 68
BCPL
B
C
Pascal
Scheme
MODULA-2
78 FORTRAN 77
Smalltalk 80
Ada 83
ICON
COMMON LISP
C++
Oberon
QuickBASIC
MODULA-3
ANSI C
90 FORTRAN 90
Eiffel
Visual BASIC
Ada 95
Java
PARTIAL to 1995
Sebesta Fig 2.1
Chapter 2
20

Ancestors of Java



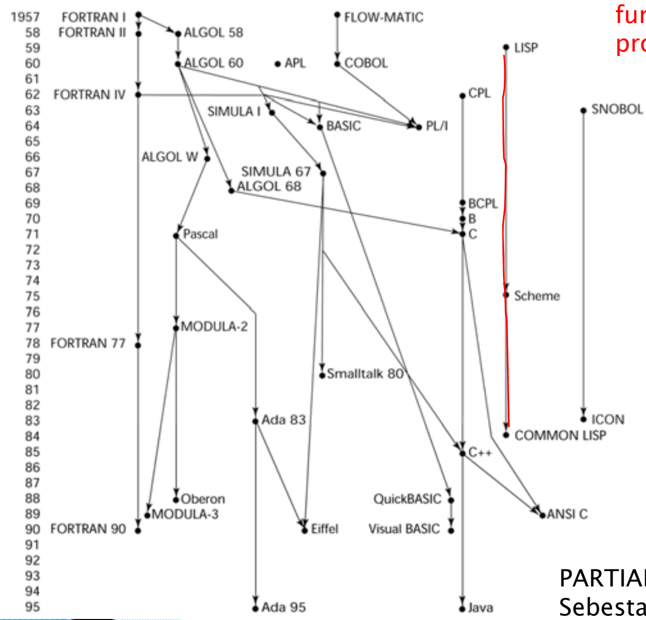
PARTIAL to 1995
Sebesta Fig 2.1

Fortran- Pascal - ADA



PARTIAL to 1995
Sebesta Fig 2.1

Ancestors of functional programming



PARTIAL to 1995
Sebesta Fig 2.1

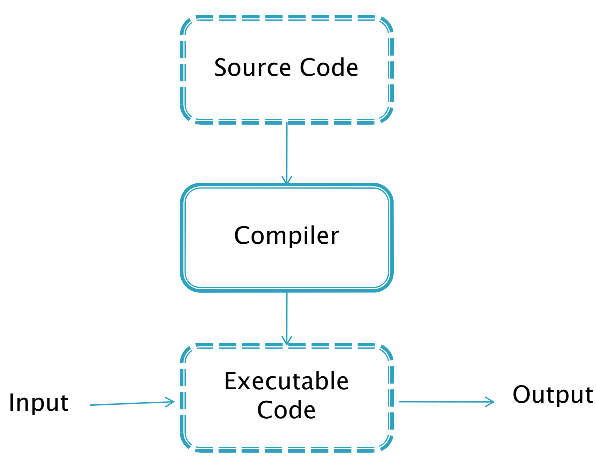

```
program pascalEx( input,output)
        type  intlisttype = array [ 1..99] of integer;
        var
                intlist: intlisttype;
                listlen, k, sum, average, result  : integer;
        begin
                result := 0;
                sum := 0;
                readln( listlen);
                if ( listlen > 0) and ( listlen < 100)  then
                   begin
                        for  k:= 1 to listlen do
                        begin
                                readln( intlist[k]);
                                sum := sum + intlist[k]
                        end;

                        average := sum /listlen;
                        for  k := 1 to listlen do
                                if ( intlist[k]   > average) then
                                        result := result + 1;
                        {Print result}
                        writeln('The number of values > average is ", result)
                   end
                else
                        writeln('Error - input list length is not legal')
        end.
```

How does this Pascal program differ from a similar Java program?

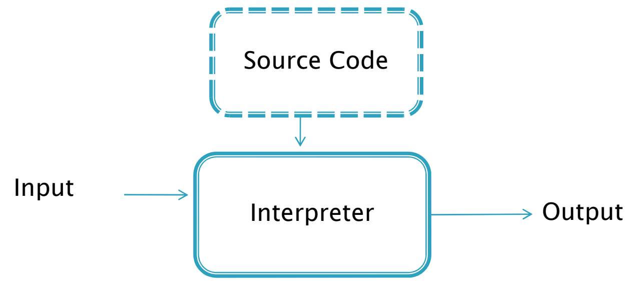
Executing programs written in a high level language

- History of Program Language Development
  - Machine code → High level source code
- Translation Needed to Run High Level Code
  - high level source code →machine code
- Compilers
- Linkers and Loaders
- Interpreters

Compilation



Interpreter



Interpreter reads and executes program statements one by one

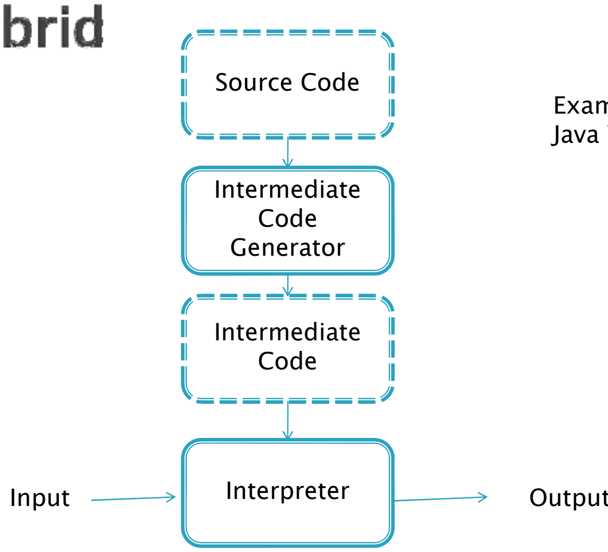
Benefits of Interpreters over Compliers

- Greater flexibility to support
  - Late binding for variables and names
  - Writing new pieces of code on the fly
- Better diagnostics
  - Source code debugging

Benefits of Compilers over Interpreters

- Better Performance (speed)
- Code Optimization
  - “Hardwire” variable location
  - Code rearrangement
  - “Inline” small subroutines

Hybrid



Example:
Java Virtual Machine

Phases of Compilation

- Scanner (Lexical analysis)
- Parser (Syntax analysis)
- Intermediate Code Generation (Semantic Analysis)
- Code Optimizer (optional)
- Target Code Generation

Front End

Back End

Details of the compilation process