Chapter 5

Exercise: Dynamic Array Implementation by Pointers

5.1 Leaning Objectives

References to literature are provided in brackets:

- Pointer [8, Ch. 4],
- `new` and `delete` [8, Ch. 4].

5.2 Exercises

In this exercise we do not implement new functionality to reach the final aim of automatic annotation of given ATCo transcriptions, but just learn how to implement dynamic C-arrays and split the code into different files.

Hint for supervisors:
see directory `03b_ExeDynArrays` for solution code

5.3 Exercises at a glance

Exercise 5.1 Split your implementation into different files.

Exercise 5.2 Read the transcriptions and annotations from a file, store the contents in appropriate C-arrays of arbitrary size (not just limited to e.g. 10).

Exercise 5.3 Read commands and utterances from a file print them to screen.

5.4 Detailed Exercise Descriptions

Exercise 5.1:

Take the solution from the previous exercises.

Split your implemenation into different files.

Use include guards (`#ifndef` `#define` or `#pragma once`).

Use e.g. a header file `FileName.h` and a source file for type `FileName.cxx` for the `struct FileName`. Use also files `AtcoCommand.h` and `AtcoCommand.cxx`. Use different files for the functionality and for the tests.

For evaluation of this exercise I will just check that your code runs and consists of multiple file, when I am evaluating the next exercise.

Exercise 5.2:

Do not use the template-class `vector` from the `STL`.[1] If you have used it during the first exercise, great!!! You know already a lot of C++. Nevertheless replace it now fully by a C-array, i.e. `arr[1000]`.

Refactor now the size of each C-array by 2 (do it one after the other). Start with the first refactoring. Your program should crash or the test should fail. Replace now the C-array by a pointer as it was shown during the lecture for the `struct Vektor`.

```
AtcoCommand gp_atcoUtterances[1000];
```

is first replace by

```
AtcoCommand gp_atcoUtterances[2];
```

and then by

```
AtcoCommand* gp_atcoUtterances;
```

Now your (hard) work starts.

Read the contents of the command file of the previous exercise into a C-array of arbitrary length, i.e. use `AtcoCommand* gp_atcoUtterances;` Now implement the array as a dynamic array and demonstrate by suitable tests that your codes runs as required.

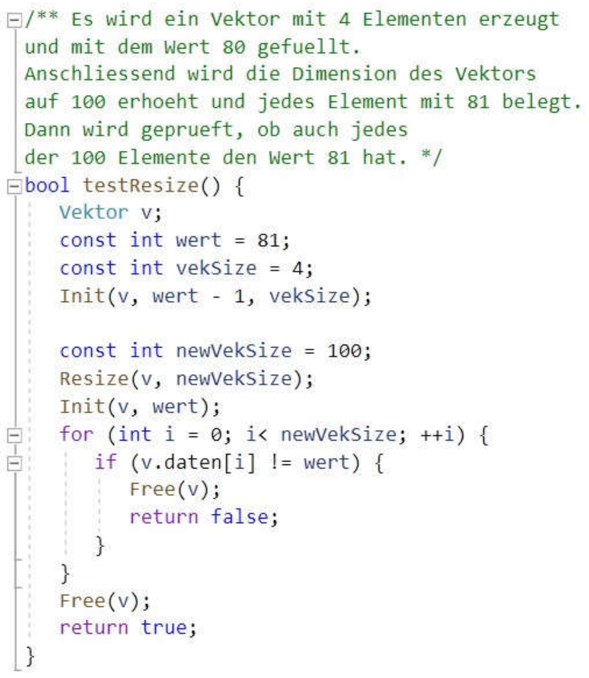
```
// dynamic array to store the commands and the word sequences
AtcoCommand* gp_atcoUtterances = nullptr;
// number of elements currently used in gp_atcoUtterances
int g_actoUtterancesCnt = 0;
// maximal number of elements currently usable in gp_atcoUtterances
int g_max_utterance_cnt = 0;
```

You need to call `new` to increase the size of the structure on the heap. Do not just use `AtcoCommand* gp_atcoUtterances = new AtcoCommand[1000];` This could be only

[1] Normally it is a good idea to use vector from STL, but for learning purposes we will reimplement this class during this lecture.

the first step. You should resize the size dynamically as needed during the runtime of your program. Implement a test (as testResize in the lecture). Make a screen dump from this test and upload it as jpg file to your SVN.

