Assignment 1

Important

Marks: This assignment is worth 25% of the overall assessment for this course.

Due Date and time: Monday, 07 September 2020, 11:59pm, via submit on Canvas. You will lose penalty marks at the rate of 10% per day or part day late, and the assignment will not be marked if it is more than 5 days late (counted from the due date/time).

Learning outcomes

The objective of this assignment is to train the student to methodologically process a given numerical data. The student shall demonstrate his/her understanding and utilisation of dynamic data structures and basic numerical computation.

Flow past a flat plate

Let us consider the flow past a flat plate which is aligned perpendicular to the mean flow direction as shown in the schematics in figure 1a. The blue arrows indicate the direction of the flow while the shaded object is the flat plate. The flow generates a wake behind the plate and exerts a pressure force upon it, similar to the force you feel when you hold your hand out while in a fast moving car. This canonical flow configuration is highly relevant in the field of fluid mechanics and understanding the flow dynamics behind the plate is of paramount importance. With the aid of high performance computing, it is possible to simulate the complex evolution of the flow with a greater deal of accuracy as shown in figure 1b.

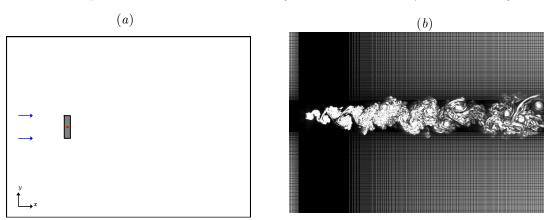


Figure 1: (a) Schematics of flow problem (b) Sample instantaneous flow field behind a flat plate superposed on the simulation grid.

For this assignment, you will process the wake data from the flat plate case. The data is provided to you in a CSV format file $(flow_data.csv)$ in the following order:

```
x,y,u,v,rho
-15.000000,-20.000000,1.051752,0.000000,0.950795
-14.602526,-20.000000,1.052630,0.001555,0.950744
-14.210199,-20.000000,1.052996,0.002728,0.950593
-13.823164,-20.000000,1.053247,0.003664,0.950348
-13.441554,-20.000000,1.053328,0.004340,0.950158
```

Each line corresponds to a point in the flow domain with coordinates (x,y). rho is the density and u and v are the fluid velocity components at that given point (u is the x component of the velocity vector while v is the y component).

This assignment consists of four tasks which will be assessed independently. For each task you are to measure the run time it takes to complete the described task using your program (see program output

Marks: 2 of 25

Marks: 5 of 25

below). Each of the four tasks must not require more than 60 seconds to run on dimefox. This means, in order to complete the task within this time limit, you may need to focus on the efficiency of your solution for each problem. Overall you have to write a single program, performing each of the four tasks sequentially. For each task you have to write your results to a file.

Task 1: Maximum/minimum flux

In this task, you shall compute the maximum and minimum flux (the maximum and minimum of rho*u and rho*v) for the data points after coordinate x = 20. You shall output first the point where the magnitude of rho*u is maximum followed by the minimum value of the magnitude of rho*u. Then you do the same for rho*v. The output should be written to a file named task1.csv and should be formatted as follows (this is just a sample how the file should look like and is not the answer):

```
x,y,u,v,rho
40.512346,-19.595387,1.007986,-0.001002,1.243862
66.899192,-0.729056,0.850117,0.000580,1.014812
69.552467,-0.729056,0.852483,0.000427,1.143862
60.961891,0.442134,0.838355,-0.000633,1.138645
```

Note: There must be no blank spaces between the values and around the commas. Each value must be written to 6 decimal places. You must write your output in exactly the same way as described. Failing to do so will result in error when comparing your output to the solution and you would lose marks. You should not assume that the data provided in flow_data.csv is in any chronological order and you must efficiently look only at points where the value of x is greater than 20. You can use file_io.c to understand how to output data to a file.

Task 2: Mean velocities on a coarser grid

Each line in the file $flow_data.csv$ is a point location in the domain. These points when joined together will create a mesh. For this task, you shall map these points onto a coarser grid, computing the new average coordinates (x,y) and the corresponding mean velocities u,v. The flow domain can be thought as divided into a two-dimensional grid such that each cell of the grid would contain multiple points, the number of which would depend on the cell upper and lower dimensions and the coordinates of the points. You shall compute the average coordinates and velocities for each cell using the formula below for all points k within a given cell:

$$q_{av} = \frac{1}{k} \sum_{i=0}^{k-1} q_i$$

where q represents x, y, u and v. While computing the averages, you must ignore any data points that lie on the coarse cell boundaries, i.e., consider only points that fall inside the given cell. The resulting averaged values of the given cell can be obtained from a score S, computed by

