Extraction of Callsigns

Table of Contents

This is the first part of the exercise to be done in November.

Learning Objectives

- Using STL
- Programming in a team, you may work in teams up to 5

Exercise at a glance

- Extract all callsigns from given utterances
- Calculate your extraction rate
- Consider also the word “*correction*”

A lot of code is mentioned here. You can download it from the SVN.

What is a Callsign?

Extract the callsign part from a given utterance, e.g.,

- "`oscar echo india november kilo direct whisky whisky nine eight five`". The expected output/callsign would be “OEINK”.
- "`good morning lufthansa one two bravo descend eight zero`". The expected output would be “DLH12B”.
- “`gruess gott ryan_air seven seven delta kilo in radar contact`”. The expected output would be “RYR77DK”.

- "`standby`": No callsign is in this utterance. The expected output would be "NO_CALLSIGN"
- "`gruess gott lupus one one zero expect ils approach three four`". The expected output would be "AYY110".

These are the easy examples, which will already cover 80% to 90% of the utterances, you will get as test cases. More challenging is already

- "`gruess gott austrian triple seven sierra identified climb flight level two three zero`". The expected output would be "AUA777S". Here the callsign was not in the first words.
- "`descending eight thousand feet direct dexon speed bird twenty nine seventeen`". The expected output would be "BAW2917". The callsign is even at the end of the utterance.
- "`speed bird twenty nine seventeen standby lufthansa four double alfa after air france taxi via november november 8 to delta four eight one`". The expected output would be "BAW2917" and "DLH4AA". "air france" is not used as a callsign.
- The callsign is not always in the beginning: "`climbing flight level two three zero austrian triple seven sierra`".

You will see more and more "interesting", i.e. challenging examples.

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\Aufgabe04\Hilfscode.`
in File `NATOAlphabet.cxx.`

```
static const std::unordered_map<char, std::string> lettersToNato =
{
{ 'A', "alfa" }, // it is alfa, not alpha
{ 'B', "bravo" },
{ 'C', "charlie" },
{ 'D', "delta" }, // there 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" },  // not xray
{ 'Y', "yankee" },
{ 'Z', "zulu" }  // not zoulou
};
```

The following code segments could also be helpful. It is in the same file:

```
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 completeDesignatorsShort.json in folder `AlleGruppen\VonJedemInSeineUmgebungZuKopieren\Aufgabe04\Data` 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` with a blank etc.

Just for understanding your task: You find here a subset in simple JSON format in SVN, e.g.:

```
{
"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"]
}
```

Detailed Exercise Description

Exercise 4-0: Who is doing what in your team

Send a short description, who will do what in your team, if you are working in a team. If you are working alone this is not necessary. It is currently enough to send a plan just for the part of the exercise described in this file.

If you are sending me this description until 2024-11-23 via email you have time for the whole exercise until 18-12-2024 otherwise until end of November.[2]

[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. You should find roughly 30 of them in the file "`completeDesignatorsShort.json`". More you find in "`completeDesignatorsLong.json`".
Internally (for my test and evaluation of your implementation) I will use a much long list, which you "never" get, but your code should be able to read also this much longer file.

[2] I hope it is clear. It is a must to send that list. You can change at any time, but you should have already now an idea, what you want to do (and I can give then early feedback whether you are working in the right direction or not).

Exercise 4-1: Read Utterance and extract callsigns

Implement a function `ReadUtteranceCheckCallsign`, which reads a file in the format shown below and extracts from each utterance (each word sequence) the callsign and compares, whether the extracted callsign is equal to the expected callsign (which is specified in the last line(s)).

The following file with name `UtterancesWithAnnotationsShort.txt` is an example:[3], [4]