The file name should be exercise-5-2-XX.jpg (for jpg files). For the first image XX is 01, for the second it is 02... If the output/code does not fit into one image, create more. The following figure 5.1 shows the test implemented in the lecture. Adapt according to your needs.



```
/** Es wird ein Vektor mit 4 Elementen erzeugt
und mit dem Wert 80 gefuellt.
Anschliessend wird die Dimension des Vektors
auf 100 erhoeht und jedes Element mit 81 belegt.
Dann wird geprueft, ob auch jedes
der 100 Elemente den Wert 81 hat. */
bool testResize() {
    Vektor v;
    const int wert = 81;
    const int vekSize = 4;
    Init(v, wert - 1, vekSize);

    const int newVekSize = 100;
    Resize(v, newVekSize);
    Init(v, wert);
    for (int i = 0; i< newVekSize; ++i) {
        if (v.daten[i] != wert) {
            Free(v);
            return false;
        }
    }
    Free(v);
    return true;
}
```

Figure 5.1: Example screen dump for testResize (you may choose a different test name

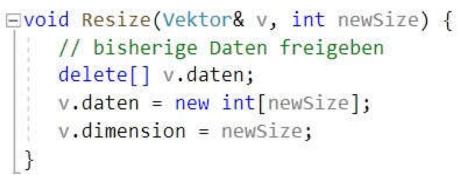
Do not forget to also call `delete` for arrays, when the heap memory is not needed anymore.

Make also a screen dump of your `Resize` function[2] and store also in SVN as file `exercise5-2-ResizeXX`[3] An example of a possible screen dump is shown in Figure 5.2.

Now continue with the other C-arrays you have in your code. Maybe you have stored the command types, which your read from file `expectedTypes.txt` in a vector or C-array. Convert them accordingly as done for `AtcoCommand`. Maybe you also stored

[2]maybe it has a different name in your code, up to you

[3]Maybe you have more than one screen dump for `Resize` if you have different implementations for different types. XX could be then 01, 02, etc.



```
void Resize(Vektor& v, int newSize) {
    // bisherige Daten freigeben
    delete[] v.daten;
    v.daten = new int[newSize];
    v.dimension = newSize;
}
```

Figure 5.2: Example screen dump for Resize (you may choose a different test name

the container of words and how often they occur in a C-array of a `struct`, so that you could sort them with e.g. heap sort.

Attention: Do not try to also make the number of commands of an utterance dynamic. Here we need a dynamic array in a dynamic array. This is also possible, but we learn easier ways together with the Copy-Constructor later. So you can keep the constant `6` as shown in code below:

```
struct AtcoCommand
{
  string m_wordSeq;
  enum { MAX_CMDS_PER_UTTERANCE = 6 };
  // do not try to also make this number dynamic
  string m_cmds[MAX_CMDS_PER_UTTERANCE];
  int    m_cmdCnt;
  /*...*/
```

The same applies for the C-array of length `MAX_LINE_LENGTH` which you might use to read lines from the input file.

```
  ifstream file(astr_filenameWithExpectedTypes, ios::in);
  \* ...*\
  char line[MAX_LINE_LENGTH];
  \* ...*\
  while (type.length() > 0 && file.good())
  {
    file.getline(line, MAX_LINE_LENGTH);
  \* ...*\
```

Keep it as it is, or use an instance of `std::string`.

Exercise 5.3:

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) is read in and the contents read in is printed to the screen.

Your main could look like this:

```
int main(int argc, char* argv[])
/***************************************************************************/
/***************************************************************************/
/***************************************************************************/
{
  if (argc > 1 && string(argv[1]) == "--test")
  {
    bool result = true;
    PERFORM_AND_OUTPUT(checkTop8InBigFile);
    PERFORM_AND_OUTPUT(checkTop5InMediumFile);
    if (true == result) {
      printScreenColorOnceVal(cout, GREEN_SCREEN_COLOR, "Tests erfolgreich\n");
      return 0;
    }
    else {
      printScreenColorOnceVal(cout, RED_SCREEN_COLOR,
        " Fehler in Tests aufgetreten! ***\n");
      return -1;
    }
  }
  ReadFileAndPrintContents();
  return 0;
}
```

`ReadFileAndPrintContents` could be:

