CS1010E: Programming Methodology

Assessed Lab 4B: Array [11%]

29 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

Priority Queue [11 %]

Problem Description

"In computer science, a priority queue is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue." – Wikipedia

In this problem we will implement a queue that contains people of high-priority. The queue will consists of a *group* of people. Some groups are designated high-priority or *VIPs*. The group will only *act* as a whole and not individually. Therefore,**two (2)** queues are necessary: 1) *Normal* queue and 2) *VIP* queue.

Once in a while, a bus will arrive at the queue. The bus will take in people from the *VIP* queue until the front of the *VIP* queue can no longer fit into the bus. We will ignore the remainder of the *VIP* queue even when there are still smaller groups behind. The bus will then take in people from the *Normal* queue until the front of the *Normal* queue can no longer fit into the bus. We will *again* ignore the remainder of the *Normal* queue even when there are still smaller groups behind.

The input to this *simulation* will be given as a sequence of Passenger (*encoded with* `'P'`) and a sequence of Buses (*encoded with* `'B'`). The following is an example of a simulation:

```
P 4 12  | VIP:  4, Normal: 12     Q1 = {4}     Q2 = {12}
B 20    | Bus cap: 20             Q1 = {}      Q2 = {}
P 10 6  | VIP: 10, Normal:  6     Q1 = {10}    Q2 = {6}
P 12 8  | VIP: 12, Normal:  8     Q1 = {10,12} Q2 = {6,8}
B 20    | Bus cap: 20             Q1 = {10}    Q2 = {8}
P 0 12  | VIP:  0, Normal: 12     Q1 = {10}    Q2 = {8,12}
P 2 0   | VIP:  2, Normal:  0     Q1 = {10,2}  Q2 = {8,12}
B 9     | Bus cap:  9             Q1 = {10,2}  Q2 = {12}
B 23    | Bus cap: 23             Q1 = {}      Q2 = {12}
```

The simulation example is simply to illustrate the commands. The actual encoding used for Bus is the number `0` while for Passenger is the number `1`. At the end of this simulation, 0 VIPs and 12 passengers are not in the bus. However, this is not our concern as a simulator.

NOTE: removal of groups from any queue can be done via shift-left operation on the queue array.

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*
6. Arrays: including *2D arrays*

Final Objective

Given a sequence of simulation commands (*i.e.,* `"B #"` or `"P # #"`), find out how many passengers are stranded at the terminal at the end of the day.

Assumptions

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

- ▷ $1 \leq$ `number(commands)` ≤ 100 (*the number of commands in the sequence*)
- ▷ $1 \leq$ `passenger` ≤ 100 (*the passengers group size*)
- ▷ $1 \leq$ `bus_size` ≤ 100 (*the bus size*)

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 the sequence of simulation commands and print the simulation commands back. The input sequence is given as:

- The first line consists of **one (1)** `integer` number n corresponding to the number of simulation commands
- The next n lines consists of the simulation commands:
  - Bus Commands: has the format `'0 #'` where `'#'` is an `integer` number
  - Passenger Commands: has the format `'1 # #'` where `'#'` are `integer` numbers

Hint:

- Use multiple `printf` for `integer`
- Read one more `integer` for `'0'` and two more `integer` for `'1'`, differentiated by `if-else`

Sample Run:

Inputs:

```
5
1 0 8
0 15
1 6 5
1 5 1
0 10
```

Outputs:

```
1 0 8↩
0 15↩
1 6 5↩
1 5 1↩
0 10↩
```

Save your program in the file named `priority1.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 priority1.c priority2.c
```

Task 2/5: ***[Counting]*** **[3%]**

Write a program to read the sequence of simulation commands and print total number of both VIPs and normal passengers. The input sequence is given as:

- The first line consists of **one (1)** `integer` number n corresponding to the number of simulation commands
- The next n lines consists of the simulation commands:
  - Bus Commands: has the format `'0 #'` where `'#'` is an `integer` number
  - Passenger Commands: has the format `'1 # #'` where `'#'` are `integer` numbers

Hint:

- Use two accumulators

Sample Run:

Inputs:

```
5
1 0 8
0 15
1 6 5
1 5 1
0 10
```

Outputs:

```
25↩ | 0+8+6+5+5+1=31 passengers
```

Save your program in the file named `priority2.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 priority2.c priority3.c
```

Task 3/5: ***[Ordering]*** **[7%]**

Write a program to read the sequence of simulation commands and print the non-zero series of total passenger (*combination of both VIP and Normal*) in order of arrival. The input sequence is given as:

- The first line consists of **one (1)** `integer` number n corresponding to the number of simulation commands
- The next n lines consists of the simulation commands:
  - Bus Commands: has the format `'0 #'` where `'#'` is an `integer` number
  - Passenger Commands: has the format `'1 # #'` where `'#'` are `integer` numbers

Hint:

- Use index to indicate where to insert the next element
- Note that there is **NO** an additional `[space]` at the end

Sample Run:

Inputs:

```
5
1 0 8
0 15
1 6 5
1 5 1
0 10
```

Outputs:

```
8 11 6↩ | passengers: [0+8], [5+6], [5+1]
```

Save your program in the file named `priority3.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 priority3.c priority4.c
```

Task 4/5: ***[Double Ordering]*** **[9%]**

Write a program to read the sequence of simulation commands and print the non-zero series of VIP passenger and non-sero series of normal passengers in order of arrival. The input sequence is given as:

- The first line consists of **one (1)** `integer` number n corresponding to the number of simulation commands
- The next n lines consists of the simulation commands:
  - Bus Commands: has the format `'0 #'` where `'#'` is an `integer` number
  - Passenger Commands: has the format `'1 # #'` where `'#'` are `integer` numbers

Hint:

- Use two indexes for two arrays
- Note that there is **NO** an additional `[space]` at the end

Sample Run:

Inputs:

```
5
1 0 8
0 15
1 6 5
1 5 1
0 10
```

Outputs:

```
6 5↩    | VIP passengers
8 5 1↩  | normal passengers
```

Save your program in the file named `priority4.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 priority4.c priority5.c
```

Task 5/5: ***[Priority Queue]*** **[11%]**

Write a program to read the sequence of simulation commands and print the number of stranded passengers at the end of the day. The input sequence is given as:

- The first line consists of **one (1)** `integer` number n corresponding to the number of simulation commands
- The next n lines consists of the simulation commands:
  - Bus Commands: has the format `'0 #'` where `'#'` is an `integer` number
  - Passenger Commands: has the format `'1 # #'` where `'#'` are `integer` numbers

Sample Run:

Inputs:

```
5
1 0 8
0 15
1 6 5
1 5 1
0 10
```

Outputs:

```
5 6↩ | 5 VIPs and 6 normal passengers stranded
```

Sample Run:

Inputs:

```
5
1 0 8
0 15
1 0 5
1 5 0
0 10
```

Outputs:

```
0 0↩ | no one stranded
```

Save your program in the file named `priority5.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*