CS1010E: Programming Methodology

Assessed Lab 1B: Operations [8%]

8 February 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

Half-Round Odd [8 %]

Problem Description

“A tie-breaking rule that is less biased is round half to odd. By this convention, if the fraction of y is 0.5, then q is the odd integer nearest to y. Thus, for example, +23.5 becomes +23, as does +22.5; while 23.5 becomes 23, as does 22.5.

“This method also treats positive and negative values symmetrically, and is therefore free of sign bias. More importantly, for reasonable distributions of y values, the average value of the rounded numbers is the same as that of the original numbers. However, this rule will introduce a towards-zero bias when y 0.5 is odd, and a towards-infinity bias for when it is even.

“This variant is almost never used in computations, except in situations where one wants to avoid rounding 0.5 or 0.5 to zero; or to avoid increasing the scale of floating point numbers, which have a limited exponent range. With round half to even, a non infinite number would round to infinity, and a small denormal value would round to a normal non-zero value. Effectively, this mode prefers preserving the existing scale of tie numbers, avoiding out of range results when possible for even based number systems (such as binary and decimal).” – Wikipedia

`integer` division operation in C can only perform *truncation* of the decimal point. We will explore the possibility of performing a *half-round odd* rounding mechanims.

Extension of such mechanism uses the ground value other than `1`. We will use ground value of `10` with half of `10` is `5`. Hence, any `integer` value ending in `5` is rounded up to the nearest odd multiple of `10`.

Final Objective

Given a value, perform the half-round odd without using any selection statement such as `if`, `switch`, or `?:` operator and repetition statement such as `while`, `do-while`, or `for`. However, you may use

Example

Table 1 below shows the half-round odd mechanism and two simpler mechanisms called the half-round up and truncation (half-round zero). Important differences are highlighted. Note that C `integer` division is performing truncation by default.

Value	Half-Round Zero	Half-Round Up	Half-Round Odd
10	10	10	10
14	10	10	10
15	10	20	10
16	10	20	20
24	20	20	20
25	20	30	30
26	20	30	30
34	30	30	30
35	30	40	30
36	30	40	40
44	40	40	40
45	40	50	50
46	40	50	50
54	50	50	50
55	50	60	50
56	50	60	60

Table 1: Summary of rounding of important values.

Assumptions

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

▷ $10 \leq$ `value` $\leq 2^{30}$ (*the value to be rounded odd*)

Restrictions

The following restriction(s) is/are imposed on the solution:

▷ You cannot use selection statements such as (*but not limited to*) **`if`**, **`switch`**, or `?:` operator

▷ You cannot use repetition statements such as (*but not limited to*) **`while`**, **`do-while`**, or **`for`**

▷ You cannot use `<math.h>` library

Tasks

The problem is split into 5 tasks. 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.

Task 1/5: ***[Half-Round Zero]*** **[1%]**

Write a program that reads an **`integer`** and print the given number half-rounded zero. You may **not** use selection/repetition statement in this task. *Hint*:

- Let `num` be the number
- Consider `m = num / 10` by **`integer`** division
- What is the relationship between `m` and the answer?

Sample Run:

Inputs:	Outputs:
`14`	`10`↩

Sample Run:

Inputs:	Outputs:
`15`	`10`↩

Sample Run:

Inputs:	Outputs:
`19`	`10`↩

Sample Run:

Inputs:	Outputs:
`20`	`20`↩

Save your program in the file named `roundodd1.c`. No submission is necessary.

