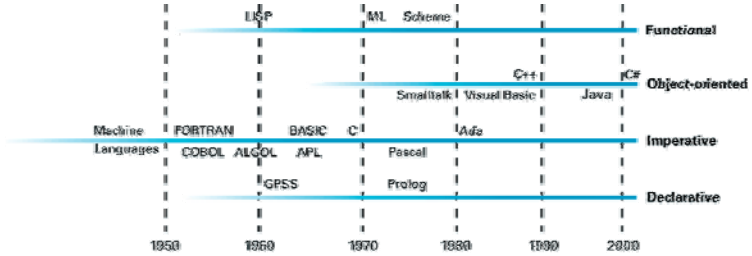
Programming language

Evolution of programming languages



- Why do people need/invent so many different programming languages?
  - Isn't C good enough?

Outline

- Imperative paradigm
  - A sequence of commands that manipulate data to produce the desired results. (C, JavaScript, Fortran, Matlab)
- Object-oriented paradigm
  - A collection of objects that can perform actions and interact with other objects. (C++, Java, C#, VisualBasic)
- Declarative paradigm
  - Describe the problem to be solved rather than the algorithms. (Prolog, Verilog, VHDL, Lex/Yacc, AMPL, SQL, HTML, latex)
- Functional paradigm
  - A composition of functions (in math sense) that accept inputs and produce outputs. (LISP, Mathematica)

Imperative paradigm

- High level languages that simplify the machine languages
  - Programmers need to describe the procedures (how to do)
- Programming primitives
  - Declarative statements: define variable/function names
  - Imperative statements: assignment, if-then-else, for-loop,…
  - Comments: explanation of statements
  - Directives: assist compiler/interpreter for code generation

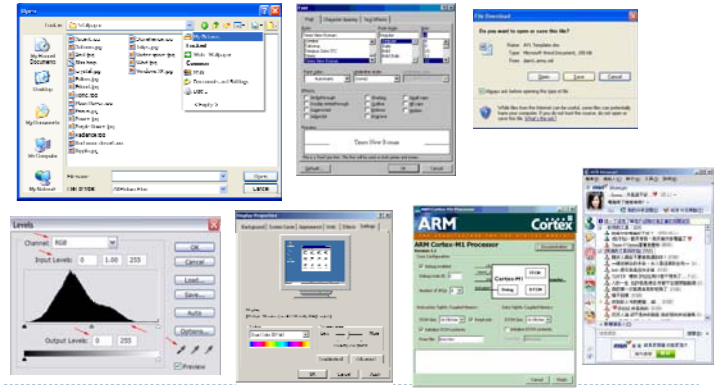
```
#include <stdio.h>
/* cntrl d terminates */
int main(int argc, char *argv[]) {
    int c;
    while( ( c = getchar() ) != EOF ) {
        putchar( c );
    }
}
```

Object-orient paradigm

- **Object**: data abstraction+procedures to process the data
- Three important concepts in OOP
  - Use **inheritance** to relate objects ➔achieve code reuse.
  - Use **polymorphism** to describe variation ➔ allow dynamic binding.
  - Use **encapsulation** to hide information. ➔ allow each object be modified independently.
- In OOP, you can concentrate on one object at a time.
  - Very good for developing large systems, such as window system, network protocol, etc.

Case study: window system

- How to create so many different types of windows (easily)?



Specification

- All windows have some basic properties
  - Location, size, resizable, shown/hidden
- All the windows need some basic functions
  - Be able to sense the mouse movement/clicks, …
  - Be able to be created, destroyed, shown, hidden, …
- Different types of windows behave differently for input
  - Button: when mouse clicks, it shows sunken figure
  - Menu item: when mouse clicks, it pops a submenu
  - …
- Windows have interaction with each others
  - Ex: The child window needs be closed with its parent window.

Using imperative programming

- Method 1: define structs for different windows and write functions for them
  - Most functions will be similar
  - When changing one property, you need to change all structs
- Method 2: define a big struct that contains everything and write functions for it.
  - Inefficient: the functions will be full of if-then-else statements
  - Very difficult to debug and to maintain.
- DON'T do either of them. We will discuss more in the software engineering.

Declarative paradigm

- A programming paradigm that expresses the problem to be solved rather than the algorithms.
  - Imperative languages need to describe algorithms explicitly.
  - Uses backend engine to "solve" problems.
  - It is usually domain specific.
    - Prolog, HTML, Verilog, VHDL, Lex/Yacc, AMPL, SQL
  - Many languages hybrid declarative and imperative paradigms.

Case study: HTML

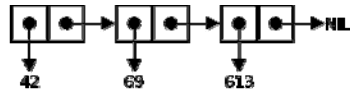
- Hyper Text Markup Language: describes the display and format of text, graphics, hyperlink to other html files…

Functional paradigm

- Computation=evaluation of math functions.
  - The output value of a "function" depends only on the arguments that are input to the function
- It avoids state and mutable data.
  - Imperative programming emphasizes changes states.
  - We will see an example for their differences.
- It uses recursion instead of iteration (loop)

Case study: Lisp

- The first functional programming language
- Syntax
  - Atom: symbol or number
  - List: consists of 0 or more expression
    - Ex: (42 69 613)
    - The first atom in the list is an "operator"
      - Ex: (+ (* 3 (+ 1 (- 4 2 (+ 3 4))))) outputs ?
  - Recursion:
    - compute n!




```
(defun factorial (n)
      (if (<= n 1)
        1
        (* n (factorial (- n 1)))))
```

List evaluation

- Ex: (+ 1 2 3 (* 4 5) 6)

Step 1. (+ 1 2 3 X 6)

- X = (* 4 5)

Step 2. the operation is +, add all atoms

Step 3. found 1, integer

Step 4. found 2, integer

Step 5. found 3, integer

Step 6. found (* 4 5), list.

evaluates the list first

Step 7. the operator is *, multiply all atoms

Step 8. found 4, integer

Step 9. found 5, integer

Step 10. end of list, evaluate 4*5=20

Step 11. found 6, integer

Step 12. end of list, evaluate 1+2+3+20+6=32