I. Boolean (Logical) Operations

1. Review of Binary logic

Three basic logical operations are commonly used in binary logic: 'and', 'or', and 'not'. Table 1 lists these operations plus one common compound operation (xor). The table includes the MATLAB syntax for carrying out these four operations and one syntax for writing them out (there are many alternatives). These operations can be used individually or in combination to yield a desired logic. In digital circuits these operations are carried out by "logic gates" that can be wired together to provide an amazing variety of processes. This first part of the lab will look at using these operations in MATLAB.

Table 1: Four Basic Logical Operations

Name	Function	MATLAB operator	A Handwritten
and	true if both connected statements are true	A & B	A • B
or	true if either connected statement is true	A | B (i.e., shift \)	A + B
not	makes a true statement false or vise-versa	~A	$\bar{A}$
xor	true if A & B differ, false if both are the same	xor(A, B)	A ⊕ B

Based on the above definitions fill out the "Truth Table" (Table 2) for these operations.

Note: The second row has been filled out to help you (the first row is the header/title row). In the second row, both A and B are 'turned on' (have values of 1). For the ~A column, the column reads as "Not A" meaning the solution will be 1 if A is not "turned on" and will be 0 if A is "turned on". For the A or (|) B column, the column reads as "A or B" meaning the solution will be a 1 if either A or B (or both) are turned on and the solution will be a 0 if neither are turned on. For the A and (&) B column, the column reads as "A and B" meaning the solution will be 1 if both are turned on and will be a 0 if either A or B are turned off. Lastly, the xor(A,B) column reads as "A differs from B" meaning the solution will be 1 if A and B are different and will be 0 if A and B are the same.

Table 2: Truth Table for basic Logical Operations

A	**B**	**~**A	A **or (|)** B	A **and (&)** B	**xor(**A,B**)**
True(1)	True(1)	0	1	1	0
True(1))	False(0)				
False(0)	True(1)				
False(0)	False(0)				

2. **Creating logical combinations (using Bboard.m)**

These basic logical operations can be combined to create a more complex logic. This is particularly necessary when there are more than two initial logical variables (A and B in the table above are logical variables). When making combinations it is important to pay attention to the logic "order of operations" shown in Table 3. The logical and relational operators are bolded. Notice that NOT has the highest precedence, before common mathematical operations. At the bottom of the table is the AND followed by OR. Parentheses are often included even when not essential to make the combination clear.

Table 3: Logic Order of Operations (highest to lowest)

	Name	**Symbol(s)**
1	Parenthesis	( )
2	**logical Not** and Unary minus (neg. number)	~ -
3	Multiplication/Division	/ *
4	Addition/Subtraction	+ -
5	Colon operator	:
6	**Relational operators**	< > ==
7	**Logical AND**	&
8	**Logical OR**	|

a) Run the Bboard program (Available on the website). To run this program, both Bboard.m and Bboard.fig must be in the current directory. In the command window type: >> Bboard
This will launch the interface shown in Figure 1. (Do not launch Bboard.fig directly, it will produce an interface that will not work. Bboard.m is the actual program.)

The three input buttons on the right can be pushed to switch between 0 (off or FALSE) and 1 (on or TRUE). The current values of the buttons are stored in the variables a, b and c as shown on the interface. Each Output light is controlled by the expression in the Logical Expression box to its right (initially labeled with the nearest button). For all Bboard exercises use only logical operators (i.e., do not use mathematical operators +, * …)

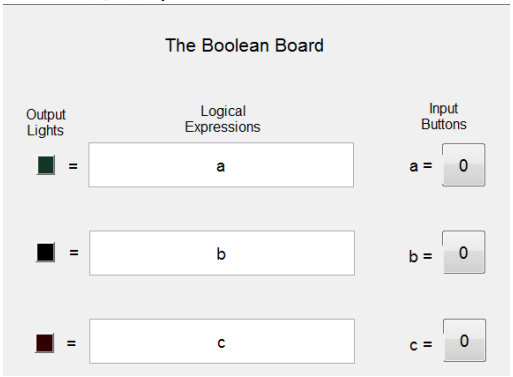


Figure 1: Start up layout of the Bboard.m program. Pressing the Input Buttons on the right will toggle their value between 0 and 1. Logic expressions for each Output Light can be written in the Logical Expression boxes.

