

# Draft der C++-Vorlesung vom 1. April 2022

---

Zu Aufgabenblatt 3:

- Beispiel Geheimzahl mit Speicherung aller Versuche (dynamischer Vektor) und einem Vektor von Strings zur flexiblen Ausgabe.  
Code: `v_3a`<sup>1</sup> mit `docu`<sup>2</sup>.
  - Auch mit `array` [Haase22, §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<sup>3</sup>.
  - Benötigt die Compileroption `-std=c++11`
- 

## Pointer → Iterator aus §6.1

Motivation für Konzept der Iteratoren am Bsp. Vektor (`v_5c`):

Indezzugriff → Pointerzugriff → Iteratorzugriff → Range-For

- statisches C-Array in `main:17-18` und Loop mit Indexzugriff in `main:29-33`.
  - Pointer auf Arrayabfang (`main:20-21`) und Verbindung mit Indexzugriff in `main:32`.
  - Loop mit Pointer als Laufvariable in `main:45-48`.
  - Nun mit `vector` und Loop mit Indexzugriff in `main:59`.
  - Loop mit Iterator als Laufvariable in `main:64-68`.
  - Nun mit `list`: kein Loop mit Indexzugriff möglich (/sinnvoll).
  - Loop mit Iterator als Laufvariable in `main:88-91`.
  - Range-For-Loop in `main:96-99`.
- 

## Matrizen aus §5.1.4

Im ersten Teil beschäftigen wir uns mit **vollbesetzten** (dense) **Matrizen** in C++, siehe auch [Haase22, §5.1.4 (zu überarbeiten)] (`bsp514_b`).

Unsere Matrix  $A$  habe  $NROW = 4$  **Zeilen** und  $MCOL = 3$  **Spalten**

$$A_{NROW \times MCOL} := \begin{pmatrix} 4 & -1 & -0.5 \\ -1 & 4 & -1 \\ -0.5 & -1 & 4 \\ 3 & 0 & 1 \end{pmatrix}.$$

Wir demonstrieren für verschiedene Varianten der Speicherung die Matrixdeklaration und -definition, sowie den Zugriff auf das Element `A2,1` (Indizes fangen mit 0 an).

Falls die Matrixdimensionen als Konstanten zur Compilezeit bekannt sind, dann kann ein *statischer* Datentyp gewählt werden, andernfalls ist ein *dynamischer* Datentyp notwendig, siehe Beispiel `bsp514_b`<sup>4</sup>.

---

<sup>1</sup>[http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/v\\_3a.zip](http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/v_3a.zip)

<sup>2</sup>[http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/v\\_3a/html](http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/v_3a/html)

<sup>3</sup><https://en.cppreference.com/w/cpp/header/random>

<sup>4</sup>[http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/bsp514\\_b.cpp](http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/bsp514_b.cpp)

1. Static matrix via `array`, 2D-access, see also `bsp514_b.cpp:68`:  

```
int const NROW(4), MCOL(3); // constant dimensions
array<array<double,MCOL>,NROW> a{ 4,-1,-0.5, -1,4,-1, -0.5,-1,4,
3,0,-1 };
```

Each row consists of the fixed size array `array<double,MCOL>` with access to element  $A_{2,1}$  via `a[2][1]`.
2. Static matrix via `array`, 1D-access, see also `bsp514_b.cpp:90`:  

```
array<double,NROW * MCOL> a{ 4,-1,-0.5, -1,4,-1, -0.5,-1,4, 3,0,-1 };
```

The matrix elements are stored continuously stored in memory (ordered as in the initializer list above) and accessed as vector with special index calculation, e.g., we access element  $A_{2,1}$  as `a[2*MCOL+1]`.
3. Matrix as vector with static initialization and 2D-access, see also `bsp514_b.cpp:26`:  

```
vector<vector<double>> const a({{4,-1,-0.5}, {-1,4,-1}, {-0.5,-1,4},
{3,0,-1}});
```

Each row  $k$  consists of the vector `vector<double>` whose length depends on the appropriate number of elements in the initializer list, i.e., `a[k].size()` returns the number of columns of row  $k$ . Access to element  $A_{2,1}$  via `a[2][1]`.  
Without `const` in the declaration, the dimensions of `a` could be changed afterwards.
4. Matrix as vector with static initialization and 1D-access, see also `bsp514_b.cpp:48`:  

```
vector<double> const a{ 4,-1,-0.5, -1,4,-1, -0.5,-1,4, 3,0,-1 };
```

The matrix elements are stored continuously stored in memory (ordered as in the initializer list above) and accessed as vector with special index calculation, e.g., we access element  $A_{2,1}$  as `a[2*MCOL+1]`.  
Without `const` in the declaration, the dimensions of `a` could be changed afterwards.
5. Dynamic matrix dimensions with 2D access, see also `bsp514_b.cpp:114`:  

```
int nrow,mcol; cin >> nrow >> mcol;
vector<vector<double>> a(nrow, vector<double>(mcol) );
```

