ENGR30003: Numerical Programming for Engineers
Assignment 1 Notes, Semester 2, 2017

August 13, 2017

This document details important points you should consider when doing your assignment. If something is not clear, ask your tutors during the workshops or ask your question on the Discussion Board, under the Assignment 1 Questions. Any unresolved issues to be directed to head tutor Chitrarth Lav (`clav@student.unimelb.edu.au`)

Points to Consider

- Only use type `int` and `float` for integers and floating point numbers respectively. When writing out a float to output make sure the format is `%.6f` to restrict it to a 6 decimal place number. Writing to file needs to be done according to the format given in the assignment to be marked correct by the system.
- Each task is independent of the other so you can work on each task individually and test it out. Make sure you adhere to the formatting and output file headers as shown in the Assignment.
- Read the data into an appropriate data structure for each task. You may choose to read it in once and reuse this structure for all tasks but the task functions might need to be modified.
- Write the header for each file as described in the assignment. Your solution would be marked wrong by the system even if you have the right solution if you get your headers wrong.
- Your code should not contain any magic numbers. Examples include but are not limited to using `malloc(1000 * sizeof(int))` or `for (i = 0 ; i < 20; i++)`. Use `#define` at the top of the file to define 1000 and 20.
- Remember to free your data structure and any other structure you allocate dynamically. Statically allocated structures are not allowed.

Tasks

- Process the command line arguments (file pointer and resolution) in `main.c`.
- Implement the timing functionality to estimate time for each task in `main.c`. You may reuse the timing functionality from `gettimeofday.c` provided on LMS.

- **Task 1**:
  - Only look at points (lines in `flow_data.csv`) where the `x` coordinate is greater than 20.
  - Maximum difference between two numbers in a range is when you take the difference between the maximum and the minimum numbers from the range.
  - Write out the point with the maximum value first, followed by the point with the minimum value next. Do this first for `u` and then for `v` in the output file.
- **Task 2**:
  - Crux of the problem is to identify the points which fall into a given cell. Find the floor and ceiling of any given cell in both `x` and `y` and use these to find the points in the chosen cell to compute the average point.
  - To test if your sorting works correctly, try writing two files, one before sorting and one after sorting. Choose any line at random from the sorted file and search for it in the unsorted file. If this random point doesn't exist, it is likely you sorted only S and did not keep track of the `x,y,u,v` for a given S.
  - The final version of your code must output only the sorted version.
- **Task 3**:
  - The larger the threshold, the more points it would contain.
  - The % is to be calculated by taking the ratio of points under a certain threshold and the points under the maximum threshold.
- **Task 4**:
  - Only look through the points where the `x` coordinate is relevant to choose the maximum `u`.
  - The magnitude of `y` coordinate chosen should be the output to file.
  - With increasing `x`, the value of `y` should be increasing. An example of what you can expect the written wake profile to look like is shown in Figure 1.

Compiling and Running

- When you are done writing and want to test out your implementation, it is highly recommended that you compile and run the code on `dimefox`. For that you would need to transfer your code files to the server.
- If you are a Mac or Linux user, you can transfer files using the following command:

  ```
  scp main.c username@dimefox.eng.unimelb.edu.au:
  ```

  where you are transferring `main.c` to your home drive on `dimefox`. If you are a Windows user, you need to use either a file transfer client, like FileZilla or WinSCP or you can install a terminal capable of transferring such as MobaXterm. If you have neither installed and don't know how, see your tutors in the workshop and get it fixed as soon as possible.

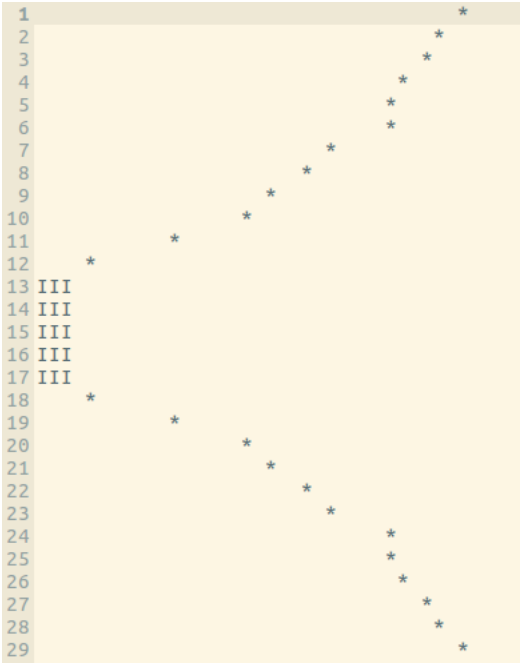


Figure 1: Output from `task4_2.txt`

- To compile and run your code on `dimefox`, use the following command:

