CS 1111: RESOLVING AMBIGUITY
In class activity

• You have 3 sheets of paper, do not do anything with them yet!

• Use the paper as follows:
  – Paper #1: Make a paper airplane
  – Paper #2: Describe in detail how you made it

• DO NOT TOUCH PAPER #3 YET!!!
In class activity

• You have 3 sheets of paper, do not do anything with them yet!

• Use the paper as follows:
  – Paper #1: Make a paper airplane
  – Paper #2: Describe in detail how you made it

• Now, hand your instructions to someone at least two seats away from you, and have them use their paper #3 to make your airplane ONLY by following your directions. STRICTLY!!!

• When both of you are done, compare your airplanes
Now...let's see how well they fly. • Throw them at me!
How is computing used?
Computing

- Hardware: The physical means by which computing is done. The device itself
- Operating System: The thing that allows the software to interface with the hardware
- Software: Written in programming languages, software uses the hardware to perform tasks

- Art of Computer Science (Problem Solving)
  - How to come up with a solution to a problem
  - How to verify the correctness of that solution
- Programming Skill
  - How to automate the solution
Software Development Cycle

- Start with Requirements and Specification – This is harder than you might think
- Design a solution, THEN implement and test until your system meets the specification
- Deploy the system, and take in feedback for further improvement.
• You can have errors when implementing a system. Here are 4 broad categories:
  – Syntax error: code violates the rules of the programming language, and thus cannot be run
  – Runtime error: code runs, but enters an illegal state or tries to do something impossible (such as int divided by zero).
  – Logical error: this causes your program to operate incorrectly, but not crash. That is, syntax is correct, the code doesn’t crash, but the output is incorrect.
    • What you are trying to do isn’t what happens
  – Semantic error: System produces nothing meaningful (such as you are expecting a percentage, but get a whole number because you forgot to divide by 100)
    • Kind of a subset of logic errors.
What does a programming Language Do?

- A programming language turns high-level, human readable language, into machine instructions:

<table>
<thead>
<tr>
<th>High-Level Language</th>
<th>Assembly Language</th>
<th>Machine Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>z = 0</td>
<td>ADD R3 R2 R3</td>
<td>10100001001011</td>
</tr>
<tr>
<td>x = 3</td>
<td>SUB R0 R0 R1</td>
<td>01010001111011</td>
</tr>
<tr>
<td>y = x</td>
<td>BZERO 4</td>
<td>01110001110101</td>
</tr>
<tr>
<td>while x ! = 0:</td>
<td>BRANCH 0</td>
<td>00100110101010</td>
</tr>
<tr>
<td>z = z + y</td>
<td>MOVE R2 R3</td>
<td>01110010101101</td>
</tr>
<tr>
<td>x = x - 1</td>
<td>HALT</td>
<td>10111101011111</td>
</tr>
<tr>
<td>y = z</td>
<td></td>
<td>111111111111</td>
</tr>
</tbody>
</table>
Algorithms

• An **algorithm** is a step by step list of instructions to solve a problem
  – These steps must be followed **EXACTLY**
    • If you ever find yourself shouting at the computer “Come on, you know what I mean”, it doesn’t. Computers do exactly what you tell them, but will be VERY passive-aggressive about it

• Ways to describe an algorithm
  – Psuedocode (“kinda” code)
  – Flowchart (Diagram)

• Think of the general solution first before you try to write code to solve the problem!
WHAT MAKES A GOOD ALGORITHM?
Good Algorithm

• Unambiguous
  – There are precise instructions that cannot be misinterpreted, that explain what to do each step AND what step to go to next

• Executable
  – Each step can be carried out in practice

• Terminating
  – It will eventually come to an end

• Don’t think about implementation yet, focus on “how do I solve this problem.”
Psuedocode

- Not formal code
  - Informal description of algorithm

- The syntax is informal, the algorithm is not!
  - It’s still specific, detailed, and followable
  - No specific syntax

- Can be translated into a high-level language by simply adding syntax

- Clearly indicates sequence of actions the program will take.
Psuedocode
Control Structures

- **Sequence**
  - A series of statements that execute *one after another*

- **Condition (if)**
  - Used to *decide* which of two or more different statements to execute based on certain conditions (that is, “do this OR do that”)

- **Repetition (loop)**
  - To repeat statements *while* certain conditions are true

- **Subprogram / named action**
  - A small part of another program solving a certain problem
Sequence

walk, walk, walk, walk, walk, walk, right-turn-180-degree, sit
Conditional

If (condition):
  statement1
else:
  statement2

- Condition
  - true: statement 1
  - false: statement 2
- ....
Repetition

- Repeatedly walk 6 steps
- Repeatedly walk until you are in front of the chair
- Right-turn-180-degree
- Sit

```
while (condition):
    statement1
    statement2
    statement3
...```

true

false
Subprogram

A meaningful collection of sequence, conditions, repetitions, and subprograms
Psuedocode practice

• Teach the robot to sing the song “if you’re happy and you know it, clap your hands”

• You may assume the robot-1111 computer knows what to do when it is instructed to “sing,” “clap,” “stomp,” “shout”, ...
“If You’re Happy…” Sequence

Sing “If you’re happy and you know it, clap your hands!”
Clap
Clap

Sing “If you’re happy and you know it, and you really want to show it”
Sing “If you’re happy and you know it, clap your hands.”
Clap Clap
“If You’re Happy...”
Using Loops

Repeat 2 times
Sing “If you’re happy and you know it, clap your hands!”

Repeat 2 times
Clap

Sing “If you’re happy and you know it, and you really want to show it”
Sing “If you’re happy and you know it, clap your hands.”

Repeat 2 times
Clap
“If You’re Happy…”

Using Loops

Repeat 2 times

Sing “If you’re happy and you know it, and you really want to show it”

Sing “If you’re happy and you know it, clap your hands.”

Repeat 2 times

Clap

Repeat 2 times

Clap

These are all sub programs!
Activity: Collatz Conjecture

• Let’s say we gave the robot the follow pseudocode:
  Let X be your age in years
  Repeat as long as X is not 1:
    If X is even:
      Divide X by 2
    Otherwise:
      Multiple X by 3 and add 1
  Clap as many times as you repeated

For you, find out how many times you would clap?
Activity: Collatz Conjecture

Let's say we gave the robot the following pseudocode:

Let $x$ be your age in years

Repeat as long as $X$ is not 1:

If $X$ is even:
    Divide $X$ by 2

Otherwise:
    Multiply $X$ by 3 and add 1

Clap as many times as you repeated

How do we tell?
Activity: Collatz Conjecture

- Make it unambiguous!

Let $X$ be your age in years
Let $\text{count}$ be 0 (zero)
Repeat as long as $X$ is not 1:
- If $X$ is even:
  - Set $X$ to be $X$ by 2
  - Set $\text{count}$ to be $\text{count} + 1$
- Otherwise:
  - Multiple $X$ by 3 and add 1
  - Set $\text{count}$ to be $\text{count} + 1$
Repeat $\text{count}$ time
Clap