  - Each of the `nrow` elements in `vector<...>` represents a matrix row which is initialized by a `vector<double>` with `mcol` uninitialized double values.
  - `vector<double>(mcol,0.0)` initializes all matrix entries with value 0.
  - Warning: `vector<double>()` results in column size 0, see `bsp514_b.cpp:115`. In this case the column size has to be set in all rows afterwards, e.g.,  

```
a.[k].resize(mcol);
```
Access to element  $A_{2,1}$  via `a[2][1]`.
6. Dynamic matrix dimensions with 1D access:  

```
int nrow,mcol; cin >> nrow >> mcol;
vector<double> a(nrow*mcol);
a[2*mcol+1]
```

---

## Bsp. Graph

Ein Graph wie der angegebene<sup>5</sup>, unvollständige Graph, besteht aus **Knoten** (knots/vertices) welche über **Kanten** (edges) verbunden sind.

Im Beispielcode `graph`<sup>6</sup> wird die Kantenbeschreibung obigen Graphs eingelesen und daraus für jeden Knoten dessen sämtliche Knotennachbarn ermittelt.

1. Jede Kante besitzt stets genau **2 Knoten**, jedoch ist die Anzahl der Kanten **unbekannt**. Somit ist z.B. folgende Datenstruktur zur Speicherung der Kantenbeschreibung sinnvoll:

```
vector<array<int,2>> edges; // main.cpp:15
```

D.h., jede Kante speichert die Indizes der sie definierenden Knoten.

2. Das Einlesen der Kantenbeschreibung erfolgt über die Funktion `read_edges_from_file(name, edges);` // main.cpp:16 welche vom Vorlesungsbeispiel `file_io`<sup>7</sup> übernommen und adaptiert wurde. Die einzigen Änderungen zum Original sind `graph.cpp:18` und die Signatur der Funktion.

3. Die Knoten in obigem Graphen besitzen eine **unterschiedliche** Anzahl von **Nachbarknoten** welche mittels

```
auto n2n=get_node2nodes(edges); // main.cpp:30
```

aus der Kantenbeschreibung bestimmt werden. Das Schlüsselwort **auto** legt für die Variable `n2n` denselben Datentyp wie den Rückgabotyp der Funktion fest.

4. Bislang wissen wir noch nicht wieviele Knoten unser Graph besitzt. Unter Annahme einer fortlaufenden Numerierung wird diese **Knotenanzahl** in `graph.cpp:43-51` bestimmt.

5. Damit können wir eine Datenstruktur festlegen, welche für jeden **Knoten** dessen noch zu bestimmende **Nachbarnknotenindizes** jeweils in einem dynamisch wachsenden Vektor speichern wird. Mit

```
vector<vector<int>> n2n(nnode); //n2n(nnode,vector<int>()) // graph.cpp:54
```

wird die **Anzahl der Nachbarnknoten** stets mit **0** initialisiert.

6. Der Algorithmus in `graph.cpp:54-61` liest aus der Kantenbeschreibung für jede **Kante k** die Indizes ihrer beiden **Knoten** und hängt diese wechselseitig **beim anderen** Knoten als Nachbarnknoten an, die **Vektoren der Nachbarn** wachsen also dynamisch.

```
{
    const int v0 = edges[k][0];
    const int v1 = edges[k][1];
    n2n[v0].push_back(v1); // add v1 to neighborhood of v0
    n2n[v1].push_back(v0); // and vice versa
}
```

Die aktuelle Anzahl der Nachbarn für Knoten `k` ist hierbei `n2n[k].size()`.

7. Abschließend werden die ermittelten Indizes der Nachbarnknoten in `graph.cpp:65` für jeden Knoten aufsteigend sortiert [Haase22, §11.3.2].

---

<sup>5</sup>[http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/graph\\_1.pdf](http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/graph_1.pdf)

<sup>6</sup><http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/graph.zip>

<sup>7</sup>[http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/file\\_io.zip](http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS22/file_io.zip)

---

Fun with ASCII<sup>8</sup>: *char\_sum.cpp*

- How to achieve 103% performance?
- { HARDWORK, KNOWLEDGE, ATTITUDE, BULLSHIT };
- `magic=96`  $\rightarrow$  `magic=0` : BILLGATES (the third!)

## Literatur

[Haase22] Gundolf Haase: Einführung in die Programmierung mit C++ (2022), *www*<sup>9</sup>.

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

---

<sup>8</sup><http://www.torsten-horn.de/techdocs/ascii.htm>

<sup>9</sup>[http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script\\_programmieren.pdf](http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf)