# Programming Languages and Algorithms

## System vs. Application

- The similarity
  - Both software
- The distinction
  - Scope of users

## How Different?

- System software (OS)
  - Serving all users
  - I.e., everyone needs it
  - Known functionalities
- Application software (MS Office)
  - Serving those users who need the specific functions
  - i.e., **not** everyone needs it
  - User-specific functionalities

## Again, Examples

- MS Office
- Internet Explorer
- Real Player
- And more

## How applications are created

- Terminology
- Program composition
- Programming languages
- Programming language classification

## Terminology

- Application
  - Short for application software
  - Short for application program
- Program (noun)
  - Specific, countable
- Software
  - General, not countable
  - Usually meant a **collection** of programs

## 3P's

- Program
  - A set of rules for your computers to follow in order to achieve a goal
- Programmer
  - A person who produces the program
  - A person who makes a living from producing programs
- Programming language
  - A way for a programmer to write about the set of rules

## As If

- Computers are creatures from outer space
- They speak strange languages
- Programmers study their languages
- Therefore, they are able to tell the computers what to do

## The 3P Relationship

**P**rogrammers write **p**rograms using **p**rogramming languages.

## Writing vs. Programming

- Setting the theme
- Structuring
- Writing
- Proof-reading
- Defining the problem
- Planning the solution
- Coding the program
- Testing the program
- **Documenting the program**
  - commenting

## Program Composition 101

Think of it just like any of your composition course (English or Chinese).

## Defining The Problem

- Input
- Output
- Problem to be solved

## Planning The Solution

- Algorithms
  - Ways of solving the same problem
  - Some fast; some slow
  - Some long; some short
- Formats
  - Pseudo code
  - Flow chart

## The Concept of an Algorithm

- A set of steps that define how a task is performed.
- A program is a representation of an algorithm.

  A process is the activity of executing a program.

## Pseudo Code

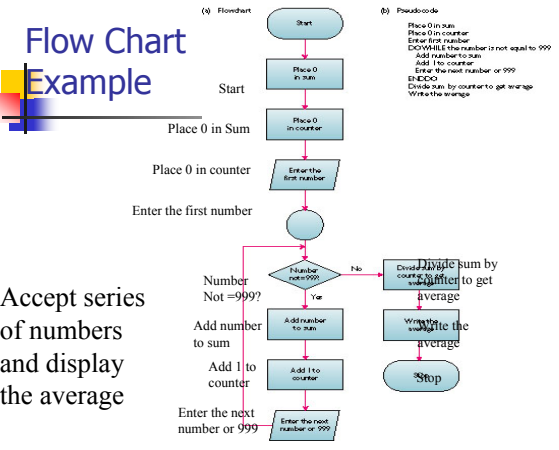
- Realizing the algorithms
- Just like drafts
  - Step by step in rough description

## The procedure Greetings in pseudocode

**procedure** Greetings
Count ← 3;
**while** (Count > 0) **do**
(print the message "Hello" and
Count ← Count +1)

## Flow Chart Example

Accept series of numbers and display the average



## Example: searching

- Searching an item in an ordered list (e.g. search a word in a dictionary)
  - # items = N
- Sequential: one-by-one from the beginning (end)
  - Actions: Average: N/2, Min: 1, Max: N
- Binary: Divide into 2 sub-lists, and repeat
  - Actions: Max: log2(N)
- If $N=2^{20}$, $N/2=2^{19}=0.5M$, $\log_2(N)=20$;
- A good algorithm is very efficient.
  - (comparison: hardware CPU: 1G -> 3G,, $->$$$$$$)

## Coding The Program

- Sentence by sentence, word by word
- Detailed description in the chosen language

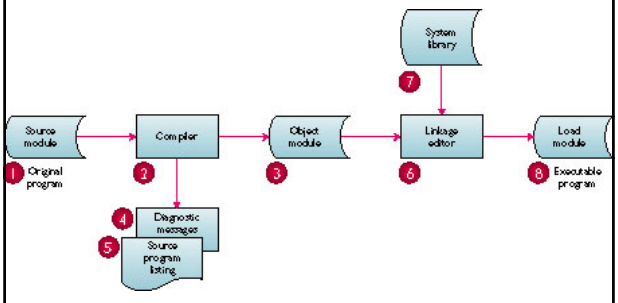
## Testing The Program

- Translation – compiler
  - Translates from **source** module into **object module**
  - Detects syntax errors
- Link – linkage editor (linker)
  - Combines **object module** with libraries to create **load module**
  - Finds undefined external references (run-time errors)
- Debugging
  - Run using data that tests all statements
  - Logic errors

