Practical 03

By the end of this set of problems, you will be familiar with:

- `read`
- `write`
- `open` and `close`

All `programs` *must* contain `implicit none`. While not strictly necessary, it will eliminate an entire class of bugs.

It is a very good idea to always use the following compiler flags for all these problems: `-Wall -Wextra -fcheck=all -g`. Try to ensure you have no warnings. You may also want to use `-std=f2008` or `-std=f2018`, depending on the version of `gfortran` you are using. This will ensure you stick to standard Fortran.

You should use the lecture materials for help/inspiration, but please don't copy and paste! There is some value to be had in typing up the programs yourself.

There are very likely too many exercises here for the time you have available. It's ok if you don't get through them all!

There may be several ways to solve each problem. If you have time, you might like to try different approaches.

Use a separate file and program for each exercise.

Hello World Revisited

1. Modify your `hello_input.f90` program to write to a file instead of to screen.
2. Now change it to read the name and number from a different file.
   - Hint: consider whether you want to read a file that looks like:
     ```
     Peter 42
     ```
     or like
     ```
     Peter
     42
     ```
     What's the difference? How would you modify your program to read it the other way instead?
3. Take the input and output filenames from the user
   - Hint: you will need a fixed `len character` to read the filenames into, but it will likely having trailing spaces that you will need to deal with when you `open` the file. Is there an intrinsic function that can help?

Further

1. Modify your program from step 2 to read the name and number with a `namelist`.
2. Look up the `position` and `status` arguments to `open`. Can you modify your program to keep a log of all responses, instead of overwriting the output file?
3. (More challenging) Allow the user to specify the input file on the command line. You can use the intrinsic subroutine `get_command_argument` to read command line options.

Earliest letter

1. Write a function that takes a `character(len=:)` and returns the earliest letter alphabetically, e.g. `earliest("rhinoceros")` would return `"c"`. You can assume the input is always in lower case.
   - Hint: you can compare `character`s with `<`

Number to string

1. Write a function that returns the "length" of an integer: the number of digits, plus the sign if it's negative. For example, `1234` as an input would return `4`, while `-56789` would return `6`.

- Hint: is there a mathematical function that tells you the size of a number in base 10?

2. Write a function that returns the format string needed to fully print an integer in the smallest width. For example, `1234` as an input would return `(i4)`, while `-56789` would return `(i6)`. Your function from step 1 will be helpful!
   - Hint: you can `write` to a `character`
   - Hint 2: how would you describe `(i4)` in terms of format codes?
3. Using your function from step 1, write a function that takes an `integer` and converts it to a string, returning an `allocatable character` of the correct length. Check your result looks the same as the input!
   - Hint: you can use both previous functions
   - Hint 2: you will need to `allocate` the result before writing to it

String to number

1. Write a function that takes a `character` and converts it to an integer

Further

1. How many bad inputs can you find?
2. How many of the bad inputs can you provide error checking for?

Euler/pi output

1. If you completed either the "Euler Integration" or "Calculating π" exercises from the previous practical, add inputs using namelists and outputs in a nice table to file.

Boxes of stars

Warning, this one is surprisingly tough! Try breaking it down into smaller steps!

1. Write a program that reads two numbers from the user and draws a box made of stars of the corresponding height and width. An example of the output might be:

```
$ ./starboxes
Enter height and width:
4 8
********
*      *
*      *
********
```

2. Read a single `character` from the user to use instead of stars

Further

1. Take some text to print in the box