Test your program using the following command: `./a.out < test1.in | diff - test1_1.out`

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

```
cp roundodd1.c roundodd2.c
```

Task 2/5: *[Half-Round Up]* [2%]

Write a program that reads an `integer` and print the given number half-rounded up. You may **not** use selection/repetition statement in this task. *Hint*:

- Let `num` be the number
- Consider `m = num / 10` by `integer` division
- Consider `m = (num + d) / 10` by `integer` division, for some value of `d`

Sample Run:

Inputs:

```
15
```

Outputs:

```
20↩ | round up
```

Sample Run:

Inputs:

```
14
```

Outputs:

```
10↩ | round down
```

Sample Run:

Inputs:

```
30
```

Outputs:

```
30↩ | no rounding
```

Save your program in the file named `roundodd2.c`. No submission is necessary.
Test your program using the following command: `./a.out < test1.in | diff - test2_1.out`
To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp roundodd2.c roundodd3.c
```

Task 3/5: *[Even Number]* [5%]

Write a program that reads an `integer` and print the number half-rounded up if the second last digit is *even* and print `0` is the second last digit is *odd*. Note that `0` is an even number. You may **not** use selection/repetition statement in this task. *Hint*:

- Let `m` be the result of Task 2
- Let `digit` be the 2^{nd} last digit
- Consider *modulo* operation on `digit`
- Consider *multiplication* with `m`

Sample Run:

Inputs:

```
5
```

Outputs:

```
10↩ |  5 -> second last: 0 -> even -> UP
```

Sample Run:

Inputs:

```
15
```

Outputs:

```
0↩  | 15 -> second last: 1 -> odd  -> ZERO
```

Sample Run:

Inputs:

```
24
```

Outputs:

```
20↩ | 24 -> second last: 2 -> even -> DOWN
```

Save your program in the file named `roundodd3.c`. No submission is necessary.
Test your program using the following command: `./a.out < test1.in | diff - test3_1.out`
To proceed to the next task (*e.g., task 4*), copy your program using the following command:

```
cp roundodd3.c roundodd4.c
```

Task 4/5: ***[Check Digit]*** **[7%]**

Write a program that reads an `integer` and the number half-rounded up if the second last digit is *even* **and** the last digit is `5`. Otherwise, print `0`. You may **not** use selection/repetition statement in this task. *Hint*:

- Let `m` be the result of Task 3
- Let `last` be the last digit
- Check if the last digit is `5`:
  - Let `lower` be `0` when `last < 5` and `1` when `last >= 5`
    Consider `integer` division operation `last / div` for some value of `div`
  - Let `upper` be `1` when `last <= 5` and `0` when `last > 5`
    Consider `integer` division operation `(d - last) / div` for some value of `d` and `div`
  - Consider `lower * upper`, what value of `last` will it be `1`?
- Consider *multiplication* with `m`

Sample Run:

Inputs: Outputs:

```
14                        0↩   | last digit not 5, second last odd
```

Sample Run:

Inputs: Outputs:

```
15                        0↩   | last digit     5, second last odd
```

Sample Run:

Inputs: Outputs:

```
24                        0↩   | last digit not 5, second last even
```

Sample Run:

Inputs: Outputs:

```
25                        30↩  | last digit     5, second last even
```

Save your program in the file named `roundodd4.c`. No submission is necessary.

Test your program using the following command: `./a.out < test1.in | diff - test4_1.out`

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

```
cp roundodd4.c roundodd5.c
```

Task 5/5: ***[Half-Round Odd]*** [8%]

Write a program that reads an **`integer`** and print the given number half-rounded odd. You may **not** use selection/repetition statement in this task. *Hint*:

- Let `round` be the result of half-rounded up from Task 2
- Let `even` be `1` if second last digit is *even* and `0` if it is *odd modified* from Task 3
- Let `five` be `1` if last digit is `5` and `0` if not `5` *modified* from Task 4
- Let `cond = (1 - even) * five`
- Result is `((cond * (round - 1)) + ((1 - cond) * round)) * 10`

Sample Run:

Inputs: Outputs:

```
15                        10↩ | round odd down
```

Sample Run:

Inputs: Outputs:

```
16                        20↩ | round up
```

Sample Run:

Inputs: Outputs:

```
24                        20↩ | round down
```

Sample Run:

Inputs: Outputs:

```
25                        30↩ | round odd up
```

Sample Run:

Inputs: Outputs:

```
35                        30↩ | round odd down
```

Save your program in the file named `roundodd5.c`. No submission is necessary.

Test your program using the following command: `./a.out < test1.in | diff - test5_1.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*