## Distinction

- Source code
  - Your creation
- Object modules
  - Machine code of your creation
- Load modules
  - Machine code of pre-installed functions
- Executable programs
  - Combination of your and pre-installed machine code

## Illustrated




## Documenting The Program

- Comments within source code
- In plain English
- Convenient for
  - The programmer him/herself who needs to read the program later
  - Somebody else who needs to read the program

## Questions so far?

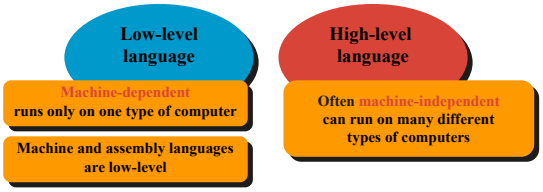
## Programming Languages

Quite a bit of them!

## Types of languages

- Old vs. new
- Low level vs. high level
- Procedural vs. non-procedural
- Object-oriented vs. non-object-oriented
- Mark-Up vs. programming language

## What are low-level languages and high level languages?

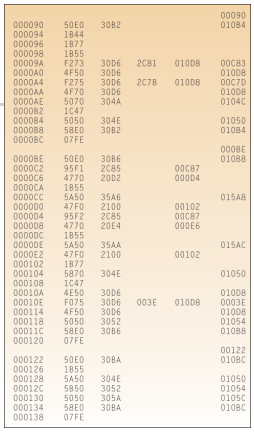



## Low or High

- Low level
  - Close to binary
  - Computer dependent
  - Ex. Machine Language, Assembly Language
- High level
  - Close to human readable style
  - Computer independent
  - **Compiler** translates from high-level language to machine language
  - Ex. All the other languages

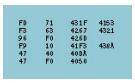
## Machine Language

- **Only language that computer directly recognizes**



## Machine Language

- Written in strings of 0 and 1
- Only language the computer understands
- All other programming languages are translated to machine language




## Assembly Language

- Mnemonic codes
  - Instructions made up of symbolic instruction codes, meaningful abbreviations and codes
  - Source program contains code to be converted to machine language
- Names for memory locations
- Assembler translates from Assembly to machine language

## Assembly Language

```
*        THIS MODULE CALCULATES THE REGULAR TIME PAY
CALCSTPY EQU   *
         ST    14,SAVERTPY
         SR    4,4
         SR    7,7
         SR    5,5
         PACK  DOUBLE,RTHRSIN
         CVB   4,DOUBLE
         PACK  DOUBLE,RATEIN
         CVB   7,DOUBLE
         ST    7,RATE
         MR    4,7
         ST    5,RTPAY
         L     14,SAVERTPY
         BR    14
*        THIS MODULE CALCULATES THE OVERTIME PAY
CALCOTPY EQU   *
         ST    14,SAVEOTPY
TEST1    CLI   CODEIN,C'O'
         BH    TEST2
         SR    5,5
         A     5,=F'0'
         ST    5,OTPAY
         B     AROUND
TEST2    SR    4,4
         SR    7,7
         SR    5,5
         PACK  DOUBLE,OTHRSIN
         CVB   4,DOUBLE
         PACK  DOUBLE,RATEIN
         CVB   7,RATE
         MR    4,7
         MR    4,=F'1.5'
         ST    5,OTPAY
AROUND   L     14,SAVEOTPY
         BR    14
*        THIS MODULE CALCULATES THE GROSS PAY
CALCGPAY EQU   *
         ST    14,SAVEGPAY
         SR    5,5
         A     5,RTPAY
         A     5,OTPAY
         ST    5,GRPAY
         L     14,SAVEGPAY
         BR    14
```

## FORTRAN

- 1954
- Represent complex mathematical formulas
- C/C++ has replaced FORTRAN
- Note: FORTRAN 90, 95, 2000 include features like object oriented programming (OOP)

## COBOL

- Designed for business applications
- COmmon Business-Oriented Language
- English-like statements make code easy to read, write, and maintain
- 1959
- Large complex data files
- Formatted business reports
- SQL for example has replaced COBOL

## COBOL

```
*   COMPUTE REGULAR TIME PAY
    MULTIPLY REGULAR-TIME-HOURS BY HOURLY-PAY-RATE
        GIVING REGULAR-TIME-PAY.

*   COMPUTE OVERTIME PAY
    IF OVERTIME-HOURS > 0
        COMPUTE OVERTIME-PAY = OVERTIME-HOURS * 1.5 * HOURLY-PAY-RATE
    ELSE
        MOVE 0 TO OVERTIME-PAY.

*   COMPUTE GROSS PAY
    ADD REGULAR-TIME-PAY TO OVERTIME-PAY
        GIVING GROSS-PAY.

*   PRINT GROSS PAY
    MOVE GROSS-PAY TO GROSS-PAY-OUT.
    WRITE REPORT-LINE-OUT FROM DETAIL-LINE
        AFTER ADVANCING 2 LINES.
```

