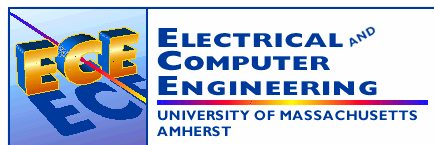

ECE 122

Engineering Problem Solving with Java

Lecture 3

Expression Evaluation and Program Interaction



Outline

- **Problem: How do I input data and use it in complicated expressions**
- **Creating complicated expressions using basic Java types (int, float, char, etc)**
- **Representing data**
- **Obtaining various forms of data from the keyboard**

Characters

- A `char` variable stores a single character
- A ***character set*** is an **ordered** list of characters, and each character corresponds to a unique number
- Character literals are delimited by single quotes:

`'a'` `'x'` `'7'` `'$'` `'.'` `'\n'`

→ `'7'` is not equivalent to `7` is not equivalent to `"7"`

Characters

- The **ASCII character set** is still quite popular
 - Eight-bits per byte.
- → (`char` is a 'primitive data type'; `String` is a class)
 - Because `String` is a class, it has many methods (operations) that can be performed on `String` objects!
- The ASCII characters include

uppercase letters	A, B, C, ...
lowercase letters	a, b, c, ...
punctuation	period, semi-colon, ...
digits	0, 1, 2, ...
special symbols	&, , \, ...
control characters	carriage return, tab, ...

Boolean

- A `boolean` value represents a true or false condition
- A boolean also can be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

Arithmetic Expressions

- An *expression* is 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	% (modulus operator in C)

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

- If both or either parts are floating point, results are floating point.

14/3.0 = 14.0/3 = 14.0/3.0 = 3.5

- The remainder operator (%) returns the remainder after dividing the second operand into the first and takes the sign of the numerator; only integers also

-14 % 3 equals? -2

8 % -12 equals? 8

16.0 % 4.0 equals invalid operands

Operator Precedence

- Operators can be combined into complex expressions (variables or literals – doesn't matter)

```
result = total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right ('associate left to right')
- Parentheses can be used to force the evaluation order
 - Can be nested too.....

Operator Precedence

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

$$a + b + c + d + e$$

1 2 3 4

$$a + b * c - d / e$$

3 1 4 2

$$a / (b + c) - d \% e$$

2 1 4 3

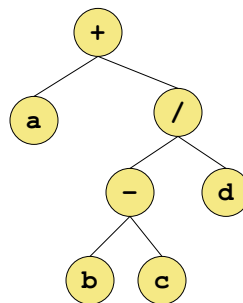
$$a / (b * (c + (d - e)))$$

4 3 2 1

Expression Trees

- The evaluation of a particular expression can be shown using an *expression tree*
- The operators lower in the tree have higher precedence for that expression

$$a + (b - c) / d$$



Assignment Revisited

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

Always the last step

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

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

What's this?

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

Assignment Revisited

- 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)

Increment and Decrement

- The increment and decrement operators use only one operand
- The *increment operator* (++) adds one to its operand
- The *decrement operator* (--) subtracts one from its operand
- The statement

```
count++;
```

is functionally equivalent to

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

Increment and Decrement

- The increment and decrement operators can be applied in *postfix form*:

```
count++
```

- or *prefix form*:

```
++count
```

- When used as part of a larger expression, the two forms can have different effects
- Because of their subtleties, the increment and decrement operators should be used with care

Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

Numeric Primitive Data

- Sizes and Ranges of storable values below.
- Use size as 'appropriate' but if in doubt, be generous.

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
float	32 bits	+/- 3.4×10^{38} with 7 significant digits	
double	64 bits	+/- 1.7×10^{308} with 15 significant digits	

Data Conversions

- Sometimes it is convenient to convert data types
- For example, we may want to treat an integer as a floating point value during a computation
- Be careful with conversions. Can lose information! (Why is one byte not enough to store 1000?)
- Widening conversions: safest; tend to go from a small data type to a larger one
 - such as a short to an int)
 - more space (magnitude) normally; can lose precision
 - int or long to float; long to double...
- Narrowing conversions can lose information;
 - Tend to go from a large data type to a smaller one (such as an int to a short) (Can lose magnitude & precision!)

Data Conversion

- Conversions must be handled carefully to avoid losing information
- *Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)
- In Java, data conversions can occur in three ways:
 - assignment conversion
 - promotion
 - casting

Assignment Conversion

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
- If `money` is a `float` variable and `dollars` is an `int` variable, the following assignment converts the value in `dollars` to a `float`

```
money = dollars
```
- Only widening conversions can happen via assignment
- Note that the value or type of `dollars` did not change

Casting

- **Casting** is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted
- For example, if `total` and `count` are **integers**, but we want a **floating point result** when dividing them, we can cast `total`:

```
result = (float) total / count;
```

Data Conversion

- **Data conversion (promotion)** happens automatically when operators in expressions convert their operands
- For example, if `sum` is a **float** and `count` is an **int**, the value of `count` is converted to a floating point value to perform the following calculation:

```
result = sum / count;
```

Expression Examples

- Type conversion
 - Examples:
`int x = 150;`
`float y, z;`
`y = x / 60;`
`z = (float) x / 60;`
 - Results for y and z are different
 - `y = 2;`
 - `z = 2.5;`

Could you do these on a test?

Expression Examples

- `int a = 1;`
- `int b = 2;`
- `int c = 3;`
- `int d = 4;`
- `int e = a+b*c-d=?`
- `int f = (a+b)*(c-d)=?`

Interactive Programs

- Programs generally need input on which to operate
- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from various sources, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

Reading Input

- The following line creates a `Scanner` object that reads from the keyboard:

```
Scanner scan = new Scanner (System.in);
```
- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
answer = scan.nextLine();
```

Reading Input

- The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used
- See `Echo.java`
- The `nextLine` method reads all of the input until the end of the line is found
- The details of object creation and class libraries are discussed further in Chapter 3

Input Tokens

- Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types
- See `GasMileage.java`

Summary

- **Understanding the order of expressions is an important issues**
 - Note that assignment (=) always happens last
- **Java provides the approaches to **convert** data**
 - Assignments
 - Promotion
 - Casting
- **Program interaction is an important part of program usability**