CS101, Mock
Boolean Conditions, If-Then

Boolean Expressions and Conditions

The physical order of a program is the order in which the statements are *listed.* The logical order of a program is the order in which the statements are *executed*. With conditional structures and control structures that we will examine soon, it is possible to change the order in which statements are executed.

Boolean Data Type

To ask a question in a program, you make a statement. If your statement is true, the answer to the question is yes. If your statement is not true, the answer to the question is no. You make these statements in the form of *Boolean expressions*. A Boolean expression asserts (states) that something is true. The assertion is evaluated and if it is true, the Boolean expression is true. If the assertion is not true, the Boolean expression is false.

In Java, data type **boolean** is used to represent Boolean data. Each **boolean** constant or variable can contain one of two values: **true** or **false**.

Relational Operators

A Boolean expression can be a simple Boolean variable or constant or a more complex expression involving one or more of the relational operators. Relational operators take two operands and test for a relationship between them. The following table shows the relational operators and the Java symbols that stand for them.

Java Symbol	*Relationship*
==	Equal to (note **two** equal signs!)
!=	Not equal to (there is no <>)
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

For example, the Boolean expression

number1 < number2

is evaluated to **true** if the value stored in **number1** is less than the value stored in **number2**, and evaluated to **false** otherwise.

When a relational operator is applied between variables of type **char**, the assertion is in terms of where the two operands fall in the collating sequence of a particular character set. For example,

character1 < character2

is evaluated to **true** if the character stored in **character1** comes before the character stored in **character2** in the collating sequence of the machine on which the expression is being evaluated. Although the collating sequence varies among machines, you can think of it as being in alphabetic order. That is, *A* always comes before *B* and *a* always before *b*, but the relationship of *A* to *a* may vary. This is an artifact of the way the alphabet was defined in the ASCII code. For ASCII, it turns out the A < a.

We must be careful when applying the relational operators to floating point operands, particularly equal (==) and not equal (**!**=). Integer values can be represented exactly; floating point values with fractional parts often are not exact in the low-order decimal places. Therefore, you should compare floating point values for near equality. For now, *do not compare floating point numbers for equality*.

For example consider the program below:

```
class NumberTest {
        public static void main(String[] argv) throws Exception {
                float f = 1111111111;
                f = f + 1;
                System.out.println(f);
        }
}
```

The output is:

1.11111117E9

This is the same as 1111111117, not the expected 1111111112 that is the correct value. This is due to roundoff errors with the IEEE 754 format. In any case, this shows that if we compare for exact equality with an expected floating point value, we may not get a match. But if we compare for a data range, then we should be ok.

Boolean Operators

You might recall we studied Boolean operators when we discussed algorithms. To review, a simple Boolean expression is either a Boolean variable or constant or an

expression involving the relational operators that evaluates to either true or false. These simple Boolean expressions can be combined using the logical operations defined on Boolean values. There are three Boolean operators: AND, OR, and NOT. Here is a table showing the meaning of these operators and the symbols that are used to represent them in Java.

Java Symbol	*Meaning*
&&	AND is a binary Boolean operator. If both operands are true, the result is true. Otherwise, the result is false. **True** True False **False** False False
||	OR is a binary Boolean operator. If at least one of the operands is true, the result is true. Otherwise, the result is false. **True** True True **False** True False
!	NOT is a unary Boolean operator. NOT changes the value of its operand: If the operand is true, the result is false; if the operand is false, the result is true. **Not Value** **True** False **False** True

If relational operators and Boolean operators are combined in the same expression in Java, the Boolean operator NOT (**!**) has the highest precedence, the relational operators have next higher precedence, and the Boolean operators AND (**&&**) and OR (||) come last (in that order). Expressions in parentheses are always evaluated first.

For example, given the following expression (**stop** is a **bool** variable)

```
count <= 10 && sum >= limit || !stop
```

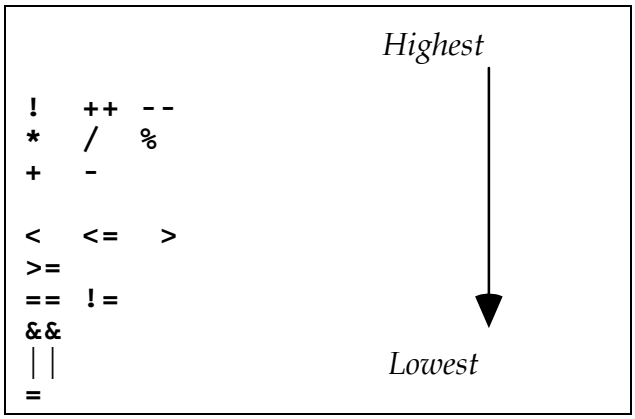
!stop is evaluated first, the expressions involving the relational operators are evaluated next, the **&&** is applied, and finally the || is applied. The evaluation is done in left-to-right order and halts as soon as the result is known.

