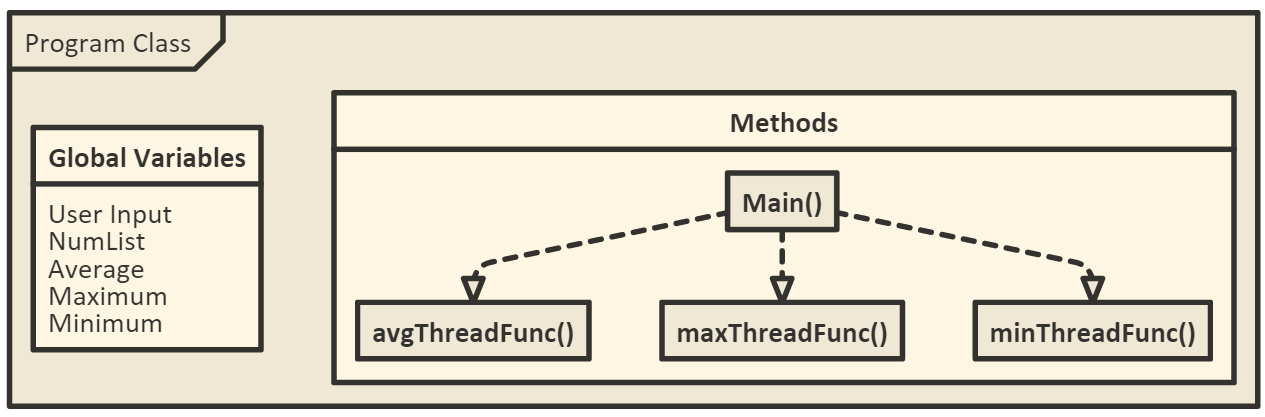
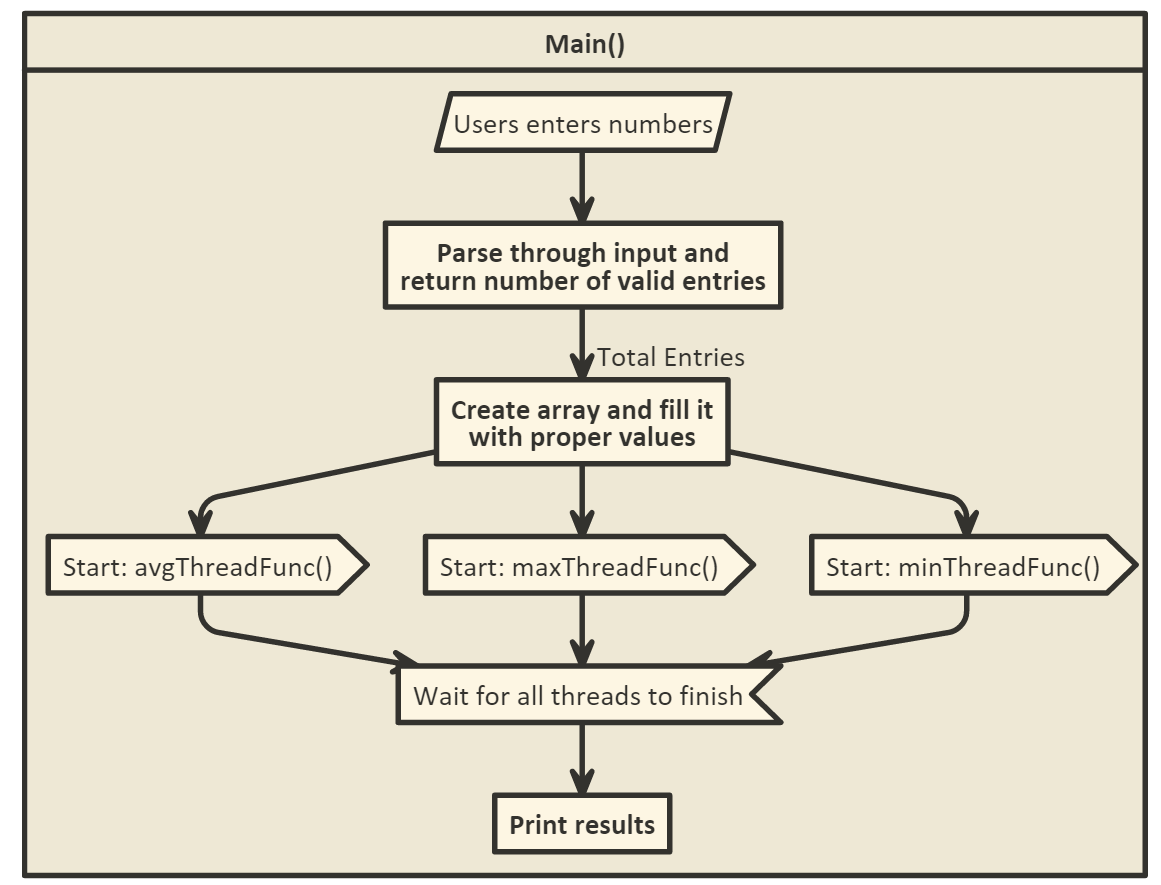
Threading Program Documentation