Important Note: The lights on the left are not paired with the input buttons on the right. This can be visually confusing. Turning on the 'a' input button does not automatically turn on the first light. The first light is controlled by the first logical expression. Turning on the 'a' input button will only turn on the first light if the first logical expression says it should.

b) Try out the interface: Spend some time getting an idea how the interface works. Without changing the logic click on the input buttons and observe what happens. Now try some different logic in each logical expression box (e.g. type "a & b" in the first box "a | c" in second box and "xor(b, c)" in the last box. Then try various button positions).

c) **Assignment Problem 1:** Bboard Challenge: Setup the logic in the boxes so:
1) Bottom light: on if any number of the buttons are on (i.e., = 1)
2) Top light: on if and only if all three buttons are on
3) Middle light: on if any two buttons or more are on.

Try out several cases to show the interface is working. As a test case to turn in, switch b off and the other two on, copy the Bboard interface to a Word document using the Alt-PrintScrn keyboard function (press the Alt and PRT SC keys simultaneously to copy interface to the clip board and then paste into word). You will be adding more to this document from the next part of the lab.

3. Truth Tables

Making a table that shows the possible outcomes for all possible inputs is often helpful in understanding a given logical expression. These are known as Truth Tables and one was created in filling in Table 2. This section will a) walk through creating the Table 2 Truth Table in MATLAB and b) then examine making a truth table for combinations of the basic operators.

a) Creating a truth table in MATLAB for the basic operations

MATLAB's ability to do operations on vectors can facilitate making the Truth Table easily. Steps needed:

a. Setup Input Variables as Column Vectors: Create two variables A & B that follow the pattern of A and B in Table 2 (use the numbers 0 and 1). Make sure A & B are column vectors. Transpose if necessary (e.g. if A is a row vector, then use A = A' to make it a column). For example, coding variable A would look like A = [1 1 0 0]'; Now you create vector B.
b. Create an array of inputs & calculations: Combine the input variables and the various calculations (shown in the column titles of Table 2) into a single row vector. This can be done by enclosing the various input and result vectors in square brackets to make them a single array.

Example: cals = [A, B, ~A, A|B, A&B, xor(A,B)]

Notice that I didn't suppress this line of code (add a ; at the end) because it's my final answer, so I want MATLAB to print the results in my command window. Run your script. Compare these results to the hand calculations in Table 2. They should match.

⇨ **Show:** this Truth Table to your instructor or assistant.

4. **Assignment Problem 2** Creating a truth table for a combination of logical operators

Overview: Use the following steps to create a Truth Table for the middle light logic in Challenge 1 (Middle Light Challenge: on if any two buttons or more are on). Write your code within a script and include: 1) a line that calculates the number of rows required in the truth table (explained below) and 2) commands to create the truth table.

Detailed Steps:

a. Number of cases: First it is a good idea to determine the number of possible arrangements of the input variables. This will indicate the correct number of rows in the truth table. This number is a function of the number of input variables in the expression. For true or false logic variables, simply raise two to the power of the number of input variables in the expression (e.g. rows = $2^{\#\text{ of inputs}}$, for Table 2 the number of rows = $2^2 = 4$). What would it be for the Boolean Board logic with three input buttons, Include this calculation in your script.
b. Setup Input Variables as Column Vectors: Now create input vectors (or an array) covering all possible combinations of the input. One way to do this:
   For the first variable, make the first half of the rows 1 and the second half 0.
   For the second variable, start half as many rows of 1 as the first variable (i.e, the first $1/4^{th}$ of the rows will have zeros in this case), then add an equal number of zeros (i.e. put zeros in the second $1/4^{th}$ of the rows. This will fill in the first half of the rows, the same pattern will need to be repeated in the second half of the rows. For example: We'll have 8 total rows for this problem so the second variable should start with B = [1 1 0 0…] The last variable should be alternating 1 and 0 until full.
c. Create an Array of Inputs and Calculations: As before create a column vector carrying out the desired calculation by using square brackets. Don't forget to put the input variables as the first three rows

like we did in the practice exercise. Hint: Something that might make the coding easier is to create a variable equal to the logical expression and that write the variable in the column vector (will make the column vector code less messy).