It is a good idea to use parenthesis to make your expressions more readable, e.g:

```
(((count <=10) && (sum>=limit))  ||  (!stop))
```

This also helps avoid difficult-to-find errors if the programmer forgets the precedence rules.

The following table summarizes the precedence of some of the Java operators we have seen so far.



If-Then and If-Then-Else Statements

The If statement allows the programmer to change the logical order of a program; that is, make the order in which the statements are executed differ from the order in which they are listed in the program. The If-Then statement uses a Boolean expression to determine whether to execute a statement or to skip it. The format is as follows:

```
if (boolean_expression)
        statement;
```

The statement will be executed if the Boolean expression is true. If you wish to execute multiple statements, which is called a *block*, use curly braces:

```
if (boolean_expression) {
        statement1;
        statement2;
        …
        statement99;
}
```

Although the curly braces are not needed when only a single statement is executed, some programmers always use curly braces to help avoid errors such as:

```
if (boolean_expression)
        statement1;
        statement2;
```

This is really the same as:

```
if (boolean_expression)
        statement1;
statement2;
```

Such a condition commonly arises when initially only a single statement is desired, and then a programmer goes back and adds additional statements, forgetting to add curly braces.

We can also add an optional **else** or **else if** clause to an if statement. The else statement will be executed if all above statements are false. Use else if to test for multiple conditions:

```
if (boolean_expression1)
        statement1;                     // Expr1 true
else if (boolean_expression2)
        statement2;                     // Expr1 false, Expr2 true
else if (boolean_expression3)
        statement3;                     // Expr1, Expr2 false, Expr3 true
…
else
        statement_all_above_failed;   // Expr1, Expr2, Expr3 false
```

Here are two examples of equivalent if statements:

```
if (number < 0)
        number = 0;
sum = sum + number;
```

```
if (number < 0) {
        number = 0;
}
sum = sum + number;
```

This third example is different due to the placing of the curly braces. It lumps together the statements number=0 and sum=sum+number only if number is less than 0:

```
if (number < 0) {
        number = 0;
        sum = sum+ number;
}
```

Here is another example.

```
System.out.println(“Today is a “);
if (temperature <= 32)
{
   System.out.println("Cold day.");
   System.out.println(“Sitting by the fire is appropriate.");
}
else
{
   System.out.println ("nice day.  How about taking a walk?”);
}
```

There is a point of Java syntax that you should note: There is never a semicolon after the right brace of a block (compound statement).

Finally here is an example using else-if, also referred to as a **nested if statement**:

```
if (y==false)
if (z < 50) {
}
else {
  ….
}
```

There may be confusion as to what the final else statement goes to. Does it match up with z<50? or with y==false? The rule is that the else is paired with the most recent if statement that does not have an else. In this case, the final else statement is paired with (z < 50). The above is equivalent to:

```
if (y==false) {
        if (z < 50) {
        …
        }
        else {
         ….
        }
}
```

If we wanted the else to match up with y==false, we should change the braces accordingly:

```
if (y==false) {
        if (z<5) { … }
}
else {
        …
}
```

In nested If statements, there may be confusion as to which **if** an **else** belongs. In the absence of braces, the compiler pairs an **else** with the most recent **if** that doesn't have an **else**. You can override this pairing by enclosing the preceding **if** in braces to make the **then** clause of the outer If statement complete.

Common Bug! Confusing = and ==

The assignment operator (=) and the equality test operator (==) can easily be miskeyed one for the other. What happens if this occurs? Fortunately, the program will not compile. Look at the following statements.

```
int i=0;
i == i + 1;
System.out.println(i);
```

This code fragment generates an error during compilation. i==i+1 will be flagged as an improper instruction.

Look at the next statement going the other direction:

```
int i=0;
if (i=1) {
        System.out.println("Value is 1");
}
else {
        System.out.println("Value is 0")l
}
```

This code will also be flagged as an error by the compiler. i=1 does not return a Boolean, and we must make a Boolean comparison in the if-statement.

Fortunately, these common problems are discovered by the Java compiler. However, if you start to program in C or C++, these statements will **not** be flagged as errors by the compiler because they are valid statements. Unfortunately, they are probably not statements you wish to make and will likely result in a program that does not function correctly.