Practical 03

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

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

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

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

Boxes of stars

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 a third number, which is the number of boxes to print

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

Number to string

1. Write a function that takes an `integer` and converts it to a string, returning an `allocatable character` of the correct length. Don't forget the minus sign!

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?

Input files and dump files

1. Take your Euler integration program from the previous set of exercises and modify it to read the timestep, end time and initial condition from file.
2. Further modify it so it writes the current timestep and value of y to another file.

Further

1. (Less challenging) Read the filename for the output file from the input file
2. Use a namelist to read the input values. Don't forget to set default values.
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.