This should result in a four-column table: three input columns and the resulting logic.

5. Notes on Logical Expressions in MATLAB (vectors and numbers).
   - Notice that in these truth table calculations the logic has been done between two vectors. For vector (or matrix) logic calculations the two vectors/matrices must have the same dimensions.
   - In most programming languages logical operators can only be used between logical variables (i.e., variables restricted to the values of 0 or 1). MATLAB allows the use of numerical variables in logical expressions. It automatically converts any numeric variable to a logical one where any nonzero number is converted to a logical 1; zero values become logical zero (see help for the logical function).
   - The ***any*** and the ***all*** functions can be handy. Look up their help file (e.g., >> help any or >> help all).

II. Relational Operators in MATLAB

Table 4 summarizes the basic relational operators used in MATLAB. These operations will test the condition and result in a 1 (if true) and a zero (if false). These operators are quite similar to what you are used to but have some minor differences to accommodate their use in programing. Notice that to test for equality two equal signs in a row are needed; a single equal sign would be an assignment and MATLAB would attempt to store the result of the right-hand side in a variable on the left hand side. This will often result in an error.

Table 4: Relational Operators

<	less than	>	greater than
<=	less than or equal	>=	greater than or equal
==	Equal	~=	not equal

Try the examples at the command line. Relational Operators applied to different classes of variables:

1. Two scalars ➔ results in a single value, 0 if false, 1 if true, resulting variables are of class "logical"
   e.g. >> x = (7 < 6) ➔ x = 0 (false)
2. A scalar and a vector (array) ➔ results in a logical vector(array) of the same size as the original vector (array), compares the scalar to each value in the vector and returns a true or false for each comparison
   e.g., y = (2 >= [1 2 3]) ➔ y = [1 1 0] (true, true, false)
3. Two vectors of the same size ➔ results in a logical vector of the same size as the two vectors compares the two vectors element by element (two arrays will work similarly)
   e.g. z = ([1 2 3] ~= [3 2 1]) ➔ z =[1 0 1] (true, false, true)
4. Text strings
   a. text strings can be defined by enclosing in single quotes (literals)
   b. text strings are vectors of type char where each character is a separate element
   c. work like other scalars and vectors except ASCII value is compared

   e.g. h = ('D' == 'Dons') ➔ h = [1 0 0 0] (true, false, false, false)

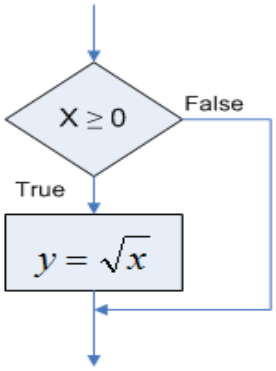
III. Conditional Statements

One of the key tools in procedural programming is the conditional statement (an "if" statement) that allows a program to "branch" and do different things depending on the value of a variable. "if" statement can be used to do something only if a given condition is true. Logical and relational operators are used to construct conditions that provide the test for these statements. The section below shows how the basic if/else structure works. Setup the function requested and try the different commands. Notice how the logic is represented by a "flowchart" in the diagrams to the right of the text.

