CS1003/1004: Intro to CS, Spring 2004

Lecture #3: Intro to Programming Languages

Janak J Parekh
janak@cs.columbia.edu

Administrivia

- Buy those textbooks – the Papyrus guy is after me!
- Third TA
- Labs start this week
  - Section 2 for 1114 has been moved & increased to 40 students
    - Room is a little hard to get to – see instructions on the class website
  - Labs are more recitations than labs per se
  - Consolidation?
  - At least one set of OH in 251 ET
- Register for the webboard
- AcIS training sessions
- Office hours
- Who hasn't registered for a lab?

Agenda

- Finish up UNIX tutorial, talk about HW#0
- Segue into programming
  - What exactly does the code do, and why?
  - General programming concepts you need to know
- HW#1 to be released this week
  - Programming is *very* easy, and very short: more a piggyback off of HW#0 than anything else
  - Check the website
  - You've got plenty of time, so *relax*

UNIX redux

- filename~: not the same thing as ~/filename
  - The latter is a "backup" file generated by editors like emacs
- Files in UNIX are case-sensitive
  - HelloWorld.java vs. helloworld.java vs HELLOWORLD.java
- "cd" by itself is equivalent to "cd ~" or "cd ~/"
  - However, ~/ lets you reference files/directories *absolutely* as well, which cd doesn't

UNIX (II)

- Two sets of files: those on the server vs. on your computer
  - Use FTP to move things back and forth…
- Other questions from last time?

So, what to do for HW#0?

- *Not* freak out
- Let's do it right now, step by step
- *Please* ask me questions now if you don't get it…
- Steps:
  - Get HelloWorld.java or hello.c onto CUNIX account
  - Go into CUNIX and run compiler
  - Run the code
- What does the code mean?

What does the code mean?

Java

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

Class declaration

Main function declaration

Print statement (function call)

C

```
#include <stdio.h>
int main() {
  printf("Hello world!\n");
}
```

Preprocessor statement

Why do we program this way?

- A machine generally processes very primitive calculator-like instructions:
  - "Get first number from memory"
  - "Get second number from memory"
  - "Add the two numbers"
  - "Store the results back in memory"
- All of this is in binary code (machine language)
  - An "operation" might be 01110010100101001001010100010101
  - We'll learn how this works later
- In short: yuck!

One step up

- Instead of using hard-to-read machine language, use textual representations
  - LD R1, x (*load the value of X into R1 in the CPU*)
  - LD R2, y
  - ADD R0, R1, R2
  - etc.
- *Assembly* language: considered "second-level" language
- Still really annoying: what we want is "x + y"

3rd-generation languages

- Started in the 50s/60s with FORTRAN and COBOL
- Idea: take a higher-level description of what we want to do, and let the computer *translate* it into the machine language as specified before
- Called *compiler* because it might take a single high-level command, and compile a sequence of low-level commands
  - Input high-level language as text, store binary commands in *executable file*
- Alternative: *interpret* commands on the fly and issue low-level statements to the processor (BASIC does this)
- C is compiled; Java between compiled and interpreted

4th-generation languages

- Very high-level languages; historically intended for user-friendliness
- Many "application-specific" languages
  - Matlab might be construed as one
  - Rapid development tools (database languages, Visual Basic, etc.)
- Tends to do a lot of the work *itself*
- We'll focus on 3rd-generation languages in this course; skills can be used in 4GLs

Different kinds of 3GLs

- C and Java are *procedural* or *imperative* languages
  - You define *procedures*, or sets of steps, to solve
  - Java is also considered an *object-oriented* language
- Not the only way to program
  - Declarative programming: you declare "facts": Excel
  - Functional programming: you develop "functions", and then build them up; very similar to a set of equations
  - Won't look at these, although there is some conceptual overlap
- Object-oriented programming: model on top of the others that specify how to organize information and code; we'll talk about this later

Elements of procedural programming

- Procedure declaration
  - Mathematical function is a decent model, actually
  - What are the inputs?
  - What are the outputs?
- Declarative statements: define terminology to be used later in the program
- Imperative statements: actually perform actions related to what we want
- In C and Java, each declarative/imperative statement **must** end with a semicolon
- Comments: not actually processed; merely for human readability

General model of procedural programming

- Get some information from user
- Process the information
- Give the user some results
- How does Hello World follow this model?
  - Input: we don't need anything: we already know what we're going to output
  - Process: nothing to process, since we already know the output
  - Results: print out "Hello world!"
- Some other simple examples…

Compiling

- The compiler takes the source code you write in *text form* and produces binary output
- As it goes along, it checks your source for *syntax errors*
  - Errors may be cryptic at times
  - There are errors which the compiler won't be able to detect (*semantic errors*)
- If there are no errors, it spits output, and quits
- You can then run your program on the machine
  - For Java, must run through an *interpreter*
  - For C, it's machine code: just run it!