COMP 110

Review for midterm exam

Darrell Bethea
May 25, 2011
Announcements

- Yesterdays slides updated

- Midterm on tomorrow in SN014
  - Closed books, no notes, no computer

- Program 3 due Tuesday
Questions?
Today in COMP 110

- A whirlwind tour of almost everything we have covered so far
  - These slides are essentially extracted from earlier lectures
Hardware vs. Software

- Hardware – physical machine
  - CPU, Memory
- Software – programs that give instructions to the computer
  - Windows XP, Games, Eclipse
Hardware

- CPU – the “brain” of your computer
- Memory – stores data for the computer
  - How much the “brain” can remember
  - Main memory
  - Auxiliary memory
Memory

- Measured in bytes
- 1 byte = 8 bits
- Bit is either 0 or 1
- Language of the computer is in bits
Programming Languages

Your Program

Compiler

Machine Language (Bits)

High-level language (human readable)

Low-level language (computer readable)
Algorithms and pseudocode

- Algorithm – a set of instructions for solving a problem

- Pseudocode – combination of code and English used to express an algorithm before writing algorithm into code
Variables

- Used to store data in a program
- The data currently in a variable is its **value**
- Name of variable is an **identifier**
- Can change value throughout program
- Choose variable names that are meaningful!
How to use variables

- **Declare** a variable
  - `int number;`
- **Assign** a value to the variable
  - `number = 37;`
- **Change** the value of the variable
  - `number = 513;`
Keywords

- Reserved words with predefined meanings
- You cannot name your variables keywords
- if, else, return, new
What kind of value the variable can hold

Two kinds of types.

- **Primitive type** – indecomposable values
  - Names begin with lowercase letters
  - `int`, `double`, `char`, `float`, `byte`, `boolean`, some others

- **Class type** – objects with both data and methods
  - Names by convention begin with uppercase letter
  - `Scanner`, `String`, `Student`
Change a variable’s value

Syntax:
- variable = expression;

Example:
- sleepNeeded = 8;
- sleepDesired = sleepNeeded * 2;
Primitive Types: small to big

double
float
long
int
short
byte
Assignment Compatibilities

- You can only put small things into bigger things
- \texttt{byte} -> \texttt{short} -> \texttt{int} -> \texttt{long} -> \texttt{float} -> \texttt{double}
  - \texttt{myShort} \neq \texttt{myInt};
  - \texttt{myByte} \neq \texttt{myLong};
  - \texttt{myFloat} = \texttt{myByte};
  - \texttt{myLong} = \texttt{myInt};
You can ask Java to change the type of values which would violate the compatibility rule.

- `myFloat = myDouble;`
- `myByte = myInt;`
- `myShort = myFloat;`

- `myFloat = (float)myDouble;`
- `myByte = (byte)myInt;`
- `myShort = (short)myFloat;`
Unary operators
- +, -, ++, --, !

Binary arithmetic operators
- *, /, %, +, -
  - rate*rate + delta
  - 1/(time + 3*mass)
  - (a – 7)/(t + 9*v)
Modular Arithmetic – %

- **Remainder**
  - $7 \% 3 = 1$ (7 / 3 = 2, remainder 1)
  - $8 \% 3 = 2$ (8 / 3 = 2, remainder 2)
  - $9 \% 3 = 0$ (9 / 3 = 3, remainder 0)
Parentheses and Precedence

- Expressions inside parentheses evaluated first
  - (cost + tax) * discount
  - cost + (tax * discount)

- Highest precedence
  First: the unary operators: +, -, ++, --, !
  Second: the binary arithmetic operators: *, /, %
  Third: the binary arithmetic operators: +, -

Lowest precedence
Errors

- Syntax error – grammatical mistake in your program
  - `int n3 = n1 + n2, // Need a ‘;’, not a ‘,’`

- Run-time error – an error that is detected during program execution
  - `int n3 = n1 / n2; // But n2 == 0`

- Logic error – a mistake in a program caused by the underlying algorithm
  - `int n3 = n1 - n2; // But we meant to sum.`
Strings

- A string (lowercase) is a sequence of characters
  - “Hello world!”
  - “Enter a whole number from 1 to 99.”

