CS1010E: Programming Methodology

Assessed Lab 3A: Functions & Recursions [10%]

15 Mar 2017

Instructions

Please read all the instructions very carefully!

1. This is an **Open Book** assessment:
   - You are allowed to bring any printed materials and calculator
   - You are NOT allowed to use other electronic devices besides the lab's computer
   - You are NOT allowed to talk with your friends, to talk with invigilators please raise your hand
   - You are NOT allowed to access the internet except to the `plab` server via `SSH terminal`
2. This lab assessment consists of **one (1)** problems with several tasks:
   - The tasks are intended to guide you in solving the problem
   - Each task should have **its own separate file** where the task number is written at the back: `task3.c` is used for task 3
   - To proceed to the next level (*e.g., from task 2 to task 3*), copy your program using the command `cp task2.c task3.c`
   - Fill in your **Name**, **Matric** (*starts with* `A`), and **NUSNET ID** (*starts with either* `A` *or* `E`)
3. Numerical and precision guides:
   - **Two (2)** types of *input* numbers: `real` (*may have decimal point*) and `integer` (*no decimal point*)
   - `integer` may contain leading *zeroes*: always use `scanf("%d")` to ensure *decimal* representation
   - `integer` has a range of -2^{31} to $+2^{31}-1$, `unsigned integer` has a range of 0 to $+2^{32}-1$
   - Always use `double` for `real` number input for high precision, but numbers that differs by less than `0.001` are considered *equal*
4. Starting the tests:
   - Use the program `SSH Secure Shell Client`
   - Login to `plab` server using the given `username` and `password`
5. Testing and debugging guides:
   - You may open **two (2)** or more `SSH Terminal`: 1 for *coding* and 1 for *compilation + testing*
   - Assumption stated in the task is considered to always hold and no checking is necessary
   - Assumption NOT stated in the task will be tested in hidden input: *always think of worst case*
   - Test case outputs are organized by task number and test case number:
     - Task number `T` on test case number `C` have output file `testT_C.out`
     - *For example*: task number 2 with test case number 3 have output file `test2_3.out`
   - Test case inputs are the same for all tasks: *e.g.,* `test2.in`
6. Marking:
   - Grading is done *automatically* using CodeCrunch: only the largest correct task is considered
   - For instance: Task 1 is *empty (i.e., not done at all)*, Task 2 is *correct*, Task 3 is *incorrect*
     ↦ mark for Task 2 is taken
   - The mark for each task is given on the right side, it is a *cumulative* mark
7. Time management suggestion: [Total Time: **1 hour 30 minutes**]:
   - *Coding*: approx. **1 hour** (±**30 minutes** for debugging)
   - *Ending*: approx. last **5 minutes** ensures that you save the filename correctly

Multi-Factorial [10 %]

Problem Description

"Multifactorial is a generalization of the factorial and double factorial,

$$\begin{aligned} n! = n!^{(1)} &= n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 \\ n!! = n!^{(2)} &= n \times (n-2) \times (n-4) \times \ldots \\ n!!! = n!^{(3)} &= n \times (n-3) \times (n-6) \times \ldots \\ &\ldots \\ n!!\ldots!! = n!^{(k)} &= n \times (n-k) \times (n-2k) \times \ldots \end{aligned} \tag{1}$$

"etc., where the products run through positive integers." – Wolfram

Note that the value of $0!^{(k)} = 1$ for all possible k. Although $n!^{(k)}$ rises more slowly as k increases, the value can still be extremely large for sufficiently large n and k. As such, we will use *modular arithmetic* to force the result within the range of `integer`. Therefore, the multifactorial formula can be rewritten as in Formula 2.

$$n!_m^{(k)} = \big(n \times (n-k) \times (n-2k) \times \ldots\big) \bmod m \tag{2}$$

Since we will be using `modulo` operator in this problem, you need to be familiar with the theorems from modular arithmetic. The **two (2)** theorems necessary for modular arithmetic are reproduced below:

$$(x + y) \bmod m = ((x \bmod m) + (y \bmod m)) \bmod m \tag{3}$$

$$(x \times y) \bmod m = ((x \bmod m) \times (y \bmod m)) \bmod m \tag{4}$$

Concepts Tested:

1. Input/Output: `scanf` and `printf`
2. Modulo & Boolean Arithmetic: `%`, `||`, `&&`, `==`, etc
3. Selection Statement: `if` and/or `if-else`
4. Repetition Statement: `while` and/or `for` as well as *nested repetition*
5. Function: including *simple recursion*

Final Objective

Given three `integer` numbers n, k, and m find $n!_m^{(k)}$.

