MEMORY, ARITHMETIC, DECISIONS

Robert A. Fulkerson
University of Nebraska at Omaha
College of Information Science and Technology

Memory Concepts

- Variable names such as a, b and c correspond to locations in the computer’s memory.
- Every variable has
  - a name
  - a type
  - a size
  - a value
- a = input.nextInt(); stores whatever the user types into the location associated with a.

Memory Concepts

- Whenever a value is placed in a memory location, it replaces whatever was there before.
- This includes keyboard input, such as
  
  ```
  int foo = input.nextInt();
  ```
  and assignment statements, such as
  
  ```
  int math = 6 + foo;
  ```
  or
  
  ```
  foo = 10;
  ```
Memory Concepts

```java
import java.util.Scanner;

public class MemoryConcepts {
    public static void main( String[] args ) {
        Scanner input = new Scanner( System.in );
        int x;
        System.out.print( "Enter value: " ); // prompt
        x = input.nextInt(); // read
        System.out.print( "Enter value: " ); // prompt
        x = input.nextInt(); // read
        System.out.print( "Enter value: " ); // prompt
        x = input.nextInt(); // read
        x = 5; // assign
        System.out.printf("The value of x is %d\n", x);
    }
}
```

Arithmetic

- Programs mainly perform arithmetic calculations.
- For integer arithmetic, the following operations are available:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Algebra</th>
<th>Java Operator</th>
<th>Java Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>f + 7</td>
<td>+</td>
<td>f + 7</td>
</tr>
<tr>
<td>Subtraction</td>
<td>p - c</td>
<td>-</td>
<td>p - c</td>
</tr>
<tr>
<td>Multiplication</td>
<td>a * b</td>
<td>*</td>
<td>a * b</td>
</tr>
<tr>
<td>Division</td>
<td>x / y</td>
<td>/</td>
<td>x / y</td>
</tr>
<tr>
<td>Modulus</td>
<td>Long division remainder</td>
<td>%</td>
<td>x % s</td>
</tr>
</tbody>
</table>

Integer Arithmetic

- Right now, our programs will deal with `int` values, which are whole number values.
- Integer division (`/` and `%`) produces only integer results.
- `numerator / denominator` will tell you how many times `denominator` goes into `numerator` completely.
- `numerator % denominator` will tell you what’s left over (the remainder) after `denominator` goes into `numerator` as many times as it can.
Integer Arithmetic with / (division)

```java
public class JavaDivision {

    public static void main(String[] args) {
        int x = 10;

        System.out.printf("%dn", x / 2); // 5
        System.out.printf("%dn", x / 3); // 3
        System.out.printf("%dn", x / 4); // 2
        System.out.printf("%dn", x / 5); // 2
        System.out.printf("%dn", x / 6); // 1
        System.out.printf("%dn", x / 11); // 0
    }
}
```

Integer Arithmetic with % (modulus)

```java
public class JavaModulus {

    public static void main(String[] args) {
        int x = 10;

        System.out.printf("%dn", x % 2); // 0
        System.out.printf("%dn", x % 3); // 1
        System.out.printf("%dn", x % 4); // 2
        System.out.printf("%dn", x % 5); // 0
        System.out.printf("%dn", x % 6); // 4
        System.out.printf("%dn", x % 11); // 10
    }
}
```

Order of Precedence Rules

- Operators are evaluated in an expression according to the rules of precedence.
- Expressions within () are evaluated first, and are evaluated from the inside out if there are nested parenthetical expressions.

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Operations</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>* / %</td>
<td>Multiplication, Division, Modulus</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>+ -</td>
<td>Addition, Subtraction</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>=</td>
<td>Assignment</td>
<td></td>
</tr>
</tbody>
</table>
Order of Precedence

- `int x = 5 + 6 - 7 * 8;`

- `int average = (1 + 2 + 3 + 4 + 5) / 5;`

- `int zz = 5 * 60 % 7 / 8 + 7 * 6;`

- `x` is -45, `average` is 3 and `zz` is 42

Equality and Relational Operators

<table>
<thead>
<tr>
<th>Algebra</th>
<th>Java Operator</th>
<th>Example</th>
<th>What?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=</code></td>
<td><code>==</code></td>
<td><code>x == y</code></td>
<td><code>x</code> is equal to <code>y</code></td>
</tr>
<tr>
<td><code>!=</code></td>
<td><code>!=</code></td>
<td><code>x != y</code></td>
<td><code>x</code> is not equal to <code>y</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&gt;</code></td>
<td><code>x &gt; y</code></td>
<td><code>x</code> is greater than <code>y</code></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td><code>&lt;</code></td>
<td><code>x &lt; y</code></td>
<td><code>x</code> is less than <code>y</code></td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td><code>&gt;=</code></td>
<td><code>x &gt;= y</code></td>
<td><code>x</code> is greater than/equal to <code>y</code></td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td><code>&lt;=</code></td>
<td><code>x &lt;= y</code></td>
<td><code>x</code> is less than/equal to <code>y</code></td>
</tr>
</tbody>
</table>

Decision Making: the if structure

- The `if` structure in Java allows a program to make a decision to perform or skip an action.
- The behavior of the program is based on the truth of a Boolean assertion/condition
- When an `if` statement is encountered:
  - If the condition is met (true), the action of the `if` structure is executed
  - If the condition is not met (false), the action of the `if` structure is skipped entirely
Example program using if

```java
import java.util.Scanner;

public class Comparison {
    public static void main ( String[] args ) {
        Scanner input = new Scanner ( System.in );
        System.out.print ( "Enter first integer: ");
        int number1 = input.nextInt ();
        System.out.print ( "Enter second integer: ");
        int number2 = input.nextInt ();
        if (number1 == number2)
            System.out.printf ( "%d == %d
", number1, number2 );
        if (number1 != number2)
            System.out.printf ( "%d != %d
", number1, number2 );
        if (number1 < number2)
            System.out.printf ( "%d < %d
", number1, number2 );
        if (number1 > number2)
            System.out.printf ( "%d > %d
", number1, number2 );
        if (number1 <= number2)
            System.out.printf ( "%d <= %d
", number1, number2 );
        if (number1 >= number2)
            System.out.printf ( "%d >= %d
", number1, number2 );
    }
}
```

Order of Precedence Rules
- Expressions within () are evaluated first, and are evaluated from the inside out if there are nested parenthetical expressions.

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>* / %</td>
<td>Left-to-right</td>
<td>Multiplicative</td>
</tr>
<tr>
<td>+ -</td>
<td>Left-to-right</td>
<td>Additive</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>Left-to-right</td>
<td>Relational</td>
</tr>
<tr>
<td>== !=</td>
<td>Left-to-right</td>
<td>Equality</td>
</tr>
<tr>
<td>=</td>
<td>Right-to-left</td>
<td>Assignment</td>
</tr>
</tbody>
</table>