CS 112  Introduction to Programming

Data Conversions; Assignment Operators; for Loops

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Admin

- PS2 to be posted today
Recap: Data Types

Why data types?
- Define (1) data representation/storage, (2) allowed operations, and (3) semantics of operations (e.g., 1 / 2 vs 1.0 / 2.0)

Java is a strong typed language: every literal, every variable, every operation has a type

```java
int nA;
nA = 4;
int nB = 1;
int total = nA * 4 + nB * 3;
System.out.println( total / (nA + nB) );
double GPA = 3.0 + 0.8;
char lastNameInitial = 'Y';
```
Variable Assignments

- A variable can't be used until it is assigned a value.
  - ```
    int x;
    System.out.println(x);  // ERROR: x has no value
  ```

- You may not declare the same variable twice.
  - ```
    int x;
    int x;  // ERROR: x already exists
  ```
  - ```
    int x = 3;  // OK: declare and initialize
    int x = 5;  // ERROR: x already exists
    x = 5;      // this is OK
  ```
Recap: Type System
Issue

```java
int nA = 4;
int nB = 1;
int total = nA * 4 + nB * 3;
System.out.println( total / (nA + nB) );
```

Integer result, not what we want

```java
float nA = 4;
float nB = 1;
System.out.println( (nA * 4 + nB * 3) / (nA + nB) );
```

Not intuitive type; 4 is int?

Which * operation?
Outline

- Admin and recap
- Primitive data types
  - why data types?
  - storage and representation
  - operations
  - expressions
  - data conversions
### Issue

- Sometimes it is more efficient and natural to represent data as one type, but during a computation, we may want to get desired result in a different type.
  - e.g., raw grade points and # of grades as integers, but GPA as double.

- Sometimes we just write mixed-type expressions.
  - $4.0 / 8$
Data Conversion

- Data conversion is the conversion of data from one type to another, e.g.,
  - an int → a double,
  - a double → an int,
  - an int → a string

- Java data conversion is **per-operator**, occurring when the operator is evaluated according to the precedence rule

- Java has two types of data conversion
  - Implicit (automatic/predefined) data conversion
  - Explicit (cast) data conversion
Implicit) Predefined Data Conversion

- Seeing a mixed operation, Java tries a set of predefined data conversion rules
  - If successful, you get the results
  - If not, you get a compiler error

Discussion: How may you define a conversion rule?
- E.g., 4.0 / 8
Predefined Data Conversion Rule: Arithmetic (numeric) Promotion

- Occurs \textit{automatically} when the operands of a binary arithmetic operator are of different types
  - if either operand is \texttt{double}, the other is converted to \texttt{double}
  - otherwise, if either operand is \texttt{float}, the other is converted to \texttt{float}
  - otherwise, if either operand is \texttt{long}, the other is converted to \texttt{long}
  - otherwise, both operands are converted to \texttt{int}

Examples:
- \(4.0 / 8\) (which is it: /\texttt{double}, /\texttt{float}, /\texttt{int})
- \(4 / 8.0\) (which is it: /\texttt{double}, /\texttt{float}, /\texttt{int})
- \(4 / 8\) (which is it: /\texttt{double}, /\texttt{float}, /\texttt{int})
Example: Mixed Arithmetic Expression

\[
2.5 + \frac{10}{3} \times 2.5 - \frac{6}{4}\\
\]

\[
2.5 + 3 \times 2.5 - \frac{6}{4}\\
\]

\[
2.5 + 7.5 - \frac{6}{4}\\
\]

\[
2.5 + 7.5 - 1\\
\]

\[
10.0 - 1\\
\]

\[
9.0 \text{ (not 9!)}\\
\]
Practice: Mixed Arithmetic Expression (Offline)

\[
7 \div 3 \times 1.2 + 3 \div 2 \\
\]

\[
2 \times 1.2 + 3 \div 2 \\
\]

\[
2.4 + 3 \div 2 \\
\]

\[
2.4 + 1 \\
\]

\[
3.4
\]
Example: Mixed Assignment

- Automatic mixed type assignment allowed only if allowed by automatic numeric promotion

```c
int x;
x = 2.5; // ERROR: incompatible types
```

```c
double myGPA = 4;
```

```c
double avg = 11 / 2;
```

<table>
<thead>
<tr>
<th>myGPA</th>
<th>4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Predefined Data Conversion Rule:
Numeric to Java String

- Occurs *automatically* when one operand is a number and the other a string in the “+” operator.

