STUDENT OUTLINE

Lesson 15 – Boolean Algebra - Loop Boundaries

INTRODUCTION: Conditional loops often prove to be one of the most difficult control structures to work with. **This lesson will give you more strategies that can be used for defining the beginning and ending conditions for loops in your programs.**

The key topics for this lesson are:

A. Negations of Boolean Assertions
B. Boolean Algebra and DeMorgan's Laws
C. Application of DeMorgan's Laws
D. Generating Random Values

VOCABULARY: BOOLEAN ASSERTIONS DE MORGAN'S LAWS
BOOLEAN ALGEBRA

DISCUSSION:

A. Negations of Boolean Assertions

A.K.A. A CONDITIONAL

1. A Boolean assertion is simply an expression that results in a true or false answer. For example,

```
a > 5     0 == b     a <= b
```

are all statements that will result in a true or false answer.

2. To negate a Boolean assertion means to write the opposite of a given Boolean assertion. For example, given the following Boolean assertions noted as A, the corresponding negated statements are the result of applying the not operator to A.

!A

A	not A
`5 == x`	`5 != x`
`x < 5`	`x >= 5`
`x >= 5`	`x < 5`

3. Notice that negation of Boolean assertions can be used to re-write code. For example:

TAKES LONGER FOR HUMAN BEINGS TO READ

```
if (!(x < 5))
  // do something...
```

can be rewritten as

BETTER

```
if (x >= 5)
  // do something ...
```

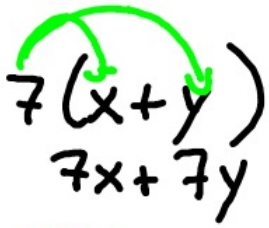
B. Boolean Algebra and DeMorgan's Laws

1. Boolean Algebra is a branch of mathematics devoted to the study of Boolean values and operators. Boolean Algebra consists of these fundamentals operands and operations:

   operands (values): **`true`**, **`false`**

   operators : and (`&&`), or (`||`), not (`!`)

   (note: Java has other Boolean operators, such as ^ (XOR – "exclusive or") and equivalence.)

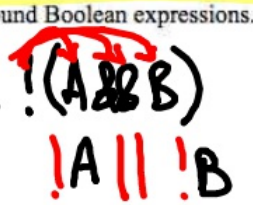


2. There are many identities that have been developed to work with compound Boolean expressions. Two of the more useful identities are DeMorgan's Laws, which are used to negate compound Boolean expressions.

   DeMorgan's Laws:

   not (A or B) = not A and not B

   not (A and B) = not A or not B



   The symbols A and B can take on the Boolean values, **`true`** (1) or **`false`** (0).

3. Here is the truth table that proves the first DeMorgan's Law.

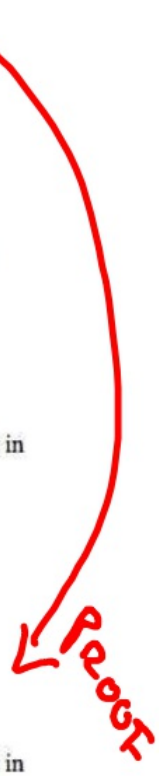
A	B	not (A or B)	not A	not B	not A and not B
1	1	0	0	0	0
1	0	0	0	1	0
0	1	0	1	0	0
0	0	1	1	1	1

   Notice that columns with the titles *not (A or B)* and *not A and not B* result in the same answers.

4. Following is the truth table that proves the second DeMorgan's Law.

A	B	not (A and B)	not A	not B	not A or not B
1	1	0	0	0	0
1	0	1	0	1	1
0	1	1	1	0	1
0	0	1	1	1	1

   Notice that columns with the titles *not (A and B)* and *not A or not B* result in the same answers.



