Spike: Spike No. 7
Title: Task 7 - Performance Measurement

Author: Khang Trinh - 102118468

Goals / Deliverables:

The goal of this spike is to learn how to test code, identify performance issues and make improvements where needed.

Technologies, Tools, and Resources used:

- Visual Studio 2017
- Microsoft Excel

Useful Links:

1. About visual studio's build settings (/O1 and /O2)
   https://docs.microsoft.com/en-us/cpp/build/reference/o1-o2-minimize-size-maximize-speed?view=msvc-160
2. Guide on how to do csv output
   https://www.youtube.com/watch?v=x2niMA5tzGo&ab_channel=YunusKulyyev
   It's tiresome to manually enter each individual output like a non-lazy person :D

Tasks undertaken:

Step 1. Learn how to measure the time a function takes to execute
To do that, we need to know about `std::chrono`. In short, it's a library used for time tracking.

```
// 1. get start time
auto start = steady_clock::now();
// 2. do some work (create, fill, find sum)
vector<int> v(size, 42);
total = accumulate(v.begin(), v.end(), 0u);
// 3. show duration time
auto end = steady_clock::now();
duration<double> diff = end - start;
```

Step 2. Learn how to do csv output.
It's fine manually entering data into an excel sheet for a test if it only has 10 or so cycles, but if you're testing in the millions, then you'd need a more efficient way, and one of which is making the program do that itself.

```cpp
#include <iostream>
#include <fstream>
int main()
{
    std::ofstream myfile;
    myfile.open("example.csv");
    myfile << "column1,column2,column3,\n";
    myfile.close();
    return 0;
}
```

Step 3. Stress test using for loop

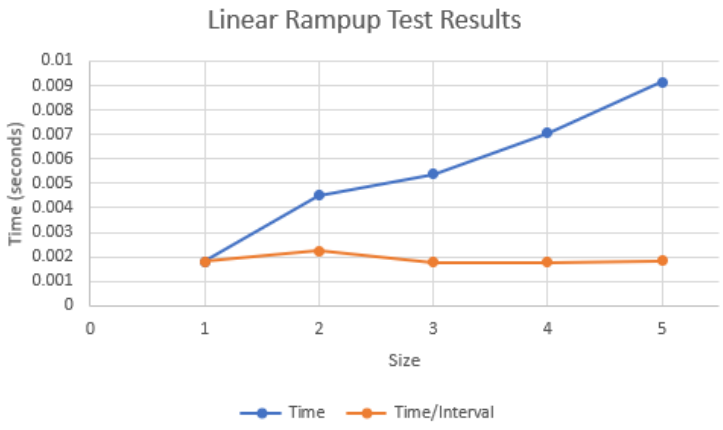
A reminder to put your output and timer code *inside* the loop to time each operation as well as outputting the result into the csv file. The number of loops

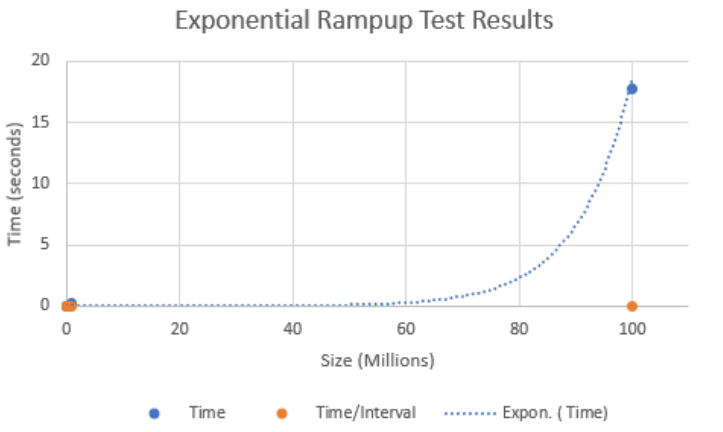
```cpp
for (auto size = 1ull; size < 1000000000ull; size *= 100)
```

Step 4. Convert the csv output into a readable graph

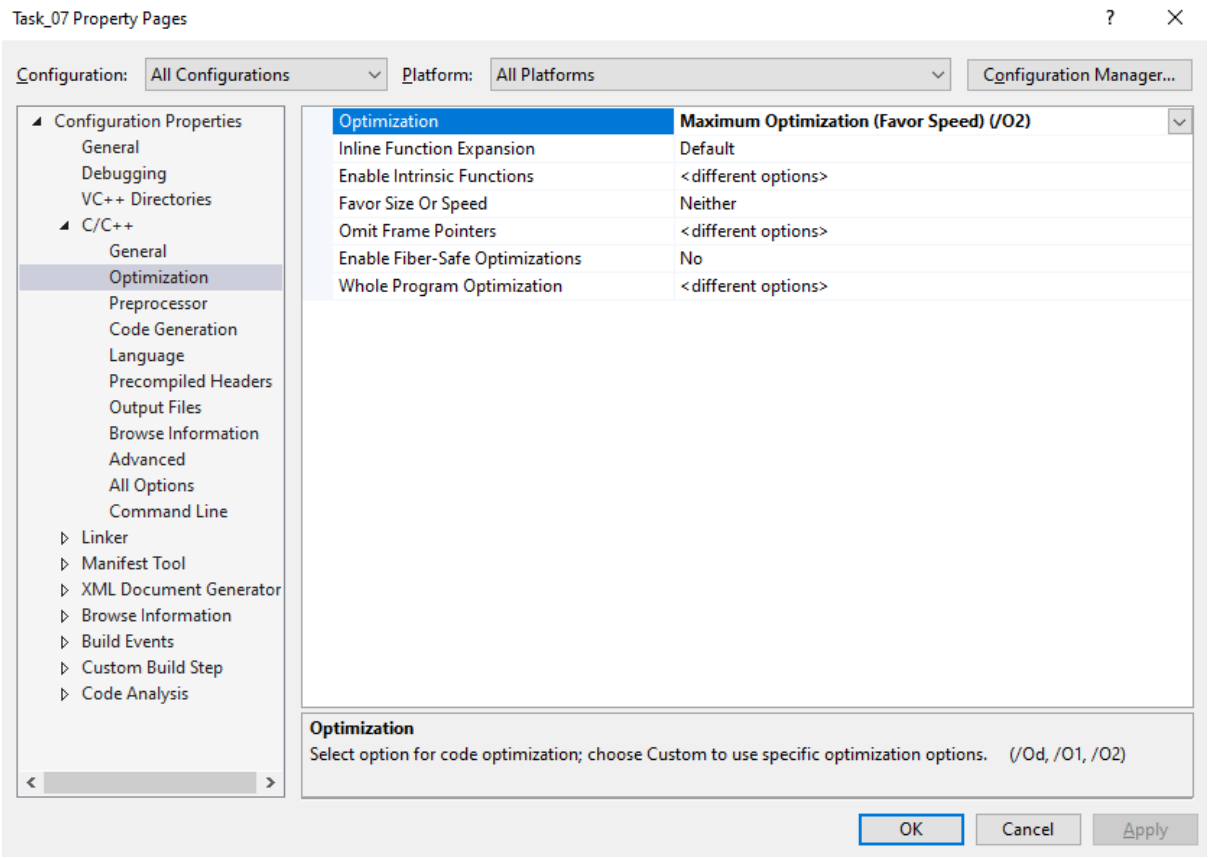
There are different types of tests you may be asked to perform. Make sure that when it comes to plotting the data into graphs that you choose the suiting graph type to match the data.

	A	B	C	D
1	Linear rampup test			
2	Size	Time	Time/Interval	
3	1	0.001818	0.001818	
4	2	0.004497	0.002248	
5	3	0.005368	0.001789	
6	4	0.007048	0.001762	
7	5	0.009145	0.001829	
8	Exponential rampup test			
9	Size	Time	Time/Interval	
10	1	1.03E-05	1.03E-05	
11	100	2.60E-05	2.60E-07	
12	10000	0.001787	1.79E-07	
13	1000000	0.177447	1.77E-07	
14	1E+08	17.76601	1.78E-07	

