THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

COMP 110
Introduction to Programming

Fall 2014

Time: TR 9:30 – 10:45

Room: G100 (Genome Sciences Bldg.)

Jay Aikat

FB 314, *aikat@cs.unc.edu*

Previous Class

- What did we discuss?

Announcements

- Women in Computer Science (WiCS) meeting: Aug 26, 7 pm FB008 (Brooks bldg.)
- Assignment1: **due Wed, Aug 26 11:55 PM**
- Office hours and labs (see webpage)
- Evening - doors unlocked
- Program Header and comments
- Labs vs. assignments
- Piazza usage
- Registering for class

Acknowledgement: Some slides have been adapted from Dr. Haohan Li's lectures.

Teaching Assistants

Graduate TAs:
- Jacob Bartel
- Junpyo Hong (JP)
- Gautam Sanka

Jacob Bartel

Gautam Sanka

Junpyo Hong (JP)

Teaching Assistants

Undergraduate TAs:
- Spencer Byers
- Faith Collins
- Max Daum
- Anna Flynn
- Sarah Gabr
- Navaneet Galagali
- Kevin Martin

Faith Collins

Spencer Byers

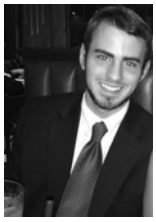
Max Daum

Teaching Assistants

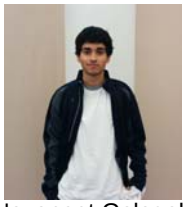
Undergraduate TAs:

- Spencer Byers
- Faith Collins
- Max Daum
- Anna Flynn
- Sarah Gabr
- Navaneet Galagali
- Kevin Martin

Anna Flynn



Kevin Martin



Navaneet Galagali

Sarah Gabr

Arithmetic Expressions

- *Expression* - a combination of one or more operands and their operators
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Remainder	**%**

- If either or both operands associated with an arithmetic operator are floating point, the result is a floating point

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3        equals?    4

8 / 12        equals?    0
```

- The remainder, or **mod**ulus, operator (%) returns the remainder after dividing the second operand into the first (only works with integer types)

```
14 % 3        equals?    2

8 % 12        equals?    8
```


Unary vs. Binary Operators

- Unary operators
  - has only one operand
  - example: - (negative, not subtraction)

    -5

- Binary operators
  - has two operands
  - example: - (subtraction)

    5 - 3

Operator Precedence

- Determines the order in which operators are evaluated:
  1. multiplication, division, and remainder
  2. addition, subtraction, and string concatenation
  3. arithmetic operators with the same precedence are evaluated from left to right
- Parentheses can be used to force the evaluation order (just like in math)

Operator Precedence (PEMDAS)

- Parentheses: 6 * (5 + 7) vs. 6 * 5 + 7
- Exponents (powers, roots – 2^5 $36^{1/2}$)
- Multiplication / Division / Mod
- Addition / Subtraction
- Left to right
- Which is these is correct?

  30 / 5 * 3 = 6 * 3 = 18 ← **this one!**

  30 / 5 * 3 = 30 / 15 = 2

Operator Precedence

- What is the order of evaluation in the following expressions?

```
a + b + c + d + e        a + b * c - d / e
  1   2   3   4            3   1   4   2