```
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  CALL_YOU_BACK
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 CLIMB 230 FL
2019-02-15__11-39-07-00:
speed bird twenty nine seventeen standby lufthansa four double
alfa after air france taxi via november november eight to
delta four eight one
  BAW2917  CALL_YOU_BACK
  DLH4AA GIVE_WAY AFR none
  DLH4AA TAXI VIA N N8
  DLH4AA TAXI TO D481
2019-02-15__11-39-12-00:
```

[3] The file structure is always:

- Line 1: one-word file name, which always starts with "20*. The line can end with a colon ":" or just with a number.
- Line 2: an utterance consisting of an arbitrary number of words, in seldom cases the line could be empty.
- Line 3 (is optional): Might contain the first word "Csgn" followed by a colon ":" and then an arbitrary number of strings separated by white spaces might follow. All in the same line. The line starting with "Csgn" might also be missing. These are the number of callsign, which are currently known (are in the air). This key "Csgn" is not used in this example, see later examples.
- Line 3, 4, … or Line 4, 5: The extracted commands. For you, only the callsign is interesting. Sometimes there might be more than one callsign, but this is the exception. These lines contain more than one word.
- And then it starts again with a one-word line with next file name.

[4] When you are working in a group/team, one team member could e.g. write the code to process these files.

```
 hi united states of america two fox one foxtrot identified
climb flight level two three zero
  AAL2F1F  INIT_RESPONSE
  AAL2F1F CLIMB 230 FL
```

The function `ReadUtteranceCheckCallsign` has at least three parameters:

1. parameter 1 specifies the full filename of the file you want to read in from disk (this could be e.g. the string "R:`\VorlesungsUnterlagen\Betreuer\Uebungen\CodeDerAufgabenLoesungen\Aufgabe04\Data\UtterancesWithAnnotationsShort.txt`"
2. and parameter 2 is a boolean parameter. If set to `true,`
   - the one word filename,
   - the keyword sequence and expected and
   - extracted callsigns

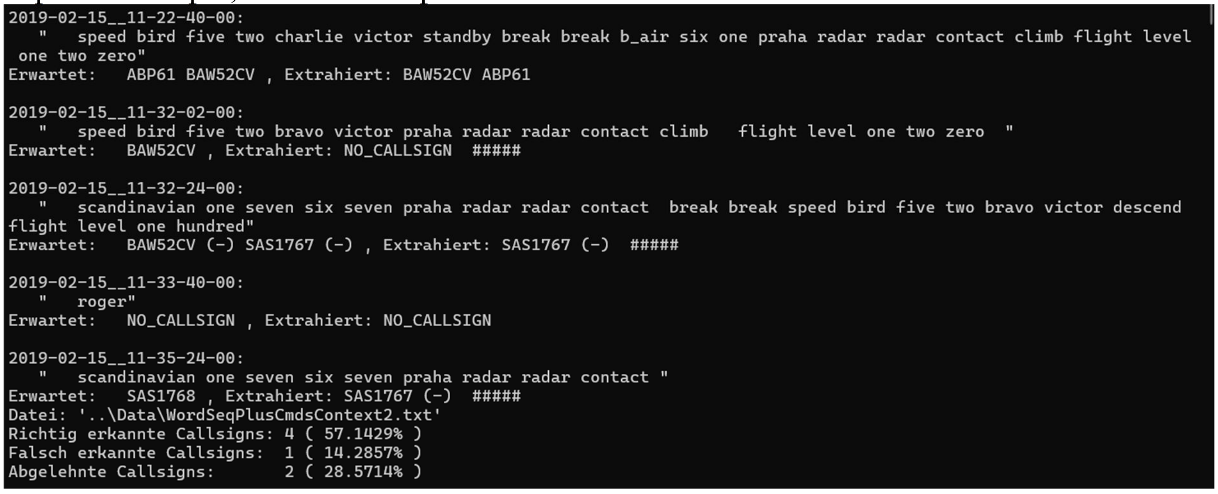
   are printed to `cout`.

   If the function detects a deviation between extracted and expected callsign it outputs after the extracted callsign the string "####".

   If the parameter is `false`, no output is printed to `cout`.
3. The third output parameter (new class *Evaluation* to be implemented by you) counts the number of read callsigns and also the number of correctly and wrongly extracted callsigns.[5] If parameter 2 is true, also output these values to cout. If you extract NO_CALLSIGN, but a callsign is expected, count this as a rejection, i.e. the class also needs a rejection counter.[6]

The function returns true, when no error occurs, while reading and processing the file, e.g. the file does not exist. The return value can also be true, if not all callsigns are correctly extracted.

A possible output, when second parameter is true could be:



```
2019-02-15__11-22-40-00:
   "   speed bird five two charlie victor standby break break b_air six one praha radar radar contact climb flight level
 one two zero"
Erwartet:   ABP61 BAW52CV , Extrahiert: BAW52CV ABP61

