EECE.2160: ECE Application Programming

Spring 2018

Programming Assignment #3: A 1-Bit Boolean Calculator

Due **Tuesday, 2/20/18**, 11:59:59 PM

1. Introduction

In this assignment, you will work with C conditional statements to implement a simple Boolean calculator program. This program will also introduce you to Boolean operators, which we will explore in much more detail later this semester.

2. Deliverables

Submit your source file by uploading it directly to your Dropbox folder. Ensure your source file name is ***prog3_boolean.c***. You should submit only the .c file. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

3. Specifications

Input: Your program should prompt the user to enter a simple Boolean expression of the form *a op b*, where *a* and *b* are 1-bit operands and *op* is one of the following operators: `&` (AND), `|` (OR), `^` (XOR). See Section 6: Boolean Operations for more details on these operators and their desired behavior. Examples include:

- `1 & 1`
- `0 | 0`
- `1 ^ 0`

Output: Given a valid expression, your program should calculate the result and reprint the entire expression as well as its result. For example, the expressions listed above will produce the following output:

- `1 & 1 = 1`
- `0 | 0 = 0`
- `1 ^ 0 = 1`

NOTE: Your program should not print the result of an expression if any error occurs. See the next page of the assignment for details on possible input errors.

See Section 5: Test Cases for more sample program runs.

Error checking: Your program should print a descriptive error message under any of the conditions below. For examples of valid error messages, see Section 5: Test Cases:

1. Any of the inputs are incorrectly formatted and therefore cannot be read correctly using `scanf()`
   - **Your error message should include the number of input values that were entered correctly—for example:**

     ```
     Error: 2 values entered correctly
     ```

   - `scanf()` returns the number of values correctly read, which you can store in a variable to test later. Say you have the following line of code:

     ```
     nVals = scanf("%d %d %d", &v1, &v2, &v3);
     ```

     If the user enters:
     - `1 2 3` → `nVals == 3`
     - `1 2 a` → `nVals == 2` (`'a'` is not part of a valid `int`)
     - `1.2 2 3` → `nVals == 1` (`'.'` is not part of a valid `int`, but `1` is read correctly)
     - `X1 2 3` → `nVals == 0` (`'X'` is not part of a valid `int`)

2. Either (or both) of the operands cannot be represented as a single bit (0 and 1 are the only valid 1-bit values)
   - **Your error message should print the invalid input(s)—for example:**

     ```
     Error: first input (3) requires > 1 bit
     ```

3. The operator entered is not a valid operator
   - **Your error message should print the invalid operator—for example:**

     ```
     Error: invalid operator X
     ```

NOTE: If the inputs are correctly formatted (i.e., `scanf()` can read all input values), then your program may generate multiple error messages! For example, if the user enters the expression `3 X 4`, then your program should print:

```
Error: first input (3) requires > 1 bit
Error: second input (4) requires > 1 bit
Error: invalid operator X
```

If `scanf()` cannot read all input values, your program should only print the error message related to a formatting error. For example, if the user enters the expression: `X 3 4`, then your program should print:

```
Error: 0 values entered correctly
```

4. Hints and Tips

Using bitwise operators: The `&`, `|`, and `^` symbols aren't operators just designed for this program—they're valid, built-in operators in C. You can use them in an expression just as you can use arithmetic operators (`+`, `-`, `*`, `/`). You don't need to write conditional statements to basically implement the truth tables shown in Section 6.

For example, assume your input variables are called `in1` and `in2`, you know the user entered the `&` operator, and you want to assign the result of that operation to a variable called `result`. An inefficient way to handle this operation is through if statements—the code below is an example of what NOT to do:

```
if (in1 == 0 || in2 == 0)
     result = 0;
else if (in1 == 1 && in2 == 1)
     result = 1;
```

The following simple statement produces the exact same result:

```
result = in1 & in2;
```

Unsigned values: Since both input values should only be 0 or 1, you can represent these values using unsigned integers (data type `unsigned int` or simply `unsigned`). Unsigned values are strictly non-negative (0 or positive).

The format specifier used to read and print unsigned values is `%u`. The example code snippet below demonstrates the use of this format specifier—it declares two unsigned integers, prompts for and reads their values, and then prints those values to the screen:

```
unsigned x, y;
printf("Enter two values: ");
scanf("%u %u", &x, &y);
printf("x = %u, y = %u\n", x, y);
```

4. Hints and Tips (continued)

Using if/else if vs. multiple if statements: Remember: in an if/else if statement, only one of the cases is executed, even if multiple conditions are true. For example, in the code below, only the first assignment statement (`z = z + 3`) will be executed, even though the first two conditions are both true:

```
int x = 1;
int y = 2;
if (x == 1)
     z = z + 3;
else if (y == 2)
     z = z - 10;
```

If you want both assignment statements that change z to be executed, use multiple if statements without the else:

```
int x = 1;
int y = 2;
if (x == 1)
     z = z + 3;
if (y == 2)
     z = z - 10;
```

5. Test Cases

Your output should match these test cases exactly for the given input values. I will use these test cases in grading of your program, but will also generate additional cases that will not be publicly available. Note that these test cases do not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.

I've copied and pasted the output from each test case below, rather than showing a screenshot of the output window. User input is underlined in each test case, but it won't be when you run the program.

Test Case 1:

```
Enter Boolean expression: 1 & 0
1 & 0 = 0
```

Test Case 2:

```
Enter Boolean expression: 0 ^ 1
0 ^ 1 = 1
```

Test Case 3:

```
Enter Boolean expression: 1 | x
Error: 2 values entered correctly
```

Test Case 4:

```
Enter Boolean expression: 4 + 1
Error: first input (4) requires > 1 bit
Error: invalid operator +
```

Remember, if you are using Visual Studio, to get your program to terminate with a message saying, "Press any key to continue …", use the **Start Without Debugging** command (press Ctrl + F5) to run your code.

If you need to insert extra code at the end of your program to get that program to pause when executing (for example, an infinite loop or the system("pause") function), please remember to comment out or remove that code prior to submitting your program.

6. Boolean Operations

The C language supports three binary Boolean operations: AND, OR, and XOR. These operations use the binary operators shown below. (Remember that a binary operator works with two values.)

Boolean operation	C bitwise operator	Example expression
AND	`&`	`X & Y`
OR	`|`	`A | 0`
XOR (exclusive or)	`^`	`1 ^ B`

Since this assignment deals only with 1-bit values, you only need to understand how these operations handle the values 0 and 1. The table below—called a truth table—shows the outcome for all three operators on all possible pairs of 1-bit values:

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Remember, you do not need to write conditional statements to implement these truth tables—simply use the appropriate operator (&, |, ^) with your input variables.