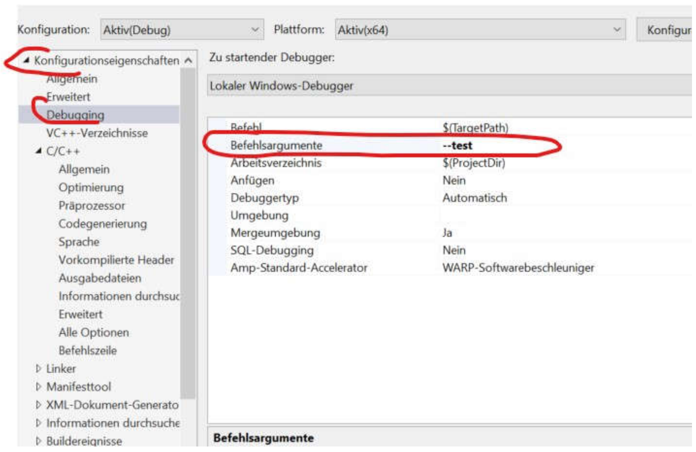
```
void ReadFileAndPrintContents()
/* We read the commands and utterances from a file and just print them to the screen
*/
{
  string filename = AppendFilenameToPath("data\\WordSeqPlusCmds.txt");

  if (!ReadCommandsFromFile(filename))
  {
    return;
  }

  cout << "We read " << g_actoUtterancesCnt << " different files from the file "
    << filename << ".\n";
  // Output of files with words and extracted commands
  for (int i = 0; i < g_actoUtterancesCnt; ++i)
  {
    cout << gp_atcoUtterances[i].m_filename.mstr_filename << " ("
      << gp_atcoUtterances[i].m_filename.m_day << "-"
      << gp_atcoUtterances[i].m_filename.m_month << "-"
      << gp_atcoUtterances[i].m_filename.m_year << ", "
      << gp_atcoUtterances[i].m_filename.m_hour << ":"
      << gp_atcoUtterances[i].m_filename.m_min << ":"
      << gp_atcoUtterances[i].m_filename.m_sec << "):\n ";
    cout << gp_atcoUtterances[i].m_wordSeq << ":\n ";
    for (int j = 0; j < gp_atcoUtterances[i].m_cmdCnt; ++j)
    {
      cout << gp_atcoUtterances[i].m_cmds[j] << "; ";
    }
    cout << endl;
  }// for i
} // ReadFileAndPrintContents
```

The above program benefits from the possibility that you can pass parameters via the command line, when you call a C++-program. You can provide parameters via the IDE of Visual Studio as shown in Figure 5.3.

Execute your program now in test mode (parameter –test) and create a screen dump. The screen dump could look like shown in Figure 5.4

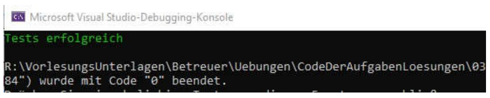


Zu startender Debugger:
Lokaler Windows-Debugger

Befehl	$(TargetPath)
Befehlsargumente	**--test**
Arbeitsverzeichnis	$(ProjectDir)
Anfügen	Nein
Debuggertyp	Automatisch
Umgebung	
Mergeumgebung	Ja
SQL-Debugging	Nein
Amp-Standard-Accelerator	WARP-Softwarebeschleuniger

Befehlsargumente

Figure 5.3: Passing parameters via command line to a C++-program



```
Tests erfolgreich

R:\VorlesungsUnterlagen\Betreuer\Uebungen\CodeDerAufgabenLoesungen\03
84") wurde mit Code "0" beendet.
```

Figure 5.4: Possible output of main, when run in test mode

Execute your program now NOT in test mode and create a screen dump, when you read your input file. The screen dump could look like shown in Figure 5.5. Store your screen in SVN in the filen with ame `exercise5-3-ProgramTest01.jpg`.

Store the image in SVN with the filename `exercise5-3-ProgramRuns01.jpg`. Later, after deadline for code uploading, you will get another test file from me and you will just replace your input file with the word sequences and commans by the one from me and upload the new screen output also to SVN as file `exercise5-3-ProgramRuns02.jpg`.

