Software Development
Programming & Languages

Programming: A Five-Step Procedure

- Define the problem
- Design a solution
- Code the program
- Test the program
- Document the program
Five Generations of Programming Languages

- **Machine Languages**, 1945--
- **Assembly Languages**, 1950s--
- **High-Level Languages**, 1960s--, FORTRAN, COBOL, BASIC, Pascal, C, C++
- **Very-High-Level Languages (4GL)**, 1970s-- PRG III, SQL, Intellect, NOMAD, FOCUS
- **Natural Languages**, 1980s--

Applications of some important programming languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Origin</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN</td>
<td>FORmula TRANslator(1954)</td>
<td>Scientific</td>
</tr>
<tr>
<td>COBOL</td>
<td>Common Business-Oriented Language(1959)</td>
<td>Business</td>
</tr>
<tr>
<td>BASIC</td>
<td>Beginner’s All-purpose Symbolic Instruction Code (1965)</td>
<td>Education, Business</td>
</tr>
<tr>
<td>Pascal</td>
<td>Named after French inventor Blaise Pascal(1971)</td>
<td>Education, systems programming</td>
</tr>
<tr>
<td>C</td>
<td>Invested at Bell Labs(1972)</td>
<td>Systems programming, general use</td>
</tr>
</tbody>
</table>
Hello!
How may I help you?
Who are my customers in Chicago?
   Just a sec. I’ll see.
   The customers in that city are:
<table>
<thead>
<tr>
<th>I.D.</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballard</td>
<td>Ballard and Sons, Inc.</td>
</tr>
<tr>
<td>Fremont</td>
<td>Henry Fremont Associates</td>
</tr>
<tr>
<td>Greenlake</td>
<td>Greenlake Consortium</td>
</tr>
<tr>
<td>Wallingford</td>
<td>Wallingford, Inc.</td>
</tr>
</tbody>
</table>
   What can I do for you now?
What is Fremont’s balance?
   Hang on. I’ll see.
   | Accounts Receivable | 563.47 |
   | Unapplied Credit    | 79.16  |
   | Balance              | 484.31 |
   What else can I do for you?
Give me Fremont’s phone number!
   Please wait while I check the files?
   (312)789-5562
   What can I do for you now?

Language Translators

- An assembler is a program that translates the assembly-language program into machine language.
- A compiler is a language translator that converts the entire program of a high-level language into machine language before computer executes the program.
- An interpreter is a language translator that converts each high-level language statement into machine language and executes it immediately, statement by statement.
Object-Oriented Programming

Object-oriented programming (OOP) is a programming method that combines data and instructions for processing that data (called methods) into a self-sufficient “object” that can be used in other programs. It involves three important concepts:

- **Encapsulation**: Encapsulation means an object contains (encapsulates) both data and the instructions for processing it.

- **Inheritance**: Inheritance allows traits of a class of objects to be inherited by its subclass.

- **Polymorphism**: Polymorphism allows a message to produce different results based on the object that it sent to. (A message is a call of some method).
Visual Programming

- Visual programming is a method of creating programs in which the programmer makes connections between objects by drawing, pointing, and clicking on diagrams and icons.

Internet Programming

- **HTML (HyperText Markup Language)** is a type of programming language that embeds simple commands within standard ASCII text documents to provide an integrated, two-dimensional display of text, graphics, and sounds.

- **VRML (Virtual Reality Markup Language)** is a type of programming language used to create three-dimensional Web-pages.

- **Java** (developed by Sun Microsystems) is a programming language that allows users to create applications (called Applets) that can be downloaded and executed on the remote host via the internet connection.
Flowchart and pseudocode for averaging numbers

1. You can design a program in many ways, but usually the first step is to figure out the requirements of the program—determine exactly what you want the program to do. In this example, the requirement is to add a sequence of numbers from 1 to a number specified by the user.

2. One of the traditional ways to design a program is to use a flow chart. The top of the flow chart indicates input from the user. Rectangles contain statements, and diamonds indicate decisions.