Example

The following table lists the values of the first few multifactorials for $n = 1, 2, 3, \ldots$

$n!$	$1, 2, 6, 24, 120, 720, \ldots$
$n!!$	$1, 2, 3, 8, 15, 48, 105, \ldots$
$n!!!$	$1, 2, 3, 4, 10, 18, 28, 80, 162, 280, \ldots$
$n!!!!$	$1, 2, 3, 4, 5, 12, 21, 32, 45, 120, \ldots$

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ $0 \leq$ `n` $\leq 2^{14} = 16384$ (*the value* n)
- ▷ $1 \leq$ `k` $\leq 2^{10} = 1024$ (*the value* k)
- ▷ $2 \leq$ `m` $\leq 2^{14} = 16384$ (*the value* m)

Tasks

The problem is split into 5 tasks with 4 number of testcases given. In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.
- If the test(s) give(s) **NO** message(s), it means your program is correct.

Task 1/5: ***[Input/Output]*** **[1%]**

Write a program to read three **integer** numbers n, k, and m, and print the value of $n \times k$ mod m.

Sample Run:

Inputs:	Outputs:
`7 3 11`	`10↩ | 21 mod 11`

Save your program in the file named `multifact1.c`. No submission is necessary.

Test your program using the following command(s):

```
./a.out < test1.in | diff - test1_1.out
./a.out < test2.in | diff - test1_2.out
./a.out < test3.in | diff - test1_3.out
./a.out < test4.in | diff - test1_4.out
```

To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp multifact1.c multifact2.c
```

Task 2/5: ***[Simple Recursion and/or Iteration]*** **[3%]**

Write a program to read three **integer** numbers n, k, and m, and print the value of $(n-k)$ mod m, $(n-k+1)$ mod m, ..., n mod m. Note that there is **NO** additional **[space]** at the end.

Hint:

- **Iteration:** what is the initial number, what is the final number, what is the update operation?
- **Recursion:** what is the base case, what is the recursive case, when and what to print?

Sample Run:

Inputs:	Outputs:
`7 3 11`	`4 5 6 7↩ | 4 mod 11, 5 mod 11, 6 mod 11, 7 mod 11`

Save your program in the file named `multifact2.c`. No submission is necessary.

Test your program using the following command(s):

```
./a.out < test1.in | diff - test2_1.out
./a.out < test2.in | diff - test2_2.out
./a.out < test3.in | diff - test2_3.out
./a.out < test4.in | diff - test2_4.out
```

To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp multifact2.c multifact3.c
```

Task 3/5: ***[Modular Arithmetic]*** **[6%]**

Write a program to read three **integer** numbers n, k, and m, and print the value of $\big(n \times (n-1) \times (n-2) \times \ldots \times (n-k)\big)$ mod m.

Hint:

- **Iteration:** what is the initial number, what is the final number, what is the update operation?
- **Recursion:** what is the base case, what is the recursive case, what to return?
- **Modulo:** where to insert the `modulo` operator according to Formula 3 and 4.

Sample Run:

Inputs:	Outputs:
`7 3 11`	`4↩ | 840 mod 11`

Save your program in the file named `multifact3.c`. No submission is necessary.

Test your program using the following command(s):

```
./a.out < test1.in | diff - test3_1.out
./a.out < test2.in | diff - test3_2.out
./a.out < test3.in | diff - test3_3.out
./a.out < test4.in | diff - test3_4.out
```

To proceed to the next task (*e.g., task 4*), copy your program using the following command:

```
cp multifact3.c multifact4.c
```

Task 4/5: ***[Factorial]*** **[8%]**

Write a program to read three `integer` numbers n, k, and m, and print the value of $\big(n \times (n-1) \times (n-2) \times \ldots \times \ldots \times 1\big) \bmod m$.

Hint:

- **Iteration:** what is the initial number, what is the final number, what is the update operation?
- **Recursion:** what is the base case, what is the recursive case, what to return?
- **Modulo:** where to insert the `modulo` operator according to Formula 3 and 4.
- **Fun Fact:** although *inconsequential* to your algorithm, note that $n!$ mod $m = 0$ **if** $n \geq m$.

Sample Run:

Inputs:	Outputs:
`7 3 11`	`2↩ | 7! mod 11 = 5040 mod 11`

Save your program in the file named `multifact4.c`. No submission is necessary.

