Compilation & Interpretation
New Topic

• What’s going on behind the scenes with the Processing “Play Button”?  
• AKA: How are my English-like Processing instructions understood by the computer?

Page 26 of book

(Many of the following slides were designed by Prof. Brian Kell for the CSC 101 class)
Basic Components of a Computer

• All computers, large or small, have the same basic parts
  – Central processor
  – Main memory
  – Input/output (I/O) devices
    • Including networking
  – Auxiliary storage devices
The von Neumann Architecture

All modern computers follow the logical model of computing called the **von Neumann Architecture**; Stored-program design: Program instructions executed from main memory

Computer organized into four main sections:

- **Central Processing Unit (CPU)**
  - Controls all of the computer’s operations
  - Performs all the calculations
- **Main Memory**
  - **Active** data and programs
- **Auxiliary Storage**
  - Disks and other storage devices
- **I/O Subsystem**
  - Input and output devices ("peripherals")
Program Execution Cycle

• Programs exist in main memory while they are running
  – Everything in memory is just binary bits
    • Think back to types (data is just bits laid out in a particular way!)
    • Bit patterns can represent many types of information, even computer instructions

• The CPU executes programs one instruction at a time
  – **Fetch** a binary word from main memory
  – **Decode** the instruction in that word
  – **Execute** the instruction
  – Repeat billions of times per second

• Computers are very dumb
  – They can only do very simple things (*machine language* instructions)
Programming Languages

• Low-level languages
  – Machine language
    • Extremely simple executable commands in binary code
    • Example: add two simple integers
      0001001000110111010 (get a number from main memory and load it into Register “R1”)
      01100010110000111010 (add a different number from main memory into that same Register “R1”)
      00111111001110111010 (store the result from Register “R1” in a different location in main memory)
  – Assembly language
    • Similar to machine language, except mnemonic codes are used to make it easier for humans to read
    • Example: add two simple integers
      LOAD 0237 R1 (get a number from main memory and load it into Register “R1”)
      ADD 02C3 R1 (add a different number from main memory into that same Register “R1”)
      STOR 0F3B R1 (store the result from Register “R1” in a different location in main memory)
Programming Languages

• Programs written in high-level languages (that humans can understand) must be translated into (possibly long sequences of) low-level (machine language) instructions that the CPU can execute

• Two methods of translation are used:
  – Compilation (or compiling)
  – Interpretation (or interpreting)
Compilation

• The entire program is translated by a compiler into machine language to form a binary executable file (*.exe)
  – The executable file contains all the program’s binary instructions
    • Happens once where software is produced
    • The binary executable file is how you get software
  – Machine specific (not portable [tied to CPU])
  – Compilation invented in 1952 by Grace Hopper
    • She retired from the Navy as an admiral in 1986 at age 80
Compilation

• The **entire program is translated** by a compiler into machine language to form a binary executable file (*.exe)

  – All error checking has to occur before translation, so no part of program can be executed until all errors are removed and translation occurs

  – Does this sound like Processing?
Interpretation

• Although **compilation** translates a whole program, all at once, to create a binary executable file...

• **Interpretation** happens **every time** a program runs
  – The program is translated at *run time*, one line at a time, on the computer that’s running the program
    • No binary executable file is ever produced
  – Not as efficient as compilation
    • Translation happens every time the program is run
  – But, it’s not machine specific (so it is portable) as long as an interpreter exists on desired machine
Interpretation

- Interpretation happens **every time** a program runs
  - The program is translated at *run time*, one line at a time, on the computer that’s running the program
    - No binary executable file is ever produced
  - Errors are not found until the line they are on is encountered
  - Programs could partially execute before a syntax error is found!

- Does this sound like Processing?
Compilation vs. Interpretation

- **Compilation**
  - Translation all at once
  - Like translating a book

- **Interpretation**
  - Translation happens “on the fly”
  - Like telling a story to someone who doesn’t speak your language

The recently discovered eighth book: *Harry Potter and the Filler of Big*

Gerald Ford telling Deng Xiaoping about his time at Hogwarts...
Java

• Java (that language Processing is built on) actually uses a hybrid approach
  – Compiles to bytecode statements (middle-level instructions)
    • Requires that all syntax errors removed first before compiles
  – Bytecode statements interpreted at run-time
Processing

• Processing adds another level
  – Processing (high-level) compiled to Java (high-level)
    • Requires that all syntax errors removed first before compiles
  – Java is compiled to bytecode statements
  – Bytecode statements are interpreted at run-time