SEN-Übung 2.3

Schett Matthias

6. April 2013

1. Aufgabe 1

1.1. Lösungsidee

Es soll ein Programm erstellt werden dass Testdaten nach folgender Grammatik ausgibt:

```
TestData = Header { Value "," } .
Header = "Values" "/" Counter ":".
Counter = Digit {Digit} .
Value = Digit {Digit} .
Digit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" |
    "0".
```

Die einzelnen Daten sollen mittels Zufallszahlengenerator ermittelt werden. Zu Beginn wird der Dateikopf geschrieben, die übernimmt die interne writeFileHeader() Funktion. Danach werden die einzelnen Werte ermittelt und ebenfalls in die Datei geschrieben Die Angaben zum Dateinamen und der Anzahl der zu generierenden Testdaten, wird mittels Kommandozeilenparamenter übergeben.

Beispiel TestDataGenerator.exe TestFile1.txt 300
Generiert ein TestFile1.txt mit 300 Testwerten.

1.2. Programmcode in C++

Der Programmcode Aufgabe befindet sich im Anhang

1.3. Testfälle

Als Testfälle werden 3 Dateien erzeugt TestFile1.txt mit 10, TestFile2.txt mit 20 und TestFile3.txt mit 24 Einträgen. Der Aufruf sieht daher so aus:
TestDataCreator.exe TestFile1.txt 10 TestFile2.txt 20 TestFile3.txt 24

2. Aufgabe 2

2.1. Lösungsidee

Aufgabe 1 soll nun so erweitert werden, dass die Grammatik der erstellten Dateien geprüft wird und eine Schnittmenge aller Test Daten in eine Result.txt geschrieben werden. Dafür gibt es nun die Methoden checkGrammar(). checkGrammar() prüft zuerst den Dateikopf und anschließend den Rest der Datei bei einem Fehler wird ünkown file formatünd die Stelle an der sich der Fehler befindet ausgegeben.

Anschließend werden unter zuhilfe Name von std::set(siehe 2.1.1) alle Werte der Datei gepseichert.

Das wird mit allen Dateien wiederholt. Anschließend wird über den Inhalt der std::set Datenstruktur iteriert und jeder Eintrag in die Result.txt geschrieben.

2.1.1. Informationen zu std::set

Die Datenstruktur std::set ist eine dynamische Datenstruktur ähnlich der einzel und doppelt verketteten Listen. Alle in ihr befindlichen Einträgen sind aufsteigend sortiert. Weiters sind alle Einträge in einem std::set einzigartig, dieser Umstand wird bei jedem insert() geprüft, dadurch eignet sie sich besonders für diese Aufgabe.

2.2. Programmcode in C++

Der Programmcode befindet sich im Anhang

2.3. Testfälle

Die Testfälle sind indentisch mit denen aus Aufgabe 1

A. Programmcode in C++

A.1. CheckGrammar Modul Header

```
1  ////////////////////////////////////
2  // Workfile    : GrammarCheck.cpp
3  // Author      : Matthias Schett
4  // Date        : 06-04-2013
5  // Description : Creates test data
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include "scanner.h"
11 #include <fstream>
12 #include <string>
13 #include <iostream>
14 #include <set>
15
16 bool checkGrammar(std::string fileName, std::set<double> &
       valueSet);
```

A.2. CheckGrammar Modul Implmentierung

```
1  ////////////////////////////////////
2  // Workfile    : GrammarCheck.cpp
3  // Author      : Matthias Schett
4  // Date        : 06-04-2013
5  // Description : Creates test data
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9  #include "GrammarCheck.h"
10 #include <set>
11 using namespace std;
12
13 void printError(string fileName, scanner & scan){
14     cout << "unkown file format for file " << fileName << endl
           ;
15     cout << "Error is near: " << scan.get_string();
16 }
17
18 bool isTbWord(scanner & scan){
19     return scan.symbol_is_identifier();
```

```
20 }
21
22 bool isTbInt(scanner & scan){
23     return scan.symbol_is_integer();
24 }
25
26 bool isTbReal(scanner & scan){
27     return scan.symbol_is_real();
28 }
29
30 bool isTbDiv(scanner & scan){
31     return scan.symbol_is_division();
32 }
33
34 bool isTbColon(scanner & scan){
35     return scan.symbol_is_colon();
36 }
37
38 bool isTbComma(scanner & scan){
39     return scan.symbol_is_comma();
40 }
41
42 bool checkHeader(scanner & scan){
43     if(isTbWord(scan)){
44         string word = scan.get_string();
45         if(word == "Values"){
46             scan.next_symbol();
47             if(isTbDiv(scan)){
48                 scan.next_symbol();
49                 if(isTbInt(scan)){
50                     if(scan.get_integer() < 1000){
51                         scan.next_symbol();
52                         if(isTbColon(scan)){
53                             scan.next_symbol();
54                             return true;
55                         }
56                     }
57                 }
58             }
59         }
60     }
61     return false;
62 }
63
```

```
64  // Checks for duplicates and removes them -> also shows the
        caller if there was a duplicate
65  bool checkForDuplicate(set<double> & valueSet, double value){
66      pair<set<double>::iterator, bool> ret = valueSet.insert(
            value);
67      return ret.second;
68  }
69
70  void checkValues(scanner & scan, set<double> & valueSet,
        string fileName){
71
72      while(!scan.symbol_is_eof()){
73          if(isTbInt(scan)){
74              int value = scan.get_integer();
75              checkForDuplicate(valueSet, value);
76          } else if(isTbReal(scan)){
77              double value = scan.get_real();
78              checkForDuplicate(valueSet, value);
79          } else if(isTbComma(scan)){
80              // DO NOTHING
81          } else {
82              printError(fileName, scan);
83          }
84
85          scan.next_symbol();
86      }
87  }
88
89  bool checkGrammar(string fileName, set<double> & valueSet){
90      ifstream file(fileName);
91      scanner scan (file);
92
93      bool result = true;
94      if(checkHeader(scan)){
95          checkValues(scan, valueSet, fileName);
96      } else{
97          printError(fileName, scan);
98          result = false;
99      }
100     file.close();
101     return result;
102 }
```