- The conversion is per-operator, affecting only its operands.

- This produces the convenient *string concatenation* operation.
Java String Concatenation Conversion: Examples

1 + "abc" + 2 is "1abc2"
"abc" + 1 + 2 is "abc12"
1 + 2 + "abc" is "3abc"
"abc" + 9 * 3 + 1 is "abc271"
4 - 1 + "abc" is "3abc"
Examples

- See IntOps.java

- Fix the GPA.java program
User Forced (Explicit) Conversion: Type Casting

- **type cast**: An explicit, FORCED conversion from one type to another.

**Syntax:**

```
(type) expression
```

- Type casting has **high precedence** and casts only the item immediately next to it.

- You can cast either up (promotion) or down (truncate)
**Type Casting Examples**

```java
double result = (double) 19 / 5;   // 3.8

int result2 = (int) result;        // 3

double x = (double) 1 + 1 / 2;     // 1.0

double y = 1 + (double) 1 / 2;     // 1.5
```
Outline

- Admin and recap
- Primitive data types
  - why data types?
  - storage and representation
  - operations
  - expressions
  - data conversions
  - assignment as operator
Update vs. Algebra

What happens here?

```java
int items = 3;
items = items + 1;  // ???
```

```
items = items + 1;
```

In programming language, assignment is just an operator, with a lower precedence than the arithmetic operators: the right is assigned to the left var.
Example: Ruler

```java
public class Ruler {
    public static void main(String[] args) {
        String ruler = "1";
        ruler = ruler + " 2 " + ruler;
        ruler = ruler + " 3 " + ruler;
        ruler = ruler + " 4 " + ruler;
        System.out.println(ruler);
    }
}
```

```
% java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
Since increment updates are common, Java introduces shorthand:

```java
count = count + increment;

count += increment;
```

These expressions have the same effect:

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

when increment is 1
### Modify-and-assign

**shortcuts to modify a variable's value**

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable ++;</td>
<td>variable = variable + 1;</td>
</tr>
<tr>
<td>variable --;</td>
<td>variable = variable - 1;</td>
</tr>
<tr>
<td>variable += value;</td>
<td>variable = variable + (value);</td>
</tr>
<tr>
<td>variable -= value;</td>
<td>variable = variable - (value);</td>
</tr>
<tr>
<td>variable *= value;</td>
<td>variable = variable * (value);</td>
</tr>
<tr>
<td>variable /= value;</td>
<td>variable = variable / (value);</td>
</tr>
<tr>
<td>variable %= value;</td>
<td>variable = variable % (value);</td>
</tr>
</tbody>
</table>

```c
int x = 2;
double gpa = 3.8;

x += 3; // x = x + (3) -> 5;
gpa --; // gpa = gpa - 1.0 -> 2.8;
x *= 2; // x = x * 2 -> 10;
x *= 2 + 1; // x = x * (2+1) -> 30;
```
General: Assignment/Modify-and-Assign as Operators

- Assignment operators in a complex expression

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

```
answer += sum / 4 + MAX * lowest;
```

Then the result is used to calculate in the variable on the left hand side.
(Offline) Practice

- Compile the list of operators that we covered and their precedence levels
Example: SavingsSuccess

- What is the result of adding $1000 on Jan. 1 of each year to a fund that returns 6% per year, for 30 years?

http://www.1stock1.com/1stock1_141.htm
Outline

- Admin and recap
- Primitive data types
  - for Loops
The **for** Statement: Syntax

Reserved word

The *initialization* portion is executed once before the loop begins

The statement is executed until the *condition* becomes false