      a / (b + c) - d % e
        2     1     4   3

    a / (b * (c + (d - e)))
      4     3     2     1
```


Integral Expressions

- All operands are integers
- Result is an integer

- Examples:

  2 + 3 * 5

  3 + x – y / 7

  x + 2 * (y – z) + 18

Floating-point Expressions

- All operands are floating-point numbers
- Result is a floating-point

- Examples:

  12.8 * 17.5 – 34.50

  x * 10.5 + y - 16.2

  7.0 / 3.5

Mixed Expressions

- Operands of different types
- Examples:

  2 + 3.5

  6 / 4 + 3.9
- Integer operands yield an integer result
- Floating-point operands yield a floating-point result
- If both types of operands are present, the result is a floating-point number
  - **implicit type coercion**
- Precedence rules are followed

Type Conversion (Casting)

- Used to avoid implicit type coercion
- Syntax
  (*dataTypeName*) *expression*
- *Expression* evaluated first, then type converted to *dataTypeName*
- Examples:
  (**int**) (7.9 + 6.7) = 14
  (**int**) (7.9) + (**int**)(6.7) = 13

Questions

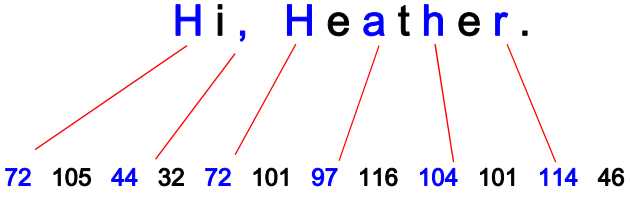
1. (5 + 4) % 6
   9 % 6
   3
2. (5 + 6) % 3.5
   11 % 3.5
   0.5
3. (double) (13) / 2
   13.0 / 2
   6.5
4. (double) (13 / 2)
   (double) (6)
   6.0

It's all About Data

- Software is data
  - numbers, characters
  - instructions, programs
- Hardware stores and processes data
  - read, write
  - add, subtract, multiply, divide

Representing Text Digitally

- All information in a computer is *digitized*, broken down and represented as numbers.



Corresponding upper and lower case letters are separate characters.

Language of a Computer

- **Machine language**: the most basic language of a computer
- A sequence of 0s and 1s
  - binary digit, or *bit*
  - sequence of 8 bits is called a *byte*
- Every computer directly understands its own machine language
  - why can't Windows programs run on Apple computers?

Bit Permutations

1 bit	2 bits	3 bits	4 bits	
0	00	000	0000	1000
1	01	001	0001	1001
	10	010	0010	1010
	11	011	0011	1011
		100	0100	1100
		101	0101	1101
		110	0110	1110
		111	0111	1111

Each additional bit doubles the number of possible permutations

Bit Permutations

- Each permutation can represent a particular item
- There are 2^N permutations of N bits
  - N bits are needed to represent 2^N unique items

How many items can be represented by

1 bit ?	**2^1 = 2 items**
2 bits ?	**2^2 = 4 items**
3 bits ?	**2^3 = 8 items**
4 bits ?	**2^4 = 16 items**
5 bits ?	**2^5 = 32 items**

Binary Numbers

- *N* bits to represent 2^N values
- *N* bits represent values 0 to 2^N-1
- Example: 5 bits
  - 32 unique values (0-31)
  - 00000 = 0
  - 11111 = 31

2^4 2^3 2^2 2^1 2^0

16 + 8 + 4 + 2 + 1

Decimal to Binary

114 → 1 1 1 0 0 1 0

Number	Place Value	Digit
114	$2^6 = 64$	1
50	$2^5 = 32$	1
18	$2^4 = 16$	1
2	$2^3 = 8$	0
2	$2^2 = 4$	0
2	$2^1 = 2$	1
0	$2^0 = 1$	0

Questions: Binary Numbers

- What's the maximum value a 6-bit number can represent? 63
- What's the decimal representation of 111010? 58 = 32+16+8+2
- What's the binary representation of 35? 100011

The class String

- String
  - sequence of zero or more characters
  - enclosed in double quotation marks
  - null or empty strings have no characters
  - numeric strings consist of integers or decimal numbers
  - length is the number of characters in a string
- The class String is used to manipulate strings
- Examples:
  - "Hello World"
  - "1234"
  - "45.67"
  - ""

Strings

- Every character has a position in the string (starting with 0)

```
"Hello World"
 0123456789...
```

- The length of the string is the number of characters in it
  - what's the length of `"Hello World"`?

    11
    (count the space)

Parsing Numeric Strings

- In Java, input from the user comes in the form of a string
  - we need to know how to get the number values out of the string
- Numeric String
  - a string with only integers or decimal numbers
  - "6723", "-823", "345.76"

String Concatenation

- A string cannot be split between two lines

X
```
String greeting = "How are you doing
                  today";
```

- Concatenation (**+**) - produces one string where the second string has been appended to the first string

```
String greeting = “How are you doing” +
                  “ today?”;
```

greeting **How are you doing today?**

String Concatenation

- Operator + can be used to concatenate two strings or a string and a numeric value or character
- Precedence rules still apply
- Example:

```
String str;
int num1 = 12, num2 = 26;
str = "The sum = " + num1 + num2;
```

str **The sum = 1226**

31

Questions

What is the result of the following String concatenations?