2019-02-15__11-32-02-00:
   "   speed bird five two bravo victor praha radar radar contact climb   flight level one two zero  "
Erwartet:   BAW52CV , Extrahiert: NO_CALLSIGN  #####

2019-02-15__11-32-24-00:
   "   scandinavian one seven six seven praha radar radar contact  break break speed bird five two bravo victor descend
flight level one hundred"
Erwartet:   BAW52CV (-) SAS1767 (-) , Extrahiert: SAS1767 (-)  #####

2019-02-15__11-33-40-00:
   "   roger"
Erwartet:   NO_CALLSIGN , Extrahiert: NO_CALLSIGN

2019-02-15__11-35-24-00:
   "   scandinavian one seven six seven praha radar radar contact "
Erwartet:   SAS1768 , Extrahiert: SAS1767 (-)  #####
Datei: '..\Data\WordSeqPlusCmdsContext2.txt'
Richtig erkannte Callsigns: 4 ( 57.1429% )
Falsch erkannte Callsigns:  1 ( 14.2857% )
Abgelehnte Callsigns:       2 ( 28.5714% )
```

The details of the output are not so important, but something should be output, which help me to understand, that your code is doing the right things and helps you also to test and debug your code.

[5] Attention. It could happen that you expect only one callsign, but you extract three different ones and vice versa.

[6] The interface of Evaluation is shown below.

It should be already clear, that this is not a trivial function, which requires writing some tests. Test first is a good idea. You are writing the test anyway, so do it early. Then you benefit from it, see next exercise.

It should be also clear, that this function is not only reading the file, but also needs to call another function (to be implemented by you) that performs the callsign extraction.

Later you will get some more files from me. Then you can easily evaluate the performance of your extraction algorithm and especially see, where your function still fails.

Here you see an example file with an empty utterance (which does not make sense, but just for your testing):

```
2019-02-15__11-32-02-00:

  Csgn: AUA774X BAW52CV
  NO_CALLSIGN NO_CONCEPT
2019-02-15__11-32-24-00:

  ABP61 INIT_RESPONSE
  ABP61 CLIMB 120 FL
2019-02-15__11-32-40-00:
  b_air six one praha radar radar contact climb flight level one two zero
   Csgn: ABP61
  ABP61 INIT_RESPONSE
  ABP61 CLIMB 120 FL
2019-02-15__11-33-40-00:
  roger
   Csgn: ABP61
  NO_CALLSIGN AFFIRM
```

And here is an example, with the keyword “Csgn“:

```
2019-02-15__11-22-40-00:
  speed bird five two charlie victor standby break break b_air six one praha
radar radar contact climb flight level one two zero
   Csgn: ABP61 BAW52CV
  BAW52CV STANDBY
  ABP61 INIT_RESPONSE
  ABP61 CLIMB 120 FL
2019-02-15__11-32-02-00:
  speed bird five two bravo victor praha radar radar contact climb   flight
level one two zero
  Csgn: AUA774X BAW52CV
  BAW52CV INIT_RESPONSE
  BAW52CV CLIMB 120 FL
2019-02-15__11-32-24-00:
  scandinavian one seven six seven praha radar radar contact  break break
speed bird five two bravo victor descend flight level one hundred
```

```
  SAS1767 INIT_RESPONSE
  BAW52CV DESCEND 100 FL
2019-02-15__11-33-40-00:
   roger
   Csgn: ABP61
  NO_CALLSIGN AFFIRM
2019-02-15__11-35-24-00:
  scandinavian one seven six seven praha radar radar contact
   Csgn: ABP61 SAS1768
  SAS1768 INIT_RESPONSE
```

The above example also shows (in yellow) that an utterance may contain more than one callsign. Three callsign and more are very, very seldom.

Interface of class “Evaluation”. Please implement this interface. The private part you can change as you need. You can also add more attributes and also more methods.

