Selection Statements

CS 180
Sunil Prabhakar
Department of Computer Science
Purdue University
Problem

- Write a program that tells a patient if their total cholesterol measure is too high or not.
  - The measure is an integer and is too high if it exceeds 239.
- Your program should read in the measure and output an appropriate evaluation.
Choices

- Clearly, in order to solve this problem, we need to be able to choose which of the alternative messages to print.
- All programming languages provide this ability to choose: selection statements.
- Java provides **if-else** and **switch** statements.
Flow of control

- Once a statement is executed, the next statement of the program is executed.
- Calling a method transfers the control to the statements in the method.
- Once the method returns, control returns to statement that made the call.
- Changing this flow of control is achieved using **if** and **switch** (and other) statements.
- These are called control flow statements.
public class CholesterolCheck {
    public static void main(String[] args) {
        int chLevel;
        chLevel = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter your cholesterol measure"));

        if (chLevel > 239) {
            System.out.print("Your cholesterol level is too high.");
        } else {
            System.out.print("Your cholesterol level is not too high.");
        }
    }
}
if (chLevel > 239)
  System.out.print("... is too high.");
else
  System.out.print("... l is not too high.");

Depending upon the value of chLevel, one or the other branch is executed, not both.
if-else syntax

The boolean expression is a special type of expression which can have one of two values: true or false values.

- If the expression evaluates to true, the if-statement is executed; otherwise the else-statement is executed.
Multiple conditional statements

- We can have multiple statements for the if and/or else branches.
- Braces are used to combine multiple statements into a single block.
if-else Blocks Control Flow

```java
if (<boolean expression>)
{
    if-statement1;
    if-statement2;
    ...
}
else
{
    else-statement1;
    else-statement2;
    else-statement3;
    ...
}
```

Diagram:
- **if** block
  - If the boolean expression is true, execute:
    ```
    previous statement
    if-statement1;
    if-statement2;
    ...
    ```
  - If the boolean expression is false, execute:
    ```
    false
    else-statement1;
    else-statement2;
    else-statement3;
    ...
    ```
- **else** block
  - Execute:
    ```
    true
    if-statement1;
    if-statement2;
    ...
    ```
- **next statement**
public class CholesterolCheck {
    public static void main(String[] args) {
        int chLevel;
        chLevel = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter your cholesterol measure"));

        if (chLevel > 239) {
            System.out.println("Your cholesterol level is too high.");
            System.out.println("You should probably see a doctor.");
        } else {
            System.out.println("Your cholesterol level is not too high.");
            System.out.println("Don't forget to exercise.");
        }
    }
}
### Boolean Expressions

- **boolean** is a primitive data type.

- A boolean value can only be either **true** or **false**

- A simple boolean expression compares two values using a relational operator, E.g.
  - `chLevel > 239`
  - `height < weight`
  - `gpa == 3.0`

- The operands can be either variables or literal values.
Relational Operators

The following operators can be used to compare numeric data types:

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

Do not confuse with assignment (=).
Complex boolean expressions

- Boolean expressions can be combined using boolean operators to form more complex expressions.
  - Analogous to normal conditional statements.

For example,
- given three int variables i, j, and k:
  - \((i > j) \land (k == 5)\)
  - evaluates to true only if the value stored in i is greater than the value stored in j AND the value stored in k is equal to 5; false otherwise.
## Boolean Operators

- Boolean operators take boolean expressions as operands.

<table>
<thead>
<tr>
<th>Boolean Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Not (negation). Takes only one operand</td>
</tr>
<tr>
<td>^</td>
<td>Exclusive-OR</td>
</tr>
</tbody>
</table>

*These are two "pipe" characters*
Boolean Operators (contd)

- `bool1 && bool2` is **true** if both `bool1` and `bool2` are **true**; otherwise it is **false**
  - `(x > 2) && (x<10)` is **true** for `x=3`; **false** for `x=11`;

- `bool1 || bool2` is **true** if either `bool1` or `bool2` (or both) are **true**; otherwise it is **false**
  - `(x>2) || (x<10)` is always true.

