Programming for Engineers Portfolio : Quarter 1, Set 3, es1219

Submission information

This is the final set in the portfolio for the first quarter. Once you've finished all the exercises, you should submit the output of `./prepare_submission.sh` via blackboard. To run `prepare_submission.sh`, just run it like a program:

```
./prepare_submission.sh
```

The output will then appear as a tar.gz file one directory level up. Every time you run the script it will produce a new output file, with the date and time attached. The date and time is irrelevant for assessment purposes, it is simply to make it easier to manage different versions.

The submission script simply grabs *everything* in that directory, so you should clean up any temporaries or unneeded files to reduce size. If the `.tar.gz` file it creates is much more than 2 MB in size, then you may have some temporaries or other things which could be removed. In general submission size is not a problem, though try to not to submit 100MB+ files - it will probably work, but you'll find uploads are quite slow, especially from outside college.

The submission system is set to open at 18:00 on Monday 4th, with the assessment name "Q1PortfolioExercises". A reminder: the delay is to encourage people to think about how to manage and backup files, particularly when working with laptops which might fail.

You can submit as many times as you like to blackboard, and the last submission will be taken as your final submission.

Sanity check : Mon 11th Nov at 9:00

On Monday 11th at 9:00 a copy of all the current submissions will be grabbed from blackboard, and some basic sanity checks will be run. These won't perform a full assessment, but are intended to do some basic "sanity checks" to make sure that the right type of file has been submitted, and that everything is in roughly the right place.

Final deadline : Fri 15th at 18:00

The final submission deadline is at the end of reading week / mid-term on Friday 15th at 18:00.

Exercises

These portfolio exercises test the ability to create and work with simple functions and structs. They do not require recursive functions.

Where exercises specify particular functions, you should make sure that the function exists and the prototype (name, parameters, return type) matches the specification. The assessment process will be checking that these functions exist, as well as checking that the program works.

1. Create a file called `e1-add_5.cpp`, which should contain a function definition for `add_5`. The function should take an integer as a parameter, and return that integer plus 5. When compiled and run, the program should read an integer, call the function `add_5`, and then print the result out.

2. Create a file called `e2-poly.cpp`, which should contain a function with the prototype:

   ```
   float polyval(float x);
   ```

   The function should return the value of the following polynomial:

   $-2 + -4(x + 1x)$

   When compiled and run, the program should read a float, call `polyval` on that value, and then print the result out.

3. Modify the file `e3-program.cpp` at the positions marked "TODO" so that:

   a. There is a new function named `f` which returns the expression `x * 10 + x * x`; and

   b. The main function calls `f`, rather than using the expression.

   The modified program should still have the same behaviour in terms of input and output.

4. The source file `e4-program.cpp` does not compile due to a missing function declaration. Add a function declaration in the place indicated with "TODO" so that it compiles.

5. Modify the source file `e5-quadratic-root.cpp` to add a function definition for `root`. The function `root(a,b,c)` should return the real root of the quadratic $0 = a + bx + cx^2$ using the standard quadratic formula. If there are two real roots the function should return the larger of the two roots. You do not need to consider the case where there are no real roots.

6. Create a source file called `e6-enumerate.cpp`. Add a function definition called `enumerate` which takes two integer parameters `x` (parameter 0) and `y` (parameter 1) and returns a vector of integers. The return value should contain the integers in the range [x,y], and you can assume that $x < y$. The source file should also contain a main function which reads two parameters `x` and `y` (in that order), calls `enumerate(x,y)`, and prints out the elements of the vector, one line per element.

7. Create a source file called `e7-triple.cpp`, which defines a struct called `triple` consisting of three members in the following order: a `string` called `a`; an `int` called `b`; and a `float` called `c`. Your main function does not need to do anything with the type, but you should check that the source file compiles.

8. Modify the file `e8-complex-div.cpp` to add a definition of the `div` function below the main function. The implementation should follow the standard rules for a/b in complex arithmetic, though the exact expressions used are up to you. (Any minor variations in the specific output due to expression choice will be automatically taken into account during assessment.)

9. Create a file called `e9-filter.cpp` which contains a function with the following prototype:

   ```
   vector<int> filter(vector<string> values);
   ```

   The function should convert the input values to integers, and then return only those values where the square of the value is less than or equal to 5. Any values which make it through the filter should appear in the same order as in the input. When compiled and run, the program should read all incoming input strings from `cin` into a vector, call the `filter` function, and then print out the results separated by spaces.

10. The file `e10-dft.cpp` contains the prototype for a function called `dft`, and a main function that calls that function. Implement the function `dft` using the following information:

    - The `complex<float>` type is provided by the C++ standard library, and provides standard numerical operations and elementary functions over the complex numbers in terms of `float` real and imaginary components.
    - The input vector always has a length of at least 2.
    - The output vector is of size m, which is passed as the second parameter.
    - If we define the function as a relationship $\mathbf{y} = \text{dft}(\mathbf{x})$, where $\mathbf{x} = x[0], x[1], .., x[n-1]$ are the input values, and $\mathbf{y} = y[0], y[1], .., y[m-1]$ are the output values, then the relationship between each output element and the input elements is:

$$y[a] = \sum_{b=0}^{m-1} x[b] \times \exp(-2\pi ab/n), \quad \text{where } 0 \leq a < m$$

    - If you run the program using the input file `e10-input.dat`, then the output should be *approximately* equal to `e10-output.txt`. You may get small minor variations depending on how you implemented it.

    There is a pre-compiled reference version of the program available as `e10-dft-ref`, which can act as a golden reference.

    Note: This exercise is much more complex than any of the other exercises, and is intended to give people something that is a bit more of a challenge, while also providing a useful program that does something. But please bear in mind that this is weighted exactly the same as all the other portfolio components, so it is only worth 1/30th of this portfolio, which is itself only worth 5% of the module, which is itself only worth 18.1% of the year. So this question is worth 0.17% of the module, or 0.03% of the year. A few people may find it fairly easy; others will find it challenging but fun; others may find it very difficult at this stage in their learning. Everyone should decide what the time investment versus reward is - not all marks are worth getting if they impact on higher value activities elsewhere.

    Note: You don't need to know what the function is actually doing mathematically, though it's not too hard to work out that it's some kind of fourier transform. We (the ELEC40004 team) are aware that you aren't even close to doing this in the maths course yet (it's in chapter 11+12, I believe), so you shouldn't go racing off trying to understand it. Often you'll be asked to implement something where it isn't clear how it works - but you do have a clear spec and reference inputs and outputs so you can still deal with the problem and translate the mathematical definition into an operational piece of code.

Note: if you want to play around with this file to see how it behaves, don't forget to copy to a new file name so that it doesn't conflict with your submitted portfolio.