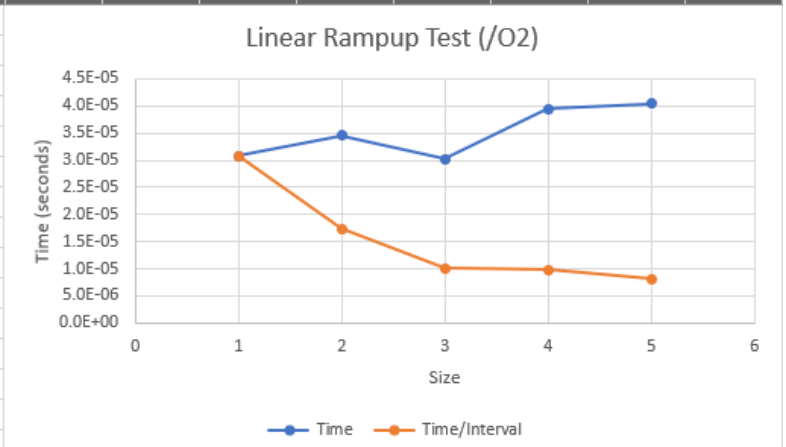


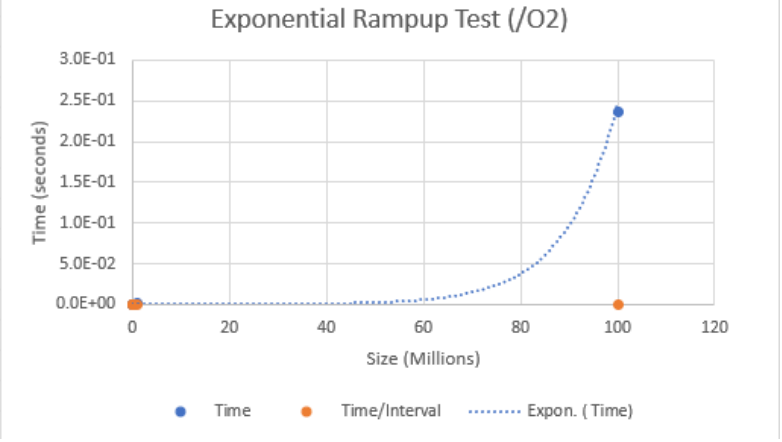


Try repeating the same test with different build options, as this will also affect the time taken. In this example, we're trying with a /O2 setting, which favors speed



	A	B	C	D
1	Linear rampup test			
2	Size	Time	Time/Interval	
3	1	3.08E-05	3.08E-05	
4	2	3.46E-05	1.73E-05	
5	3	3.02E-05	1.01E-05	
6	4	3.94E-05	9.85E-06	
7	5	4.05E-05	8.10E-06	
8	Exponential rampup test			
9	Size	Time	Time/Interval	
10	1	2.40E-06	2.40E-06	
11	100	2.10E-06	2.10E-08	
12	10000	2.47E-05	2.47E-09	
13	1000000	0.001729	1.73E-09	
14	1E+08	0.236538	2.37E-09	