3. Pseudocode refers to expressing program instructions in English as a way of describing the operation of the program without going into detail and worrying about following the syntax of a particular computer language.

4. There are many techniques for designing complex programs. In fact, there are programs that can help you design software and keep track of the flow of data through the components of your application. These software-engineering tools are most frequently used for large projects that involve many programmers.
### Machine Language

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Machine Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>CSEG SEGMENT PARA PUBLIC 'CODE'</td>
</tr>
<tr>
<td>0000</td>
<td>ASSUME CS:CSEG, DS:CSEG</td>
</tr>
<tr>
<td>0002</td>
<td>ENTPT PROC FAR ;entry point</td>
</tr>
<tr>
<td></td>
<td>mov ax, ce</td>
</tr>
<tr>
<td></td>
<td>mov ds, ax           ;set up addresses</td>
</tr>
<tr>
<td>0004</td>
<td>mov si, 0            ;clear SI (counts+points to data)</td>
</tr>
<tr>
<td>0007</td>
<td>mov al, 0            ;clear AX (stores total)</td>
</tr>
<tr>
<td>0009</td>
<td>02 84 001C R</td>
</tr>
<tr>
<td></td>
<td>count: add al, [DATA+si] ; add memory value to AX</td>
</tr>
<tr>
<td></td>
<td>; location of memory is</td>
</tr>
<tr>
<td></td>
<td>; DATA plus value in SI</td>
</tr>
<tr>
<td>000D</td>
<td>46 inc si            ; add one to SI</td>
</tr>
<tr>
<td>000E</td>
<td>B3 FE 05</td>
</tr>
<tr>
<td>0011</td>
<td>75 FE 00 05</td>
</tr>
<tr>
<td>0013</td>
<td>A2 0021 R</td>
</tr>
<tr>
<td>0016</td>
<td>84 4C mov AH,4ch</td>
</tr>
<tr>
<td>0018</td>
<td>80 00 mov AL, 00</td>
</tr>
<tr>
<td>001A</td>
<td>CD 21 int 21H</td>
</tr>
<tr>
<td>001C</td>
<td>0C 0E 0B 12</td>
</tr>
<tr>
<td>0021</td>
<td>00 DATA db 12, 14, 09, 11, 18 ; bytes of data</td>
</tr>
<tr>
<td></td>
<td>RESULT db 00         ; place to store result</td>
</tr>
<tr>
<td>0022</td>
<td>ENTP ENDP CSEG ENDS</td>
</tr>
</tbody>
</table>

### Assembly Language

#### A BASIC Program

5 PRINT 10 PRINT " Student Grade Report : Jennifer Jonson" 11 PRINT 15 PRINT "Class " ; " Grade Number " 20 FOR A=1 TO 5 do ; this 5 times 30 READ CLASS$, GRADES 40 IF GRADES= "A" THEN NUM=4 50 IF GRADES= "B" THEN NUM=3 60 IF GRADES= "C" THEN NUM=2 70 IF GRADES= "D" THEN NUM=1 80 IF GRADES= "E" THEN NUM=0 90 PRINT CLASS$, " GRADES ; 100 PRINT USING " * NUM" 110 TOTAL=TOTAL+NUM 120 NEXT A 130 AVERAGE=TOTAL/5 140 PRINT " ; " 150 PRINT " " AVERAGE " ; " 160 PRINT USING " * AVERAGE" 1000 DATA "Freshman English " ; A 1010 DATA " Roman Empire " ; B 1020 DATA " Elementary French " ; B 1030 DATA " Pre-Calculus " ; C 1040 DATA " Volleyball 1 " ; A

output

<table>
<thead>
<tr>
<th>Student Grade Report</th>
<th>Jennifer Jonson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Grade Number</td>
</tr>
<tr>
<td>Freshman English</td>
<td>A 4.00</td>
</tr>
<tr>
<td>Roman Empire</td>
<td>B 3.00</td>
</tr>
<tr>
<td>Elementary French</td>
<td>B 3.00</td>
</tr>
<tr>
<td>Pre-Calculus</td>
<td>C 2.00</td>
</tr>
<tr>
<td>Volleyball 1</td>
<td>A 4.00</td>
</tr>
</tbody>
</table>

