Draft der C++-Vorlesung vom 24.März 2023

1. Strukturierte Programmierung: Siehe [Haase23, §4].
   Kurze Demo der Loops-Examles 0, 1, 2, 3: Kommentare, Einrückungen etc.

2. Demonstration der Nutzung von `string`, `vector`, `array` [Haase23, §2.3.1-2.3.3].
   Code: demoString[1], demoVector[2]. strings_c_cpp[3]

3. Dynamischer Vektor [Haase23, §5.1.1] und dessen Nutzung als Input- oder Output-parameter in Funktionen.
   Code: v_2c[4] mit docu[5].

4. Beispiel Geheimzahl mit Speicherung aller Versuche (dynamischer Vektor) und einem Vektor von Strings zur flexiblen Ausgabe.
   Code: v_3a[6] mit docu[7].

   Reprise 17. März

   - Auch mit `array` [Haase23, §5.1.2] von Strings möglich (`main.cpp:51`).
   - Zufallszahlengenerierung (`main.cpp:108-112`) im Intervall $[anf, ende]$ mit C+++-Zufallszahlen `#include <random>`, zur Vertiefung[8].
   - Benötigt die Compileroption `-std=c++11` (oder höheres wie `-std=c++17`)

5. Trennung in Header- und Source-Files (Deklaration vs. Definition)
   Signum-Fkt. im Beispiel v_4c[9]: [clever[10]]
   Um mehrfaches Einbinden desselben Headerfiles in einem Sourcefile zu verhindern, wird das *header guarding* angewandt (`v_4c/fkt.h`):

```
#ifndef FKT_H_INCLUDED
   #define FKT_H_INCLUDED
   ...                                   // Unsere Deklarationen
#endif
```

   Dies wird durch das **global eindeutigen** Macro `FKT_H_INCLUDED` garantiert.
   Neue C++-Lösungsmöglichkeit:
   Stattdessen Voranstellen von `#pragma once`[11].

6. IO-System und File-IO: Die Ein- und Ausgabe von Daten in Programmen erfolgt über Datenströme (*stream*). Sie kennen bereits die Datenströme `cout` und `cin` mit ihren zugehörigen Operatoren << und >>. Zusätzlich haben wir die Fehlerausgabe über `cerr` welche ebenfalls mit `#include <iostream>` inkludiert wird.
   Diese Datenströme lassen sich aus/in Files umlenken [Haase23, §8], was bei größeren Datenmengen natürlich einen enormen Vorteil darstellt.

   - Statt mit `cout << d` geben wir eine double-Variable `d` in das File *out.txt* aus, siehe dazu Beispiel v_5a[12]:

```
#include <fstream>                       // ofstream
#include <iostream>
```

[1] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/demoString.cpp
[2] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/demoVector.cpp
[3] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/strings_c_cpp.zip
[4] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/v_2c.zip
[5] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/v_2c/html
[6] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/v_3a.zip
[7] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/v_3a/html
[8] https://en.cppreference.com/w/cpp/header/random
[9] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/v_4c.zip
[10] https://stackoverflow.com/questions/1903954/is-there-a-standard-sign-function-signum-sgn-in-c-c
[11] https://de.wikipedia.org/wiki/Include-Guard
[12] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/v_5a.zip

```
using namespace std;
...
ofstream out_file("out.txt");                    // ASCII-File als Ausgabefile
out_file << d;
```

- Wir haben ein Text/ASCII[13]-file benutzt, Binärfiles sind ebenfalls möglich[14].
- Statt zwei String-Variablen `c1`, `c2` mit `cin>>c1>>c2` über die Tastatur einzugeben, lesen wir diese vom ASCII-File *my_in.txt* ein.

```
#include <fstream>                                    // ofstream
#include <iostream>
using namespace std;
...
ifstream in_file("my_in.txt");                   // ASCII-File als Eingabefile
in_file >> c1 >> c2;
```

- Siehe auch das ausführliche Beispiel Simple_FileIO[15].
- Wenn die Operatoren `<<` und `>>` für einen Datentyp definiert sind, so sind obige IO-Operation mit cin/cout oder via Files möglich. Anderfalls müssen diese beiden Operatoren für den neuen Datentyp definiert werden [Haase23, §9.8].

7. Einlesen eines Vektors vom ASCII-File: Das Beispiel file_io[16] (html[17]) demonstriert das Lesen/Schreiben eines double-Vektors via ASCII-Files.
   - Das kurze Hauptprogramm demonstriert die Anwendung der Leserfunktion (`main.cpp:16`) und der Schreibfunktion (`main.cpp:24`).
   - Die Lesefunktion `read_vector_from_file` (`file_io.cpp:30`) öffnet das File und bricht mit einer Fehlermeldung ab, falls das Inputfile nicht gefunden wird.
   - In obigem Erfolgsfalle wird der Vektor in der Funktion `fill_vector` solange mit Daten gefüllt (und dabei dynamisch verlängert, `file_io.cpp:14`) bis das Fileende erreicht ist. Zeilen `file_io.cpp:15-24` dienen nur der möglichen Fehlerbehandlung [Stroustrup10, p.364] und `file_io.cpp:25` verkürzt den Vektor auf die nötige Länge.
   - Die Schreibfunktion `write_vector_to_file` ist selbserklärend.

Literatur

[Haase23] Gundolf Haase: Einführung in die Programmierung mit C++ (2023), *www*[18].

[Stroustrup10] Bjarne Stroustrup: Einführung in die Programmierung mit C++. Pearson Studium, München (2010).

[13] https://de.wikipedia.org/wiki/Textdatei
[14] http://www.cplusplus.com/forum/general/21018/
[15] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/Simple_FileIO.zip
[16] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/file_io.zip
[17] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/file_io/html/file__io_8cpp.html
[18] http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf