Exercise 5.1 – Basic IO stream formatting

- The goal of this exercise is to use the `iostream` manipulators to format output on the terminal
  - Be sure to include the `<iomanip>` header for this exercise.
- Approach – reading and writing integers in decimal, hex and octal
  a) Write a small program that reads in an integer from `cin`.
  b) Write it out in decimal, hexadecimal and octal format on three separate lines using the `iostream` manipulators to specify the formatting.
  c) Change the reading part of the program so that it expects a hex number instead of a decimal number.

- Approach – Reading and writing floating points in various formats
  d) Extend the main program so that it also reads three floating point numbers `f1`, `f2` and `f3`.
  e) Print `f1`, `f2` and `f3` separated by spaces in scientific format (i.e. `0.33e05`). Do you need to repeat the scientific mode manipulator in front of each floating field?
  f) Print `f1`, `f2` and `f3` again, but now change the formatting using the `setw()` manipulator such that each number is printed in a 20 character field. Do you need to repeat the `setw()` manipulator in front of each field?
  g) Print `f1`, `f2` and `f3` once more, but now in fixed precision notation with 3 figures behind the dot and aligned left in their fields (e.g. "`15.253`")

Exercise 5.1 – Basic IO stream formatting

h) Insert code above the code you designed for question f) that prints a line with the text headers '`ValueA`', '`ValueB`' and '`ValueC`' spaced in such a way that they align with the numbers below.

i) Finally, add code to print a line of dashes ('`-`') between the header and the values. You are not allowed to use more than one (**1**) dash character in this statement. Hint: use the `setw()` and `setfill()` manipulators.

Exercise 5.2 – Reading a file

- The goal of this exercise is to read a file with integers to the end.
- Approach
  a) Write a program that opens the file `ex5.2/data1.txt` using the class `ifstream`.
  b) Add code that verifies that the file was opened OK. Check that this code works by temporarily changing the name of the file to be opened to a different – non-existent – one, e.g. `data2.txt`. (Hint: use `operator!()`)
  c) Look at the contents of file `ex5.2/data1.txt`. You will see that it contains only integers, separated by spaces and/or newlines. Write a loop that reads in one integer at a time until you reach the end of the file. Print each integer as you read it in. Think about how you will know when you have reached the the end of the file.
  d) Change the loop so that it introduces an *additional* condition to stop reading: when you have read an integer with value zero.

Exercise 5.3 – Word counting

- The goal of this exercise is to count the number of lines, words and characters in a text file, similar to the UNIX utility `wc`
- Approach - Write a program that counts the *lines* of a text file
  a) Write a program that takes a filename as argument when it is invoked (e.g. '`unix> myProgram theFile.txt`'), opens that file and checks if the file was opened successfully.
  (Hint: use `main(int argc, char* argv[])`. You can use the file `ex5.3/example.txt` as input.
  b) Write a loop the read the file line by line using `ifstream::getline(char* buf, int buflen) ;`. Think about how you will know when you are at the end of the file.
  c) In the loop, count the number of lines you read and report the line count to standard output at the end of the program.
  d) Run your program on `example.txt` and cross check your result with the unix utility `wc -l <file>`
- Approach – Augment the program such that it counts the number of *characters* as well.
  e) Add code in the loop that processes each line that counts the number of characters in that line and use that number to calculate the total number of characters in the file.
  Cross check your result with '`wc -c example.txt`'

Exercise 5.3 – Word counting

- Approach – Modify the program such that also counts the number of *words* in the file

  f) Add code in the loop that reads and processes each line as follows:

  g) Construct an `istringstream` object (declared in `<sstream>`) passing the `char[]` line buffer as constructor argument. This creates an *input stream* representation of the current line you are processing and allows you reread the line using the C++ streamer operators

  h) Read a `char[]` string from the `istringstream`. The read will stop as soon as you encounter a white space in the line represented by the stream. Repeat the reading procedure until you are at the end of the stream (how do you know that?). The number of successful reads is the number of words on the line. Explain why.

  i) Use the number of words per line to calculate the total number of words in the file. Report the total number of words at the end of your program and cross check your result with '`wc example.txt`'