**Assignment 3: Due – Sunday, July 21 at 9 PM**

**Turning in:** All **written responses & code** should be placed in your git repository on GitHub in a subfolder called HW03 by the due date. If you are part of a team, only one set of files will be needed for all students who are members of the team.

All scripts and programs will be graded **on Euler**. To make sure you are using the same compiler that is used for grading, issue the following command as soon as you log into Euler: module load gcc/latest. Make sure that your responses are well-tested on Euler in this environment.

**BEFORE YOU START:**

1. You must use Slurm to run your programs on Euler. Running compute-heavy programs on the head node can destabilize the entire cluster. You should exclusively use the slurm\_shortgpu partition on Euler (see Slurm early slides in the semester).
2. Some helper files might be found on GitHub at <https://github.com/nicolsen/ME459Upstream/tree/master/HW06>
3. Please post your questions on Piazza.

**Task 1**.

One very common task in data processing is string manipulation. In this task, you’ll have to write a C program that reads a string provided as the first command line argument. Pass the string to a function that you write. This function should count the total number of characters in the string excluding the null character. Return this value to the main program and pass it to the provided function outputT1. We are going to test your program by passing it all sorts of strings: from empty strings, to strings that are 256 characters long. We won’t pass it a string longer than 256 characters.

Notes:

* Your program will receive only one argument.
* A good overview of how to handle command line arguments is available [here](https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm).
* We will compile your program with the following command:

gcc task1.c output.c -o task1

* We will run your program with the following command:

./task1 *somestring*

(*When testing your program, you may need to put your argument* somestring *in quotes for bash to treat it as a single argument.*)

* You may not use the standard library function strlen from string.h when writing your answer.

**Task 2.**

The purpose of this task is to learn how to time how long it takes the CPU to execute a portion of your code. The timing should be done as explained in the [GNU documentation](https://www.gnu.org/software/libc/manual/html_node/CPU-Time.html). What you will get reported is CPU time; i.e., the time the CPU took to solve your problem. Keep in mind that the elapsed time can be larger since the operating system engages in time slicing and will execute other programs while your program is in flight.

Given a matrix of dimension 1000 by 1000, containing all random numbers stored in double precision, you will have to multiply this matrix by a vector of size 1000 whose entries are defined as follows: the first entry will be 0.5, the next one will be -0.5, the next one 0.5, the next one -0.5; i.e., the entries alternate between 0.5 and -0.5.

For program output, you will need to pass two values to the function outputT2:

* The first value is the [norm two](http://mathworld.wolfram.com/L2-Norm.html) of the vector
* The second value is the amount of time, in milliseconds, that the CPU took to multiply carry out and then evaluate the norm-2 of .

Notes:

* The way the matrix is stored is as follows: in one array of doubles of size 1,000,000, you will store the first row of A in entries 0 through 999, the second row of in entries 1000 through 1999 of the big array, etc. This is called “row-wise storage of ”. This array should be *dynamically-allocated*.
* You should not generate the random values yourself, instead you will pass your dynamically-allocated array to the function randomT2 which will fill it with random numbers.
* There are several ways to multiply a matrix by a vector. The strategy that you must implement is as follows: to get the first entry in you will multiply the first row of with the vector; to get the second entry in you will multiply the second row of with the vector; to get the third entry in you will multiply the third row of with the vector; etc.
* We will compile your program with the following command:

gcc task2.c output.c -o task2 -lm

* We will run your program with the following command:

./task2

**Task 3.**

The purpose of this problem is to familiarize with the gdb debugger on Euler and to understand better how pointer arithmetic works. To this end, you will have to use the flag -g when compiling your code with gcc to include debug information in the executable. Consider the code in the text-box at right. Use the gdb debugger to step through the code and answer the following questions:

int main() {

int d;

char c;

short s;

int \*p;

int arr[2];

p = &d;

\*p = 10;

c = (char)1;

p = arr;

\*(p + 1) = 5;

p[0] = d;

\*((char \*)p + 1) = c;

return 0;

}

1. What is the size of variable p on Euler?
2. What is the address of p?
3. What is the address of c?
4. What is the value of arr[0] after the assignment on line 16?
5. What is the value of arr[0] at the end of the program?
6. Explain why the value of arr[0] changes

Report your responses to questions a) through f) in your written response.

Notes:

* The source code shown in the image above is available as task3.c in the upstream repository. It can be compiled with the following command:

gcc -g task3.c -o task3

* gdb should be invoked with Slurm using the following command:

srun –p slurm\_shortgpu --pty -u gdb task3

**Task 4.**

In one or two sentences, highlight what the following types of testing try to accomplish:

1. Unit Testing
2. Integration Testing
3. Functional Testing
4. End-to-end Testing
5. Performance Testing
6. Usability Testing
7. Acceptance Testing
8. Beta Testing

Are you aware of any other type of testing that is important? If so, please briefly summarize what it tries to accomplish

**Task 5. (Bonus)**. This task is a continuation of Task2.

Solve Task 2 again, but this time multiply the matrix and the vector in the following way: the array should be computed as the first column of scaled by the first entry in , to which you add the second column of scaled by the second entry of , to which you add the third column of scaled by the third entry of , etc. Make sure your code produces the same outcome as for Task 2.

Additionally, answer the following in writing: Assume that the amount of time required to solve the problem in Task 2 is T2, while the amount of time required to solve the problem in Task 5 is T5.

* + How do T2 and T5 compare? When answering this question, make sure you run the codes several times to get a sample of T2 values and T5 values – just to avoid a situation where you get some sort of outlier results.
  + Do you have any insight into why the values might be different?

Notes:

* The norm 2 of should be the same, since the values in should be like the ones you calculated in Task 2.
* You should also use the same helper functions, randomT2 and outputT2, for this task.
* We will compile your program with the following command:

gcc task5.c output.c -o task5 -lm

* We will run your program with the following command:

./task5