

Rechtlicher Hinweis

Diese Präsentation ist urheberrechtlich geschützt und darf nur im Rahmen von Lehrveranstaltungen der Friedrich-Schiller-Universität Jena verwendet werden. Eine Nutzung durch Verbreitung oder Veröffentlichung dieses Materials - auch in Auszügen - ist strengstens untersagt und wird die Geltendmachung von Unterlassungs- und Schadenersatzansprüchen durch die Friedrich-Schiller-Universität Jena zur Folge haben.

Legal notice

These slides are protected by copyright and may only be used as part of courses at the Friedrich Schiller University Jena. Any use through the dissemination or publication of this material - even in extracts - is strictly prohibited and will result in the assertion of injunctive relief and claims for compensation by the Friedrich Schiller University Jena.

Informatik I (B.Sc. Physik)

Arrays / Felder in C++

Dr. Paul Bodesheim

(Paul.Bodesheim@uni-jena.de)



**FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA**

**Fakultät für Mathematik und Informatik
Lehrstuhl für Digitale Bildverarbeitung**

SoSe 2020

1 Motivation

2 Vektoren in C++

- Vektoren anlegen und verändern
- Vektoren und Funktionen

3 Mehrdimensionale Felder

Inhalt

1 Motivation

2 Vektoren in C++

- Vektoren anlegen und verändern
- Vektoren und Funktionen

3 Mehrdimensionale Felder

Komplexe Datenstrukturen

- Mehrere Elemente des gleichen (Basis-) Typs handhaben
- Beispiel: mehrere Messwerte in einer Zeitreihe (Temperaturen)
- Begriffe für komplexe Datenstrukturen in der Informatik: Feld, Reihung, Array
- Elemente lassen sich über Index (ganzzahligen Wert) "ansprechen"
- Arrays in C++: Vektoren (vector)
- Es gibt noch weitere Möglichkeiten, Felder in C++ zu verwenden
 - 处理相同（基本）类型的多个元素
 - 示例：一个时间序列（温度）中的多个测量值
 - 计算机科学中复杂数据结构的术语：字段，序列，数组
 - 元素可以通过索引（整数值）来寻址
 - C ++中的数组：向量（向量）
 - 在C ++中还有其他使用字段的方法

Inhalt

1 Motivation

2 Vektoren in C++

- Vektoren anlegen und verändern
- Vektoren und Funktionen

3 Mehrdimensionale Felder

标准模板库 (STL) 中的 (Template) 类 向量
 向量类型的向量可以在不了解模板和类的情况下使用
 在标题中包含适当的库: #include <vector>
 标准名称空间也有帮助: 使用名称空间 std;
 复杂数据类型: 向量
 尖括号中字段元素的 (基本) 数据类型, 例如:
 向量<int>, 向量<double>, 向量<bool>, ...
 载体的声明:

Was sind Vektoren in C++?

- (Template-) Klasse vector aus der Standard-Template-Library (STL)
- Ohne Wissen über Templates und Klassen können Vektoren vom Typ vector verwendet werden
- Entsprechende Bibliothek in Header einbinden: **#include** <vector>
- Standard-Namensraum ist ebenso hilfreich: **using namespace** std;
- **Komplexer Datentyp**: vector
- (Basis-) Datentyp der Elemente des Feldes in Winkelklammern, z.B.:
vector<**int**>, vector<**double**>, vector<**bool**>, ...
- Deklaration von Vektoren:

```
vector<int> v1; // Vektor der Laenge 0
vector<int> v2(10); // Vektor der Laenge 10
```

Vektoren verändern

向量的大小可以在运行时更改

向量是动态场

“回推”功能，用于添加元素：v1.push_back(1);

通过以下方式调用函数：

<变量名称>.<功能名称>(<参数列表>)

(来自面向对象的编程)

变量作为使用功能的对象

函数 pop_back 删除最后一个元素：v1.pop_back();

