C Programming Assessment Feedback
2018

This document acts as a general feedback for the streamline calculation exercise. A summary of the key points of the exercise are outlined, including the aspects that were looked for when marking. The typical issues found are outlined also.

1 Summary

In general, the submissions for this task were of a high standard. Many students have taken to programming well even though it is there first exposure to it, and can often be a tricky skill to pick up in just a few weeks. However, the number of high marks obtained shows how well the cohort completed this task, and the skill of programming in general.

When marking, they two areas considered were correctness and clarity. These were split down into a few more areas as follows:

- *Correctness*
  - streamlines start at the correct location
  - terminating conditions were given by the box constraints and the maximum iteration constraint
  - velocity and integration method were coded correctly
  - the program output a suitably formatted tecplot file
  - the extension integration method was coded correctly
- *Clarity*
  - functions were used where appropriate
  - code was clear with sufficient comments, good indenting and suitable variable names
  - the code could handle incorrect inputs (such as a negative number of streamlines) without crashing

2 Correctness

Almost all students submitted a code that produced an answer, which is encouraging, and marks were awarded for this. Almost all students correctly coded the velocity equations and Euler integration method. Overall, however, more marks were lost for obtaining the fully correct answer, than on submitted a clear, well structured code.

A simply pseudo-code is given below.

Issues typically arose when the number of streamlines asked for were odd. Often, when an odd number of streamlines was asked for, in initial y distribution became incorrect when

```
Get user data
for nn=0 → nn<Nstream do
    Set x[nn,0]=-1, calculate y[nn,0]
    Export x[nn,0] and y[nn,0] to file
    for tt=0 → tt<Tmax-1 do
        Calculate u[nn,tt] and v[nn,tt]
        Calculate x[nn,tt+1] and y[nn,tt+1]
        if x[nn,tt+1] or y[nn,tt+1] outside box then
            Break
        end if
        Export x[nn,tt+1] and y[nn,tt+1] to file
    end for
end for
Export to file
```

it would otherwise be correct for an even number. Some submissions would only allow an even number of streamlines to be input. I think some students had problems with the streamline at $x = 0$, hence tried to avoid plotting it. There was no need to have a catch for an this as the code should calculate the $x = 0$ streamline and it should terminate at $x \approx -0.16$, however if the streamline did look to stop but then continue, often on a circular arc, then this was not penalised.

Most submissions contained the streamline box constraints, but some lacked the maximum iteration counter constraint asked for. This could have been part of the problem with the previous point, where the code gets into an infinite loop because the $x = 0$ streamline never gets to the box constraints.

A final common point was that sometimes the first point of the streamline would not be exported. This was due to exporting the updated location after the initial point had been updated. Arrays could be utilised to get around this problem. Also, some submissions exported a final point beyond the maximum box constraints, and this was due to writing the file before checking the box constraint.

On the extension, many students successfully coded the extension problem. Where possible, marks were awarded to those students who may not have the correct final answer due to a previous error, but did code the new extension parts correctly.

3 Clarity

Most submissions were well commented and structured, with suitable variable names. However, there were a couple of common issues to outline.

Commenting was mostly done well. It is worth remembering that more comments are almost always a good thing. Ensure any complicated pieces of code are very well commented. All functions should also be commented to explain what they do.

The naming of variables was slightly more mixed than comments. Many students lost a mark here because their variable names were too abstract e.g. `u, v, x`. Even though these may represent velocities and location respectively, this is something that needs to be included in variable names. For full marks, students needed to ensure that variable names were descriptive and that the handout did not need to be consulted to find out what symbols meant. A few submissions tried to explain their variable names within the comments. This is a good habit to get in to, but the names themselves need to still be as descriptive as possible.

Indenting was generally well done. When it comes to indenting, my preferred approach is to indent all code within a set of curly-brackets `{}` where every line of code within one set of `{}` are on the same indent level, and then whenever another set of `{}` are introduced, a new level of indenting is used. This clearly allows the visualisation of how the loops and conditionals are used e.g.

```c
#include <stdio.h>
int main() {

        double a=0, ii;

        for(ii=1; ii<=10; ii++) {
                a=a+1;
                if (a==3) {
                        break;
                }
        }

        return(0);
}
```

A common issue was that some codes could not handle an incorrect input say if the number of streamlines requested were negative. A couple of times the code would crash, or would just not run and not display an error message to the user. This is simple to check for and should be done for user inputs. Also, do not assume that if the user inputs an incorrect answer once, that they will not do this again. A few codes did successfully check for an incorrect user input the first time, but when another wrong answer was input for the same number, the code would continue without recognising the problem. This is simple to check for by using a `while` or `do-while` loop instead of an `if` statement.

Although marks were not awarded for the suitable use of functions, much of the problem could have been effectively utilised using functions so it is worth briefly considering when it is suitable to use functions. In general, functions should be used as much as possible to keep your code modular, making it easier to test and easier to read. For example, you could siphon-off the user-input, output, and calculations to functions. Your main function would therefore be kept small, and this is ideal. The velocity equations were a prime candidate for placing in a function, and may have avoided quite a bit of repetition of similar lines of code. Also ensure functions are suitably named (`function1` is not an acceptable name for a function), and comment what the function does.