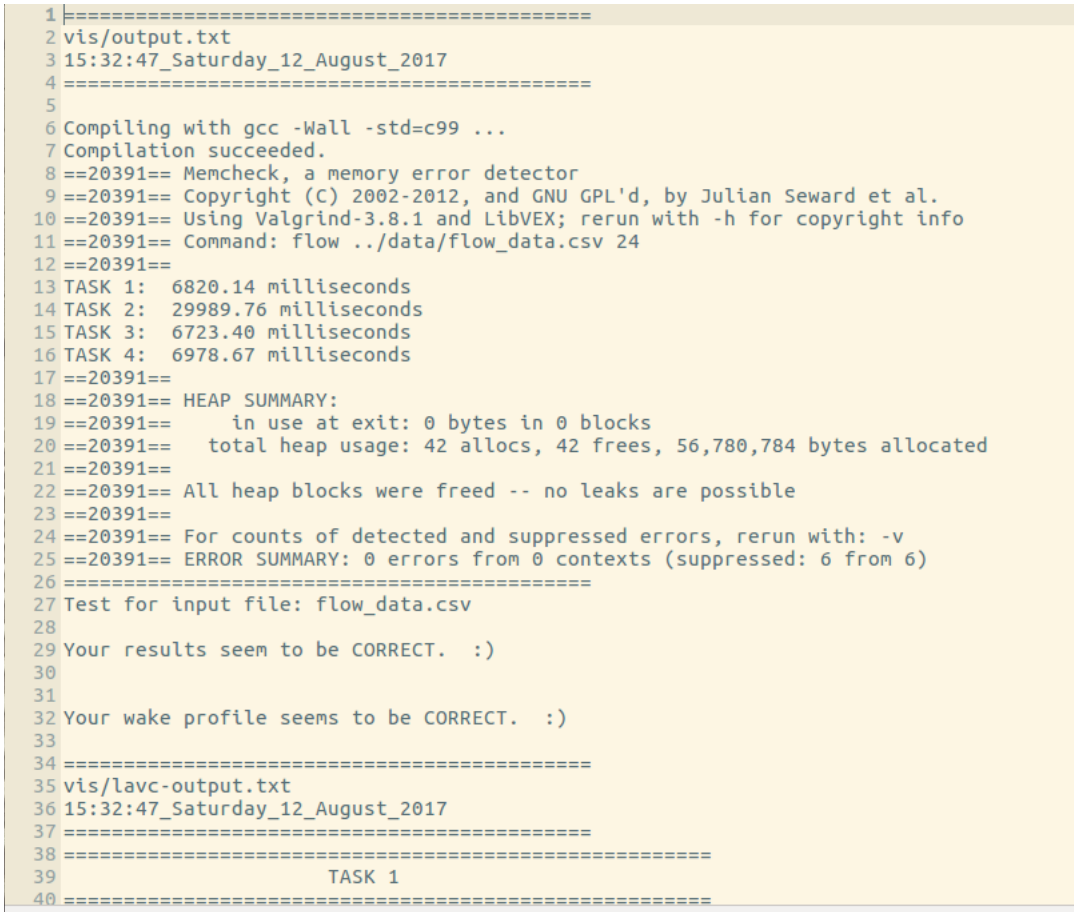
  ```
  gcc -Wall -std=c99 *.c -o flow -lm
  valgrind ./flow flow_data.csv 24
  ```

  You should have no warnings or errors when you compile. Also executing with `valgrind` should be done often to check if you are allocating and freeing memory properly.

Submission and Verifying

As mentioned in the Assignment, you will be using the command `submit` on `dimefox` to submit your assignment and `verify` to check the result of the submission. Here are some key points you should watch out for:

- When you submit your work, make sure you submit all .c and .h files. Sometimes students end up submitting only the .c files and missing header files would result in compilation error.
- The `submit ENGR30003 A1 *.c *.h` means you are submitting for the course `ENGR30003` under the project `A1`. It will not work if you end up using this command to submit after the deadline. To submit late assignments you must use: `submit ENGR30003 A1.late *.c *.h`
- When you want to verify if your submission was successful, try using `verify ENGR30003 A1 > feedback.txt`. You can check the contents of the file with either `nano` or with `more`. The format of a successfully submitted `feedback.txt` would look as Figure 2.



```
1  ==========================================
2  vis/output.txt
3  15:32:47_Saturday_12_August_2017
4  ==========================================
5
6  Compiling with gcc -Wall -std=c99 ...
7  Compilation succeeded.
8  ==20391== Memcheck, a memory error detector
9  ==20391== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
10 ==20391== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
11 ==20391== Command: flow ../data/flow_data.csv 24
12 ==20391==
13 TASK 1:  6820.14 milliseconds
14 TASK 2:  29989.76 milliseconds
15 TASK 3:  6723.40 milliseconds
16 TASK 4:  6978.67 milliseconds
17 ==20391==
18 ==20391== HEAP SUMMARY:
19 ==20391==     in use at exit: 0 bytes in 0 blocks
20 ==20391==   total heap usage: 42 allocs, 42 frees, 56,780,784 bytes allocated
21 ==20391==
22 ==20391== All heap blocks were freed -- no leaks are possible
23 ==20391==
24 ==20391== For counts of detected and suppressed errors, rerun with: -v
25 ==20391== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
26 ==========================================
27 Test for input file: flow_data.csv
28
29 Your results seem to be CORRECT.  :)
30
31
32 Your wake profile seems to be CORRECT.  :)
33
34 ==========================================
35 vis/lavc-output.txt
36 15:32:47_Saturday_12_August_2017
37 ==========================================
38 ======================================================
39                       TASK 1
40 ======================================================
```

Figure 2: Sample `feedback.txt`

Line 7 shows that the program compiled without warning or errors; lines 13-16 show the execution of the program with the output of the timings; lines 19,20 and 22 show that there were no memory leaks in the program; lines 29 and 32 show that the output from the sample solution seems to match the correct solution. There is no guarantee that you have the correct solution if this message is relayed, however if it says the result is incorrect, you definitely have the wrong answers. All you can do is make sure this message is displayed when you check your `feedback.txt` *and then pray to the* `dimefox` *gods to accept your offering as the one true solution.*

Marking Scheme

This section details how your code implementation is going to be marked.

Task 1: Maximum Velocity Difference [5/25 Marks]

- Read the data correctly without memory leaks [1 Marks]
- Insert the data into an appropriate data structure [1 marks]
- Efficiently compute the correct solution [2 marks]
- Output the correct results and timings [1 marks]

Task 2: Mean Velocities on a Coarser Grid [8/25 Marks]

- Read the data correctly without memory leaks and put data into the relevant data structure [1 mark]
- Choose a relevant data structure for the coarse grid [1 mark]
- Efficiently track number of points in each cell and compute the average [2 marks]
- Efficiently compute the score correctly for each cell [1 mark]
- Use of relevant comparison function for sorting [1 mark]
- Output the sorted results and timings [2 marks]

Task 3: Velocity and It's Statistics [5/25 Marks]

- Read the data correctly without memory leaks and put data into the relevant data structure [1 mark]
- Efficiently compute the correct points and percentage under each threshold [3 marks]
- Output the correct results and timings [1 mark]

Task 4: Wake Profile Visualization [4/25 Marks]

- Read the data correctly without memory leaks and put data into the relevant data structure [1 mark]
- Choose correct `x` coordinate closest to given `x` [0.5 marks]
- Efficiently compute the `y` heights [1 mark]
- Output the obtained `x` location and the corresponding `y` height correctly [0.5 marks]
- Efficiently compute the spacing for each height [0.5 marks]
- Output the right wake profile [0.5 marks]

Implementation [3/25 Marks]

- Compiler warnings [-1 marks]
- Code is not readable (lots of uncommented stuff, bad formatting) [-1 marks]
- Magic numbers in the code [-1 marks]
- Incorrect or no exception handling (no usage of `assert` to check) [-1 marks]
- Variable size arrays on the stack [-1 marks]
- Incorrect command line parameter parsing [-1 marks]
- No or insufficient authorship information [-0.5 marks]

- No or insufficient comments [-1 marks]
- Minor code quality problems [-0.5 marks]
- Some code quality problems [-1 marks]
- Major code quality problems [-2 marks]
- Serious code quality problems [-3 marks]