```
class Evaluation {
public:
   Evaluation()  { Reset(); }

    // main function for checking the expected, against extracted callsignn
    void CheckCallsign(const std::set<std::string>& antwortSet,
       const std::vector<std::string>& extraktions,
       const std::set<std::string>& ar_callsignSetThisUtter,
       bool ab_output);
    void PrintStatistics(std::string astr_dateipfad) const;

    int GetTotal() const { return mi_total; }
    int GetWrongRecogn() const { return mi_wrongRecogn; }
    int GetCorrectRecogn() const { return mi_correctRecogn; }
    int GetRejectedRecogn() const { return mi_rejectedRecogn; }

private:
   void Reset();
   int mi_wrongRecogn;
   int mi_correctRecogn;
   int mi_rejectedRecogn;
   int mi_total;
};
```

In the next exercise you see an example of a test, which uses this interface.

Exercise 4-2: Unit Tests

Write ***enough*** tests for the function `ReadUtteranceCheckCallsign`. In one test it should be tested, whether the expected output value `true` is returned and in one other test the expected value should be `false`. Test 3 tests e.g., whether the third parameter contains the correct and expected values. More tests are always better.

The following function shows a possible test:

```
bool ReadAllError()
{
   unique_ptr<ExtractionBase> p_extractor = make_unique <ExtractionHHe123>();
   ReadUtterAndGoldFromFile read(p_extractor.get());
   string filename = "..\\Data\\WordSeqPlusCmdsContext.txt";
   Evaluation eval;
   read.ReadUtteranceCheckCallsign(filename, g_outputToScreen, eval);

   EXPECT_EQ(0, eval.GetCorrectRecogn());
   EXPECT_EQ(4, eval.GetWrongRecogn());
   EXPECT_EQ(0, eval.GetRejectedRecogn());

   RETURN_TRUE_IF_NOT_GOOGLE_TEST;
}
```

`ExtractionHHe123` is just a dummy strub class, which always return the callsign HHe123.
It is a derived class of `ExtractionBase.`
Use also this interface for your classes for callsign extraction, which you will need in the next exercise (not described in this document).

```
class ExtractionHHe123 : public ExtractionBase
{
public:
   virtual ~ExtractionHHe123() {};
   virtual std::vector<std::string> extractCallsign(std::string phrase,
      const std::set<std::string>& ) override
   {
      // a callsign, which is always wrong
      return { "HHe123" };
   }
};
```

The base class is just an interface class (in C++ called abstract class).

```
class ExtractionBase
{
public:
   virtual ~ExtractionBase() {};
    // Extraction all callsigns from phrase,
    //    by considering the known callsigns in callsignSetThisUtter
    virtual std::vector<std::string> extractCallsign(std::string phrase,
       const std::set<std::string>& callsignSetThisUtter) = 0;
    virtual std::vector<std::string> extractCallsign(std::string phrase,
       const std::set<std::string>& callsignSetThisUtter, bool /* ab_pilot*/) {
       return extractCallsign(phrase, callsignSetThisUtter);}
};
```

The last bool parameter of the second method `extractCallsign` defines (with value true), whether the pilot is speaking. If the second method is not specified in the derived classes, the first method is called by called.
This bool value ab_pilot MIGHT be helpful for implementation of some heuristics, because the callsign is mostly said by the air traffic controller (ATCo) in the first words, whereas in pilot utterances the callsign is mostly said at the end of the utterance (except for initial calls, i.e. the first conversation between ATCo and pilot initiated by the pilot).

Your later implementation of the abstract class, will also need to read the file with the three letter codes AFR, DLH etc. You find a first example in `completeDesignatorsLonger.json`.

In the file googleTestEmulation.h you find a definition of the makros `EXPECT_EQ` and `RETURN_TRUE_IF_NOT_GOOGLE_TEST`. This enables you to use google-tests without adding the whole implementation of google-tests to your SVN.[7]

