Armstrong State University
Engineering Studies
MATLAB Marina – If-Else Statements Primer

Prerequisites

The If-Else Statements Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, scripts, variables, arrays, and logic expressions. Material on these topics is covered in the MATLAB Marina Introduction to MATLAB module, MATLAB Marina Variables module, MATLAB Marina Arrays module, and MATLAB Marina Logic Expressions module.

Learning Objectives

1. Be able to use if-else statements to selectively execute blocks of statements.
2. Be able to use if-else statements for inline error handling.

Terms

logical true, logical false, logic expression, test, condition, inline error handling

MATLAB Functions, Keywords, and Operators

if, elseif, else, end

If-Else Statements

Conditional program structures (`if-else`, `switch`) allow programs to selectively execute blocks of statements depending upon the result of a logic expression. By default, all program statements are executed sequentially. The simple `if-else` statement has the general form shown in Figure 1a for no else block or Figure 1b. The compound `if-else` statement (`if-elseif-else`) has the general form shown in Figure 1c.

```
if (logic expression)
    statements to be executed if logic expression true
end
```

Figure 1a, General Form of Simple if-else Statement (no else block)

```
if (logic expression)
    statements to be executed if logic expression true
else
    statements to be executed if logic expression false
end
```

Figure 1b, General Form of Simple if-else Statement

The simple `if-else` is characterized by a single logic expression and one or two blocks of statements to selectively execute. The compound `if-else` (`if-elseif-else`) is

characterized by two or more logic expressions and generally three or more blocks of statements to selectively execute. There can be an unlimited number of `elseif` blocks in the compound `if-else` statement. The `else` block is optional for both the simple and compound versions of the `if-else` statement and does not have a separate logic expression. Note that `if-else` statements must be terminated with the `end` keyword.

```
if (first logic expression)
    statements to be executed if first logic expression true
elseif (second logic expression)
    statements to be executed if second logic expression true
else
    statements to be executed if both logic expressions false
end
```

Figure 1c, General Form of Compound if-else Statement

For `if-else` statements, the statements in the `if` and `elseif` blocks are only executed if the corresponding logic expression for the block is true. The block of statements for the `else` block is executed only if all logic expressions in the `if` and `elseif` portions evaluate to false. If the `else` block is omitted, no statements are executed when the logic expressions in the `if` and `elseif` portions evaluate to false.

The logic expression(s) (tests, comparisons) for `if-else` statements should result in scalar Boolean results (true or false) and are typically a combination of relational and logic operators although in some cases mathematical operations are also part of the comparison. Some examples of legal logic expressions for `if-else` statements are: a Boolean constant, a variable containing a Boolean value, a logical or relational operation on two scalars, a compound logical expression resulting from the logical AND or OR of two or more logic expressions, or a MATLAB function that returns a Boolean value.

The MATLAB programs of Figure 2a and 2b use `if-else` statements to determine the larger of two numbers.

```
num1 = input('Enter number 1: ');
num2 = input('Enter number 2: ');
if (num1 > num2)
    disp('num1 is greater than num2')
end
if (num1 < num2)
    disp('num1 is less than num2')
end
if (num1 == num2)
    disp('num1 is equal to num2')
end
```

Figure 2a, MATLAB Program to Determine Larger of Two Numbers

The program of Figure 2a uses three simple `if-else` statements with no `else` blocks. Each of the `if-else` statements tests for one of three possible cases and executes the block of code contained if the test is true. The program of Figure 2b uses one compound `if-else` statement to perform the same tests. The `if` and `elseif` blocks handle two of the cases and if these are both false the `else` block is executed.

```
num1 = input('Enter number 1: ');
num2 = input('Enter number 2: ');
if (num1 > num2)
    disp('num1 is greater than num2')
elseif (num1 < num2)
    disp('num1 is less than num2')
else
    disp('num1 is equal to num2')
end
```

Figure 2b, MATLAB Program to Determine Larger of Two Numbers

The program of Figure 2b can be interpreted as follows:

- Two numbers are read from the user and saved in the variables num1 and num2.
- The condition (num1 > num2) is checked. If this condition is true the statements in the `if` block are executed.
- If the first condition is false, the next condition (num1 < num2) is checked. If this condition is true the statements in the `elseif` block are executed.
- If all previous conditions are false, the statements in the `else` block are executed.

The operation of the program of Figure 2b is also shown by the flow chart of Figure 2c.

