Chapter 8

Exercise: Extracting Callsigns and other Numbers

8.1 Exercises

In this exercise we will implement more functionality to extract the concepts from a controller utterance. We will extract numbers from a sequence of words, which do not belong to the callsign and handle correction. It is not expected that you can handle all cases, but at least some of them (>50%).

8.2 Exercises at a glance

Exercise 8.1 Extract the callsign code from a given utterance.

Exercise 8.2 Extract all numbers (given as a word sequence) from a sequence of words, which do not belong to a callsign.

Exercise 8.3 Process also the word correction.

8.3 Detailed Exercise Descriptions

Your code should be in the folder `Aufgabe08/Code`, i.e. your screen dumps are expected in SVN in folder `Aufgabe08/Images`.

Exercise 8.1:

Extract the callsign part from a given utterance.

```
"oscar echo india november kilo direct whisky whisky nine eight five" //--> OEINK
"good morning lufthansa one two bravo descend eight zero" // --> DLH12B
  gruess gott ryan_air seven seven delta kilo in radar contact --> RYR77DK
```

```
  standby --> NO_CALLSIGN
gruess gott austrian triple seven sierra identified climb flight level two three zero
             //   AUA777S not expected from you
gruess gott lupus one one zero expect ils approach three four //--> AYY110
```

Here you find the data structures, which could be used for mapping from a letter to the name in the NATO alphabet. Attention you will need the opposite map, which maps from NATO alphabet to the ASCII character. You will also find this code segments in SVN in folder `AlleGruppen/VonJedemInSeineUmgebungZuKopieren/Aufgabe08/Hilfscode`.

```
static const std::unordered_map<char, std::string> lettersToNato =
{
  { 'A', "alfa" },
  { 'B', "bravo" },
  { 'C', "charlie" },
  { 'D', "delta" }, // their is also an airline 'delta', with the three letter code DAL
  { 'E', "echo" },
  { 'F', "foxtrot" }, // also fox is said
  { 'G', "golf" },
  { 'H', "hotel" },
  { 'I', "india" },
  { 'J', "juliett" },
  { 'K', "kilo" },
  { 'L', "lima" },
  { 'M', "mike" },
  { 'N', "november" },
  { 'O', "oscar" },
  { 'P', "papa" },
  { 'Q', "quebec" },
  { 'R', "romeo" },
  { 'S', "sierra" },
  { 'T', "tango" },
  { 'U', "uniform" },
  { 'V', "victor" },
  { 'W', "whiskey" },
  { 'X', "x-ray" },
  { 'Y', "yankee" },
  { 'Z', "zulu" }
};
```

The following code segments could also be helpful:

```
static const std::unordered_map<char, std::string> numbersToNato =
{
  { '1', "one" },
  { '2', "two" },
  { '3', "three" },
  { '4', "four" },
  { '5', "five" },
  { '6', "six" },
  { '7', "seven" },
  { '8', "eight" },
  { '9', "nine" },
  { '0', "zero" },
  { '.', "decimal" }
};
```

```
static const std::unordered_map<std::string, int> numbersToNatoMultipleDigits =
{
{ "ten", 10 },
{ "eleven", 11 },
{ "twelve", 12 },
{ "thirteen", 13 },
{ "fourteen", 14 },
{ "fifteen", 15 },
{ "sixteen", 16 },
{ "seventeen", 17 },
{ "eighteen", 18 },
{ "nineteen", 19 },

{ "twenty", 20 },
{ "thirty", 30 },
{ "fourty", 40 },
{ "fifty", 50 },
{ "sixty", 60 },
{ "seventy", 70 },
{ "eighty", 80 },
{ "ninety", 90 },

{ "hundred", 100 },
{ "thousand", 1000 }
};
```

In SVN in the file `designators.json` in folder `AlleGruppen/VonJedemInSeineUmgebungZuKopieren/Aufgabe08/Hilfscode` you find a subset of the three letter airline designators[1]. DLH stands for hansa or lufthansa, GTW for united states of america, DAL for delta. BAW for speed_bird or speedbird or speed bird etc.

Just for understanding your task you find here a subset of the 20-50 lines in SVN:

```
{
      "ABP": ["b_air"],
      "ACA": ["canada"],
      "AEG": ["airest", "eastern", "east air", "east"],
      "AFR": ["air_france", "france"],
      "BER": ["air_berlin", "berlin"],

      "DLH": ["lufthansa", "hansa"],
      "GEC": ["lufthansa cargo"],
      "LCI": ["lufthansa india"],
      "LHT": ["lufthansa technik"],

      "DAL": ["delta"],
      "DAT": ["deltair", "delta air"],

      "MHV": ["snowcap", "snow cap"],
      "NLY": ["flyniki", "fly niki", "fly_niki"]
}
```

[1] There are more than 7000 combinations, but we will use only some of them in our test data. Combinations not provided here, are not used in our test data – I hope so.

