

# 1 Exercises

1. Recall that we define the factorial of a non-negative integer as

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1 & \text{if } n > 0 \end{cases}$$

For this exercise you will write a 64 bit assembly program that will be called by a C++ program to calculate a user input factorial. The C++ program will handle all the input and output and the assembly program will calculate the factorial.

Specifically, the C++ program will ask the user for the input, it will call the assembly program to do the calculation and return the result to the C++ program, then the C++ program will display the result.

The assembly program will check to see if the input number (that came from the C++ program) is greater than or equal to 0. If the input is valid, the assembly program will do the factorial calculation and return the result. If the input was negative, then the assembly program will return  $-1$ . The C++ program is not to check the user's input, it will simply check the output of the assembly program, if the assembly program output is  $-1$ , the C++ program will display an error message and if not it will display the output of the assembly program.

Several runs of the program are below.

```
./prog
Input n: -3
Error: Input must be greater than or equal to 0.
```

```
./prog
Input n: 0
0! = 1
```

```
./prog
Input n: 12
12! = 479001600
```

```
./prog
Input n: 10
10! = 3628800
```

```
./prog
Input n: 20
20! = 2432902008176640000
```

2. The Fibonacci sequence is a sequence of positive integers, the first two integers are both 1 and every entry in the sequence after that is the sum of the two previous entries.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946 . . .

For this exercise you will write a 64 bit assembly program that will be called by a C++ program to calculate a Fibonacci number. The C++ program will handle all the input and output and the assembly program will calculate the Fibonacci number.

Specifically, the C++ program will ask the user for the input, it will call the assembly program to do the calculation and return the result to the C++ program, then the C++ program will display the result.

The assembly program will check to see if the input number (that came from the C++ program) is greater than 0. If the input is valid, the assembly program will calculate the  $n^{\text{th}}$  Fibonacci number (where  $n$  is the user's input) and return the result. If the input was 0 or negative, then the assembly program will return  $-1$ . The C++ program is not to check the user's input, it will simply check the output of the assembly program, if the assembly program output is  $-1$ , the C++ program will display an error message and if not it will display the output of the assembly program.

Several runs of the program are below.

```
./prog
Input n: -1
Error: input must be greater than or equal to 1.
```

```
./prog
Input n: 4
3
```

```
./prog
Input n: 10
55
```

```
./prog
Input n: 20
6765
```

```
./prog
Input n: 50
12586269025
```

3. Write a program in NASM that will load an array of values from the argument list (100 maximum), then double each entry in the array, then finally print out the contents of the altered array. Use only one array in this program.

An example run is below. Remember that command-line arguments are placed on the stack, use this to determine the number of elements for the array and to read the array into the program.

```
./arraydouble 1 2 3 50 123 9 256  
Contents: 2 4 6 100 246 18 512
```

4. Write a program in NASM that will load an array of values from the argument list (100 maximum), print out the array, then reverse the array, then finally print out the contents of the altered array. Use only one array in this program. As a hint, you can reverse an array by interchanging the first and last elements, then the second and second to last elements, then the third and the third to last elements, and so on. If you use this method then remember that you would stop halfway down the array.

An example run is below. Remember that command-line arguments are placed on the stack, use this to determine the number of elements for the array and to read the array into the program.

```
./arrayreverse 1 3 2 6 5 9 10 123 2  
Contents: 2 123 10 9 5 6 2 3 1
```

## 2 Submissions

For this submission, zip up all of the files into one compressed file and upload it to the MyClasses page for this assignment. Put each exercise into its own directory, place all 4 of the directories into a directory named Lab10, right click on the Lab10 directory and select Compress from the pop-up menu to create the compressed file.