- String (capital S) is a class in Java, not a primitive type
String animal = “aardvark”;
System.out.println(animal);
aardvark
String Concatenation

String animal = “aardvark”;
String sentence;
sentence = “My favorite animal is the ” + animal;

My favorite animal is the aardvark
Strings methods

- `myString.length();`
- `myString.equals("a string");`
- `myString.toLowerCase();`
- `myString.trim();`

- Many others
String output = myString.substring(1, 8);
### String Indices

<table>
<thead>
<tr>
<th>U</th>
<th>N</th>
<th>C</th>
<th>i</th>
<th>s</th>
<th>G</th>
<th>r</th>
<th>e</th>
<th>a</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

`String output = myString.substring(1, 8);`
# Escape Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>Double quote</td>
</tr>
<tr>
<td>'</td>
<td>Single quote</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
</tbody>
</table>
Keyboard Input

Scanner kb = new Scanner(System.in);
int num = kb.nextInt();
// this is a comment

/* This is also
   a comment */
Boolean Expressions

- An expression that is either true or false
- Examples:
  - It is sunny today (true)
  - 10 is larger than 5 (true)
  - Today is Saturday (false)
import java.util.*;

public class FlowChart {
    public static void main(String[] args) {
        System.out.println("Give me an integer:");
        Scanner keyboard = new Scanner(System.in);
        int inputInt = keyboard.nextInt();

        if (inputInt > 10) {
            System.out.println("big number");
        } else {
            System.out.println("small number");
        }
    }
}
# Java Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

Example expressions:
- `variable <= 6`
- `myInt > 5`
- `5 == 3`
Can be either *true* or *false*

```java
boolean sunny = true;
boolean cloudy = false;

if (sunny || cloudy) {
    // walk to school
}
```
AND

if ((temperature > 50) && (temperature < 75))
{
    // walk to school
}

OR

if (sunny || cloudy)
{
    // walk to school
}
The ! (NOT) operator

- !true is false
- !false is true
- Example: walk to school if it is NOT cloudy

```java
if (!cloudy)
{
    // walk to school
}
```
switch (year) {
    case 1:
        System.out.println("freshman");
        break;
    case 2:
        System.out.println("sophomore");
        break;
    case 3:
        System.out.println("junior");
        break;
    case 4:
        System.out.println("senior");
        break;
    case 5:
        System.out.println("super senior");
        break;
    default:
        System.out.println("unknown");
        break;
}

Controlling expression

Case labels

Break statements

Default case: all other values
Loops

- Loop: part of a program that repeats
- Body: statements being repeated
- Iteration: each repetition of body
- Stopping condition

Start

Enough sandwiches?

Make sandwich

Distribute sandwiches

No

Yes

38
Types of Loops

- **while**
  - Safest choice
  - Not always most elegant
  - Loop iterates 0 or more times

- **do–while**
  - Loop iterates AT LEAST once

- **for**
  - Similar to **while**, but often more convenient syntax
  - Most useful when you have a known # of iterations you need to do
Using a `while` loop

```java
int n = 1;
while (n <= 10) {
    System.out.println(n);
    n = n + 1;
}
```
Using a **do-while** loop

```java
int n = 1;
do {
    System.out.println(n);
    n = n + 1;
} while (n <= 10);
```

Don’t forget the semicolon!
Using a `for` loop

```java
int n;

for (n = 1; n <= 10; n++)
{
    System.out.println(n);
}
```
Infinite loop example

```java
int n;

for (n = 1; n <= 10; n = 0)
{
    System.out.println(n);
}
```
The break statement

```java
for (int item = 1; item <= 5; item++)
{
    System.out.print("Enter cost of item #" +
                     item + ": ");
    amount = keyboard.nextDouble();
    total = total + amount;
    if (total >= 100)
    {
        System.out.println("You spent all your money.");
        break;
    }
    System.out.println("Your total so far is "+ total);
}
System.out.println("You spent "+ total);
```
Identify the loop body from pseudocode

Output instructions to the user
Initialize variables
Prompt user for input
Read a number into variable next
sum = sum + next;
Prompt user for input
Read a number into variable next
sum = sum + next;
Prompt user for input
Read a number into variable next
sum = sum + next;
...
Output the sum

Repeated statements become your loop body

Statements that are only done once are not part of your loop body
Variables used in your loop need to be initialized (set to a value) before the loop

next
  ◦ Read a number into variable next
  ◦ We read a new value for next before using it during each iteration of the loop so we do not need to initialize it

sum
  ◦ \( \text{sum} = \text{sum} + \text{next}; \)
  ◦ sum is on the right side of an assignment statement. sum MUST have a valid value before the loop starts.
Ending a loop

- Count-controlled loops
  - If you know the number of loop iterations
  - `for (count = 0; count < iterations; count++)`

- User-controlled loops
  - Ask-before-iterating
  - Sentinel value
Nested loops example: friendly greetings

```java
for (int stdLineA = 1; stdLineA <= 3; stdLineA++)
{
    for (int stdLineB = 4; stdLineB <= 6; stdLineB++)
    {
        System.out.println("Student "+ stdLineA + " shakes Student "+ stdLineB + "’s hand.");
    }
}
```