- `!bool1` is **true** if `bool1` is **false**, and **false** if `bool1` is **true**
  - `!(x>2)` is **true** for `x=1`; and **false** for `x=3`;

- `bool1 ^ bool2` is **true** if `bool1` and `bool2` are **different**; otherwise it is **false**
  - `(x>2) ^ (x<10)` is false for `x=3`; and true for `x = 11`;
Definition of Boolean Operators

- Truth table for boolean operators

|   |   | p && q | p || q | !p | p^q |
|---|---|--------|--------|----|-----|
| false | false | false | false | true | false |
| false | true  | false | true  | true | true |
| true  | false | false | true  | false | true |
| true  | true  | true  | true  | false | false |

- Sometimes true and false are represented by 1 and 0 (NOT in Java).
- In C and C++, 0 is **false**, everything else is **true**.
Examples of boolean expressions.

```java
int i, j;
byte b, c;
float f, g;
double d, e;

i < j  f >= i  d > 9.3  2 == c  j != i  g <= (b*c + d)

(i > j) && (f >= i)  (d > 9.3) || (2 != d)  !(c <= j) ^ (j != i)

((i > j) && (f >= i)) || ((d > 9.3) || (2 != d)) ^ (!(c <= j) ^ (j != i))
```

© Sunil Prabhakar
Problem

- Write a program that tells a patient how to interpret their total cholesterol measure. The measure is an integer. A cholesterol measure
  - Less than 200 is "Desirable"
  - 200-239 is "Mildly High"
  - 240 and above is "High"

- Your program should read in the measure and output an appropriate evaluation.
The Nested-if Statement

The then and else block of an if statement can contain any valid statements, including other if statements. An if statement containing another if statement is called a nested-if statement.

```java
if (chLevel > 239)
    System.out.print("... is too high.");
else
    if (chLevel > 199)
        System.out.print("... is mildly high.");
    else
        System.out.print("... is normal.");
```
Sample control flow

```java
if (chLevel > 239) {
    // too high.
} else if (chLevel > 199) {
    // mildly high.
} else {
    // normal.
}
```

© Sunil Prabhakar

Thursday, February 9, 2012
if (chLevel < 1) {
    System.out.print("There is an error in your input");
}
...

else is Not Required

System.out.print("...
    error ...");

next statement
Caution: Dangling else

```java
if (chLevel > 199) {
    if (chLevel > 239)
        System.out.print("Too High");
    else
        System.out.print("Mildly High");
} else
    System.out.print("Normal");
```

Each `else` paired with nearest unmatched `if` -- use braces to change this as needed.
Boolean Variables

- Boolean values can be stored in `boolean` variables -- a primitive datatype.
- Can be used in boolean expressions.

```java
boolean hasWon, isFinalLevel;

isFinalLevel = false;
...

isFinalLevel = (gameLevel == 10);
hasWon = (numberOfZombies == 0);

if (hasWon)
    if (isFinalLevel)
        System.out.println("WOW -- you beat the game!");
    else
        startNextLevel();
else
    restartSameLevel();
```
A method that returns a boolean value is a Boolean method.

A call to this method can be used as a boolean value.

```java
public boolean isGameOver(){
    if(numberOfHumans < 1) || (numberOfZombies<1)
        return true;
    else
        return false;
}
```

```java
if(isGameOver ())
    if(numberOfZombies < 1)
        System.out.println("You WON!!");
    else
        System.out.println("Sorry, you lost!!");
else
    System.out.println("Battle on…");
```
## Operator Precedence Rules

<table>
<thead>
<tr>
<th>Group</th>
<th>Operator</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subexpression</td>
<td>( )</td>
<td>Innermost first</td>
</tr>
<tr>
<td>Postfix increment and decrement</td>
<td>++, --</td>
<td>Right to Left</td>
</tr>
<tr>
<td>Unary operators</td>
<td>++, --, - , !</td>
<td>Right to Left</td>
</tr>
<tr>
<td>Multiplicative</td>
<td>*, /, %</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Additive</td>
<td>+ , -</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Relational</td>
<td>&lt;, &lt;=, &gt;, &lt;,</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Equality</td>
<td>!=, ==</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Boolean AND</td>
<td>&amp;&amp;</td>
<td>Left to Right</td>
</tr>
<tr>
<td>Boolean OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>=</td>
<td>Right to Left</td>
</tr>
</tbody>
</table>
Increment and Decrement

- The increment (++) and decrement (--) operators can precede the operand
  - x++; ++x; y--; --y;

- Their effect on the operand is the same, however, they vary only in terms of the timing of the increment or decrement.