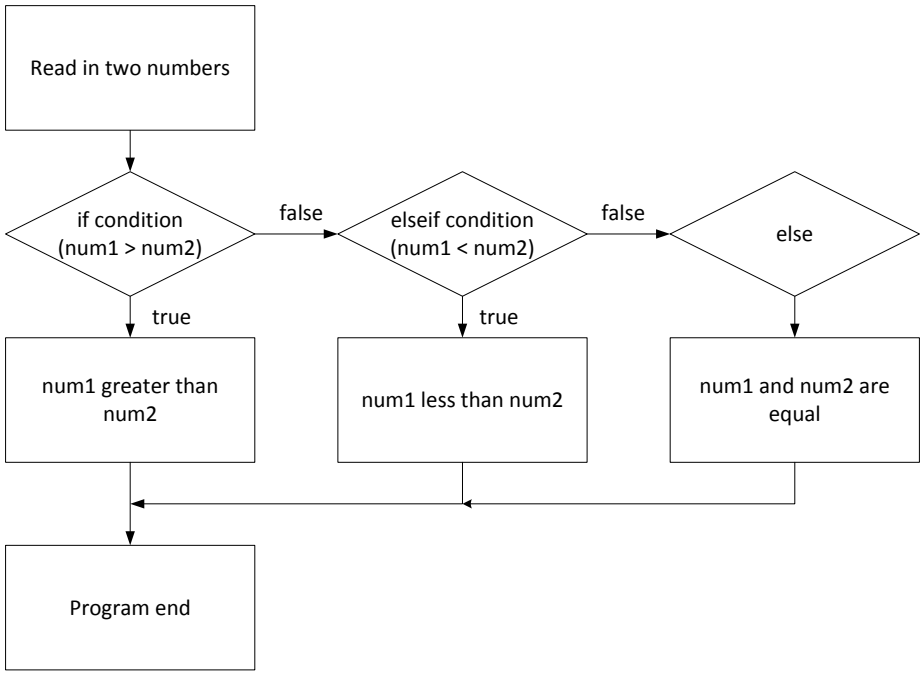

Figure 2c, Flowchart for MATLAB Program to Determine Larger of Two Numbers

In a compound `if-else` statement, only one of the logic expressions should evaluate to true. The MATLAB `if-else` statement is generally used when one has four or less outcomes to choose among. More than four outcomes can often be better handled using a `switch` statement.

Inline Error Handling

An exception is an unexpected error. Exception (error) handling statements can be interspersed with the program statements dealing with the error where it occurs. This is called inline error handling. Inline error handling is relatively straightforward to implement using a conditional statement to check for the error but generally makes the program more difficult to maintain. The MATLAB program of Figure 3a shows a program that performs division and the program of Figure 3b shows how inline error handling can be added in this case to detect and avoid a divide by zero error.

```
% read in two numbers
num1 = input('Enter number 1: ');
num2 = input('Enter number 2 (not 0): ');

% perform division
num3 = num1/num2;
disp(num3);
```

Figure 3a, Program to Perform Division

```
% read in two numbers
num1 = input('Enter number 1: ');
num2 = input('Enter number 2 (not 0): ');

% perform division
if (num2 == 0)
   num3 = NaN;
   disp('divide by zero');
else
    num3 = num1/num2;
end

disp(num3);
```

Figure 3b, Exception Handling Using Inline if-else for Divide by Zero

The statements where the error may occur should be written and tested first without the error handling. Once the statements have been verified to work for the cases except the error cases, then the statements for the inline error handling are added. Depending on how the potential error is checked for, the original operation will be in either the if block or the else block and the

error handling operation will be in the other block. For the program of Figure 3b, the original division operation is in the else block and the error handling operation is in the if block.

MATLAB has a formal exception handling system. When errors are detected, instead of displaying a message an exception can be thrown using the `error` function. If the program does not include `try-catch` blocks to handle the exception, an error message is displayed and the program is exited. MATLAB's exception handling system will be covered in more detail in the MATLAB Marina Exception Handling module.

Nested If Statements

Nested if statements are `if-else` statements that contain another `if-else` statement in one of the conditionally executed blocks of statements. Nested if statements can be used instead of using compound logic expressions involving logically ANDing logical expressions in if statements.

The MATLAB program of Figure 4a determines if a metal rod's length is within an accepted range for the part. The rod's nominal length is 4.5 cm and should be within 0.1 cm of the nominal length to be acceptable.

```
% read in metal rod length
rodLength = input('Enter length of metal rod: ');
% determine if rod's length is within accepted range
if ((rodLength >= 4.4) & (rodLength <= 4.6))
    disp('accept');
else
    disp('reject');
end
```

Figure 4a, rodLength Program Implementation using Compound Comparison

```
% read in metal rod length
rodLength = input('Enter length of metal rod: ');
% determine if rod's length is within accepted range
if (rodLength >= 4.4)
    if (rodLength <= 4.6)
        disp('accept');
    else
        disp('reject');
    end
else
    disp('reject');
end
```

Figure 4b, rodLength Program Implementation using Nested if

The MATLAB program of Figure 4b shows another implementation of the program of Figure 4a using nested if statements instead of a compound logic expression. For this example, the program of Figure 4a is the better choice of implementations.

Nesting structures typically makes programs more difficult to debug and maintain and should only be used if necessary. One common use of nested if statements is to ensure that invalid data is not processed (inline error handling). The program of Figure 4c shows inline error handling added to the program of Figure 4a. The error handling ensures that the rod length is greater than zero.

```
% read in metal rod length
rodLength = input('Enter length of metal rod: ');
% determine if rod's length is within accepted range
if (rodLength >= 0.0)
   if ((rodLength >= 4.4) & (rodLength <= 4.6))
      disp('accept');
   else
      disp('reject');
   end
else
    disp('invalid rod length');
end
```

Figure 4c, rodLength Program with Inline Error Handling

Last modified Tuesday, September 09, 2014