**Spike:** Task 8

**Title:** Performance Measurement

**Author:** Sam Huffer, 101633177

# Goals / deliverables:

* Demonstrate (through this spike report) the following performance and measurement concepts:
  1. Measurement of single and multiple functions’ execution times.
  2. Linear and exponential ramp-up testing of function execution time, comparing to ramp-down testing.
  3. How repeatability will vary depending on test conditions.
  4. Comparison of functions providing the same functionality.
  5. Whether there’s any difference in execution time between debug and release settings in Visual Studio.
  6. Whether turning down / off compiler optimisation makes a difference.
* Record and chart the data collected for each section.

# Technologies, Tools, and Resources used:

* Visual Studio 2017
* Microsoft Word
* Microsoft Excel

# Tasks undertaken:

1. *Single Tests: Demonstrate how to measure both single and multiple function execution time.*
   * Requested clarification.
2. *Ramp-Up Test: Execute and show (numbers/make a chart) both linear and exponential ramp-up testing of function execution time. Is there a difference to ramp-down tests?*
   * I examined the code for both of the ramp-up test methods to get an understanding of how they operated, then ran the code as is.
   * I ran the code 5 times, and noted down all of the results for each iteration in Excel.
   * Below the linear and exponential data sets, I configured some cells to calculate the average time and time per item for each quantity of items tested in both the linear and exponential data sets.
   * I used the average data sets to create line charts, labelling the charts and axes, and displaying individual data points and trendlines appropriately.
3. *Repeatability: Show (with numbers/chart) how repeatability will vary depending on test conditions.*
   * Requested clarification
4. *Function Comparison: There are two "char in string” counting functions provided (code sample 1). Clearly show the difference in performance (if any), and show if the speed difference is linear to string size. (Note, you will probably want to create random strings of the various size to test with.)*
   * I examined the code for both of the string counting methods to get an understanding of how they operated. Particularly, I had a look at how the time recording was done in the ramp-up testing methods, and replicated that around the methods being tested for this part of the spike. taking how it recorded the time it took to execute a chunk of code, and implemented that around the count\_char…() methods. Ran the code several times using strings of different lengths, and recorded the results.
   * I adopted the existing test string and created 5 more test strings of varying lengths and compositions, and ran the code 5 times for each, noting down all of the results for each iteration in Excel.
   * Below the linear and exponential data sets, I configured some cells to calculate the average time each of the two methods took to calculate each string.
   * I used the average data sets to create a line chart comparing the two methods, labelling the charts and axes, including trendlines, and displaying individual data points as appropriate for the chart.
5. *IDE Settings: Show what, if any, is the difference in execution time between debug settings and release settings. (Remember to have a task that runs for long enough that it matters.)*
   * Double check this is what I think it is.
6. *Compiler Settings: Turn down/off compiler optimisation and demonstrate a difference.*
   * .

# What we found out

## General

* How to use stopwatch-like functionality in C++ to examine how long something takes to be executed.
* How to do linear and exponential ramp-up testing, and repeatability testing.

## Test results

### 1: Single Tests

### 2: Ramp-Up Test

Figure 2: the total time for the linear test method to execute was roughly linear, increasing in duration proportionate to the increasing number of items. Interestingly, its duration per item was rather variable, but the trend line did show a very slight increase in time per item as the number of items increased; I suspect that might be due to the linear disparity in average time per item for 10,000 items and 50,000 items. The exponential ramp-up method showed an exponential increase in time taken to process all items in line with the exponential increase in the number of items, but an exponential decrease in the time per item as the number of items increased exponentially. Note: while I was executing the two methods to obtain their respective data sets, I noticed that, at least several times, the exponential ramp-up method, would print out all the times and the time for the 100,000,000 item iteration would be zero point something like the rest, but it would stall before printing “done”, and once “done” was printed, the number would change to twenty-something. That said, each increase in the number of items to be processed by a factor of 10 (barring the single-item iteration) corresponds to an increase in duration by roughly a factor of 10, which the aforementioned data point conforms to, so it would not be unexpected that the zero point something values originally displayed were an algorithmic error and the twenty-something values were the accurate times.

### 3: Repeatability

### 4: Function Comparison

Figure 4: the time each method took to process the strings was, on average, proportionate to the length of the string. Diversity of characters seemed to affect efficiency as well (compare the three strings of length 111). The using count method, for whatever reason, showed the most variability, struggling particularly with the third string (“This is . . .”) but the using find first of method exhibited the steeper trendline. Whether that’s because of the former method’s variability or poorer handling of longer strings on its own part is unclear from the data collected.

### 5: IDE Settings

### 6: Compiler Settings