Lecture 18: Conditions

Sierra College
CSCI-12
Spring 2015
Mon 04/06/15

Announcements

- **General**
  - Midterms returned today, end of lecture
  - Still finishing up the two most recent assignments…
- **Schedule**
  - Check the schedule going forward, a few changes have been made
  - Spring withdraw deadline is Thurs 4/16
    - Final off-ramp: after that point, you will receive a letter grade for this class
- **Past due assignments**
  - Nothing at this time
- **Current assignments**
  - Nothing at this time
- **New assignments**
  - Next program should be posted Tues, we'll discuss in lab Weds

Some General Observations

- I've done lots of grading recently (HWs and midterm)
- Some general instructor observations:
  - **Read the assignment completely, and make sure your code does all that is asked**
    - Use the provided rubric as a checklist
  - **Check your results closely, especially against any sample outputs provided**
    - Just returning results isn't enough, they need to be the right results
  - **Make sure your code compiles cleanly**
    - If I can't run it, I can't grade it effectively (and I won't fix it for you)
  - **Do a code cleanup before submitting**
    - Indents inside braces, add comments/whitespace, updated header, label outputs
    - Make it readable *("quick grading is usually favorable grading")*
  - **Check Canvas and review any grading feedback**
    - Make sure you understand what you might have missed…
    - … so you can FIX it for next next time
  - **Above all: start early and allow time to ask questions (don't get "stuck")**
    - Murphy's Law: something will ALWAYS go wrong, esp. as a deadline approaches!

Lecture Topics

- **Last time**:
  - Midterm exam
- **Today**: 3 topics
  - Mid-semester survey
  - Lecture: conditions
    - Beginnings of next program (accurate age calcs)
  - Midterm return and recap

Flows of Control

- **Flow of control**: the order of execution of instructions in a program
- The 4 basic flows of control in programming:
  - **Sequential execution**
    - Instructions are executed in sequential order
  - **Method call**
    - Jump to another method, evaluate it, then return to calling step
  - **Selection** (Ch.5)
    - Deciding which execution "fork in the road" to take, depending on some true-or-false logic
  - **Looping** (Ch.6)
    - Executing a set of instructions a proscribed number of times, or until some condition is met

Decision-Making in Programs

- **Decision-making** in programming involves choices about which "fork in the execution path" should be taken
- Decision-making may go by multiple terms
  - Selection, branching, if/if-else statements, switch/case statements, …
  - We'll use "selection" (book uses "decisions")
- In order to implement selection, we need two things:
  - **Conditions**, the logical syntax which determines which path to take
  - **Selection structures**, which implement those path decisions
- We will consider **conditions** in isolation, first…

What Are Conditions?

- Conditions are simple ***boolean*-valued expressions**
  - They are evaluated at run-time, based on existing variables' values
  - They represent the outcome of some decision-making
  - They evaluate to one simple ***boolean*** values: either *true* or *false*
    - They answer "questions" with a simple yes-no response
- Conditions allow us to start implementing **program logic**
  - They determine which code execution path to take thru the various selection and looping structures
- Conditions are formed using 3 new types of **operators:**
  - **2 Equality operators**: ==, !=
  - **4 Relational operators**: >, >=, <, <=
  - **3 Logical operators**: &&, ||, ! (AND, OR, NOT)

Condition Examples

```
//assume all of the following data exists and has been initialized
boolean  c1, c2, c3, c4, c5;
int inputNum, ticketId, winningNumber;
SimpleDate today;
Motor dcMotor;

c1 = (  (inputNum < MIN_VALUE)  ||  (inputNum > MAX_VALUE)  );
c2 = (  (inputNum >= MIN_VALUE) && (inputNum <= MAX_VALUE) );
c3 = ( ticketId == winningNumber );
c4 = ( today.getMonth() != 10 );
c5 = !dcMotor.isPoweredOn();
```

- What are the actual T/F values for these??
  - It all depends upon the individual data values at run-time

Conditional Operators Across Languages

language	Java	C	C++	HTML	Java Script	PHP	VB.NET
Equality operators	== !=	== !=	== !=	N/A	==, === !=, !==	== !=	= <>
Relational operators	> >= < <=	> >= < <=	> >= < <=	N/A	> >= < <=	> >= < <=	> >= < <=
Logical operators	&& || !	(AND) (OR) (NOT)	&& || !	&& || !	N/A	&& || !	&&, and ||, or !	And, AndAlso Or, OrElse, Xor Not

