STUDENT LESSON

Boolean Algebra and 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

**VOCABULARY:** ASSERTION BOOLEAN ASSERTIONS

BOOLEAN ALGEBRA DE MORGAN'S LAWS

**DISCUSSION:** A. Negations of Boolean Assertions

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.

1. 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 ! operator to A.

|  |  |
| --- | --- |
| **A** | **!A** |
|  |  |
| 5 == *x* | 5 != *x* |
| *x* < 5 | *x* >= 5 |
| *x* >= 5 | *x* < 5 |

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

**if** (!(*x* < 5))

// do something...

can be rewritten as

**if** (*x* >= 5)

// do something ...

This is important because we understand positive statements much more easily than statements that contain one or more !s.

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 fundamental operands and operators:

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

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

(Note: Java has other Boolean operators, such as ^ (XOR – “exclusive or”) and equivalence. This curriculum does not cover these other operators because they are not part of the AP subset.)

2. There are many identities that have been developed to use 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:

!(A || B) -> ! A && ! B

!(A && B) -> ! A || ! B

The symbols A and B represent the Boolean values, **true** or **false**.

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** |  | **!(A||B)** | **!A** | **!B** | **!A&&!B** |
|  |  |  |  |  |  |  |
| true | true |  | false | false | false | false |
| true | false |  | false | false | true | false |
| false | true |  | false | true | false | false |
| false | false |  | true | true | true | true |

Notice that columns with the titles *! (A || B)* and *! A && ! B* result in the same answers.

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** |  | **!(A&&B)** | **!A** | **!B** | **!A||!B** |
|  |  |  |  |  |  |  |
| true | true |  | false | false | false | false |
| true | false |  | true | false | true | true |
| false | true |  | true | true | false | true |
| false | false |  | true | true | true | true |

Notice that columns with the titles *! (A && B)* and *! A || ! B* result in the same answers.

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 through both terms inside of the parentheses, except that the operator switches from *and* to *or*, or vice versa.

!(A && B) -> ! A || ! B

!(A || B) -> ! A && ! B

C. Application of DeMorgan's Laws

1. The casino game of craps involves rolling a pair of dice. The rules of the game are as follows. (Refer to Lesson A6, *Libraries and APIs* if you need to review the Random class)

* 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.

2. The following sequences of dice rolls give 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 A14.1,** *C****raps.java*** | 4. A complete program, *Craps.java*, is provided in Handout A14.1. The application of DeMorgan's Laws occurs in the getPoint() method. The **do-while** loop has a compound 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. In other words, when the loop is done, what will be true?

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 Part 6 above, the negation of the assertion follows.

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

9. Applying DeMorgan's law results in

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

Rewriting each half of the expression gives

(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. In other words, we haven’t won yet.

11. The second half of the boundary condition (value != 7) means we have not yet "crapped" out (i.e., rolled a 7). In other words, we also haven’t lost yet, so we must keep rolling the dice.

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

|  |  |
| --- | --- |
| **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. Thinking in the positive sense is typically easier. Determine what will be true when the loop is done. Negate this condition to stay in the loop. |

**ASSIGNMENT:** Handout A14.1, *Craps.java*

Worksheet A14.1, *Boolean Algebra*