- The postfix operators are applied AFTER the variable’s value is used.

- The prefix operator are applied BEFORE
Example

```java
int x=2, y=10;
x = y++;  // X is: 10
Y is: 11
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```java
int x=2, y=10;
x = y--;  // X is: 10
Y is: 9
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```java
int x=2, y=10;
x = ++y;  // X is: 11
Y is: 11
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```java
int x=2, y=10;
x = --y;  // X is: 9
Y is: 9
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```

```java
int x=2, y=10, z;
z = x++ * --y;  // X is: 3
Y is: 9
Z is: 18
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
System.out.println("Z is:" + z);
```

```java
int x=2, y=10;
x = --x * ++y;  // X is: 11
Y is: 11
System.out.println("X is:" + x);
System.out.println("Y is:" + y);
```
Announcements

- Midterm exam
  - Monday Feb 13th
  - 6:30pm - 7:30pm PHY 112
- Special help session
  - 3-6pm Sunday Feb 12th
  - LWSN 1142
- Piazza questions
- Recitations will discuss exam topics
- Next week: Prof. Korb will lecture.
Examples

- Write a program that classifies triangles
  - by their sides
  - by their angles

- Write a program that classifies quadrilaterals by their sides and one angle
  - consider only parallelograms, rectangles, squares and rhombi.
class TriangleClassifier {
    public static void main(String args[]) {
        int side1, side2, side3;
        String type;
        // read in all three side lengths
        if (side1 == side2)
            if (side1 == side3)
                type = "Equilateral";
            else
                type = "Isosceles";
        else
            type = "Scalene";
        System.out.println("This is a " + type + " triangle. ")
    }
}
```java
class TriangleClassifier2 {
    public static void main(String args[]) {
        double angle1, angle2, angle3, maxAngle;
        String type;
        . . . // read in all three angles
        maxAngle = Math.max(angle1, Math.max(angle2, angle3));
        if ((maxAngle - 90.0) < 0.0000001)
            type = " right-angled";
        else
            if (maxAngle > 90)
                type = "n obtuse";
            else
                type = "n acute";
        System.out.println("This is a" + type + " triangle.");
    }
}
```
Quadrilateral Logic

- **oppositeSidesEqual**: 
  - true
  - false

- **hasRightAngle**: 
  - true
  - false

- **allSidesEqual**: 
  - true
  - false

- **next statement**: 
  - "Square"
  - "Rectangle"
  - "Rhombus"
  - "Parallelogram"
  - "Unknown"
public class QuadClassifier {
  public static void main(String args[]) {
    int side1, side2, side3, side4;
    int anyAngle;
    // read in all four side lengths and any one angle
    if ((side1 == side3) && (side2 == side4))
      if (anyAngle == 90)
        if (side1 == side2)
          type = "Square";
        else
          type = "Rectangle";
      else
        if (side1 == side2)
          type = "Rhombus";
        else
          type = "Parallelogram";
    else
      type = "type that is unfamiliar to this program";
    System.out.println("The quadilateral is a " + type);
  }
}
public class QuadClassifier2 {
    boolean oppositeSidesEqual, allSidesEqual, hasRightAngle;
    
    oppositeSidesEqual = (side1==side3) && (side2==side4);
    allSidesEqual = oppositeSidesEqual && (side1 == side2);
    hasRightAngle = anyAngle==90;
    
    if (oppositeSidesEqual){
        if(hasRightAngle){
            if(allSidesEqual){
                type = "Square";
            } else {
                type = "Rectangle";
            }
        } else {
            type = "Rhombus";
        }
    } else {
        if(allSidesEqual) {
            type = "Rhombus";
        } else {
            type = "Parallelogram";
        }
    } else {
        type = " type that is unfamiliar to this program";
    }
}
Comparing Objects

- As with numeric types, we can compare two objects for equality and inequality
  - Relational operators (<, < , <=, <=) are not allowed for objects.

- Recall that these are reference types.
  - Thus, we are really testing for equality of the references, i.e., are the two variable referencing the same object or not?

- If we want to compare their contents, we need special methods.
Comparing Objects

Student student1, student2, student3;
student1 = new Student();
student2 = new Student();
student3 = student2;

if(student1 == student2)
    System.out.println("Equal");
else
    System.out.println("Not Equal");
Comparing Objects