A.3. TestDataCreator Modul Header

```
1  ////////////////////////////////////
2  // Workfile    : TestDataCreator.h
3  // Author      : Matthias Schett
4  // Date        : 06-04-2013
5  // Description : Creates test data
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include "RandomGen.h"
11 #include <fstream>
12 #include <string>
13 #include "GrammarCheck.h"
14
15 // Creates TestData
16 void CreateTestData(int argc, char *argv[]);
17 // Creates a single Test File
18 void CreateTestFile(std::string fileName, size_t numOfEntries)
   ;
```

A.4. TestDataCreator Modul Implmentierung

```
1  ////////////////////////////////////
2  // Workfile    : TestDataCreator.cpp
3  // Author      : Matthias Schett
4  // Date        : 06-04-2013
5  // Description : Creates test data
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9  #include "TestDataCreator.h"
10 #include <iostream>
11 #include <set>
12 using namespace std;
13
14 // Writes the file header
15 void writeFileHeader(ofstream & file, size_t numOfEntries){
16     file << "Values" << "/" << numOfEntries << ":\n";
17 }
18
19 // Generates random values between 0 and 1000
20 int getRandomNumber(){
21     return rgen::GetRandVal(0, 1000);
```

```
22 }
23
24 // Writes the values to the file
25 void writeDataToFile(ofstream & file){
26     file << getRandomNumber() << ",";
27 }
28
29 // Checks if the file has .txt as file extension
30 string checkForFileExtension(string fileName){
31     if(fileName.find_last_of(".txt") == string::npos){
32         fileName.append(".txt");
33     }
34     return fileName;
35 }
36
37 void CreateTestFile(std::string fileName, size_t numOfEntries)
      {
38     ofstream file;
39     file.open(checkForFileExtension(fileName));
40     // Init random generator once for each file
41     rgen::Init();
42
43     writeFileHeader(file, numOfEntries);
44     for(size_t i = 0; i <= numOfEntries; i++){
45         writeDataToFile(file);
46     }
47
48     file.close();
49     cout << "Test Data created" << endl;
50 }
51
52 // Writes overlap to file
53 void writeResultFile(ofstream & file, set<double> & valueSet){
54     for(set<double>::iterator it = valueSet.begin(); it !=
          valueSet.end(); ++it){
55         file << "  " << *it;
56     }
57     file.close();
58 }
59
60 void CreateTestData(int argc, char *argv[]){
61     if(argc < 2){
62         cout << "Es wurden zu wenig Parameter uebergeben.\
              nBeispiel: TestDataCreator.exe TestFile 1000" <<
              endl;
```

```
63      } else if(argc > 1000){
64          cout << "Es wurden zu viele Werte angefordert, es
                duerfen maximal 1000 Werte generiert werden" <<
                endl;
65      } else if((argc - 1) % 2 == 0) {
66          set<double> valueSet;
67          for(int i = 1; i < argc; i+=2){
68              int numOfEntries = atoi(argv[i+1]);
69              if(numOfEntries < 0){
70                  cout << "Anzahl der Eintraege darf nur Positiv
                        sein" << endl;
71                  numOfEntries -= -1;
72              }
73              string fileName(argv[i]);
74              CreateTestFile(fileName, numOfEntries);
75              set<double> tempSet = valueSet;
76              if(checkGrammar(fileName, tempSet)){
77                  valueSet = tempSet;
78              }
79          }
80          ofstream resultFile("Result.txt");
81          writeResultFile(resultFile, valueSet);
82      } else{
83          cout << "Fehlerhafte Parameter Uebergabe.\nBeispiel:
                TestDataCreator.exe TestFile 1000" << endl;
84      }
85  }
```

A.5. Testreiber

```
1  ////////////////////////////////////////
2  // Workfile    : Main.cpp
3  // Author      : Matthias Schett
4  // Date        : 06-04-2013
5  // Description : Creates test data
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////
9
10 #include <iostream>
11 #include "TestDataCreator.h"
12 #include <string>
13
14 using namespace std;
15
16 int main(int argc, char *argv[]){
17
18     CreateTestData(argc, argv);
19
20     cin.get();
21     return 0;
22 }
```