LAST PAGE →

5. Here is a good way to think about both of DeMorgan's Laws. Notice that it is similar to the distributive postulate in mathematics. The not operator is distributed among both terms inside of the parentheses, except that the operator switches from *and* to *or*, or vice versa.

```
not (A and B) = not A  or not B     (!(A && B) == !A || !B)

not (A or B) = not A  and not B     (!(A || B) == !A && !B)
```

C. Application of DeMorgan's Laws

COME OUT ROLL (FIRST ROLL)
CONTINUE TO ROLL UNTIL

1. The casino game of craps involves rolling a pair of dice. The rules of the game are as follows.
   - If you roll a 7 or 11 on the first roll, you win.
   - If you roll a 2, 3, or 12 on the first roll, you lose.
   - Otherwise rolling a 4,5,6,8,9, or 10 establishes what is called the point value.
   - If you roll the point value before you roll a 7, you win. If you roll a 7 before you match the point value, you lose.
   - After the point value has been matched, or a 7 terminates the game, play resumes from the top.

THE POINT

2. The following sequences of dice rolls gives these results:

7	player wins
4 5 3 7	player loses
8 6 2 8	player wins
3	player loses

3. The rules of the game are set so that the house (casino) always wins a higher percentage of the games. Based on probability calculations, the actual winning percentage of a player is 49.29%.

See Handout H.A.15.1, *Craps.java*.

4. A complete program, *Craps.java*, is provided in Handout H.A.15.1. The application of DeMorgan's Laws occurs in the `getPoint()` method. The **`do-while`** loop has a double exit condition.

```
do
{
   sum = rollDice();
}
while ((sum != point) && (sum != 7));
```

5. When developing a conditional loop it is very helpful to think about what assertions are true when the loop will be finished. When the loop is done, what will be true?

① THINK WHAT SHOULD BE TRUE OF L.C.V. POST LOOP

6. When the loop in `getPoint` is done, one of two things will be true:

   a. the point will be matched (`sum == point`).

   b. or a seven has been rolled.

   These two statements can be combined into one summary assertion statement:

```
((sum == point) || (sum == 7))
```

7. The loop assertion states what will be true when the loop is done. Once you have established the loop assertion, writing the boundary condition involves a simple negation of the loop assertion.

8. Taking the assertion developed in Section 6, the negation of the assertion follows.

```
!((sum == point)  ||  (sum == 7))
```

9. Applying DeMorgan's law results in

```
(!(sum == point)) && (!(sum == 7))
```

   Then negate each half of the expression to give

```
(sum != point) && (sum != 7)
```

10. Looking at the first half of the developing boundary condition, the statement `(sum != point)` means that we have not yet matched the point, keep rolling the dice.

11. The second half of the boundary condition `(value != 7)` means we have not yet "crapped" out (rolled a 7). Keep rolling the dice.

12. You can use the equivalent Boolean expressions in items 8 and 9 interchangeably. Choose the one that you think makes your code easier to read.

D. Generating Random Values

1. The casino game of craps involves rolling a pair of dice to generate random values within a specified range. Java has a class named `Random` that makes it easy to get seemingly random numbers into your program. Note: this class is NOT in the AP subset and another way to generate random values will be introduced later. It's a static method - `Math.random()`. ← SOON

2. To use the capabilities of the `Random` class, you must first import the class as follows:

USE FOR LAB 15.1 →

```
import java.util.Random;
```

3. A `Random` object can then be constructed with a statement like

```
Random die = new Random();
```

and then use the `nextInt` method.

```
/** From the Random class
 *
 * Returns a pseudorandom, uniformly distributed int
 * value between 0 (inclusive) and the specified value
 * n (exclusive), drawn from this random number
 * generator's sequence.
 */
public int nextInt(int n);
```

.nextInt(6)+1;

(Note that because of the exclusion on the upper boundary mentioned in the above comment, `nextInt(n)` returns a random integer in the range, 0,1, ….n-1.)

4. For example, you can get a number from 1 to 6 with an expression like:

```
// The message randomly returns 0, 1, 2, 3, 4, or 5.
// A die is simulated by adding 1 to give a value
//   in the range 1 - 6.
int dieRoll = die.nextInt(6) + 1;
```

5. The `Random` class contains additional methods for generating random **`longs`**, **`doubles`**, **`booleans`**, etc. Please consult the Java documentation for a complete explanation of its capabilities.

SUMMARY/ REVIEW: Conditional loops are some of the hardest pieces of code to write correctly. The goal of this lesson is to have a structured approach for the construction of conditional loops.

ASSIGNMENT: Lab Exercise L.A.15.1, *Rolling*.