Set up a function with the function definition line: function y = test(x)

1. Simple conditional (if/then) statement:
   a. English: If *x is greater than or equal to zero* then *y equals the square root of x.*
   b. MATLAB example (***type the box into your function and test for x = 2, x= 0 & x -2***)

```
if x >=0
        y = x^0.5;
end
```

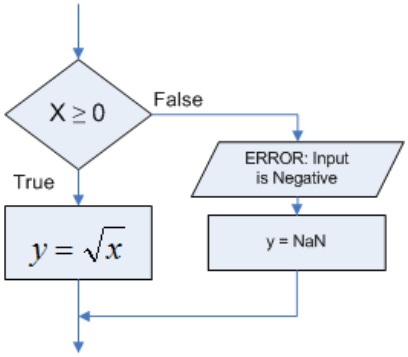


   c. MATLAB general structure

```
if logical statement
    Statement Group (executed if logical statement is true)
end
```

2. if/else structures
   a. English: If *x is greater than or equal to zero* then *y equals the square root of x* or else *display an error and set y to NaN*
   *Note: NaN in Matlab means "Not a Number"
   b. MATLAB example (***try with same tests as above***)

```
if x >=0
        y = sqrt(x);
else
        disp('input is negative')
        y = NaN;
end
```



   c. MATLAB general structure

```
if logical statement
        Statement Group (executed if logical statement is true)
else
        Statement Group (executed if logical statement is false)
end
```

IV. Applying Conditional Statements

1. **Assignment Problem 3:** A make even program (with setup)

Figure 1 gives the setup for a program to force an integer to be an even number. If the input integer is even the program should return it without change. If the input integer is odd the program should return the next number higher.

a. **Flowchart:** Complete the ***Flowchart Basics Tutorial*** available on the website – Uses MS Visio software which is available in the studio labs. With it you will create the flowchart for this problem.

b. **Code & Validation:** Develop and validate a function that matches this setup. Notice that the test cases cover all possible branches of the program for both positive and negative numbers. Use these test cases.

Problem ID _Make Even Function _____Programmer _S. Moor _____

Set Up/ Planning Type of Program: ☐ Script ☑ Function

1. Problem Statement:

Develop a function that will have an output of an even integer. If an integer is input then it is simply output as is. If an odd integer is input the program will add one and return the resulting integer.

2. Inputs: (full name, variable to be used, units)

Variable Name	Description	Units or Values	Input Source*
x	Input Value	Must be Integer	Command Line

3. Output: (full name, variable to be used, units)

Variable Name	Description	Units or Values	Output type*
x	Even output value	Integer	Command Line

4. Solution Steps (order of these two parts may be varied):

(1) Perform calculation on test case(s) (2) Identify the steps/equations to be used in code

Input	Output
2	2
3	4
-4	-4
-3	-2

How to check if a number is even:

- The rem(a, b) function determines the remainder of the division a/b
- therefore rem(x,2) == 0 will yield a logical 1 if x is even and a logical 0 if it is odd.

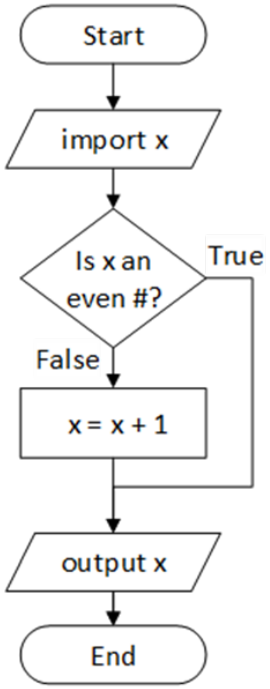


Figure 2: Above is the setup for creating a program that forces the any integer to be an even number.

2. Greater than and less than (**Read carefully Extremely important**): Testing if a variable is between two bounds takes some special attention because the approach in a program is slightly different from how it is done in algebra. If we want to express that a variable x should be between 3 and 10 in algebra you might write:

$$3 < x < 10$$

This algebra notation will not work the same in MATLAB. Try the following example at the command line in MATLAB:

```
>> x =12;
>> 3 < x < 10
```

If you type the following at the command line in MATLAB the program will return a 1 (i.e., be true). This is because MATLAB will evaluate the two conditions in succession. First, MATLAB will determine that 3 is less than x yielding a 1. Next MATLAB will evaluate the less than condition 1 < 10, which it will determine to be true resulting in 1.

