

1 Exercises

1. Write an assembler program that will ask the user for two integer inputs (positive integers) and output their sum. An example run is below.

```
./AddUserInput
Please enter first number: 234
Please enter second number: 652
Sum is 886
```

2. Write an assembler program that will ask the user for 5 integer inputs (positive integers) and output their sum. An example run is below.

```
./AddUserInput
Please enter number: 3
Please enter number: 6
Please enter number: 31
Please enter number: 76
Please enter number: 123
Sum is 239
```

3. Write an assembler program that will ask the user for the number of numbers that they want to add, then asks them for each number one at a time and finally returns the sum of the input numbers.

```
./AddUserInput
Please enter the number of numbers to add: 3
Please enter number: 123
Please enter number: 456
Please enter number: 789
Sum is 1368
```

4. Write an assembler program that will ask the user for a number and then return the factorial of that number. Recall that we define the factorial of a number as follows. If $n > 0$ then

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdots 2 \cdot 1$$

and if $n = 0$ we define $0! = 1$. For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

```
./FactorialUserInput
Please enter number: 10
10! = 3628800
```

```
./FactorialUserInput
```

```
Please enter number: 0
0! = 1
```

```
./FactorialUserInput
Please enter number: 5
5! = 120
```

5. In this exercise we are going to generate random numbers in both assembly and in C++, then using the Linux `time` command we will compare the running time of the two. One way to generate random numbers (really pseudo-random numbers) is with a linear congruential algorithm. In a linear congruential algorithm the program generates a sequence of numbers x_0, x_1, x_2, \dots by setting x_0 to the seed (starting number) and then generating each of the other numbers by $x_{j+1} = m \cdot x_j + a$. If you look up a linear congruential algorithm, you will also see that this calculation is modded by a number n . We simply let the cycling of the integer data types take care of this so we do not need to worry about modding the results. For example, if we set our seed to 243, use $m = 123$ and $a = 456$, our sequence would start out, 243, 30345, 3732891, 459146049 ...

- (a) Write an assembly program that will take value from the command-line argument list and generate a linear congruential sequence of numbers. The first argument is the number of numbers to generate, the second number is the multiplier m , the third number is the adder a , and the fourth number is the seed (the first number of the sequence x_0). If the number of arguments is incorrect then the program should print out an error message.

```
./RandUserArgs 10 123 456 243
243
30345
3732891
459146049
640389635
1458514233
3303591979
2614888049
3803650579
3992553705
```

```
./RandUserArgs 10 123 456
Incorrect number of arguments.
```

- (b) Write the same program in C++, recall that we can get command-line arguments in C++ by adding an integer and an array of strings to the argument list of the main function. The example we looked at in class is below.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main(int argc, char * argv[])
6 {
7     cout << "Number of Arguments: " << argc << endl;
8     cout << "Argument List" << endl;
9     for (int i = 0; i < argc; i++)
10         cout << argv[i] << endl;
11 }
```

To make this program match the assembly program you have written, use the data type `unsigned int` instead of just `int`. Also, C has its own built-in `atoi` function (which is where the name of ours came from). It is in the `stdlib.h` library so make sure you include this library as well as `iostream`.

- (c) Make sure that each of the two random number generating programs are working correctly. Now we will compare their speeds. In each of the two programs, comment out the line that prints the values. Only comment out the printing of the values, all the other calculation lines should be uncommented. Recompile and reassemble the two programs. Run each with the same argument list in the `time` command to generate 1000000, 10000000, 100000000, and 1000000000 random numbers. For example, generating 1000000 numbers and timing the execution would be the command.

```
time ./RandUserArgs 1000000 432381 73821 123456789
```

You will get three times, just use the one that says real. Now run the other sizes and compare the run times of your C++ and Assembly programs. Is one consistently faster than the other? If so which one and how much faster is it?

2 Submissions

Upload each of your programs (6 total, 5 assembler and one C++) to the MyClasses page for this assignment. Also, in the comments box, put the running times you got from the assembler and C++ random number generating programs.