CS1010E: Programming Methodology

Assessed Lab 2: Selections & Repetitions [9%]

01 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

$3p + 1$ Problem [100 %]

Problem Description

3 Prime + 1 problem is a modified version of the $3n + 1$ problem. Its premise is similar to the $3n + 1$ problem except for one major difference as shown in the Formula 1.

$$p_{i+1} = \begin{cases} \dfrac{p}{2} & n_i \text{ divisible by } 2 \\ \text{Largest prime factor of } (3p+1) & \text{otherwise} \end{cases} \tag{1}$$

Prime numbers are defined as a number with only 1 and itself as the divisor. It is *conjectured*[1] that the sequence will always terminate with $p = 2$. In this exercise, we will *empirically* test such claim.

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*

Final Objective

Given **two (2)** numbers n and m such that $n > m$, print the number of steps required to reach $p = 2$ when starting with $p = k$ such that $k = n, n + 1, n + 2, ..., m$.

Example

The sequence generated by Formula 1 when $p = 6$ are: $6 \rightarrow 3 \rightarrow 5 \rightarrow 2$. Thus, it takes 3 steps.
The sequence generated by Formula 1 when $p = 7$ are: $7 \rightarrow 11 \rightarrow 17 \rightarrow 13 \rightarrow 5 \rightarrow 2$. Thus, it takes 5 steps.

Assumptions

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

▷ 2 $\leq$ n $\leq$ 999 (*the value of* n)
▷ n+1 $\leq$ m $\leq$ 1000 (*the value of* m)

Tasks

The problem is split into 5 tasks with 3 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.

[1] The conjecture is from me, I tested that it terminates for $2 \leq p \leq 1000$. Note how the termination criteria is *exactly* the same as in the usual $3n + 1$ problem. It will terminate if the number p can be expressed as $p = 2^k$ for some value of k. The largest value in both cases is the same $3n + 1$ which happens when $3n + 1$ is exactly a prime number. Other than that, the value will be smaller than $3n + 1$. In fact, I *strongly* believe that this sequence will have fewer number of steps than the original $3n + 1$ for all values $p > 17$. – Adi

Task 1/5: *[Input/Output]* [10%]

Write a program to read **two (2)** `integer` numbers n and m and print the numbers back.

Sample Run:

Inputs:	Outputs:
`6 7`	`6 7`↩

Save your program in the file named `3p11.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
```

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

```
cp 3p11.c 3p12.c
```

Task 2/5: *[Simple Loop]* [30%]

Write a program to read **two (2)** `integer` numbers n and m and print all the numbers between n and m (*inclusive*) in a single line. Note that there is **NO** additional `[space]` at the end.

Sample Run:

Inputs:	Outputs:
`6 7`	`6 7`↩

Save your program in the file named `3p12.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
```

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

```
cp 3p12.c 3p13.c
```

Task 3/5: *[Nested Loop]* [60%]

Write a program to read **two (2)** `integer` numbers n and m and print all the *prime* numbers between n and m (*inclusive*) in a single line. Note that there **IS** an additional `[space]` at the end.

Sample Run:

Inputs:	Outputs:
`6 7`	`7 `↩

Save your program in the file named `3p13.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
```

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

```
cp 3p13.c 3p14.c
```

Task 4/5: *[Advanced Nested Loop]* [90%]

Write a program to read **two (2)** `integer` numbers n and m and print the number of steps required for the sequence generated by Formula 1 to reach 2 when starting with $p = n + m$.

Sample Run:

Inputs:

```
6 7
```

Outputs:

```
2↩ | 13 -> 5 -> 2
```

Save your program in the file named `3p14.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
```

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

```
cp 3p14.c 3p15.c
```

Task 5/5: *[*$3p + 1$*]* [100%]

Write a program to read **two (2)** `integer` numbers n and m and print the number of steps required for the sequence generated by Formula 1 to reach 2 when starting with $p = k$ such that $k = n, n+1, n+2, ..., m$. Note that there is **NO** additional `[space]` at the end.

Sample Run:

Inputs:

```
6 7
```

Outputs:

```
3 5↩ | p=6: 6->3->5->2, p=7: 7->11->17->13->5->2
```

Save your program in the file named `3p15.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
```

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*