Test your program using the following command(s):

```
./a.out < test1.in | diff - test4_1.out
./a.out < test2.in | diff - test4_2.out
./a.out < test3.in | diff - test4_3.out
./a.out < test4.in | diff - test4_4.out
```

To proceed to the next task (*e.g., task 5*), copy your program using the following command:

```
cp multifact4.c multifact5.c
```

Task 5/5: ***[Multi-Factorial]*** **[10%]**

Write a program to read three `integer` numbers n, k, and m, and print the value of $n!_m^{(k)}$.

Hint:

- **Iteration:** what is the initial number, what is the final number, what is the update operation?
- **Recursion:** what is the base case, what is the recursive case, what to return?
- **Modulo:** where to insert the `modulo` operator according to Formula 3 and 4.

Sample Run:

Inputs:	Outputs:
`7 3 11`	`6↩ | 7!!! mod 11 = 28 mod 11`

Save your program in the file named `multifact5.c`. No submission is necessary.

Test your program using the following command:

```
./a.out < test1.in | diff - test5_1.out
./a.out < test2.in | diff - test5_2.out
./a.out < test3.in | diff - test5_3.out
./a.out < test4.in | diff - test5_4.out
```

Useful VIM and SSH Terminal Commands

- **VIM Mode Switch:**
  - **i** i nsert (*from* Command)
  - **esc** esc ape to Command
- **Basic VIM Commands:** [mode=Command]
  - **:w** w rite file
  - **:q** q uit file
  - **:q!** q uit file (*forced*: *without saving*)
  - **:wq** w rite and q uit
- **Advanced VIM Commands:** [mode=Command]
  - **/text** find text
  - **n** find n ext text
  - **shift + n** find previous text
  - **gg=G** auto-indentation all lines
- **VIM Text Edit Commands:** [mode=Command]
  - **dd** d elete line at cursor (*cut*)
  - **yy** y ank line at cursor (*copy*)
  - **p** p aste after current cursor
  - **u** u ndo one change
  - **x** cut one character at cursor
  - **:red** red o undone changes
  - *N* **dd** d elete N lines down (N is number)
  - *N* **yy** y ank N lines down (N is number)
- **VIM Auto-Completion:** [mode=Insert]
  - **ctrl + n** complete word
  - **ctrl + x** complete line
- **Basic SSH Terminal Commands:**
  - **cd dir** open folder dir
  - **cd ..** open *parent* folder
  - **rm file** remove file file
  - **rm -r dir** remove folder dir
  - **vim file** open file in VIM
  - **ls** list files in folder
  - **ls -all** list ALL files in folder
  - **cat file** open *small* text file
  - **less -e file** open *large* text file
  - **cp f1 f2** copy f1 to f2
  - **mv f1 f2** move f1 to f2

    (*in effect, rename if in same folder*)
- **Execute Your Program in SSH Terminal :**
  - **gcc -Wall file** compile file
  - **gcc -Wall -lm file**

    compile file with math library (i.e. #define <math.h>) included
  - **./a.out** run program
  - **gcc -Wall file -o f1**

    compile file and rename executable into f1 (run using ./f1)
- **Advanced Program Execution Commands in SSH Terminal :**
  - **./a.out < f_in**

    run program with input redirection from file located at f_in

    (e.g. **./a.out < test1.in**)
  - **./a.out < f_in > f_out**

    run program with input redirection from file located at f_in and redirect the output to write into (*non-existing*) file called f_out

    (e.g. **./a.out < test1.in > output1**)
  - **diff f1 f2**

    compares the two files (f1 compared with f2) line by line (*note: no news is good news*)

    (e.g. **diff output1 test1_1.out**)
  - **./a.out < f_in | diff - f_out**

    run program with input from f_in immediately compare output with f_out

    (e.g. **./a.out < test1.in | diff - test3_1.out**)
- **SSH Terminal Emergency Commands:**
  - *Infinite loop* press **ctrl + c**
  - *End input* press **ctrl + d**

  (*better way is to use input redirection*)
- **VIM DO NOT DO LIST**
  - **ctrl + z** move to background

    (if done, type **fg** into **SSH Terminal**)
  - **ctrl + s** suspend

    (if done, press ctrl+q)
  - *Close without using* **:q**
    - on reopen, .swp file created
    - open file, choose Recover & exit VIM
    - open file again & choose Delete
- **GCC DO NOT DO LIST**
  - **gcc file -o file**

    compile file and rename into file (now, file is no longer a C program file)
    - ***pray hard...***
    - look for .file.history by typing **ls -all**
    - copy to windows using **SSH File Transfer**
    - ***hope*** latest code is at *end of file*