```plaintext
for ( initialization ; condition ; increment )
  statement;
```

Both semi-colons are always required

The *increment* portion is executed at the end of each iteration
Flowchart of a for loop

for ( initialization ; condition ; increment )
statement;

initialization

condition evaluated

true

statement

false

increment
The **for** Statement: Example

```java
for (int counter = 1; counter <= 3; counter ++)
{
    System.out.println ( counter );
}

// beginning of the next statement
```

Establish *initial value* of control variable.

Determine if *final value* of control variable has been reached.

Body of loop (this may be multiple statements)

Increment the control variable.
Flexibility of for Loop with Counter

Loop counter:
- can use any name, not just i
- can start at any value, not just 1
- only valid in the loop

Compare loop counter with target:
- < less than
- <= less than or equal to
- > greater than
- >= greater than or equal to

for (int i = 1; i <= 6; i ++)
{
    System.out.println("I am so smart");
}
Using for Loops?

I need to use the computer.  
Hold on. Let me see if anyone wrote on my Facebook wall... nope.

Now let me see if anyone replied to me on Twitter... nope.

Now let me see if anyone started following my Tumblr... nope.

Let me check Facebook again. Someone could have written something in the last 30 seconds... nope.

Now let me check Twitter again... now Tumblr...

Our sister has gone infinitely loopy.

Now Facebook... now Twitter... now Tumblr... now Facebook... now Twitter... now Tumblr... now Facebook... now Twitter... now Tumblr... now Facebook... now Twitter... now Tumblr...
for (int i = 1; i <= 3; i++)
{
    System.out.println("Now Facebook");
    System.out.println("Now Twitter");
    System.out.println("Now Tumblr");
}
Using **for Loops**

- Java's **for loop** statement performs a task many times.

```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```
What is the result of adding $1000 on Jan. 1 of each year to a fund that returns 6% per year?
Using for Loop: Counting Down

- Write a program generating output
  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!

- Requirement: loop counter starts with 10 and counts down
The update uses -- to count down.

```java
System.out.print("T-minus ");
for (int i = 10; i >= 1; i--) {
    System.out.print(i + ", ");
}
System.out.println("blastoff!");
```
Counting Down v2

- Requirement: loop counter starts with 1 and counts up:

T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!

    System.out.print("T-minus ");
    for (int i = 1; i <= 10; i++) {
        // ???
    }
    System.out.println("blastoff!");
Backup Slides
Type Conversions in Java

- Identity conversion (i.e., no conversion)

- Conversions related to primitive data types
  - widening primitive conversions
  - narrowing primitive conversions

- Conversions related to classes
  - widening reference conversions
  - narrowing reference conversions
  - we will cover these two cases later in the course; they are powerful tools to allow polymorphism

- Conversions related to Strings
  - string conversions: i.e., convert a numerical data to a string, e.g., the number 17 to the string “17”
Widening Primitive Conversions

- Widening primitive conversions are those that do not lose information about the overall magnitude of a numeric value.
- Java defines 19 primitive conversions as widening primitive conversions:
  - byte → short, int, long, float, double
  - short → int, long, float, double
  - char → int, long, float, double
  - int → long, float, double
  - long → float, double
  - float → double

- They are generally safe because they tend to go from a small data type to a larger one (such as a short to an int).
  - Can potential problems happen in some of the cases?
Narrowing Primitive Conversions

- Java defines 23 primitive conversions as narrowing primitive conversions:
  - byte → char
  - short → byte, char
  - char → byte, short
  - int → byte, short, char
  - long → byte, short, char, int
  - float → byte, short, char, int, long
  - double → byte, short, char, int, long, float

- Narrowing primitive conversions may lose either overall magnitude of a numeric value and/or precision.
Assignment during Declaration

- You can assign a value to a variable when declaring it.
  - This is called initialization

- Syntax:
  ```
  <type> <name> = <expression>;
  ```

- `int x = (11 % 3) + 12;`  
  ![x | 14]

- `double myGPA = 3.95;`  
  ![myGPA | 3.95]
Once given a value, a variable can be used in expressions:

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed
        // assuming 6% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .06;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
```