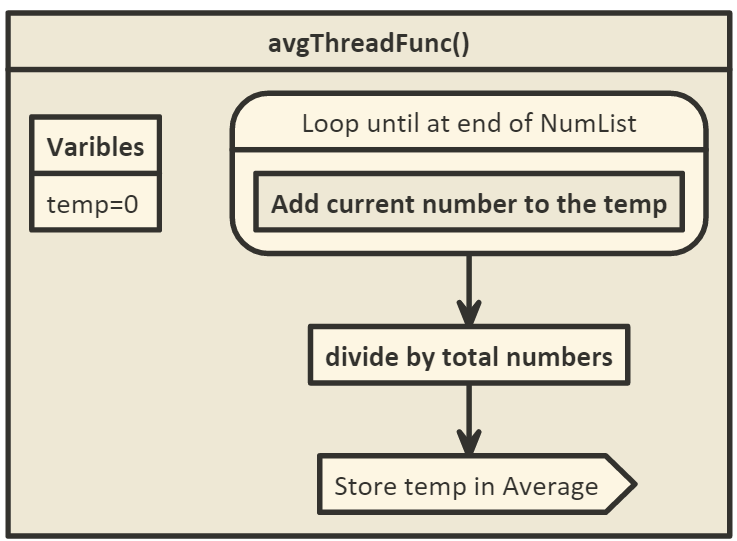
# Program Design

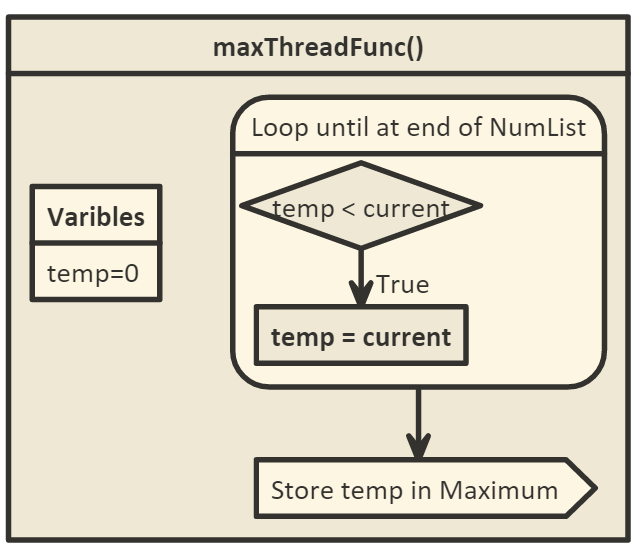
This program is created to take a series of integers and then create three separate threads to simultaneously calculate the Average, Maximum and Minimum values. The input values will be separated by spaces and if any value is not a valid integer, then the program will just ignore that value. The basic logic for this as follows:

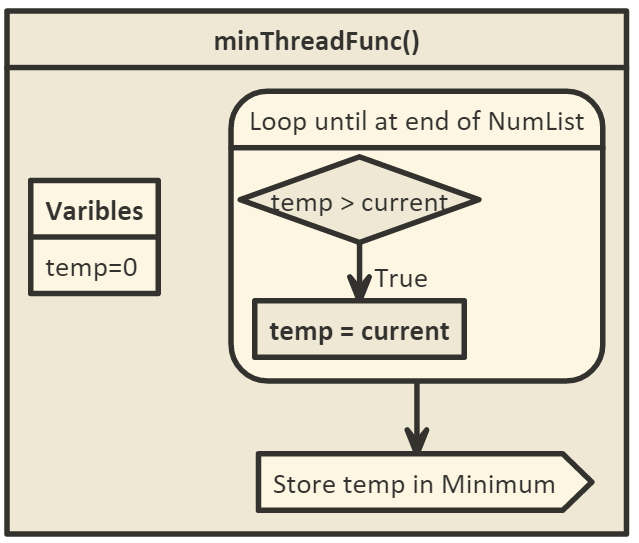
* Get the user input string
* Count the number of valid entries
* Store the valid entries in the numList global int[]
* Start each of the following methods as separate threads:
  + avgThreadFunc - Gets the average value and stores in Average global variable
  + maxThreadFunc - Gets the maximum value and stores in maximum global variable
  + minThreadFunc - Gets the minimum value and stores in minimum global variable
* wait for all of threads to exit
* print results

The program is fairly simple and could be easy modified with more functions to calculate several results. Any statistical values that could be drawn from a set of numbers can be done. Something that would probably make this better is to have a function that creates all the threads from a list, so that adding new functions is easy, and another to stop all functions. This way you wouldn’t have to edit main functions just the method and add it to the list of threads to create.

The following are diagrams of the program:  
  








# Files/Database Section

There are no files needed to test this program.

# Compile/Execution Section

This program was created in C# using .net framework 4.6. You will need the .net framework installed on your system to run it. Navigating to the bin/debug will give you the .exe needed to run the program. Also the entire solution can be open in visual studio.

# Test Cases Section

As far as preventing invalid errors go, the program will just ignore anything it parses from the user input that is not a valid integer, so there is no chance for error in the calculations.

Now when it comes to proving that the threads are running at the same time and do not wait for each other to finish we have to edit the code to create a case that shows us this. I added a few comments in the code to show where these changes need to be made. First we need to comment out the maxThread.join(); line in the main function. This will cause the main function to continue running regardless of whether the thread has finished or not. The next thing to do is to add (or uncomment) Thread.Sleep(3000) at the beginning of the maxThreadFunc(). This will cause that thread to take at least 3+ seconds to complete making it take longer than the other threads.

Now if you run this code you will see that the max value will always be zero, but both the average and minimum have been correctly calculated. This shows that the other two methods were still calculated and the main method printed the results before the max value was found, thus proving that the other two functions (or more specifically the minimum function) are not waiting to for the maximum function to finish. One important note is that this needs to be tested on a multicore system. A single core processor may attempt to finish the maximum thread before the minimum thread is started is it doesn’t have another core that can do it while it waits for that sleep function to end (this also is dependent on how the system handles the sleep function and if it switches to another thread during the sleep function).