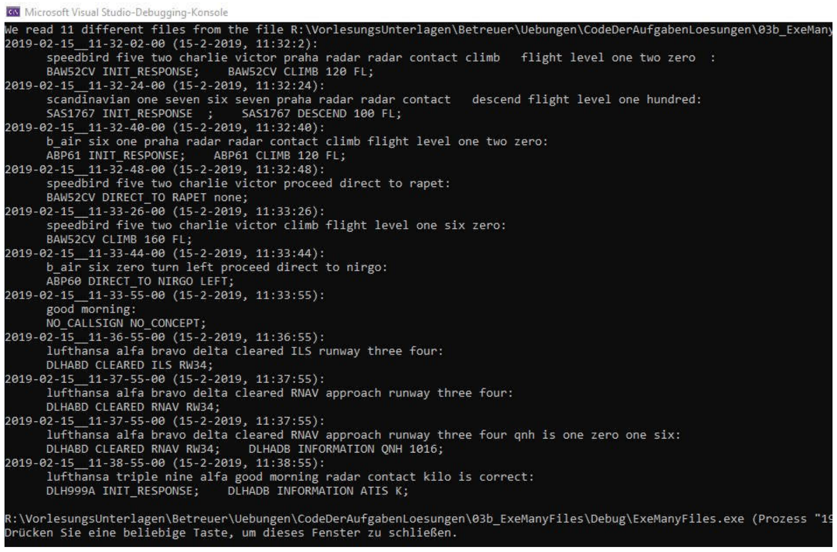
5.5 Tipps, Tricks and Constraints

If you are not developing under Windows (e.g. Linux) and not with Visual Studio, please make sure that all your files are in one directory (header, source, data files) or provide a cmake file.

Do not use path dependent file pathes at different places in your program, i.e. do not write

```
  ofsteam str("C:\\VorlesungsUnterlagen\\Betreuer\\Uebungen\\"
          "CodeDerAufgabenLoesungen\\03b_ExeManyFiles\\x.txt";
```

better use a function as e.gl `AppendFilenameToPath`, shown in the following code fragments:



```
We read 11 different files from the file R:\VorlesungsUnterlagen\Betreuer\Uebungen\CodeDerAufgabenLoesungen\03b_ExeMany
2019-02-15__11-32-02-00 (15-2-2019, 11:32:2):
      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 (15-2-2019, 11:32:24):
      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 (15-2-2019, 11:32:40):
      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 (15-2-2019, 11:32:48):
      speedbird five two charlie victor proceed direct to rapet:
      BAW52CV DIRECT_TO RAPET none;
2019-02-15__11-33-26-00 (15-2-2019, 11:33:26):
      speedbird five two charlie victor climb flight level one six zero:
      BAW52CV CLIMB 160 FL;
2019-02-15__11-33-44-00 (15-2-2019, 11:33:44):
      b_air six zero turn left proceed direct to nirgo:
      ABP60 DIRECT_TO NIRGO LEFT;
2019-02-15__11-33-55-00 (15-2-2019, 11:33:55):
      good morning:
      NO_CALLSIGN NO_CONCEPT;
2019-02-15__11-36-55-00 (15-2-2019, 11:36:55):
      lufthansa alfa bravo delta cleared ILS runway three four:
      DLHABD CLEARED ILS RW34;
2019-02-15__11-37-55-00 (15-2-2019, 11:37:55):
      lufthansa alfa bravo delta cleared RNAV approach runway three four:
      DLHABD CLEARED RNAV RW34;
2019-02-15__11-37-55-00 (15-2-2019, 11:37:55):
      lufthansa alfa bravo delta cleared RNAV approach runway three four qnh is one zero one six:
      DLHABD CLEARED RNAV RW34;    DLHADB INFORMATION QNH 1016;
2019-02-15__11-38-55-00 (15-2-2019, 11:38:55):
      lufthansa triple nine alfa good morning radar contact kilo is correct:
      DLH999A INIT_RESPONSE;    DLHADB INFORMATION ATIS K;

R:\VorlesungsUnterlagen\Betreuer\Uebungen\CodeDerAufgabenLoesungen\03b_ExeManyFiles\Debug\ExeManyFiles.exe (Prozess "19
Drücken Sie eine beliebige Taste, um dieses Fenster zu schließen.
```

Figure 5.5: Possible output of main

```
bool checkTop8InBigFile()
/* We calculate for each word in BigWordSeqPlusCmdsFile.txt how often
   it occurs and check for the first 8 with the expected counts.
*/
{
   string filename = AppendFilenameToPath("data\\BigWordSeqPlusCmdsFile.txt");

   if (!ReadCommandsFromFile(filename))
```

You could use the following implementation (of course your basic path is different):

```
string GetPathPrefix()
{
#ifdef _WIN32 // on Windows system
   return "R:\\Uebungen\\CodeDerAufgabenLoesungen\\03b_ExeManyFiles";
#else
   // \toDo not implemented
   return "./";
#endif
}

string AppendFilenameToPath(std::string astr_filename)
{
#ifdef _WIN32 // on Windows system
   return GetPathPrefix() + "\\" + astr_filename;
#else
   return GetPathPrefix() + "/" + astr_filename;
#endif
}
```