$$S = 100 \frac{\sqrt{u_{av}^2 + v_{av}^2}}{\sqrt{x_{av}^2 + y_{av}^2}}$$

Output the average values and the score S to a file **task2.csv** in descending S order for each cell. An example of what the output should look like is shown below. There must be no blank spaces between the values and around the commas and values must be written to 6 decimal places as shown:

```
S,x,y,u,v

1.186624,69.842187,5.861905,0.831445,0.019688

1.090734,79.552381,9.923571,0.874429,-0.001291

1.088278,79.552381,5.861905,0.868093,-0.003071

1.074177,79.552381,-9.886786,0.861106,-0.001160

1.070231,79.552381,14.017391,0.864510,0.000087
```

The size of the grid (number of cells in each direction) must be an input parameter, allowing the code to run different grid sizes. Your implementation would be checked for the grid resolution of 10 i.e. 10 cells in x and 10 cells in y. The domain extent for this coarse grid in x and y is -10 to 80 units and -18

Marks: 8 of 25

to 18 units respectively. Points outside these boundaries will be discarded. An example of the coarse grid is shown in Figure 2 (left). The 10 cells in x direction would span from -10 to 80 units while the 10 cells in y direction would span -18 to 18 units. The red area represents the zone in which the existing points must be discarded. Also shown is an example of a cell within this grid. As can be seen, the cell is of width Δx and height Δy and the black dots show the points in the original grid. Once you do the averaging for all these points, you will end up with one average point (shown in red).

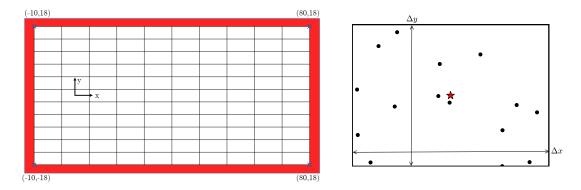


Figure 2: (left) A coarse grid with minimum and maximum extent shown by the coordinates. (right) Sample cell where the black dots show the original points and the star shows the location of the average.

Task 3: Searching in data structures

For this task, you will implement three data structures: an array, a Linked List and a perfectly balanced Binary Search Tree. The data contained in these structures will be the same; the aim is to perform a search operation and output the time taken to search in each case. The data required for this task is a subset of the data in <code>flow_data.csv</code>. First, you must pull out the data points at the domain centreline, i.e. points where <code>y=0.0</code>, and store these points in an array. Next, you must sort the data points in ascending order of the flux <code>rho*v</code>. Then, you must insert the sorted values into a Linked List and in a perfectly balanced BST. Finally, you must search for the data point in all the three data structures whose <code>rho*v</code> value is the closest to <code>75%</code> of the maximum <code>rho*v</code> flux. First find out the value which is closest to the <code>75%</code> of maximum <code>rho*v</code> in the array and then search for this value in the Linked List and BST data structures. Output your result into a file <code>task3.csv</code> in the following order

- 1. Linear search on the array (first line)
- 2. Binary search on the array (second line)
- 3. Linear search on the Linked List (third line)
- 4. Search on the balanced BST (fourth line)

In all cases, write all **rho*v** values up to and including the value that is being searched. A sample output (not the actual answer) for the **task3.csv** file is shown below, where each line (total of 4 lines) corresponds to the cases in the order shown above:

```
3.614885,3.564577,3.562721,3.544763,3.542255,3.489632,3.488729
3.614885,3.564577,3.544763,3.542255,3.488729
3.614885,3.564577,3.562721,3.544763,3.542255,3.489632,3.488729
3.614885,3.564577,3.544763,3.542255,3.488729
```

The search time for all four cases must be written to standard output, in the format described in the "Program output" section. Here, perfectly balanced refers to nearly perfectly balanced i.e. there may be uneven branches but the length of these paths should not differ by more than one node depth. The search algorithm in all cases must terminate when the exact value is identified and the last value written out must be this value.

Marks: 5 of 25

Task 4: Computing the vorticity

One of the quantities of interest for engineers to study is vorticity, which represents the rotation of the velocities about an axis. The vorticity for the given data can be defined as follows:

$$\omega = \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)$$

To compute the vorticity, there are several steps you have to follow. Since each line of the total n*m lines of the data in $flow_data.csv$ represents a point in the domain, you would first have to arrange these data points into a grid representation such that accessing any point in the domain is done through two indices, similar to a 2D array representation. n=1057 is the number of x grid points while m=397 is the y grid points. You may assume the data in $flow_data.csv$ is sorted in ascending order in y followed by ascending order in y. It is important that the arrangement of the data points is consistent with the domain. So, if as an example, the y data points were put into a 2D array y of shape y when, increasing y in y

$$\omega[i][j] = \left(\frac{V[i+1][j] - V[i][j]}{X[i+1][j] - X[i][j]} - \frac{U[i][j+1] - U[i][j]}{Y[i][j+1] - Y[i][j]}\right)$$