## BASIC

- Designed for use as simple, interactive problem-solving language
- Beginner's All-purpose Symbolic Instruction Code
- 1965
- Popularity grew with PC popularity (1970s)
- Easy to learn
- Use interpreter
  - little memory
  - Slow speed (new versions providing compilers)

## BASIC

```
REM COMPUTE REGULAR TIME PAY
Regular.Time.Pay = Regular.Time.Hours * Hourly.Pay.Rate

REM COMPUTE OVERTIME PAY
If Overtime.Hours > 0 THEN
    Overtime.Pay = Overtime.Hours * 1.5 * Hourly.Pay.Rate
ELSE
    Overtime.Pay = 0
END IF

REM COMPUTE GROSS PAY
Gross.Pay = Regular.Time.Pay + Overtime.Pay
REM PRINT GROSS PAY
PRINT USING "The gross pay is $##,###.##": Gross.Pay
```

## C

- Powerful language originally designed to write system software
- Requires professional programming skills
- 1972
- Efficient code
- Portability

## C

```
#include <stdio.h>

void main()
{
        int a, b;
        a=3;
        b=2;
        printf("This result of a+b=%d",a+b);
        printf("hello!");
}
```

## C++

- Object-oriented enhancement of C
- Includes all elements of C, plus additional features for working with object-oriented concepts
- Used to develop database and Web applications

## C++

```
// portion of a C++ program that allows users to create a new zip code from a
// string or a number and expand zip codes, as appropriate, to a 10-digit number

ZipC::ZipC( const unsigned long zipnum )
{
  ostringstream strInt;
  strInt << zipnum;
  code = strInt.str();
}

const string ZipC::getCode()
{
  return code;
}

void ZipC::setCode(const string newCode)
{
  code = newCode;
}

void ZipC::expand( const string suffix )
{

  if(code.length() == 5 &&      // small size?
     suffix.length() == 4)      // length ok?
  {
     code += '-';
     code.append(suffix);
  }
}
```

## Java

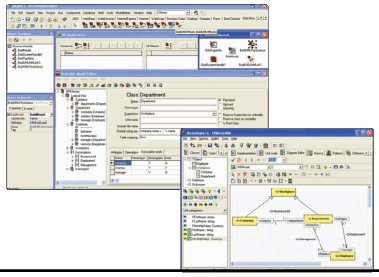
- 1996
- Cross-platform
- Java Virtual Machine (JVM)
  - Sits on top of computer's regular platform
  - Translates compiled Java code into instructions for the specific platform

## Java

```
import java.awt.Graphics;
public class FirstApplet extends
java.applet.Applet
{
        public void paint(Graphics g)
        {
                g.drawLine(0, 0, 200, 200);
        }
}
```

## What is Delphi?

- **Powerful visual programming tool**
- **Ideal for large-scale enterprise and Web applications**



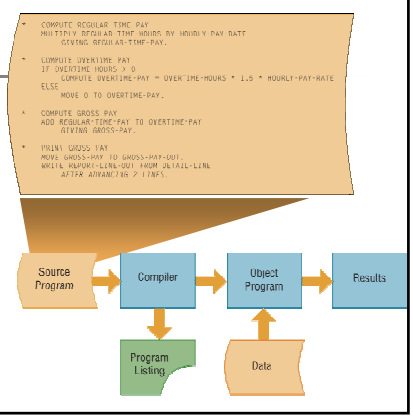
## Categories

- Machine
- Assembly
- FORTRAN
- COBOL
- BASIC
- C
- C++
- Java

Non-procedural (Machine, Assembly, FORTRAN)
low (Assembly)
high (FORTRAN)
procedural (COBOL, BASIC, C, C++, Java)
Pre 70's (COBOL, BASIC)
Post 70's (C, C++, Java)

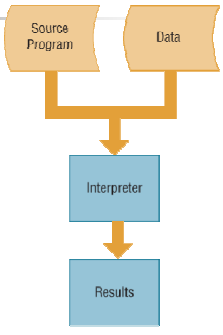
## Compiler

- **Program that converts entire source program into machine language before executing it**




## Interpreter

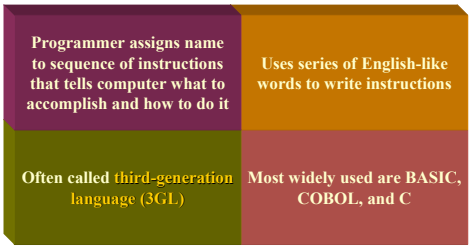
- **Program that translates and executes one program code statement at a time**
- **Does not produce object program**



