

MATLAB Marina: Iteration, while loops applications

Student Learning Objectives

After completing this module, one should:

1. Be able to list commonly used iterative operations that use while loops.
2. Be able to determine when to use for loops versus while loops for iteration.
3. Be able solve problems using while loops.

Terms

NA

MATLAB Functions, Keywords, and Operators

sentinel, flag

While Loop Applications

While loops repeat the loop body until the loop control expression is false. The terminating condition can be a count being met, a sentinel being encountered, or a flag being set. Sentinel controlled loops execute the loop body until a specific value is encountered indicating the loop should be terminated. Flag controlled loops execute the loop body until the value of the flag (typically a Boolean) changes. The loop body will have a comparison that if met changes the flag value.

Common uses of while loops include: ensuring valid input, repetitively displaying menus and performing operations based on the selections, performing input of an unknown number of elements (file input), and numerical operations that terminate based on an accuracy or tolerance rather than a set number of terms or computations.

Any operation that could be done with a for loop could also be done using a while loop (often by adding counting variables to the while loop). Generally, if one knows before entering the loop how many times something needs to be done then a for loop is the better choice than a while loop.

Ensuring Valid Input

One common use of while loops is to ensure that data read from a user or file is valid. This is an example of a sentinel controlled loop, where the sentinel is a data value that is valid. The MATLAB code of Figure 1 ensures that the user enters a number greater than zero.

```
% read number greater than zero
number = input('Enter number greater than zero: ');
while (number <= 0)
    number = input('Enter number greater than zero: ');
end
```

Figure 1. while Loop used to Ensure Number is Greater than Zero

The code segment of Figure 1 operates as follows:

- A number is read from the user. This also serves as the initialization of the loop control variable for the while loop.
- The loop control expression, `number <= 0`, is executed. If the number is greater than zero the loop body is not executed and the number read is greater than zero.
- If the number is less than or equal to zero, the loop body is executed, and another number is read.
- This is repeated from the second step until the expression is false meaning the number read is greater than zero.

If the initial read number is greater than zero, the while loop body is skipped over and never executed.

The program of Figure 2 uses the code segment of Figure 1 to ensure an end count is greater than zero.

```
% read end count
endCount = input('Enter number to count to (greater than zero):
');
while (endCount <= 0)
    endCount = input('Enter number greater than zero: ');
end

% display count from one to end count
for count = 1 : 1 : endCount
    fprintf('%d ', count);
end
fprintf('\n');
```

Figure 2. Program to Count from One to End Count

Repeating a Menu Driven Operation

Another common use of while loops is to allow a user to repetitively make choices from a menu and act on them. The program of Figure 3 repeats a menu and generates sine or cosine plots based on the menu choice. This is also an example of a sentinel controlled loop where the sentinel is the value corresponding to the quit menu choice.

The program of Figure 3 operates as follows;

- The menu is displayed, and the initial menu selection is read.
- If the selection is quit (the sentinel), the while loop body is not executed, and the program ends.
- If the selection is other than quit, the while loop body is executed and based on the menu selection either a sine or cosine plot is generated in a new figure window. The menu is displayed again, and a new selection is read.
- This is repeated from the second step until the selection from the menu is quit.

```

% read in plot type using menu
choice = menu('','sine', 'cosine', 'quit');

% generate sine or cosine plots until quit chosen
figureNumber = 0;
while(choice ~= 3)
    % generate trig function according to choice
    t = 0.0 : 0.05 : 1.5;
    if (choice == 1) % sine
        f = sin(2*pi*t);
    else % cosine
        f = cos(2*pi*t);
    end

    % plot selection in new figure window
    figureNumber = figureNumber + 1;
    figure(figureNumber);
    plot(t, f);

    % display menu and read in next choice
    choice = menu('','sine', 'cosine', 'quit');
end

fprintf('Quit\n');

```

Figure 3. Repeated Menu

The menu needs to be displayed and a new menu selection read inside the while loop body to update the while loop control variable. Otherwise, the loop would infinitely generate plots of the same curve for the initial choice.

Running Concatenation

The program of Figure 4 builds (populates) a 1D array of unknown size with values greater than zero. With the size not known beforehand, values are repetitively read in until the sentinel value is encountered. Reading from a file until the end of the file is performed similarly.

The initial array is an empty array. Values are read in and concatenated to the existing array until a negative value is entered. When the negative value is encountered the loop terminates and the array `dataValues` contains all the values read in. If the first value is negative, the array `dataValues` will be empty. The values are saved by concatenating (appending) each new value with the existing array.

```
dataValues = [dataValues, value];
```


```
% create array of non predetermined size
% start with empty array and read in and save values until
% terminating indication entered
dataValues = [];
value = input('Enter value (negative to terminate):');
while (newValue >= 0)
    % concatenate new value to existing array
    dataValues = [dataValues, value];
    % read in next value
    value = input('Enter value (negative to terminate):');
end
```

Figure 4. Program to Build Array of Unknown Size

The program of Figure 4 has a major inefficiency. The array is increasing in size by one element for each value read. To increase the array size, MATLAB is allocating memory for a new array, copying the old array and new value into the new array, and deallocating the memory for old array. This is much less efficient than if the memory for the array was preallocated. If the array size is not known beforehand, the following can be done to make the operation more efficient:

- Preallocate the amount of memory you think you will need for the operation.
- Perform the operation.
- If while performing the operation, the array fills up, allocate memory for a new larger array (typically 20% to 50% larger than the current array). Copy the old array into the new array and deallocate the memory for the old array. Perform this array resizing as needed until the operation is complete.
- Once the operation is completed, resize the array down to deallocate unused memory. Allocate memory for a new smaller array, copy the portion of the array being used into the new array and deallocate the memory for the old array.

Last modified Wednesday, March 30, 2022

 [MATLAB Marina](https://creativecommons.org/licenses/by-nc-sa/4.0/) is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.