- The format of conditional operators is remarkably identical across languages
- Learn it once in Java, and your next languages will feel very familiar

Equality Operators

Equality operator	Type (number of operands)	Meaning
= =	binary	is equal to
!=	binary	is not equal to

- Used to determine if the values of the LHS and RHS expressions are equal or not equal
  - **Binary operator**: takes two operands
  - Operands are typically two variables, or expressions, of some primitive numerical types
  - ***boolean* result**: *true* or *false*
- **A common "gotcha": using = instead of == to test equality**
  - The = operator is making a data assignment
  - The == operator is asking a question: are these two values equal, yes or no?
- **Do NOT use these operators to compare 2 objects field-by-field**
  - This is a job for every object's *equals*() method instead

Equality Operators Examples

```
13 public class OperatorsEquality {
14
15     public static void main(String [] args) {
16
17         // declarations
18         int age = 21;
19         final int MIN_AGE = 21;
20         String str1 = new String("Hello");
21         String str2 = new String("Hello");
22         String str3 = str2;
23         boolean c1, c2, c3, c4, c5;
24
25         // equality operator expressions
26         c1 = (age == MIN_AGE);
27         c2 = (age != MIN_AGE);
28         c3 = (str1 == str2); // compares object refs, not data
29         c4 = (str3 == str2); // compares object refs, not data
30         c5 = (str1.equals(str2));
31
32         System.out.println("c1 = " + c1);
33         System.out.println("c2 = " + c2);
34         System.out.println();
35         System.out.println("c3 = " + c3);    // these point to different data
36         System.out.println("c4 = " + c4);    // these point to the same data
37         System.out.println("c5 = " + c5);    // now we are comparing string data, not addresses
38
39     } // end main
40
41 } // end class
```

```
c1 = true
c2 = false

c3 = false
c4 = true
c5 = true
```

See ***OperatorsEquality.java*** in **Example Source Code**

Relational Operators

Relational Operators	Type (number of operands)	Meaning
<	binary	is less than
<=	binary	is less than or equal to
>	binary	is greater than
>=	binary	is greater than or equal to

- Used to compare the values of the LHS and RHS expressions
  - **Binary operator**: takes two operands
  - Operands are typically two variables, or expressions, of some primitive numerical types
  - *boolean* result: *true* or *false*

Relational Operators Examples

```
15      public static void main(String [] args) {
16
17          // declarations
18          int age = 21;
19          final int MIN_AGE = 21;
20          boolean c1, c2, c3, c4;
21
22          // relational operators
23          c1 = (age <  MIN_AGE);
24          c2 = (age <= MIN_AGE);
25          c3 = (age >  MIN_AGE);
26          c4 = (age >= MIN_AGE);
27
28          System.out.println("c1 = " + c1);
29          System.out.println("c2 = " + c2);
30          System.out.println("c3 = " + c3);
31          System.out.println("c4 = " + c4);
32
33      } // end main
34
35  } // end class
```

```
c1 = false
c2 = true
c3 = false
c4 = true
```

See ***OperatorsRelational.java*** in **Example Source Code**

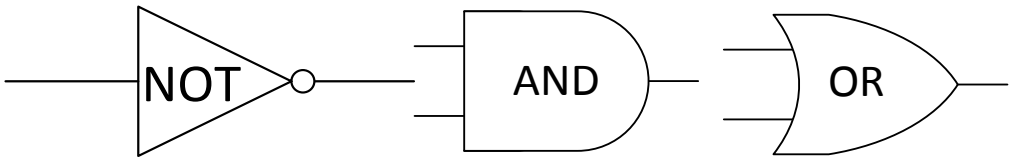
Logical Operators

Logical Operator	Type (number of operands)	Meaning
!	Unary	NOT
&&	Binary	AND
||	Binary	OR

- Used to construct more complex, compound expressions, out of simpler equality and/or relational expressions
- All operands now must be *boolean* expressions

Boolean Logic Truth Table & Gates

a	b	!a	a && b	a || b
true	**true**	false	true	true
true	**false**	false	false	true
false	**true**	true	false	true
false	**false**	true	false	false