## Non-Procedural vs. Procedural

- Non-procedural
  - Spaghetti
- Procedural
  - Structure
  - But…can still be spaghetti

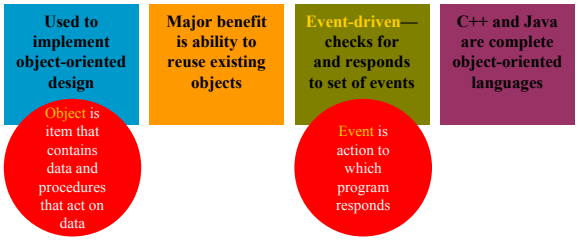
## What is a procedural language?




## Categories

- Machine — Non-procedural
- Assembly — low — procedural
- FORTRAN — high
- COBOL
- BASIC — Pre 70's
- C — Non-object-oriented — Post 70's
- C++ — Object-oriented
- Java

## What is an object-oriented programming (OOP) language?




## Object-Oriented Languages

- Object
  - Self-contained unit of data and instructions
  - Includes
    - Related facts (data)
    - Related functions (instructions to act on that data)

## Example

- Object: cat
- Data: feet, nose, fur, tail
- Functions: eat, purr, scratch, walk
- Cat: Coco, Junior

## Concept of Class

- Class
  - Defines characteristics unique to all objects of that class
- Inheritance
  - Objects of a class automatically posses all of the characteristics of the class from which it was derived

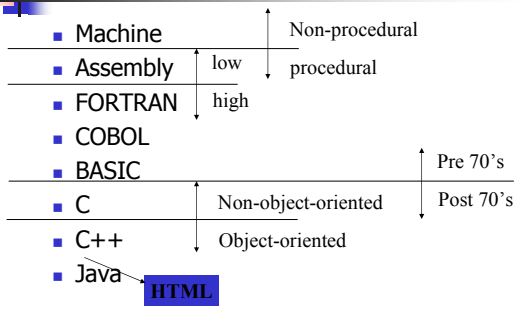
## Subclass

- Inherits characteristics from class and defines additional characteristics that are unique

## Example

- Class: Animal
  Subclass: Cat
  Subsubclass: Smart cat
  Instance: Coco

## Categories



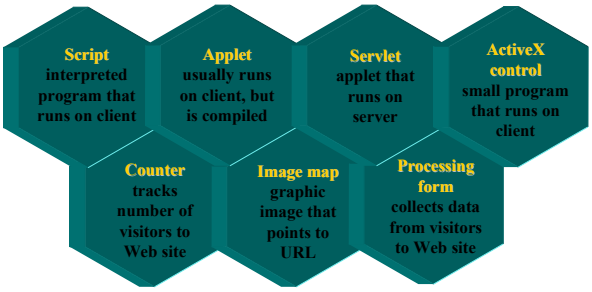

## Mark-Up Language

- Formatting text
- Examples
  - *Tex*
  - Postscript
  - HTML

## HTML

- 1989
- Use tags to structure text

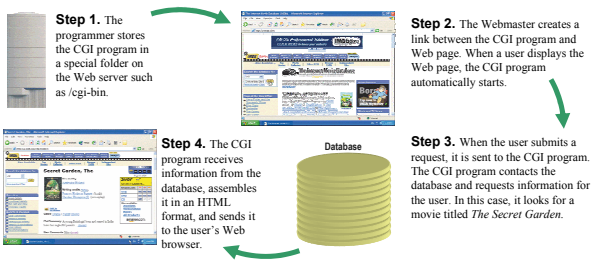
## How are special effects and interactive elements added to a Web page?



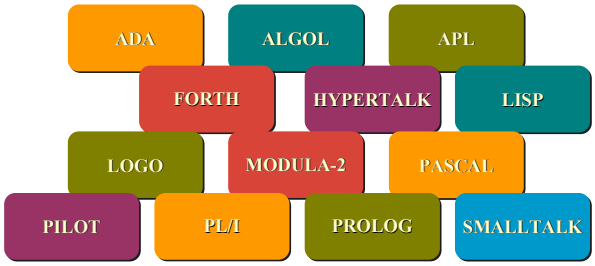

- What is the common gateway interface (CGI)?

**Communications standard that defines how Web server communicates with outside sources**

- **CGI script**—program that manages sending and receiving across CGI




## Other Programming Languages




## Learning a Programming Language?

- Enroll in courses
- Use tutorials
- **Read** sample code
- **Write** code (start small)