Average 3.20
A FORTRAN Program

```fortran
character*1 grade(5)
character*30 course(5)
write(*,*), 'Student Grade Report: Jennifer Jonson'
write(*,*), 'Class Grade Number'
total = 0
do 10, i = 1, 5
if ( grade(i) .EQ. 'A' ) gradenum=4
if ( grade(i) .EQ. 'B' ) gradenum=3
if ( grade(i) .EQ. 'C' ) gradenum=2
if ( grade(i) .EQ. 'D' ) gradenum=1
if ( grade(i) .EQ. 'F' ) gradenum=0
total = total + gradenum
write (* , 99 ) course(i) , grade(i) , gradenum
10 continue
average = total / 5
write (*,* )
write (* , 99)  average
98 format (1x , a30 , 2x , a1 , 4x , f4 . 2 )
99 format ( 'Average' , 30x , f4 . 2)
end
```

A C Program

```c
#include <stdio.h>
main() {
    char  *  course[5]=" Freshman English ", " Roman Empire ", " Elementary French ", " Pre-Calculus ", " Volleyball 1 ";
    int  a ;
    float num , total=0 ;
    printf( "\n" ) ;
    printf(" Student Grade Report : Jennifer Jonson\n ");
    printf("\n") ;
    printf(" Class\t Grade Number\n ");
    for (a=0 ; a<5 ; a++) {
        switch ( grade[a] ) {
            case 'A' : num=4 ; break ;
            case 'B' : num=3 ; break ;
            case 'C' : num=2 ; break ;
            case 'D' : num=1 ; break ;
            case 'E' : num=0 ; break ;
        }
        printf(" %s \t%c\t%g\n" , course[a] , grade[a] , num);
        total=total+num ;
    }
    printf("---------\n") ;
    printf(" Average\t\t\t%g\n" , total) ;
    printf("\n") ;
}
```
**A COBOL Program**

**PROCEDURE DIVISION**

**MAIN- PROGRAM.**
- PERFORM START-PROCESS.
- PERFORM DECISION-FOR-HONOR-ROLL
  - UNTIL END-OF-FILE-FLAG="Y"
- PERFORM WRAPITUP
- STOP RUN

**START-PROCESS.**
- OPEN INPUT RECORDS-IN OUTPUT PRINT-OUT.
- MOVE SPACES TO STUDENT-LINE.
- WRITE PRINT-LINE FROM HDG AFTER ADVANCING 2 LINES.
- WRITE PRINT-LINE FROM HDG-1 AFTER ADVANCING 2 LINES.
- WRITE PRINT-LINE FROM HDG-2 AFTER ADVANCING 2 LINES.
- WRITE PRINT-LINE FROM HDG-3 AFTER ADVANCING 1 LINE.

**155-PROCESS-CALSS**
- MOVE CLASS-DESCR(WS-INDEX) TO WS-CLASS-DESCR
- MOVE CLASS-DESCR(WS-INDEX) TO WS-CLASS-GRADE
- IF WS-CLASS-GRADE= "A"
  - MOVE 4 TO WS-CLASS-GRADE-NUM
  - ADD 4 TO WS-GRADE-SUMM
- ELSE
  - IF WS-CLASS-GRADE= 'B'
    - MOVE 2 TO WS-CLASS-GRADE-NUM
    - ADD 2 TO WS-GRADE-SUMM
  - ELSE
    - IF WS-CLASS-GRADE= 'D'
      - MOVE 1 TO WS-CLASS-GRADE-NUM
      - ADD 1 TO WS-GRADE-SUMM
    - ELSE
      - IF WS-CLASS-GRADE= 'F'
        - MOVE 0 TO WS-CLASS-GRADE-NUM
      - WRITE PRINT-LINE FROM WS-CLASS-GRADE-LINE
        - AFTER ADVANCING 1 LINE.
- 155-EXIT.
- EXIT.

