1 IO

Exercise 1.1. Write a program that prints the message `"Hello World!"` followed by a newline to the standard output of your terminal.

Exercise 1.2. Write a program that reads two integers from the standard input of your terminal and prints the sum of those integers to the terminal.

Exercise 1.3. Write a program where the user enters two words on the same line and then press `<enter>`. The program should print the two words in the opposite order from which they where read, separated by a `','` character.

Exercise 1.4. Think about *why* the user could enter two words at once in the previous assignment but the program could read them separately one at a time. What is the mechanics behind this?

Exercise 1.5. Describe what buffered IO means. What are the benefits of this compared to unbuffered IO?

Exercise 1.6. How do you remove the content of the buffer in `std::cin`?

Exercise 1.7. Write a program that reads an integer and then prints it with *at least* three digits. If the integer has fewer than three digits there should be an appropriate amount of zeroes added to the start of the initeger.

Hint: Look at `<iomanip>`: `https://en.cppreference.com/w/cpp/header/iomanip`.

Note: You should *only* use IO operations, no if-statements or loops required.

Exercise 1.8. Write a program that generates two prompts for the user: one for a number and one for a word and then prints the result of each prompt.

A prompt in this assignment means that the program freezes while waiting for input from the user, and then once the user has pressed `<enter>` the program proceeds. The second prompt should behave the same, no matter what the user entered in the first prompt.

Hint: If the user enters `1 word <enter>` then the program should read the `1` and then *ignore* the rest of that input (see: `https://en.cppreference.com/w/cpp/io/basic_istream/ignore`).

Exercise 1.9. Write a program where the user enters a line of text and print that line.

Exercise 1.10. Write a program that generates three prompts: one when the user enters an integer `a` (and potentially some other characters that should be ignored), one prompt where the user enters a line of text and finally a second integer `b`. It then prints `a + b` on one line and the line on the next line.

Exercise 1.11. Write a program where the user enter a real number (i.e. a number with a potential decimal point). Then print said real number with exactly 4 digits of precision.

Example: If the user enters `1.5` then the program should print `1.5000`. If the user enters `2.123456789` then the program should print `2.1234`.

Hint: Look at `<iomanip>`: `https://en.cppreference.com/w/cpp/header/iomanip`.

2 Repetition

Note: In these exercises you have to think about what type of loop to use. Each exercise should use a different type of loop compared to the other exercises.

Exercise 2.12. Make a program that reads a positive integer `n` and sums all integers from `1` to `n`.

Exercise 2.13. Write a program that prompts the user for an integer `n`. If the integer was negative then generate the same prompt again. Do this until the integer is positive or zero.

Hint: You do not need if-statements or `break` statements for this exercise.

Exercise 2.14. Make a program that read integers from `std::cin` and calculate their sum until the user enters something that isn't an integer. Print the sum once all integers have been read.

Hint: The expression `std::cin >> n` can be used as a condition, and fails if it didn't manage to read an integer (assuming `n` is an integer type).

3 Floating point numbers

Exercise 3.15. Think about what you expect the following program to print, and what it *actually* prints.

```
#include <iostream>

int main()
{
    float f { 0.01f };
    float sum { 0.0 };

    for (int i { 0 }; i < 10000; ++i)
    {
        sum += f;
    }

    std::cout << sum << std::endl;
    std::cout << 10000 * f << std::endl;
}
```

Why do they differ? What conclusions about *how* to use floating-point numbers can you draw from this? Remember that floating-point numbers struggle with rounding errors.

Exercise 3.16. Fix the following program so that it prints the expected result without hardcoding the value.

```
#include <iostream>

int main()
{
    float sum { 0.0f };
    while (sum < 25)
    {
        sum += 0.1f;
        std::cout << sum << std::endl;
    }
    std::cout << "Total sum: " << sum << std::endl;
}
```

Exercise 3.17. Why doesn't this program print the final value (`0.01`)?

```
#include <iostream>

int main()
{
    float f { 100.0f };
    while (f >= 0.01)
    {
        std::cout << f << std::endl;
        f = f / 10;
    }
}
```

Hint: The value `0.01` is of type `double` while `f` is of type `float`.

Exercise 3.18. You can solve the problem above by making `0.01` into a `float` instead. Do this as simply as you can (can be done by adding one character to the code).