Chapter 3

Exercise: Read transcriptions and annotations from a file

3.1 Leaning Objectives

References to literature are provided in brackets:

- File I/O , [8, Appendix] [9, Sect. 1.8, Sect. 11.4],
- Using your IDE (e.g. Visual Studio),
- Using `map` and `set` and `stringstream`

3.2 Exercises

In this exercise we will implement first functionality to reach the final aim of automatic annotation of given ATCo transcriptions.
see directory ExeFileReading for the solution code

3.3 Exercises at a glance

Exercise 3.1 Read the transcriptions from one file, store the contents in appropriate variables and count the number of occurrences of the different words. Print to screen the TOP TEN occurrences.

Exercise 3.2 Read also the command annotations from file and count how often an expected type occurs.

Exercise 3.3 Implement unit tests.

3.4 Detailed Exercise Descriptions

Exercise 3.1: You get a file with the following example format

```
2019-02-15__11-32-02-00:
   speedbird five two charlie victor praha radar radar contact climb
            flight level one two zero
   BAW52CV INIT_RESPONSE
   BAW52CV CLIMB 120 FL
2019-02-15__11-32-24-00:
   scandinavian one seven six seven praha radar radar contact
           descend flight level one hundred
   SAS1767 INIT_RESPONSE
   SAS1767 DESCEND 100 FL
2019-02-15__11-32-40-00:
   b_air six one praha radar radar contact climb flight level one two zero
   ABP61 INIT_RESPONSE
   ABP61 CLIMB 120 FL
2019-02-15__11-32-48-00:
   speedbird five two charlie victor proceed direct to rapet
   BAW52CV DIRECT_TO RAPET none
2019-02-15__11-33-26-00:
   speedbird five two charlie victor climb flight level one six zero
   BAW52CV CLIMB 160 FL
2019-02-15__11-33-44-00:
   b_air six zero turn left proceed direct to nirgo
   ABP60 DIRECT_TO NIRGO LEFT
2019-02-15__11-33-55-00:
   good morning
   NO_CALLSIGN NO_CONCEPT
...
```

The file contains first a time stamp followed by a colon (:). The next line contains a sequence of words (introduced by some blanks in the beginning of the line.) It is always one line. In the example we split it so that it fits on the page. The next line or lines contain the commands, which are extracted from the word sequence. Then a date follows again introducing the next word sequence with its commands etc. The date always starts with the characters "20".

Write a program which is able to open such a file and extract the word sequences from the file. The output of the program is, how often each word occurs. The output is sorted alphabetically (according to their ASCII codes, i.e. zulu would be before apple).

For the following input file

```
2019-02-15__11-32-02-00:
   speedbird five two charlie climb   flight level two two zero
   BAW52CV INIT_RESPONSE
   BAW52CV CLIMB 120 FL
2019-02-15__11-33-55-00:
   good morning charlie
   NO_CALLSIGN NO_CONCEPT
```

we would just expect the following output (hoping my manual counting is correct):

```
        charlie :     2
          climb :     1
           five :     1
         flight :     1
           good :     1
          level :     1
        morning :     1
      speedbird :     1
            two :     3
           zero :     1
```

Some help:

It could be a good idea to extract each line separately as a `string` from the input line and then process this `string` word by word.

The following code fragment reads a line as a complete string from an input stream, e.g. a file.

```
#include <fstream>
#include <string>
using namespace std;

enum { MAX_LINE_LENGTH = 1000 };
/** The following functions reads from stream ar_inputStream
the next empty line and returns it in the parameter
arstrLineAsString. The ampersand at string& means
that it is an output parameter (call by reference).
If such a line could not be found, i.e. end of file is reached
before false is returned, otherwise true
*/
bool GetNextLineAsString
(
  ifstream& ar_inputStream,
  string& arstrLineAsString
)
{
  char line[MAX_LINE_LENGTH];
  line[0] = '\0';
  while (strlen(line) <= 0 && ar_inputStream.good())
  {
    // the previous line was still not completely read
    ar_inputStream.getline(line, MAX_LINE_LENGTH);
  }
  if (ar_inputStream.good())
  {
    arstrLineAsString = line;
    return true;
  }
  else
  {
    arstrLineAsString = "";
    return false;
  }
} // GetNextLineAsString
```

If you have a `string` you can easily use an instance of class `stringstream`, from which you can read word by word or number by number, just how you read from `cout`. Currently you do not know, how functions are used C++, but you can easily transform the above code so that you have everything in one `main`-function (try it).

```
  ifstream file(".\\WordSeqPlusCmds1.txt", ios::in);
  string nxtLine;
  GetNextLineAsString(file, nxtLine);
  // we now transform the line to an input string stream
  // from which we can read word by word
  stringstream strStream(nxtLine); // we need to include <sstream>
  string word;
  double number;
  // of course strStream must contain a string and a number
  strStream >> word >> number;
```

Exercise 3.2: In the previous exercise 3.1 you implemented how to read the word sequences. Now you also need to input the commands and count how often a command type occurs. A command always consists of a callsign and a type. After the type the third word can also specify a second part of the command type or something else or is empty. The allowed types are specified as string.

If we e.g. specify the following allowed types `DESCEND`, `REDUCE`, `CLIMB`, `INIT_RESPONSE` `TURN` and `DIRECT_TO` we would expect the following output for the first example of exercise 3.1.

```
          CLIMB :     3
        DESCEND :     1
      DIRECT_TO :     2
  INIT_RESPONSE :     3
```

If we specify the following allowed types `REDUCE`, `CLIMB`, `TURN` and `NO_CONCEPT` we would expect the following output:

```
          CLIMB :     3
     NO_CONCEPT :     1
```

For the following input file

```
2019-02-15__11-32-02-00:
   does not matter
   BAW52CV CLEARED ILS 34
   BAW52CV CLIMB 120 FL
   BAW52CV INFORMATION QNH 1013
2019-02-15__11-33-55-00:
   does not matter for the example
   BAW52CV INFORMATION ATIS L
   BAW52CV CLIMB 130 FL
```

and the allowed types `CLEARED ILS`, `REDUCE`, `CLIMB`, `INFORMATION QNH` and `INFORMATION ATIS` we would expect the following output:

```
     CLEARED ILS :     1
           CLIMB :     2
INFORMATION ATIS :     1
 INFORMATION QNH :     1
```

You see above an example, when also the third word is used for type. The types are read from a file with the name `expectedTypes.txt`.

If we specify the allowed types `CLEARED ILS`, `REDUCE`, `CLIMB`, `INFORMATION` we would expect the following output:

```
     CLEARED ILS :     1
           CLIMB :     2
     INFORMATION :     2
```

Exercise 3.3: Do not forget to implement Unit-Tests which show (and test) that your implementation is correctly running. Maybe you add also a parameter, which specifies the filename from which you read. So you can have multiple different tests.