Here, ω , U, V, X and Y are defined as 2D arrays containing values of ω , u, v, x and y in the 2D domain. The indices in the above formula for i go from 0: n-2 and for j go from 0: m-2. For data points with indices n-1 or m-1, the value of vorticity will be given by:

$$\begin{split} \omega[n-1][j] &= \left(\frac{V[n-1][j] - V[n-2][j]}{X[n-1][j] - X[n-2][j]} - \frac{U[n-1][j+1] - U[n-1][j]}{Y[n-1][j+1] - Y[n-1][j]}\right) \\ \omega[i][m-1] &= \left(\frac{V[i+1][m-1] - V[i][m-1]}{X[i+1][m-1] - X[i][m-1]} - \frac{U[i][m-1] - U[i][m-2]}{Y[i][m-1] - Y[i][m-2]}\right) \\ \omega[n-1][m-1] &= \left(\frac{V[n-1][m-1] - V[n-2][m-1]}{X[n-1][m-1] - X[n-2][m-1]} - \frac{U[n-1][m-1] - U[n-1][m-2]}{Y[n-1][m-1] - Y[n-1][m-2]}\right) \end{split}$$

Once you've obtained the values of ω , you must report a histogram of the absolute value of ω . That is you will count the number of data points that have the $|\omega|$ within a certain lower and upper boundaries (edges). You will define these edges in the following way: Suppose you want to group your data points into n bins. Then you will compute n edges starting from 0 and ending at 25 (including 0 and 25). That is you will create equally spaced n points between 0 and 25. For instance if n = 5, your edges will be $\{0,6.25,12.5,18.75,25.0\}$. You will then have 5 bins defined by these edges as follows:

$$0 \leq |\omega| < 6.25, \quad 6.25 \leq |\omega| < 12.5, \quad 12.5 \leq |\omega| < 18.75, \quad 18.75 \leq |\omega| < 25.0, \quad 25 \leq |\omega|$$

and you are supposed to count the nodes in this bins. The number of edges (i.e. thresholds) will be defined as a command-line input by the user (assume it is always greater than 1). Finally, report your results in the following format to the file **task4.csv**:

```
threshold, points
0.000000,10000
6.250000,20000
12.500000,30000
18.750000,100
25.000000,200
```

In each line write the lower edge of that bin (to 6 decimal places) followed by the count number (an integer) separated by a comma.

Implementation

The implementation of your solution is extremely important and should be based on the provided skeleton code. A total of **3 marks** are allocated to the quality of your code:

Marks: 5 of 25

- Program compiles without warning using gcc -Wall -std=c99 on dimefox
- Exception handling (check return values of functions such as malloc or fscanf)
- No magic numbers i.e. hard coded numbers in your code (use #define instead)
- Comments and authorship for each file you submit are important (top of the file)
- General code quality (use code formatters, check for memory leaks)
- No global variables

There is a coding etiquette we are enforcing for this assignment, which is concerned with the comments in the code. Every code snippet must be provided with sufficient comments, i.e., what does the following block of code do. This is imperative in case your output is incorrect; as it makes it easier for the grader to assess if your implementation was correct. **2 marks** is allocated for the quality of comments. The following sections describe the required command line options and program output of your program.

Command-line options

When running on dimefox your program should support the following command-line options:

```
terminal: gcc -Wall -std=c99 main.c tasks.c -o flow -lm
```

for the compilation and

```
terminal: ./flow flow_data.csv 10 8
```

for the execution, where **flow_data.csv** is containing the flow related data, 10 is the grid resolution and 8 is the number of thresholds.

Program output

When running on dimefox the program should only output the following information to stdout in addition to the csv files for each task:

```
terminal: ./flow flow_data.csv 10 8

TASK 1: 200.63 milliseconds

TASK 2: 14063.82 milliseconds

TASK 3 Array Linear Search: 30.4 microseconds

TASK 3 Array Binary Search: 40.5 microseconds

TASK 3 List Linear Search: 100.21 microseconds

TASK 3 BST Search: 50.1 microseconds

TASK 3: 209.46 milliseconds

TASK 4: 221.16 milliseconds
```

Note the units of time in the output for the tasks. For each task, output the time, in relevant units, taken to perform all computation associated with each task. This can be achieved by adopting the <code>gettimeofday.c</code> file provided in the lectures. Your program should write the files <code>task1.csv</code>, <code>task2.csv</code>, <code>task3.csv</code> and <code>task4.csv</code> containing the results for the different tasks in the same folder where the code is executed.

