Aufgabe 2.1

- Exercise·2.1¶

Re-implement·your·code·from·the·previous·exercise,·so·that·dynamic·C-Arrays·are·possible·inside·the·Levenshtein·distance·class.·Do·not·use·the·STL-template·class·`vector`·for·the·matrix.·For·the·rest·of·the·code·you·may·use·the·vector·template·class.¶

new und delete und new[] und delete[] verwenden.
Entsprechend der Vorlesung für Vektor

Bitte immer noch nicht die Klasse vector<T> zu diesem Zweck verwenden.
map, vector etc. darf natürlich weiterhin an anderer Stelle verwendet werden (aber nicht für die Matrix der LD-Berechnung).

Aufgabe 2.1

Exercise·2.1¶

Re-implement·your·code·from·the·previous·exercise,·so·that·dynamic·C-Arrays·are·possible·inside·the·Levenshtein·distance·class.·Do·not·use·the·STL-template·class·`vector`·for·the·matrix.·For·the·rest·of·the·code·you·may·use·the·vector·template·class.¶

```
class·Levenshtein¶
{¶
public:¶
···//·astr1¶
···Levenshtein(¶
·····const·std::vector<char>&·astr1,··//·first·string·compared·to·astr2¶
·····const·std::vector<char>&·astr2);·//·i.e.·LD·between·¶
·····//·astr1·and·astr2·is·calculated¶
···/*·…·*/¶
¶
private:¶
···//·contains·the·matrix,·used·for·LD·calculation·as·a·vector¶
···int*·mpi_mat;·//·Use·here·a·pointer·to·int·and·not·a·C-Array¶
···/*…·*/¶
};¶
```

It might be easier to implement it as a vector<vector<int>>, but for learning progress and future exercise, we just use this approach.

Aufgabe 2.1: Testen

- Did you implement tests which show the functionality of dynamic array sizes (e.g. calling Resize or something similar) so that the array sizes is **dynamically increased** (three times is enough)?
- Did you implement tests, which show the functionality of dynamic array sizes (e.g. calling Resize or something similar) so that the array size can be **decreased at runtime**?
- Did you implement tests, which show that array size can be **decreased and increased** and runtime?

Angepasste Schnittstelle hilfreich

```
çľǎşş Ľêŵêŋşḥʧêîŋ
řụčľîç
Ľêŵêŋşḥʧêîŋ
      çộŋşʧ şʧđ  ŵêçʧộs Ţ Bǎşêţỳřê   ǎx,
      çộŋşʧ şʧđ  ŵêçʧộs Ţ Bǎşêţỳřê   ǎx,

ŵộîđ ŞêʧŊêxẄộsđş
      çộŋşʧ şʧđ  ŵêçʧộs çḥǎs   ǎşʧs,
       çộŋşʧ şʧđ  ŵêçʧộs çḥǎs   ǎşʧs,
   îŋʧ ĞêʧCụssêŋʧŇǎʧsîẏŞîćê  çộŋşʧ

   îŋʧ CǎľçĽêŵêŋşḥʧêîŋDîşʧǎŋçê
   şʧđ  şʧsîŋĝ čǎçlʧsǎçê   çộŋşʧ
```

Aufgabe 2.1: Testen

- Did you implement tests which show the functionality of dynamic array sizes (e.g. calling Resize or something similar) so that the array sizes is **dynamically increased** (three times is enough)?
- Did you implement tests, which show the functionality of dynamic array sizes (e.g. calling Resize or something similar) so that the array size can be **decreased at runtime**?
- Did you implement tests, which show that array size can be **decreased and increased** and runtime?

Aufgabe 2.2

- Exercise·2.2¶

Split·your·code·into·different·files,·e.g.¶

- → One·main·file,·which·executes·the·tests¶
- → Header·and·Source-Code·file·for·the·class·Levenshtein·distance·calculation¶
- → Header·and·Source-Code·files·for·the·different·tests¶
- → …¶

Use·include·guards·(#ifndef,···#define···or/and··#pragma·once).¶

Einige von Ihnen haben dieses schon bei der ersten Abgabe berücksichtigt.

Aufgabe 2.3

Exercise 2.3

Split your main program into two parts. If it is called with the command line parameter `--test` all the tests are executed.

Otherwise a test file (you decide which) with some ATC utterances is read in and the contents is output. Later you will also output the extracted callsigns. Currently it is enough to just use the `--test` parameter in main.

Your `main` could look like this:

```
int main(int argc, char* argv[])
{
   if (argc > 1 && string(argv[1]) == "--test")
   {
      bool result = true;
```

Dummy Check for Memory Leaks

It should be also possible to add another value to „--test“, e.g. „--test 1000“.
Then all tests are executed in a loop 1000 times.

Exercise 2.4 is then to use Windows task manager and visualize the used memory of your process, when you call 1000 times or more.
If the used memory size increases you very probable have a memory leak.