1.1 Testing your code with I/O Redirection

Your programs should not explicitly open any file. You can only use the **standard input** e.g. std::cin in C++, getchar(), scanf() in C and **standard output** e.g. std::cout in C++, putchar() , printf() in C for input/output.

However, this restriction does not limit our ability to feed input to the program from files nor does it mean that we cannot save the output of the program in a file. We use a technique called standard IO redirection to achieve this.

Suppose we have an executable program a.out, we can run it by issuing the following command in a terminal (the dollar sign is not part of the command):

```
$ ./a.out
```

If the program expects any input, it waits for it to be typed on the keyboard and any output generated by the program will be displayed on the terminal screen.

Now to feed input to the program from a file, we can redirect the standard input to a file:

```
$ ./a.out < input_data.txt
```

Now, the program will not wait for keyboard input, but rather read its input from the specified file. We can redirect the output of the program as well:

```
$  ./a.out > output_file.txt
```

In this way, no output will be shown in the terminal window, but rather it will be saved to the specified file. Note that programs have access to another standard interface which is called standard error e.g. std::cerr in C++, fprintf(stderr,…) in C. Any such output is still displayed on the terminal screen. However, it is possible to redirect standard error to a file as well, but we will not discuss that here.

Finally, it's possible to mix both into one command:

```
$  ./a.out  < input_data.txt  > output_file.txt
```

Which will redirect standard input and standard output to input_data.txt and output_file.txt respectively.

Now that we know how to use standard IO redirection, we are ready to test the program with test cases.

Test Cases

A test case is an input and output specification. For a given input there is an *expected* output. A test case for our purposes is usually represented by two files:

- test_name.txt
- test_name.txt.expected

The input is given in test_name.txt and the expected output is given in test_name.txt.expected.

To test a program against a single test case, first we execute the program with the test input data:

```
$ ./a.out < test_name.txt > program_output.txt
```

The output generated by the program will be stored in program_output.txt. To see if the program generated the expected output, we need to compare program_output.txt and test_name.txt.expected. We do that using a general purpose tool called diff:

```
$  diff -Bw program_output.txt test_name.txt.expected
```

The options -Bw tells diff to ignore whitespace differences between the two files. If the files are the same (ignoring the whitespace differences), we should see no output from diff, otherwise, diff will produce a report showing the differences between the two files.

We would simply consider the test passed if diff could not find any differences, otherwise we consider the test failed.

The tester uses this method to test your code against multiple test cases. There is also a test script accompanying this project test1.sh which will make your life easier by testing your code against multiple test cases with one command.

Here is how to use test1.sh to test your program:

- Store the provided test cases zip file in the same folder as your
- project source files Open a terminal window and navigate to your

  project folder using
  cd command
- Unzip the test archive using the unzip command:

```
$ unzip test_cases.zip
```

NOTE: the actual file name is probably different, you should replace test_cases.zip with the correct file name.

- Store the test1.sh script in your project
- directory as well Mark the script as

  executable once you download it:

```
$  chmod +x test1.sh
```

- Compile your program. The test script assumes your executable is called a.out
- Run the script to test your code:

```
$ ./test1.sh
```

The output of the script should be self-explanatory. To test your code after each change, you will just perform the last two steps afterwards.