```
String str1 = "Hello";
String str2 = "World";
String str3 = str1 + " " + str2;
```

str3 **Hello World**

```
int num1 = 10;
int num2 = 2;
String str = "The difference is " +
             (num1 - num2);
```

str **The difference is 8**

32

Storing Data in Memory

1. Instruct the computer to allocate memory (*declaration*)
   - *how much?* based on data type
   - *how to access?* name the memory location

2. Include statements in the program to put data into the allocated memory
   - *assign* a particular value to a particular memory location
   - *initialization* - putting the first (initial) value into the memory location

Types of Storage

- Named Constant
  - once the value is stored in memory, it can't be changed

- Variable
  - the value stored in memory can be changed (can *vary*)

Named Constant

```
static final dataType IDENTIFIER = value;
```

- Declared by using the reserved word final
- Always *initialized* when it is declared
- Identifier should be in ALL CAPS, separate words with underscore (_)
- Example:
  - 1 inch is always 2.54 centimeters

```
final double CM_PER_INCH = 2.54;
```


Variable

```
dataType identifier;
```

- Must be declared before it can be used
- Can be (but doesn't have to be) initialized when declared
- Identifier should start in lowercase, indicate separate words with uppercase
- Example:
  - number of students in class

```
int numStudents;
```

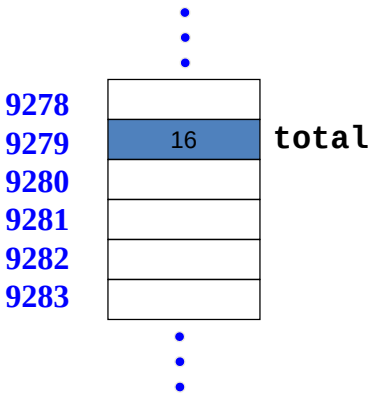
- Multiple variables (of the same data type) can be declared on a single line

```
int numStudents, numGrades, total;
```

Variable

- A variable can be given an initial value in the declaration
- When a variable is referenced in a program, its current value is used

```
int total = 16;
int base = 32, max = 149;
```



Naming Variables

- Variables should be descriptively named
- We should be able to tell what type of information the variable holds by its name
- ***Remember Java is case-sensitive!***

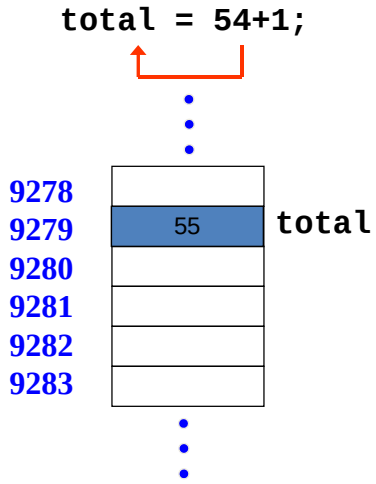
Assignment

```
variable = expresssion;
```

- Assignment Operator (=)
- *expression* can be a value (3) or a mathematical expression (2 + 1)
- The *expression* must evaluate to the same data type as the variable was declared

Assignment

- assignment statement
  - changes the value of a variable
- Expression on right is evaluated and result is stored in variable on the left
- Value that was in total is overwritten



Assignment

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4    1   3     2
```

Then the result is stored in the variable on the left hand side

Assignment

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

```
count  =  count + 1;
```

Then the result is stored back into count (overwriting the original value)

Questions

What is stored in the memory location referred to by the identifier num after each of the following statements is executed in order?

```
int num;
num = 3;
num = 5 + 4 - 2;
num = num * 2;
num = 3.4 + 5;
```

nothing
3
7
14
14

would give an error since 3.4 + 5 would not result in an int

Questions

Write Java statements to accomplish the following:

1. Declare `int` variables `x` and `y`

```
int x, y;
```

2. Update the value of an `int` variable x by adding 5 to it

```
x = x + 5;
```

3. Declare and initialize an `int` variable `x` to 10 and a `char` variable `ch` to 'B'

```
int x = 10;
char ch = 'B';
```

4. Declare `int` variables to store four integers

```
int num1, num2, num3, num4;
```

The Value of Variables

- We can use variables to save the result of expressions for use later in our programs.

```
int sum, count;
double average;

sum = 3 + 4 + 5;
count = 3;
average = (double) sum / count;
```


Questions

Which of the following are valid Java assignment statements? Assume that `i`, `x`, and `percent` have been declared as `double` variables and properly initialized.

```
1. i = i + 5;
2. x + 2 = x;
3. x = 2.5 * x;
4. percent = 10%
```

valid

invalid

valid

invalid

More Operators – Increment and Decrement

- Increment (++)
  – adds 1 to any integer or floating point

```
count++;
count = count + 1;
```

- Decrement (--)
  – subtracts 1 from any integer or floating point

```
count--;
count = count - 1;
```


Increment and Decrement

- Prefix (`++count or --count`)
  – value used in a larger expression is the new value of **`count`** (after the increment/decrement)