Within each task, marks are allocated for correct implementation of the "output format" in terms of console output and output written to the result file generated by each task.

Provided Code

The following files are provided to you for this assignment:

- main.c, where the parsing of data from command line is to be done and timing for each task implemented.
- tasks.c, where you would implement four functions max_min_flux(), coarsegrid(), searching() and calculate_vorticity(), for each task.
- tasks.h, which need not be changed and acts as a header file to link the C file to the main file.

You are free to use guide programs provided during the lectures and adapt them for your use. This could be any of the files like **file_io.c**, **linkedlist.c**, **bst.c** and more. You may also use the header files if needed. Remember to fill in your details in each file you write, such as the student name and student number.

Points to consider

- Only use type **int** and **double** for integers and floating point numbers respectively. When writing out a float to output make sure the format restricted 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. If there are multiple numbers on a line separated by a comma, leave no space in between.
- 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. Static memory allocation is not allowed.

Submission

You will need to submit your work in Canvas LMS. You can submit partially completed tasks as soon as you have them. You can submit multiple times but only the latest submission will be marked. Try not to wait until last minute to make your submission. You will be allowed to submit after the due date. But all submissions after the due date will be marked as late submissions and the late submission policy will be applied in grading your assignment.

Your files need to be packed into a single zip file which will contain the necessary .h and .c files to compile and run your code. You must name your file using your SIS Login ID. For example if your username is DaRodriguez, your zip file will be named <code>DaRodriguez.zip</code>. If the grader fails to extract your source code from the zip file you will get zero marks. Submitting individual files in Canvas will not be possible. Additionally, failure to comply with the above naming format will infer a penalty of 10% of your marks.

Your code will be checked for compile and run time errors in dimefox. Make sure that your code compiles and runs successfully in the server to avoid losing marks. That is, transfer you codes and the input CSV file to dimefox and compile and run your code to see if it works without errors. Once you are satisfied with your results, you can submit your work through Canvas as mentioned earlier.

You must also check that you have no memory leaks in your code as loss of memory from your implementation will result in mark deductions. Your submissions will be assessed using the **valgrind** debugging

tool by your grader. You can check your code in dimefox as follows:

```
valgrind ./flow flow_data.csv 10 8
```

A well-implemented code is expected to have the following output regarding memory usage:

```
==3887== HEAP SUMMARY:
==3887== in use at exit: 0 bytes in 0 blocks
==3887== total heap usage: 53 allocs, 53 frees, 56,921,168 bytes allocated
==3887== All heap blocks were freed -- no leaks are possible
```

Note that valgrind will take longer time to run compared with the normal execution. In case your submission fails to pass the memory check, you will lose marks. There are two potential areas where you can lose marks: run time error messages and heap/leak summary. Examples of run time error messages:

- Use of uninitialised value of size X: Happens when you use a variable that has not been defined or does not exist anymore or initialised.
- Conditional jump or move depends on uninitialised value(s): Happens when using an uninitialised variable to perform operations
- Invalid read/write of size X: Happens when trying to scan or write to file a variable to memory which you do not have access to.
- Process terminating with default action on signal 11: Happens when the error is so severe, your code is terminated automatically.

Examples of heap/leak summary errors are:

• total heap usage: 2,686 allocs, 29 frees, 152,138,664 bytes allocated

```
==6504== LEAK SUMMARY:
==6504== definitely lost: 0 bytes in 0 blocks
==6504== indirectly lost: 0 bytes in 0 blocks
==6504== possibly lost: 0 bytes in 0 blocks
==6504== still reachable: 16,912,796 bytes in 2,657 blocks
==6504== suppressed: 0 bytes in 0 blocks
```

You may discuss your work during your workshop, and with others in the class, but what gets typed into your program must be **individual** work, not copied from anyone else. So, do not give a copy of your work to anyone and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "**no**" when they ask for a copy of, or to see, your program, pointing out that your "**no**", and their acceptance of that decision, is the only thing that will preserve your friendship. A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode. Students whose programs are so identified will be referred to the Student Center. See https://academichonesty.unimelb.edu.au for more information.

Getting help

There are several ways for you to seek help with this assignment. First, go through all the instructions provided in this document in detail. It is likely that your question has been answered already. Second, you may also discuss the assignment on the **Discussion Assignment 1** in Canvas. However, please **do not** post any source code on the discussion board. Finally, you may also ask questions to your tutors during the workshops and if it is still unresolved, contact either the lecturer Aman Kidanemariam (aman.kidanemariam@unimelb.edu.au) or the head tutor David Rodriguez (david.rodriguezSanchez@unimelb.edu.au) directly.