```java
Student student1, student2, student3;
student1 = new Student();
student2 = new Student();
student3 = student2;

if (student3 == student2)
    System.out.println("Equal");
else
    System.out.println("Not Equal");
```

"Equal"
Comparing Object Contents

- If we want to compare the internal contents of objects, we have to use methods.
- For example, with String objects, we can use:
  - `equals()` to test equality of two strings' contents.
  - `equalsIgnoreCase()` to test equality while treating upper and lower case of the same letter as equal.
  - `compareTo()` to determine the relative position of two strings in lexicographic order.
  - Each is called on one string with the other as an argument.
Comparing Strings

String str1 = "Elephant", str2 = "eLePhant";

if (str1.equals(str2)) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}

if (str1.equalsIgnoreCase(str2)) {
    System.out.println("Equal, but for case");
} else {
    System.out.println("They are not equal");
}
Strings are compared character by character. The return value is an integer that tells us their relative order.

```java
String str1, str2;
int i;

i = str1.compareTo(str2);

if(i==0)
    System.out.println(str1 + " equals " + str2);
else
    if(i>0)
        System.out.println(str2 + " precedes " + str1);
    else
        System.out.println(str1 + " precedes " + str2);
```
equals() for Other Classes

- All classes get an equals() method for free.
- However, it may not work as expected.
- If you wish to compare objects of your classes for equality of content you should write an appropriate method.
- We will see some examples later.
The \textbf{char} Data Type

Each character of a string is an instance of a primitive type called \textbf{char}.

- In Java, a \textbf{char} variable is stored using two bytes
  - each character is encoded using an international standard called UNICODE
  - character literal are written with \textit{single quotes},
    \texttt{e.g., 'c' 'x' '०' 'स' 'س' 'ψ' '大' '👩' '👨'
  - some languages may use ASCII -- an older subset of UNICODE (1 byte per char).
Unicode Encoding

- Extended version of ASCII to accommodate world languages and common symbols.
  - Each character mapped to a code (2 bytes)
  - Often written in hexadecimal e.g., '\u1234'
  - Can convert between int and char types

```java
char ch = 'A';
int code = '\u2620';

System.out.println(ch);
System.out.println( (int) ch);
ch = (char)code;
System.out.println(ch);
```

© Sunil Prabhakar
We can compare characters with each other or with numeric

```java
char ch1='x', ch2=64, ch3='\u00a9';
int i,j;

if( ch1 == 'X' || ch2 == 99){
    ...
}
if(ch2 < i && ch3 < ch2){
    ...
}
System.out.println(ch1 +  ch2 + ch3);
System.out.println("" + ch1 +  ch2 + ch3);
```
Strings and Characters

- We can get the character at a given index of a string using charAt()

```java
char ch;
String s = "Go Purdue!!!";

ch = s.charAt(4);
System.out.println("The Character at index 4 is:" + ch);
```

- We can combine characters into a string