- Postfix (`count++ or count--`)
  – value used in a larger expression is the original value of **`count`** (before the increment/decrement)
  – increment/decrement is the last operation performed (even after assignment)

Increment and Decrement

- If **`count`** currently contains 45, then the statement

```
total = count++;
```

  assigns 45 to **`total`** and 46 to **`count`**

- If **`count`** currently contains 45, then the statement

```
total = ++count;
```

  assigns the value 46 to both **`total`** and **`count`**

Questions

- What is stored in **`total`** and **`count`** in the following statements?

```
double total = 15.5;
total++;
```

total
16.5

```
int total = 10, count = 5;
total = total + count++;
```

total	count
15	6

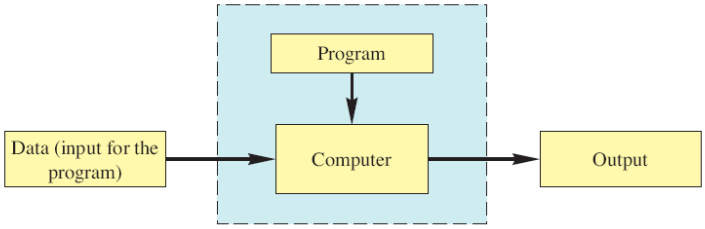
```
int total = 20, count = 3;
total = total / --count;
```

total	count
10	2

Input and Output

- Normally, a computer receives two kinds of input:
  - The program
  - The *data* needed by the program.
- The output is the result(s) produced by following the instructions in the program.

Running a Program



- Sometimes the computer and the program are considered to be one unit.
  - Programmers typically find this view to be more convenient.

Some Terminology follows…

Class Loader

- A Java program typically consists of several pieces called *classes.*
- Each class may have a separate author and each is compiled (translated into byte-code) separately.
- A *class loader* (called a *linker* in other programming languages) automatically connects the classes together.

Programmer, User, Package…

- The person who writes a program is called the *programmer.*
- The person who interacts with the program is called the *user.*
- A *package* is a library of classes that have been defined already.
  - `import java.util.Scanner;`

Arguments, Variables…

- The item(s) inside parentheses are called *argument(s)* and provide the information needed by methods.
- A *variable* is something that can store data.
- An instruction to the computer is called a *statement*; it ends with a semicolon.
- The grammar rules for a programming language are called the *syntax* of the language.

Programming

- Programming is a creative process.
- Programming can be learned by discovering the techniques used by experienced programmers.
- These techniques are applicable to almost every programming language, including Java.

Object-Oriented Programming

- Our world consists of *objects* (people, trees, cars, cities, airline reservations, etc.).
- Objects can perform *actions* which affect themselves and other objects in the world.
- Object-oriented programming (*OOP*) treats a program as a collection of objects that interact by means of actions.

OOP Terminology

- Objects, appropriately, are called *objects.*
- Actions are called *methods.*
- Objects of the same kind have the same *type* and belong to the same *class.*
  - Objects within a class have a common set of methods and the same kinds of data
  - but each object can have it's own data values.

Printing to the Screen

```
System.out.println ("Whatever you want to print");
```

- **`System`** is a class
- **`System.out`** is an object for sending output to the screen.
- **`println`** is a method to print whatever is in parentheses to the screen.

Printing to the Screen

- The object performs an action when you *invoke* or *call* one of its methods

```
objectName.methodName(argumentsTheMethodNeeds);
```


Algorithms

- By designing methods, programmers provide actions for objects to perform.
- An *algorithm* describes a means of performing an action.
- Once an algorithm is defined, expressing it in Java (or in another programming language) usually is easy.

Algorithms

- An algorithm is a set of instructions for solving a problem.
- An algorithm must be expressed completely and precisely.
- Algorithms usually are expressed in English or in *pseudocode*.

In-class Exercise 2

```
import java.util.Scanner;

public class addTwoNumbers {
    public static void main(String[] args) {
    System.out.println("Hello!");
    System.out.println("I will add two number for you.");
    System.out.println("Enter two whole numbers, one on each
line when prompted");

    int n1, n2;

    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter first number:");
    n1 = keyboard.nextInt();
    System.out.println("Enter second number:");
    n2 = keyboard.nextInt();

    System.out.println("The sum of those two numbers is");
    System.out.println(n1 + n2);
    }
}
```

Next class (Tue, Sep 2)

- Scanner methods
- String methods
- Class exercise 3

→ Reading Assignment: Chapter 1.1 and 1.2