**Task 1**

1. The ability to check input is a primary method of ensuring your program works properly without errors. Invalid input can cause even properly coded functions to fail, which can result in minor or major problems in program execution.
2. If functions are passing variables as parameters to other functions with input that is outside of the scope of the program, many programs can be caused to crash. That one bad input can have a domino effect and cause bad data to be stored and cause other bad input to be created for subsequent functions.

**Task 2:**

1. As the busy waiting loop being removed, a thread shouted all 5 times, and then each of other threads took turn. On the contrary, when each thread yielded once after shouting, each of thread took turn to shout.
2. When input validation was disabled, no thread was created or ready for execution as the input started with characters, such as Ab143, @fdal5.53, A5-45, and etc. As input starting with numbers, such as 3.14, 6.1.1.1, 4Abc-1, and etc., the numbers before the first character were extracted and considered as integer input.

**Task 3**

The problem can be solved using more brute force method of only if-else statements.

**Task 4:**

1. In your own words, explain how you implemented each task. Did you encounter any bugs? If so, how did you fix them?

For task 1, the input string is extracted from keyboard using *scanf()*. Upon passing the input string to atoi(), any characters which are not numbers (i.e. 0-9) are truncated. If so,

For task 2, the function ThreadTest() calls the function NumShouter() which prompts the user for the number of shouters and then checks to see if the type entered is a positive integer. If not it sends an error message and prompts the user for input again. Once input is correct it returns the number of shouters requested by the user. Then Occurrence() is called and the user is prompted for the amount of times to shout and then checks to see if the type entered is a positive integer. If not it sends an error message and prompts the user for input again. Once input is correct it returns the number of shouts requested by the user.

Then the tread is forked to the function Shout() once for each shouter. Shout() picks a random shout for each shouter and then loops the shout for the users request amount of times.

1. What did you learn from working on this assignment?

We learned how to use the random number generator in Nachos, How to fork multiple threads, How to simulate busy-wait loops, How to interpret command line functions, and How to thoroughly test input for validity.

1. What sort of data structures and algorithms did you use for each task?

The algorithm used in task 1 is described as the pseudo-code as follow.

*str* = get\_input; // use scanf()

*i\_str* = atoi(str); // only keep the first several number

if (strcmp(*str*, *i\_str*)==0) // nothing is truncated {

if (*str* == ‘0’) {*str* is zero;}

else if (*str*[0] == ‘-‘){*str* is a negative integer}

else {*str* is a positive integer}

} else { //something is truncated

scan each digit and raise *flag* if ***E*** is true; // refer to the description of ***E***

if (*flag* == 1){str is a character string}

else if (*str*[0] == ‘-’) {*str* is a negative decimal}

else {*str* is a positive decimal}

}

***E*** is true if

1) *str*[0] = ‘.’;

or 2) number of ‘.’in *str* is more than 1;

or 3) ‘+’ or ‘-‘ appears in *str* but not the beginning;

or 4) any digit is not ‘0-9’,’+’, ’-‘, or ’.’.

The algorithm used in task 2 is described as the pseudo-code as follow.