

Homework 4

COP 3223C Introduction to Programming with C, Section 0V06

Spring 2021

1 DESCRIPTION

For this assignment you will be working with numeric types, functions, and arrays. You will write two functions named `MultiplyCIntegers`, and `GetFactorial`. Both functions and the `main()` function should be in a single C source file named **homework4.c**. The descriptions of each function are listed below in the Function Requirements Section. Feel free to implement additional “helper” functions as you see fit.

2 DELIVERABLES

A single C source file named `homework1.c` must be submitted to Webcourses by the assignment deadline (posted in Webcourses).

3 GRADING RUBRIC

Uses correct filename (See Deliverables)	5 pts
Includes appropriate header comment (See Style Requirements)	5 pts
Uses tasteful comments (See Style Requirements)	10 pts
Follows all style requirements (See Style Requirements)	20 pts
Required functions work correctly and produce correct output (See Function Requirements)	60 pts
Total	100 pts

4 SUPER IMPORTANT INFORMATION

- **Sharing code or posting assignment details in public places will be considered academic dishonesty and will result in an automatic 0 grade for this assignment.** Feel free to have *high level* discussions about the assignment and your solution with your classmates. In general, discussions about the assignment are encouraged if they are only with students actively enrolled in this course and do not include any sharing of code.
- **Copying source code from the internet or other sources other than your own brain will be considered academic dishonesty and will result in an automatic 0 grade for this assignment.**
- Your source file must be named correctly to receive full credit. See Deliverables for the required filename. If you submit your file multiple times to Webcourses an additional “-n” will be added to the end of the filename (example: homework4.c, homework4-1.c, homework4-2.c, etc.). Files with this suffix will also be accepted.
- Your source file must contain the exact method signatures listed in Function Requirements to receive full credit.
- Your source file must compile and run to receive credit. **Submissions that do not compile will receive an automatic 0 grade.**
- **Submissions that print extra information to the console will not receive full credit.** Your submission should only produce console output if specifically requested in Function Requirements.

5 FUNCTION REQUIREMENTS

The source file you submit should contain the following functions. Please note that these functions are useful tools, which are not a part of the C Standard Library. Being developed well, you can use them in real-world projects.

`_Bool MultiplyCIntegers(char* cx, char* cy, char* res);`

Description: The function multiplies two decimal string representations of non-negative integers. The function takes two parameters `cx` and `cy`, which are string decimal representations of integers, for example,

`cx = "1237091"`

`cy = "23015"`

and returns their product as a string decimal representation: `"28471649365"`.

Your function `MultiplyCIntegers(...)` should return `1` if everything is fine, and `0` in case of an error (empty string(s), overflow(s)).

The maximum length of string representations (number of decimal digits) supported is defined by `MAX_STR_LEN` in the following `#define` pre-processor directive:

```
#define MAX_STR_LEN 1000
```

`void GetFactorial(unsigned short n, char* res);`

Description: Using the function `MultiplyCIntegers(...)` above, write a function that takes `unsigned short n` as parameter, and returns the decimal representation of `n!` as the `res` parameter.

In the `main()` function, test how it works by evaluating `n!` for `n = 10, 30, 50`.

Please note that you would get an overflow even for `unsigned long long int` data type if you tried to evaluate `50!` using standard operation `*` for integers.

6 STYLE REQUIREMENTS

- **Header Comments:** Submissions must include a header comment of the following form on the very first line of the file:

```
0 |// <Your Name>                                0 |// John Smith
1 |// NID: <Your NID>                            1 |// NID: jo123456
2 |// <Assignment Name> "<Filename>"            2 |// Homework 4 "homework4.c"
3 |// <Course ID> <Section #>, < Current Semester> 3 |// COP 3223C Section 0V06, Spring 2021
```

- **Function Comments:** All required functions must have a comment directly above them that describes what they do. Function comments should include a description of a function's expected inputs and expected outputs:

```
0 |// Takes a non-negative integer n as input and returns n! where
1 |// n! is a positive integer and n! = 1 * 2 * 3 * 4 * ... * n
0 |int factorial(int n)
1 |{
2 |    ...
3 |}
```

- **Tasteful Comments:** You should add comments throughout your code that make the code easier to read. Be careful when adding tasteful comments to your code; only lines that *need* tasteful comments should have them. Tasteful comments should be placed directly above the line or block of code they describe, or on the same line as the line they describe:

```
0 |void foo(int n)
1 |{
2 |    int i, a = 0, b = 1;
3 |
4 |    // You can put comments like this :)
4 |    for (i = 0; i < 10; i++)
5 |    {
6 |        a++; // You can put comments like this :)
7 |        b--;
8 |
9 |    // You can NOT put comments like this (bad indentation)
10 |        a = a * b;
11 |    }
12 |}
```

Please also **avoid very long comments**. Comments must usually fit one visible line in the editor.

- **Whitespace:** Variables, values, and operators should be separated by a single space in your code:

```
0 |void foo(int n)
1 |{
2 |    int a = 0, b = 1;
3 |
4 |    a = a + 1; // This is allowed
5 |    b=b+1; // This is NOT a good style because there are no spaces between b, =, +, and 1
6 |}
```

Whitespace: Your submission should include blank lines as needed to make your code easier to read. In general, you should include blank lines to separate statements that perform different tasks:

```
0 | void foo(int n)
1 | {
2 |     int i, a = 0, b = 1;           // Initialize some variables
3 |                                     // Blank line for readability
4 |     for (i = 0; i < 10; i++)       // For loop performing a new task
5 |     {
6 |         a++;
7 |         b--;
8 |     }
9 | }
```

- **Indentation:** Submissions must have consistent indentation. This means that any two lines of code inside the same code block (functions, if statements, loops, etc.) must have the exact same indentation:

```
0 | void foo(int n)
1 | {
2 |     int i, a = 0, b = 1; // Lines 2 and 4 must have the same indentation
3 |
4 |     for (i = 0; i < 10; i++)
5 |     {
6 |         a++; // Both of these lines are in the same for loop,
7 |         b--; // so they have the same indentation.
8 |     }
9 | }
```

- **Indentation (Contd.):** When a nested code block is created (like a loop inside a function), all lines of code inside the nested code block should be indented one more time than the nested block's container (I know that sounds confusing, so take a look at the example below):

```
0 | void foo(int n) // This line has an indentation of 0 (no indentation)
1 | {
2 |     int i, a = 0, b = 1; // This line has an indentation of 1 because it is inside of foo
3 |
4 |     for (i = 0; i < 10; i++) // This line has an indentation of 1 because it is inside of foo
5 |     {
6 |         a++; // Both of these lines have an indentation of 2
7 |         b--; // because they are inside a for loop inside of foo.
8 |     }
9 | }
```