# **Task 1: Input Identification**

1. Why is the ability to check input so important?
   1. The ability to check input and tell its type is very important because in an operating system, you will be getting in various amounts of input from a lot of places. You will need to be able to validate this input, assuring its type is appropriate for what it is being used for. If an improper command is given, it may also crash the system.
2. Other than simply providing the wrong type of input, what other ways can you think of for bad input to cause an error? Consider situations other than typing input when prompted.
   1. If the input entered is too large in memory size, it may cause an error to occur.
   2. If there is no input entered when input is to be expected, this should be caught, and properly handled in order to prevent an error from occurring.
   3. If an operation returns an improper data type, or returns 0, this may cause an error to occur.

# **Task 2: Shouting Threads**

1. Remove the busy waiting loop used whenever a thread shouts and run the task with 5 shouters and 5 shouts per shouter. Then have each thread yield once after shouting and run another test with the same parameters. Note your results and explain your observations. Undo any changes made to accommodate this question before submitting your assignment.
   1. Without the busy/wait loop, each thread will do all 5 shouts, and then it will move on to the next thread. This is due to the lack of any type of yielding to allow any other threads access to the CPU. Thus, each thread completes its shouting for loop before allowing the other threads to shout.
   2. After having each thread yield once after shouting, the threads will then shout once in order (0, 1, 2, 3, 4, 0…). This is because each time any thread shouts once, it yields to allow another thread access to the CPU, allowing it to shout.
2. Temporarily disable your input validation, run a minimum of 5 tests with garbage input, and note the results. How would an end user react to this? Undo any changes made to accommodate this question before submitting your assignment.
   1. Threads Input: Jason, Shouts Input: Hello
      1. Nothing happens.
   2. Threads Input: 2.6, Shouts Input: 10.5
      1. Program runs with truncated values (2 threads running 10 shouts)
   3. Threads Input: -45, Shouts Input: -1
      1. Nothing happens.
   4. Threads Input: 2, Shouts Input:
      1. Nothing happens.
   5. Threads Input: !@#$%, Shouts Input: !@#$%
      1. Nothing happens

A user would obviously be very confused when nothing happens, once garbage input is entered. When the program runs with truncated values, this would likely be a bit more understandable to the user, and would make a bit more sense.

# **Task 3: Command Line**

1. What other solutions can you think of to handle improper input on the command line?
   1. The input could be more efficiently evaluated to turn doubles into integers or further provide individualized error messages for each type of improper input.

# **Task 4: Report**

1. In your own words, explain how you implemented each task. Did you encounter any bugs? If so, how did you fix them? If you failed to complete any tasks, list them here and briefly explain why.
   1. For task 1: I created an enumeration which accounts for the different types of inputs. A thread takes this input to an evaluation function, which truncates whitespace and goes through the array character-by-character to determine which type it is. If it ever sees a character, it is determined to be of type character, if it sees a ‘–‘ or ‘.’ In the right spots, it is marked as possibly being a negative or decimal, respectively, if they are only encountered once. Then if it manages to exit the loop, it checks those Booleans to determine if it is a decimal, negative decimal, integer, or negative integer.  
      The main bugs encountered were dealing with removing leading whitespace, because there was pointer arithmetic involved, and I was ignorant to the topic. This was fixed by becoming less ignorant to the topic.
   2. For task 2: I converted user input into integers, then create a thread pointer and loop the desired number of times. A new thread is created each iteration, and forked to a shouting function. The shouting function generates a random number from 2 to 5 to wait, prints a shout, and then waits that number of times.
   3. For task 3: A string compare was used with “-A” and the number after it is assigned to a task counter. Threadtest.cc tests this to ensure that –A was used and the user asked for a valid task.
2. What did you learn from working on this assignment?
   1. We learned how to create/fork/manage threads, how to evaluate the type of an input, and how to create console commands.
3. What sort of data structures and algorithms did you use for each task?
   1. The only used data structures were arrays and enumerations. The main algorithm was the evaluation method described above.