```java
char ch1 = 'C', ch2 = 111, ch3 = '\u006c', ch4='\u0089';
String s;
s = "" + ch1 + ch2 + ch2 + ch3 + ch4;
System.out.println(s);
```
if and switch

- The **if** statement is essential for writing useful programs.
- Other control flow statements (e.g., switch and loops) can be implemented using if statements.
  - However, programs are more readable and less error-prone by using these other control flow statements.

- Next: **switch**
Converting Grades to Points

class StudentV5 {
    . . .

    public void recordGrade()
    {
        char letterGrade;
        . . .
        letterGrade = JOptionPane.showInputDialog(null, "Enter Grade").charAt(0);
        if (letterGrade == 'A')
            grade = 4;
        else if (letterGrade == 'B')
            grade = 3;
        else if (letterGrade == 'C')
            grade = 2;
        else if (letterGrade == 'D')
            grade = 1;
        else
            grade = 0;
        . . .
    }
}
Using a `switch` statement

```java
switch(letterGrade) {
    case 'A':
        grade = 4;
        break;
    case 'B':
        grade = 3;
        break;
    case 'C':
        grade = 2;
        break;
    case 'D':
        grade = 1;
        break;
    default:
        grade = 0;
}
```

Equivalent code

```java
if (letterGrade == 'A')
    grade = 4;
else if (letterGrade == 'B')
    grade = 3;
else if (letterGrade == 'C')
    grade = 2;
else if (letterGrade == 'D')
    grade = 1;
else
    grade = 0;
```
Using a **switch** statement

```java
switch (letterGrade) {
    case 'A':
        grade = 4;
        break;
    case 'B':
        grade = 3;
        break;
    case 'C':
        grade = 2;
        break;
    case 'D':
        grade = 1;
        break;
    default:
        grade = 0;
}
```

**Equivalent code**

```java
if (letterGrade == 'A')
    grade = 4;
else if (letterGrade == 'B')
    grade = 3;
else if (letterGrade == 'C')
    grade = 2;
else if (letterGrade == 'D')
    grade = 1;
else
    grade = 0;
```

Executed only if letterGrade == 'A'
Executed only if letterGrade == 'B'
Executed only if letterGrade == 'C'
Executed only if letterGrade == 'D'
Executed only if none of the above cases match
Syntax for the `switch` Statement

Switch statement syntax:

```java
switch (<integer expression>) {
    case <label 1>: <case body 1>
        <break;>
    ...
    case <label n>: <case body n>
        <break;>
    default: <default body>
}
```

- Case label: Literal value or constant.
- Case body can be any number of statements (even empty).
- Can only be of integer type: byte, short, int, long, or char.

Optional:

- Default block
- Break statements
Switch statement (cont.)

- The integer expression can have only one of the following types:
  - char, byte, short, or int (and enum types)
  - Java 7 allows Strings too.

- The label must be a literal or named constant of the same type as the integer expression
  - Each label must be unique.
  - Labels may be listed in any order.
  - The default case applies when no label matches.

- A break causes execution to break out of the switch statement to the next statement.
  - each break is optional
Simple **switch** statement

```java
switch ( N ) {
    case 1: x = 10;
    case 2: x = 20;
    case 3: x = 30;
}
```
switch with break, and default

switch ( N ) {
    case 1: x = 10;
    break;
    case 2: x = 20;
    case 3: x = 30;
    break;
    default: x = 0;
}
```java
switch ( N ) {
    case 1: x = 10;
          break;
    case 2: 
    case 3: x = 30;
          break;
    default: x = 0;
}
```
```java
int month, daysInMonth;
boolean leapYear;
    // set month (1 - 12) and leapYear appropriately
    switch (month) {
       case 2:
           if(side1 == side2)
               daysInMonth = 29
           else
               daysInMonth = 28;
           break;
       case 3:
       case 5:
       case 7:
       case 8:
       case 10:
       case 12:
           daysInMonth = 31;
           break;
       default :
           daysInMonth = 30;
    }
```
A common situation is to assign one of two alternative values depending on a condition.