```
#define EXPECT_EQ(r, e) \
       if ((r) != (e) ) {\
              std::cerr << "Test failed for: " << #r << "==" << #e << std::endl; \
              return false; \
              }
#define RETURN_TRUE_IF_NOT_GOOGLE_TEST {return true;}
```

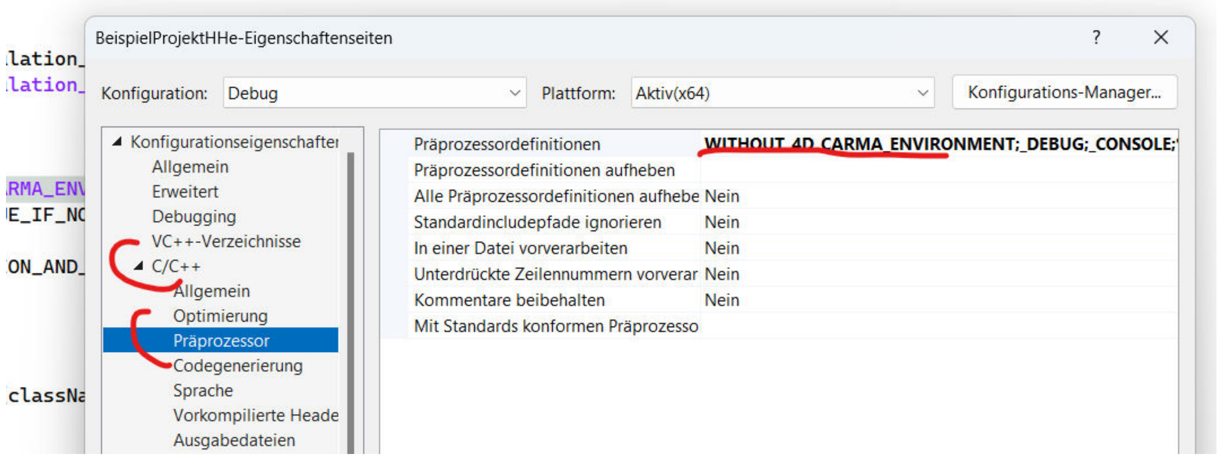
You need a class, which is extracting the callsigns.

%%

Evaluation criteria for Exercise 4-1 and 4-2

- Upload at least three screen dumps of your test code for the function `ReadUtteranceCheckCallsign` with the filenames ReadUtteranceCheckCallsignTest01.jpg, ReadUtteranceCheckCallsignTest02.jpg etc.
- Run the function `ReadUtteranceCheckCallsign` also on the file `UtterancesWithAnnotationsShort`.txt shown above.[8] Call the function with boolean parameter set to \ true. Make a screen dump of this screen output and upload it in file `UtterancesWithAnnotationsShort`.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. It should also run on my computer by just changing the file.

[7] I am also using google test, but do not rely on that. You need to define the preprocessor directive WITHOUT_4D_CARMA_ENVIRONMENT, when callsing the compiler. For the command line or for CMake-files you add -D`WITHOUT_4D_CARMA_ENVIRONMENT`
For Visual Studio you are changing the settings of your project:



BeispielProjektHHe-Eigenschaftenseiten

Konfiguration: Debug   Plattform: Aktiv(x64)   Konfigurations-Manager...

```
◢ Konfigurationseigenschafter
    Allgemein
    Erweitert
    Debugging
    VC++-Verzeichnisse
  ◢ C/C++
      Allgemein
      Optimierung
      Präprozessor
      Codegenerierung
      Sprache
      Vorkompilierte Heade
      Ausgabedateien
```

Präprozessordefinitionen	WITHOUT_4D_CARMA_ENVIRONMENT;_DEBUG;_CONSOLE;
Präprozessordefinitionen aufheben	
Alle Präprozessordefinitionen aufhebe	Nein
Standardincludepfade ignorieren	Nein
In einer Datei vorverarbeiten	Nein
Unterdrückte Zeilennummern vorverar	Nein
Kommentare beibehalten	Nein
Mit Standards konformen Präprozesso	

[8] This file does not contain the tag "Csgn:" You find the file also in data folder of "`..\AlleGruppen\VonJedemInSeineUmgebungZuKopieren\Aufgabe04`". The same file, but now with tag "Csgn" you find in file "`NumbersWithCallsignsEx2WitExpected.txt`".

Exercise 4-3: Consider “correction”

If the keyword sequence contains the word correction please also consider this, as described by the examples below:

We expect the following callsign(s) when considering 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, because the callsign is not clear.

Evaluation criteria for Exercise 4-3

- Create an input file with the above four examples and run your function ReadUtteranceCheckCallsign on it with boolean parameter set to true. Upload the resulting screen dump to file `CallsignExtrWithCorrection.jpg` and check the results.
- Furthermore, you will get an unknown file and we might have a *competition* between me and the other teams and your team.