In MATLAB, the way to test if x is between 3 and 10 is to use the following code (try it in the command window):

```
>> 3 < x & x < 10
```

With this code MATLAB will find the first condition to be true and the second to be false. Because they are connected by an ***and*** operator, the overall expression will be deemed false.

3. **Assignment Problem 4:** QC program: A part must have a diameter between 9.95 and 10.05 mm. A function is to be created that will take the actual measured diameter of a part as a command line input and return a 1 if it is between these two specifications (inclusive of the limits) and a 0 if it is not. This program will eventually be part of a larger program on the production line. It will receive its input from a laser micrometer on the line and output the result to an automatic sorter.
   a. Complete the setup sections (i.e., page 1) of the Program Development worksheet. Be sure to complete adequate hand calculation test cases in step 4.
   b. Prepare prototype code to accomplish this task in MATLAB
   c. Validate this code using test cases that test all branches of any conditional statement(s).

Assignment Summary **Due: Next Lab**

Boolean Logic – Bboard Challenge

1. Challenge: Using only logical operators, set up the logic on the Bboard interface so that:
   1) Bottom light: on if any number of the buttons are on (i.e., = 1)
   2) Top light: on if and only if all three buttons are on
   3) Middle light: on if any two buttons or more are on.

   Turn in: A screen capture of the Bboard with this setup

2. Truth Table: Write a commented script to create a truth table for these cases in the Bboard Challenge. The truth table and Bboard should match.
   Turn in: The script & the execution result.

Conditional Programs

3. Make even Program (based on Figure 2)
   Write a function that will carry out the logic in the Figure 2 flowchart for integer input.
   Turn in: a. The flow chart for this program created in Visio using techniques from the ***Flowchart Basics Tutorial*** available on the website.
   b. Complete commented code (step #5) and validation (step #6). Use provided test cases for the validation.

4. QC Program: Develop and Validate the QC prototype function outlined in section IV.3 above.
   Turn in: A complete Function Development worksheet for the setup, coding, and validation

MATLAB Review Sheet- Intro To Conditional Statements

Functions: A Basic structure of programing (review)

Function Def. Line:	function [a, b] = name(x,y)
Function Call:	[e, f] = name(x1, x2)
Some Keys:	Keep definition line name = the file name
	Do not reuse same names, Watch your directories

Conditionals:

	Name	Function	MATLAB	Handwritten
Logical Operators	and	true if both connected statements are true	A & B	A • B
	or	true if either connected statement is true	A | B	A + B
	not	makes a true statement false or vise-versa	~A	$\bar{A}$
	xor	true if A & B differ, false if both are the same	xor(A, B)	A ⊕ B

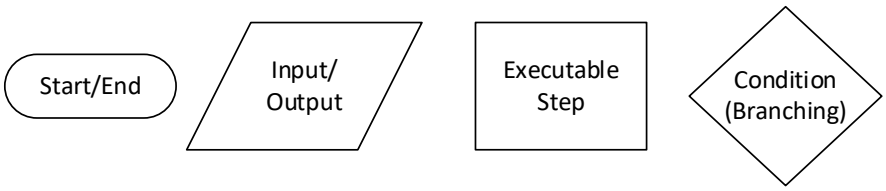
Relational Operators	<	less than	>	greater than
	<=	less than or equal	>=	greater than or equal
	= =	Equal	~=	not equal

Number of cases (rows) in a truth table = $2^{\#\text{ of input variables}}$

Basic if statements (branching)

if *logical statement*
Statement Group (executed if logical statement is true)
elseif *logical statement*
Statement Group (executed when previous statements are false and the current logical statement is true). An if structure may have any number of elseif statements.
else
Statement Group (executed if all previous logical statements are false). May only be used once. Does not include a logical statement.
end

Flowchart symbols:



Flowchart Direction: Flow charts should clearly organized and generally proceed top to bottom (sometimes they can run left to right). Avoid reversing directions except in loops.