```java
if (graduateStudent)
    passGrade = 90;
else
    passGrade = 80;
```

We can use the following ternary shortcut:

```java
passGrade = graduateStudent ? 90 : 80;
```
Short-Circuit Evaluation

- Sometimes it is unnecessary to compute all subparts of a boolean expression in order to know the overall value. E.g.,
  - \( i == j || k < 5 \)
    - if \( i \) is equal to \( j \), the expression is true no matter what the value of \( k \) is
  - \( i == j && k < 5 \)
    - if \( i \) is not equal to \( j \), the expression is false no matter what the value of \( k \) is

- Most compilers will stop evaluating a expression if its overall value is clear earlier.
  - Called Short-Circuit (Lazy) Evaluation
Short-Circuit Evaluation

Why should we care?

Can impact side effects of expressions:
- done = (i == j) || (k++ < 5)
  - k is incremented only if i was not equal to j

Can be useful
- okay = (j == 0) || (i/j > 5)
  - prevents divide by 0 error

We can force Full (Eager) Evaluation by using & instead of && and | instead of ||

Caution: &, |, ^ also denote bitwise operations if the operands are integer values not boolean.
Caution: Strings

```java
String str1, str2;
str1 = new String("Pete");
str2 = new String("Pete");
if(str1==str2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

```java
String str1, str2;
str1 = "Pete";
str2 = "Pete";
if(str1==str2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```
Caution: Wrapper Classes

```java
Integer int1, int2;
int1 = new Integer(1);
int2 = new Integer(1);
if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

```java
Integer int1, int2;
int1 = 1;
int2 = 1;
if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```
Caution: Wrapper Classes

```
Integer int1, int2;

int1 = 1;
int2 = 1;

if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");

int1 += 1;
if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");

int2 += 1;
if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```

© Sunil Prabhakar
Caution: Wrapper Classes

```java
Integer int1, int2;

int1 = new Integer(1);
int2 = new Integer(1);

if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");

int1 += 1;
if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");

int2 += 1;
if(int1==int2)
    System.out.println("Equal");
else
    System.out.println("Not equal");
```
Caution: Object Equality

- Be very careful about using == and != with both Wrapper classes and Strings.
- They can have some surprising behaviors.
- In general, when using numeric values for boolean conditions, do not use Wrapper classes -- use the primitive types instead.
**Precedence Examples**

```java
int x= 1, y=10, z=100;
boolean bool, test=false;

- x = -y + y * z;  \quad x = (-y) + (y*z);
- x ==1 && y > 5  \quad (x ==1) && (y > 5)
- 4 < x && !test  \quad (4<x) && (!test)
- bool = x!=y && y == z  \quad bool = (x!=y) && (y ==z)
- x==y || y>4 && z<2  \quad (x==y) || ((y>4) && (z<2))
```
Side effects -- 1

```c
int x = 1, y = 10;

- x = y++; x: 10  y: 11
- x = ++y; x: 11   y: 11
- x = -++y; x: -11 y: 11
- x = -y++; x: -10 y: 11
- x = -y--; x: -10 y: 9
- x = -(--y); x: -9  y: 9
- x = ++y++; ERROR!
```
Prefix vs. postfix.

- A prefix (postfix) operator is equivalent to executing the operator before (after) using the value of the variable:

  \[ z = x++ \times --y; \]

- Is equivalent to:

  \[
  \begin{align*}
  y &= y-1; \\
  z &= x \times y; \\
  x &= x + 1;
  \end{align*}
  \]

What about:

\[ z = x++ \times x++; \]
More Examples

\[ z = x++ \times x++; \]

- Is equivalent to:
  \[ z = x \times (x+1); \]
  \[ x = x+2; \]

\[ z = x++ \times --x; \]

- Is equivalent to:
  \[ z = x \times x; \]

Can be tricky -- use with care.
Side effects -- 2

```
int x= 1, y=10, z=100;
boolean bool, test=false;
x = y = z;                x: 100  y: 100  z: 100
x = y = ++z;              x: 101  y: 101  z: 101
bool = (x=11)>y           x: 11   y: 10   bool: true
bool = (x=11)>y++         x: 11   y: 11   bool: true
bool = (x=11)> ++y        x: 11   y: 11   bool: false
bool = (x=3) > y && (z=5)<10 x: 3    y: 10    z: 10   bool: false
bool = (x=3) > y & (z=5)<10 x: 3    y: 10    z: 5   bool: false
```
Alternative styles

if ( <boolean expression> ){
    ...
}
else {
    ...
}

All are equivalent -- the compiler doesn't care.