900-END-OF-JOB.
- CLOSE STUDENT-DATA.
- GRADE-REPORT.
- STOP RUN.

900-EXIT.
- EXIT.
IDENTIFICATION DIVISION
PROGRAM-I, D, PGM001
AUTHOR. JOHN DOE
DATE-WRITTEN. MARCH 8, 1995.

DATE-COMPILED
* SYSTEM : STUDENT GRADE REPORT
* RUNNING SCHEDULE : ON DEMAND
* OVERVIEW : THIS PROGRAM WILL READ STUDENT DATA,
* CALCULATE AND PRINT STUDENT'S GRADE AVERAGE.
* INPUT : STUDENT DATA
* OUTPUT : REPORT OF STUDENT'S GRADES AND THE GRADE AVERAGE

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-PC.
OBJECT-COMPUTER. IBM-PC
SPECIAL-NAME. CO1 IS TOP-OF-PAGE.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT STUDENT-DATA
ASSING TO "D: \ INPUT. DAT".
SELECT GRADES-REPORT
ASSING TO "D: \ OUTPUT. REP".

DATA DIVISION.
FILE SECTION.
FD STUDENT-DATA
RECORD CONTAINS 121 CHARACTERS
DATA RECORD IS STUDENT-REC.

01 STUDENT-REC.
03 STUDENT-NAME
10 FIRST-NAME PIC X (8).
10 LAST-NAME PIC X (8).
03 CLASS-DATA OCCURS 5 TIMES.
10 CLASS-DESCR PIC X (20).
10 CLASS-GRADE PIC X.
Program grades (input, output)

var
   A:integer
   num, total, average: real;
   grade: array[1..5] of char;
   class : array[1..5] of string [30];

Begin
   grade [1] : 'A';
   class [1] = 'Freshman English';
   class [2] = 'Roman Empire';
   class [3] = 'Elementary French';
   class [4] = 'Pre-Calculus';
   class [5] = 'Volleyball 1';
   Writeln;
   writeln ('Student Grade Report : Jennifer Jonson');
   writeln;
   writeln ('Class           Grade Number');
   A Pascal Program

For a: = 1 to 5 do
   Begin
      case grade [a] of
         'A': num: =4;
         'B': num: =3;
         'C': num : =2;
         'D': num: =1;
         'F': num : =0;
      end; (case)
      writeln (CLASS[a], ' ', GRADE[a], ' ', NUM: 4: 2);
      total : = total+num;
   end; {for}
   average: = total/5;
   writeln(' Average
   writeln (' Average
   end
A FORTRAN Program

```fortran
C COMPUTER THE SUM AND AVERAGE OF 10 NUMBERS
C
REAL NUM, SUM, AVG
INTEGER TOTNUM, COUNTR
C
SUM=0.0
C   INITIALIZE LOOP CONTROL VARIABLE
COUNTR=0
TOTNUM=10
C
C   LOOP TO READ DATA AND ACCUMULATE SUM
20 IF (COUNTR . GE . TOTNUM )  GO TO 30
READ ,  NUM
SUM=SUM + NUM
C        UPDATE LOOP CONTROL VARIABLE
COUNTR = COUNTR + 1
GO TO 20
C   END OF LOOP - COMPUTE AVERAGE
30 AVE = SUM / TOTNUM
C   PRINT RESULTS
PRINT, SUM
PRINT, AVG
STOP
END
```

An SQL query for retrieving data

```sql
SELECT <columns> FROM <tables> WHERE <condition>

[Identifies columns / data fields to be retrieved]
[Specifies tables / files from which the data is to be retrieved]
[Extracts only such records as match a specified condition]

Example:

```sql
SELECT CUSTOMER NAME , AMOUNT OWED
FROM CUSTOMER , ORDER
WHERE CUSTOMER.CUSTOMER ID NO = ORDER.CUSTOMER ID NO
```

This query creates a new record by linking the customer and order files through their common data field, Customer ID No. The new record will show the name of each customer and the amount that customer owes.