Boolean Logic Module

Boolean logic involves values being either true or false. This can be useful in programming because you can make a program run a certain way if a value is true or a different way if it is false. In the C language, false is represented by the number zero (0) and true is typically represented by the number one (1). However, any non-zero number is considered true in C as well, so positive or negative numbers can represent true. There are no boolean variables in C, but since numbers can represent true and false, you can use the int variables to represent a boolean variable. If you assign a value of zero to an int variable you can consider it false. If you assign a non-zero integer value to an int variable you can consider it true.

Assigning an integer value is one way to set a variable true or false. You could also assign it the result of a boolean expression. A boolean expression is an expression that results in either true or false. They are similar to arithmetic expressions like “2 + 2” or “3 * (4 - 5)” except their result represents true or false instead of a number. For example: “5 < 6” and “5 >= 6”. Written in words, these expressions are “5 is less than 6” and “5 is greater than or equal to 6”. The first expression “5 < 6” is true since 5 is less than 6. The second expression “5 >= 6” is false since 5 is neither greater than nor equal to 6. Here are examples of these boolean expressions in a program with their result printed to the LCD.

```
#include <pololu/3pi.h>

int main()
{
    // print result of "5 is less than 6"
    print_long(5 < 6);
    while(1);
}
```

Figure 1: true_expression.c. Prints the result of the boolean expression “5 < 6”.

```
#include <pololu/3pi.h>

int main()
{
    // Print result of "5 is greater than or equal to 6"
    print_long(5 >= 6);
    while(1);
}
```

Figure 2: false_expression.c. Prints the result of the boolean expression “5 >= 6”.
Figures 1 and 2 show boolean expressions mentioned in the previous paragraph and the result of evaluating the expressions. Since booleans are represented by numbers, print_long could be used to print the result of the boolean expressions. Figure 1 shows the result of "5 < 6", which is 1 because it is true. Figure 2 shows the result of "5 >= 6", which is 0 because it is false.

*Subsection Divide* (Relational Operators)

The symbols ‘<’ and ‘>=' represent “less than” and “greater than or equal to”, respectively. These are called relational operators, which are like the arithmetic operators ‘+’, ‘-’, ‘*’, and ‘/’ except they are purposely used for boolean expressions. Here is the list of all the relational operators.

**Relational Operators**
- < - “less than” - will evaluate as true if the left operand is less than the right operand.
- <= - “less than or equal to” - will evaluate as true if the left operand is less than or equal to the right operand.
- > - “greater than” - will evaluate as true if the left operand is greater than the right operand.
- >= - “greater than or equal to” - will evaluate as true if the left operand is greater than or equal to the right operand.
- == - “equal to” - will evaluate as true if the left operand is equal to the right operand.
- != - “not equal to” - will evaluate as true if the left operand is not equal to the right operand.

Note that the “equal to” operator uses 2 equal symbols (=) because a single equal symbol signifies assignment of a value.

Relational and arithmetic operators can be used in the same expression to make a boolean expression. For example, the expression “(3 + 3) < 4” uses the plus and less than operator. This expression also uses the parentheses to carry out the 3 + 3 first before being compared against the 4. This expression would evaluate to false since after adding 3 + 3 you get 6 and 6 is not less than 4.

*Subsection Divide* (Logical Operators)

There is another set of operators commonly used in boolean expressions which are called logical operators. Logical operators use boolean values as operands and depending on the operand values the boolean expression may result in true or false. Here is a list of the logical operators and what their result may be.

**Logical Operators**
- && - “and” - will evaluate as true only if the left operand and right operand are true.
• || - “or” - will evaluate as true if either the left or right operand are true.
• ! - “not” - precedes, goes before, a single operand. Reverses the operand, so if the operand is true the result will be false and if the operand is false the result will be true.

Here are examples of each logical operator in use.

```c
#include <pololu/3pi.h>

int main()
{
    // Print result of boolean expression with and operator
    print_long((5 < 5) && (5 >= 6));
    while(1);
}
```

Figure 3: and_operator.c. Prints result of boolean expression with and operator.

```c
#include <pololu/3pi.h>

int main()
{
    // Print result of boolean expression with or operator
    print_long((5 < 5) || (5 >= 6));
    while(1);
}
```

Figure 4: or_operator.c. Prints result of boolean expression with or operator.
Figure 5: not_operator.c. Prints result of boolean expression with not operator.

Figure 3 shows use of the and operator. The first operand “(5 < 6)” is true but the second operand “(5 >= 6)” is not true. Since both of the operands are not true, the use of the and operator makes the result false. Figure 4 shows the use of the or operator. Since one of the operands are true, even though the second operand is false, the use of the or operator makes the result true. Figure 5 shows the use of the not operator. Since the operand is true, the result is reversed and results in false.

The int data type can be used as a boolean variable. Here is an example of assigning boolean values to variables.

```
#include <pololu/3pi.h>

int main()
{
    // Print result of boolean expression with not operator
    print_long(!((5 < 6)));
    while(1);
}
```

Figure 6: boolean_variable.c. Use of boolean variable.

In figure 6, two int variables are created then used in a boolean expression. The value of the expression is assigned to an int variable and then the value of it is printed. Since the expression is true, a 1 is printed.