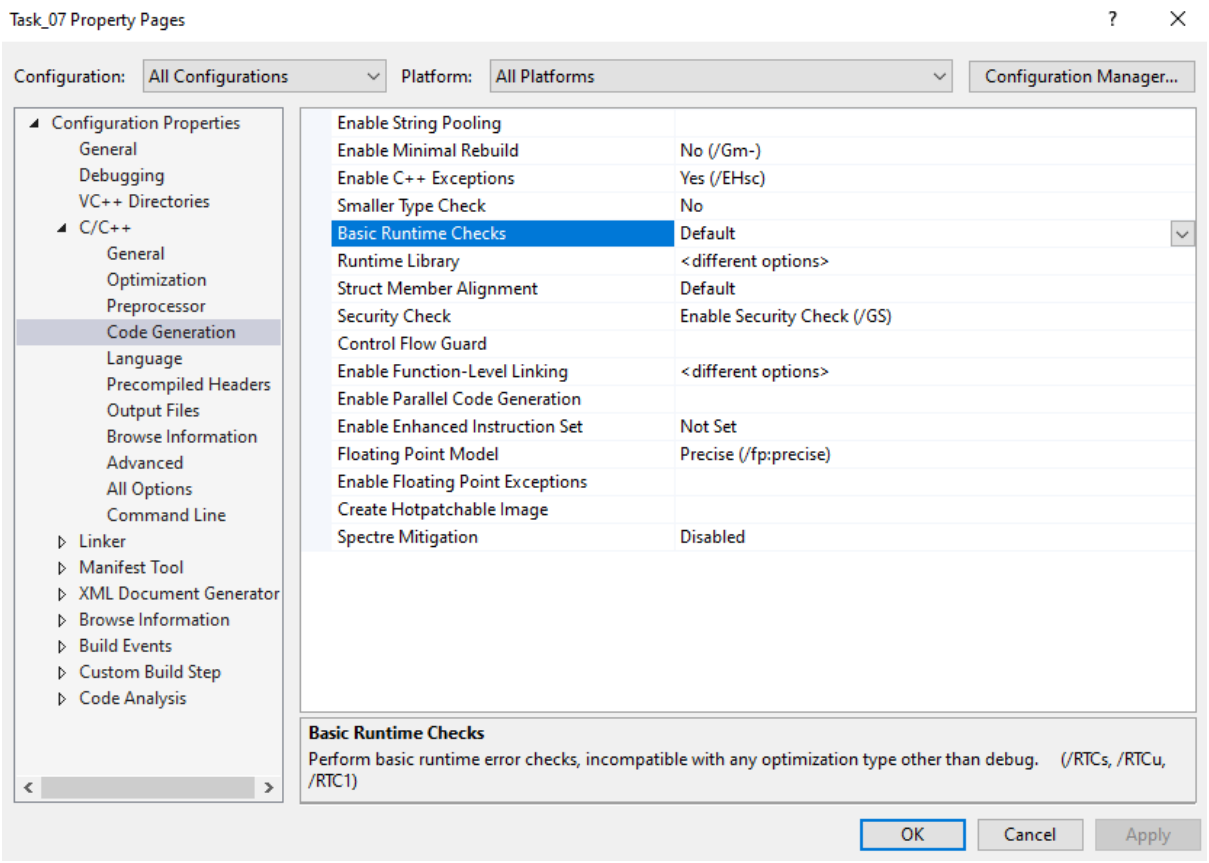


Expanded version of the code as an example

```
void exponential_rampup_test()
{
    testFile << "Exponential rampup test" << endl;
    testFile << "Size, Time, Time/Interval" << endl;

    cout << " << Exponential Ramp-up Test >> " << endl;
    int total;
    // ull (suffix) == "unsigned long long" in c
    for (auto size = 1ull; size < 1000000000ull; size *= 100)
    {
        // 1. get start time
        auto start = steady_clock::now();
        // 2. do some work (create, fill, find sum)
        vector<int> v(size, 42);
        total = accumulate(v.begin(), v.end(), 0u);
        // 3. show duration time
        auto end = steady_clock::now();
        duration<double> diff = end - start;
        cout << " - size: " << size << ", time: " << diff.count() << " s";
        cout << ", time/int: " << diff.count() / size << "s/int" << endl;

        testFile << setprecision(15) << size << "," << diff.count() << "," << diff.count() / size << endl;

        // TIP: time in nanoseconds? Cast result of chrono::duration.count() ...
        // auto _dur = duration_cast<nanoseconds>( end - start ).count();
        // cout << _dur << endl;
    }
    cout << "done." << endl;
}
```

```cpp
int main()
{
    testFile.open("TestResult.csv");

    // Simple wrapper around a linaer set of time tests
    linear_rampup_test();

    // Simple wrapper around an exponential set of time tests
    exponential_rampup_test();

    // Compare the two different methods of counting in a string
    // - show result in nanoseconds?
    string s1 = "This is a really simple string but it will do for testing.";
    int result;
    result = count_char_using_find_first_of(s1, 's');
    cout << "result: " << result << endl;

    result = count_char_using_count(s1, 's');
    cout << "result: " << result << endl;

    testFile.close();
}
```

What we found



- In order to build with an optimization setting, you'd need to also change the Basic Runtime Checks to "Default". You won't be able to debug the code line-by-line, but that's what you trade for more speed.