Implement a function `ReadUtteranceCheckCallsign`, which reads a file in the known format and extracts from each utterance (word sequence) the callsign and compares whether the extracted callsign is equal to the expected callsign (which you analyse by reading the utterance by yourself and set as desired value). The following file is an example:

```
2019-02-15__11-32-02-00:
  oscar echo india november kilo direct whisky whisky nine eight five
  OEINK DIRECT_TO WW985
2019-02-15__11-33-02-00:
  good morning lufthansa one two bravo descend eight zero
  DLH12B DESCEND 80 none
2019-02-15__11-34-02-00:
  gruess gott ryan_air seven seven delta kilo in radar contact
  RYR77DK INIT_RESPONSE
2019-02-15__11-35-02-00:
  standby
  NO_CALLSIGN NO_CONCEPT
2019-02-15__11-37-02-00:
  gruess gott lupus one one zero expect ils approach three four
  AYY110 EXPECT ILS 34
2019-02-15__11-38-02-00:
  gruess gott austrian seven seven seven sierra identified climb flight level two three zero
  AUA777S INIT_RESPONSE
  AUA777S CLIMB 230 FL
```

(./NumbersWithCallsignsEx1.txt)

First the keyword sequence `oscar echo india november kilo direct whisky whisky nine eight five` is read and the callsign `OEINK` is expected. The function has at least two parameters: parameter 1 specifies the full filename and parameter 2 is a boolean parameter. If set to `true` the keyword sequence is printed to cout, then the expected callsign is printed followed by the extracted callsign. If the parameter is false, no output is printed to cout. If the function detects a deviation it returns false, otherwise true.

Write at least two tests for the function `ReadUtteranceCheckCallsign`. In one test it should be tested whether the expected value `true` is returned and in the other test the expected value should be `false`.

8.3.1 Evaluation criteria

- Upload at least two screen dumps of your tests for the function `ReadUtteranceCheckCallsign` in `ReadUtteranceCheckCallsignTest01.jpg`, `ReadUtteranceCheckCallsignTest02.jpg` etc.
- Run the function `ReadUtteranceCheckCallsign` also on the file `NumbersWithCallsignsEx1.txt` shown above. Call the function with boolean parameter set to `true`, Make a screen dump of this screen output and upload it in file `NumbersWithCallsignsEx1.jpg` in the image folder.
- After you have uploaded your code, you will get after the deadline a new test file (currently not known to you). Run your function `ReadUtteranceCheckCallsign`

(again with boolean parameter set to `true`) on it and upload the screen dump to `callsignExtraction.jpg`

Exercise 8.2:

Use your previous implementation of your class `NumberExtractor` and extend it so that numbers belonging to a callsign are ignored.

The following code fragment shows some controller utterances from real world traffic together with the expected values:

```
"dobry den sky_travel five eight juliett ruzyne radar radar contact on present"
" heading descend four thousand feet qnh one zero two two",
                  //4000, 1022 (58 belongs to callsign)

"oscar kilo victor india kilo roger descend three thousand five hundred feet "
"squawk seven thousand", // 3500, 7000

"snow cab two hundred one descend eight thousand feet",
                    //8000 (201 belongs to callsign)

"austrian three nine two papa descend altitude one zero thousand "
"qnh one zero zero three", // 10000 1003 (392 belongs to callsign)

"fly_niki six hundred zulu contact tower now "
  "one two three point eight servus", // 123.8 (600 belongs to callsign)

"negative sir temperature two three instead of two one "
  "and dew point one three instead of one two", // 23, 21, 13, 12

"contact director one one nine eight two five goodbye"
   // 119.825 is meant, however, extraction should be 119825
```

More data for testing you have from previous test code.

Implement a function `ReadUtteranceExtractNumbers`, which reads a file in the known format and extracts from each utterance (word sequence) the numbers not belonging to the callsign.

The function has at least three parameters: parameter 1 specifies the full filename, parameter 2 is a container (e.g. a vector), which contains the expected numbers and parameter 3 is a boolean parameter. If set to `true` the keyword sequence is printed to cout, then the expected numbers are printed followed by the extracted numbers. If the parameter is false, no output is printed to cout. If the function detects a deviation it returns false, otherwise true.

Write at least two tests for the function `ReadUtteranceExtractNumbers`. In one test it should be tested whether the expected value `true` is returned and in the other test the expected value should be `false`.

8.3.2 Evaluation criteria

- Run the function `ReadUtteranceExtractNumbers` also on the file `NumbersWithCallsignsEx1.txt` shown above. Call the function with boolean

parameter set to `true`, make a screen dump of this screen output and upload it in file `ExtractedNumbersNoCallsign.jpg` in the image folder.

Exercise 8.3: If the keyword sequence contains the word `correction` consider this: We expect the following in callsign correction:

```
"oscar echo correction oscar delta india november kilo direct "
       " whisky whisky nine eight five" //--> ODINK
"good morning lufthansa correction speed bird one two bravo "
        " descend eight zero" // --> BAW12B

  gruess gott ryan air correction lufthansa eight correction lupus seven "
         " seven delta kilo in radar contact --> AYY77DK

  "gruess gott ryan air correction lufthansa correction"
         " i call you back" --> NO_CALLSIGN
```

For number extraction we would expect:

```
    "dobry den sky_travel five eight echo correction lufthansa juliett tango eight "
            " ruzyne radar radar contact on present"
    " heading descend four thousand feet qnh one zero two " "
              correction one zero one two", //4000, 1012

    "oscar kilo victor india kilo roger descend three thousand five correction "
         " four thousand feet "
    "squawk seven thousand correction heading zero one zero "
              " squawk seven zero zero zero ", // 4000, 10, 7000
```

8.3.3 Evaluation criteria

- Create an input file with the above four examples with **callsign correction** and run your function `ReadUtteranceCheckCallsign` on it with boolean parameter set to true. Upload the resulting screen dump to `CallsignExtrWithCorrection.jpg` and check the results.
- Create an input file with the above two examples with **number correction** and run your function `ReadUtteranceExtractNumbers` on it with boolean parameter set to true. Upload the resulting screen dump to `NumberExtrWithCorrection.jpg` and check the results.
- Furthermore you will get two unknown files and we have a *competition* between the bachelor and the master students. The best group gets most available additional points, the group on second position gets most available additional points minus one, etc.