Short-Circuit Evaluation

- For any logical operator, the operands are evaluated left to right
- If the result of the logical operation can be determined after evaluating the first operand only, the second operand is not evaluated.
  - If the first operand of an OR (**||**) expression is *true*, the overall result will always be ***true***
  - If the first operand of an AND (**&&**) expression is *false*, the overall result will always be ***false***

Logical Operator Examples

```
----jGRASP exec: java OperatorsLogical

Enter age > 11
age = 11 ==> child or senior: true
age = 11 ==> legal age but not yet senior: false

----jGRASP: operation complete.

----jGRASP exec: java OperatorsLogical

Enter age > 18
age = 18 ==> child or senior: false
age = 18 ==> legal age but not yet senior: true

----jGRASP: operation complete.

----jGRASP exec: java OperatorsLogical

Enter age > 65
age = 65 ==> child or senior: true
age = 65 ==> legal age but not yet senior: false

----jGRASP: operation complete.
```

```java
import java.util.Scanner;

public class OperatorsLogical {

    public static void main(String [] args) {

        // declarations
        Scanner input = new Scanner(System.in);
        int age;
        boolean c1, c2;

        // constants: preferable to hardwiring constants i
        final int AGE_CHILD = 12;
        final int AGE_ADULT = 18;
        final int AGE_SENIOR = 65;

        // user input
        System.out.print("Enter age > ");
        age = input.nextInt();

        // logical operators
        // both of these may SHORT-CIRCUIT if the first term alone is sufficient to evaluate

        c1 = ((age <= AGE_CHILD) || (age >= AGE_SENIOR));
        System.out.println("age = " + age + " ==> child or senior: " + c1);

        c2 = ((age >= AGE_ADULT) && (age < AGE_SENIOR));
        System.out.println("age = " + age + " ==> legal age but not yet senior: " + c2);

    } // end main

} // end class
```

See ***OperatorsLogical.java*** in **Example Source Code**

Equivalent or Alternate Expressions

- Sometimes it's useful to reformulate an expression for simplicity or clarity
- Provided that both expressions give the same logical results, there is no problem doing so
- Two useful principles:
  - **DeMorgan's Laws**
  - **Negation of operators**

DeMorgan's Laws

1. NOT(A AND B) = (NOT A) OR (NOT B)

   !(A && B) ←→ (!A) || (!B)

2. NOT(A OR B) = (NOT A) AND (NOT B)

   !(A || B) ←→ (!A) && (!B)

Thus, to find an equivalent expression:

a) Change any interior && to ||

b) Change any interior || to &&

c) Negate each individual operand expression

Negation of Operators

Expression	!(Expression)
a == b	a != b
a != b	a == b
a < b	a >= b
a <= b	a > b
a > b	a <= b
a >= b	a < b

- Negation of an operator simply means “everything ELSE”

Equivalence Examples

These expressions are equivalent:

`!( age > 18  &&  age < 65 )`	original expression
`!( age > 18 )  ||  !( age < 65 )`	by DeMorgan's Laws
`( age <= 18  ||  age >= 65 )`	by negation of expressions

Precedence

- **Precedence** dictates which operators are evaluated in which order (in the absence of any explicit parentheses)
  - We saw this with arithmetic operators
  - Conditional operators are now added to the list
- **Precedence table** references (the complete rules)
  - Appendix B in your textbook
  - Oracle's website (see URL in lecture module, or next slide)
- Again, the same general guidance on precedence holds:
  - In existing code, know how to **interpret** the rules of precedence
  - In new code you write, do not rely upon precedence, and instead **be explicit** in your intentions with grouping and parentheses

Operator Precedence

- This table is from Oracle's Java Trail tutorial website
- See URL in today's lecture module

- The equality, relational, and logical operators have their own places in the operator precedence hierarchy, as shown

Operator Precedence

Operators	Precedence
postfix	*expr++ expr--*
unary	*++expr --expr +expr -expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

For Next Time

- **Lecture Prep**
  - Text readings and lecture notes
  - Look over the next assignment, which should be posted on Tuesday. We'll go over this in lab Weds.
- **Assignments**
  - See slide 2 for new/current/past due assignments