MATLAB Marina: Conditional Statements, if-else Examples

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 1a 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))
    fprintf('accept\n');
else
    fprintf('reject\n');
end
```

Figure 1a. rodLength Program Implementation using Compound Comparison

The program of Figure 1b shows another implementation of the program of Figure 1a using nested if statements instead of a compound logic expression. For this example, the program of Figure 1a is the better choice of implementations; it is easier to follow and the code will be easier to maintain. Nesting structures typically makes programs more difficult to debug and maintain and should only be used if necessary.

```
% 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)
        fprintf('accept\n');
    else
        fprintf('reject\n');
    end
else
    fprintf('reject\n');
end
```

Figure 1b. rodLength Program Implementation using Nested if

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 2a shows a program that performs division and the program of Figure 2b 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;
fprintf('%0.2f / %0.2f = %0.2f\n', num1, num2, num3);
```

Figure 2a. Program to Perform Division

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 2b, the original division operation is in the else block and the error handling operation is in the if block.

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

% perform division
if (num2 == 0)
   num3 = NaN;
   fprintf('divide by zero\n');
else
    num3 = num1/num2;
end
fprintf('%0.2f / %0.2f = %0.2f\n', num1, num2, num3);
```

Figure 2b. Inline Error Handling Using if-else for Divide by Zero

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.

Inline error handling is commonly used to ensure that invalid data is not processed. The program of Figure 3 shows inline error handling added to the rod length program of Figure 1a. The inline error handling is accomplished using nested if statements.

```
% 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))
        fprintf('accept\n');
    else
        fprintf('reject\n');
    end
else
    fprintf('invalid rod length\n');
end
```

Figure 3. rodLength Program with Inline Error Handling

Last modified Monday, September 28, 2020