CS1003/1004: Intro to CS, Spring 2004
Lecture #4: Language concepts, data storage

Janak J Parekh
janak@cs.columbia.edu

Administrivia

- HW#1 is out!
  - I hope you're checking the website frequently
  - Should know everything for the HW this week
  - Programming is about 5 lines of code, so don't worry too much
- Fourth TA: Rob Tobkes
  - Info on website
  - We now have office hours 5 days a week
- Labs update
  - How'd your first lab go?
  - This week only: Suhit's combining Thursday C labs to see what works best
- Register for the webboard, or else!
- Put books on reserve?

Agenda

- Finish up language intro
- Start data representation concepts
- Hopefully everything you need for the theory part of HW#1
  - If not, I'll trim the HW#1 theory a little bit
- Some overlap with labs…
Variables

- Very often, we want to store information from user as data.
- We can do so by **declaring variables**.
  - In C or Java, a declarative statement like `datatype variablename = value;`, e.g., `int i = 5;`.
  - Conceptually similar to a mathematical variable, but we try to be more precise and assign the variable a **data type**.
- We can then assign *values* to these variables:
  - From user input.
  - As the result of some computation.
  - Even random numbers.

What data types?

- Lots; you’ll see some of them in the labs.
- Some basics…
  - `int = Integer`, generally between $-2$ billion and positive 2 billion.
  - `double = Floating-point (i.e., flexible number of decimal places)`, roughly between $-10^{308}$ and $10^{308}$ (although not an infinite number of decimals!).
  - `char = Character (such as 'a')`.
  - `Strings (i.e., words, sentences or arbitrary alphanumeric data)` are complicated.
- We’ll talk about storage shortly…

And more…

- We can even declare *arrays of variables*:
  - Since we’re not going to have 50,000 declarations at the beginning of every piece of code.
  - “`int [10];`” in C, “`int [] = new int[10];`” in Java.
  - Note array is homogeneous, not heterogeneous.
- Can get much more complicated by this, but let’s not worry about that yet.
Constants and literals

- We don’t need to declare variables for everything; as we saw, we can just literally put numbers in place when we want to do things
  - e.g., print the sum of 10 and 15
- We can also declare that certain variables are constants for sanity’s sake
  - “const double Pi = 3.141592654” in C
  - “final double Pi = 3.141592654” in Java

Assignments

- Once we’ve declared our variables, we might want to assign them values
  - x = 5;
- Can do this at declaration-time, too
  - int x = 5;
- Key concept: the above two statements are not functionally equivalent!
- Operators commonly used in assignments
  - * for multiply, + for add, - for subtract…
  - Operator precedence applies: use parentheses!

Comments

- As your code becomes more complex, you’ll want to document it a little
- In C and Java, can use “/* comment */” notation
  - Can be multiple lines
- In Java, can also use “// comment” notation
  - Single-line only
  - Sometimes works in C too, but depends on age of compiler
Control statements

- We generally want to adjust the behavior of our program based on the situation
  - Options in a menu: *if* the user clicks Save, *then* save the file.
    *Else* if the user clicks Exit, *then* Exit. And so on…
  - In older programming languages, “goto” would exist
    - Considered bad form nowadays, because it can lead to very confusing code.
  - Instead, the *if-then-else* construct is used
    - *if*(something) *do something*
      *else* if(something else) *do something else*
      *else* *do a generic thing*
  - Generally, control statement itself doesn’t need a semicolon

What’s “something”?

- A *boolean* condition
- That is, if the test clause evaluates to *true*, then the corresponding code is executed
- Use curly braces ({,}) to “group together” code to be executed
- *if*(numcredits > 20) {
  printf("You’re insane!");
}

What is a boolean value?

- In Java, there is a data type called *boolean*
- Can be assigned “true” or “false”
- In C, no such datatype; you can use an int to represent it
  - 0 is false, any nonzero value is true (1 is common)
  - Can “create” a boolean datatype, much later in the semester
- Why 0 and 1?
  - Three more slides…
What are boolean operators?

- *A logic operator* that takes one or two operands and produces a boolean result
- For numbers:
  - Equals: `==`
  - Greater than: `>`
  - Less than: `<`
- Extremely important: `=` is not `==`
  - `=` is an *assignment operator*, while `==` is a boolean *test*
  - C programmers: you will get burned by this at least once in your life
  - Java programmers: the compiler will usually warn you

Combine boolean values?

- **AND**: `&&`
  - Only true if both operands are true
- **OR**: `||`
  - Only false if both operands are false
- **NOT**: `!`
  - Takes single operand and reverses it
  - We can draw “truth tables” for each of these
  - Let’s do a few examples…

Loops

- Instead of doing something *once*, can we do something *many times* until a boolean condition is satisfied?
  - Yes, we can
  - while(something is true) do something
  - Will keep on running (potentially forever)
  - How can we make an *infinite loop* (not that we’d want to)?
  - How can we make our loops *non-infinite*?
- for statement: more complex notation for loops
  - In labs…
  - *Iteration* is the fancy term for such repetition
How is this information represented in the machine?

- **Bit** (binary digit): either 0 or 1
- Why?

What can we do with bits?

- Combine them together into larger values
- Base 2 representation of numbers…
  - Converting from decimal to binary: divide by 2 repeatedly and keep the remainder
  - Converting from binary to decimal: multiply the $i^{th}$ digit by $2^i$ (with $i$ starting at 0 for the ones’ digit)

Binary representation, cont’d.

- We can also represent characters (in general) as a binary sequence
  - ASCII: American Standard Code for Information Interchange
  - Originally used 7 bits to represent a single character
  - Now, 8 bits used == byte in most computers today
  - Google for “ASCII table”
- Finally, we can apply *logic* operators to bit values
  - AND, OR, NOT, XOR are the four basics
  - Why XOR?
  - We’ve already seen the first two…

**AND and OR**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>
NOT and XOR

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

NOT

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Logic diagrams

- Use those four building blocks to build increasingly complex logic operators, and ultimately devices
- Example: how would we diagram a AND b AND c?

Next time

- Finish up data storage
- Start talking about understanding algorithms using all our newfound information