使用方括号和索引进行元素访问（写和读）：

v2[1] = 5; //写入权限

int i = v2[1]; //读取权限

介绍：索引为门牌号码

- Die Größe eines Vektors kann zur Laufzeit verändert werden
- Vektoren sind **dynamische** Felder
- Funktion "push_back" zum **Anhängen von Elementen**: v1.push_back(1);
- Funktionsaufruf über:
<VARIABLENNAME>.<FUNKTIONSNAME>(<PARAMETERLISTE>)
(kommt von der objektorientierten Programmierung)
- Variablen als Objekte, für die eine Funktion angewendet wird
- Funktion "pop_back" zum **Entfernen des letzten Elements**: v1.pop_back();
- Elementzugriff** (schreibend und lesend) über eckige Klammern und Index:
v2[1] = 5; // schreibender Zugriff
int i = v2[1]; // lesender Zugriff
- Vorstellung: Index als Hausnummer

Indizierung von Vektoren

元素访问的索引值不得超过向量的大小

第一个元素的索引为0

大小为n的向量的最后一个有效索引是什么？ n-1

程序员负责正确的索引编制

如果系统每次访问都检查索引正确，那将太耗时

超出索引范围时的不确定行为

编译器会翻译程序，但会在运行时损坏！

不会删除或覆盖整个硬盘（这由操作系统确保），但是存在数据操纵的风险

使用函数size或length的向量的元素数（大小，长度，尺寸）：

- Index-Wert bei Elementzugriff darf Größe des Vektors nicht überschreiten
- Erstes Element bei Index 0
- Was ist der letzte gültige Index für einen Vektor der Größe n? n-1
- Programmierer ist für korrekte Indizierung verantwortlich
- Es wäre zu aufwendig, wenn das System bei jedem Zugriff korrekte Indizierung prüfen würde
- undefiniertes Verhalten, wenn Index-Bereich überschritten wird
- Compiler übersetzt das Programm, aber Schaden zur Laufzeit!
- Kein Löschen oder Überschreiben der gesamten Festplatte (dafür sorgt Betriebssystem), aber Gefahr der Manipulation von Daten
- Anzahl der Elemente (Größe, Länge, Dimension) eines Vektors über die Funktionen size oder length:

```
int s = v1.size();
```

```
v1.size() == v1.length() // true
```

Vektoren mit Elementen füllen

要么：\清空“ 向量，然后向后推
或者：使用访问运算符[]初始化向量的长度并设置值

- Entweder: “leerer” Vektor und `push_back`
- Oder: Vektor mit Länge initialisieren und Werte über Zugriffsoperator `[]` setzen

```
#include <vector>
#include <iostream>
using namespace std;

int main()
{
    vector<int> v1;
    vector<int> v2(10);

    for (int i=0; i<v2.size(); i++)
    {
        v1.push_back(i);
        cout << v1.size() << endl;
        v2[i] = 2*i;
    }
    ... // weitere Anweisungen
    return 0;
}
```

- Was gibt das Programm aus?

Anlegen von Vektoren

- Vektoren sind Objekte, die über Konstruktoren angelegt werden

- Standard-Konstruktor:

```
vector<int> v1; // Laenge 0
```

- Konstruktor mit vorgegebener Länge:

```
vector<int> v2(10); // Laenge 10
```

- Konstruktor mit vorgegebener Länge und Füllwert:

```
vector<int> v3(5,1); // Laenge 5, alle Werte gleich 1
```

- Kopier-Konstruktor:

```
vector<int> v4(v3); // Laenge 5, alle Werte gleich 1
```

- Zuweisung:

```
vector<int> v5 = v4; // Laenge 5, alle Werte gleich 1
```

- Ab Standard C++11 lassen sich initiale Werte angeben:

```
vector<int> v6{2,4,6,8,1,3,5,7,9}; // Laenge 9
```

(Compiler-Flag `-std=c++11` nicht vergessen!)

Vektor ist ein Objekt, das über Konstruktoren erstellt wird

Standard-Konstruktor:

```
vector<int> v1; // Länge 0
```

Konstruktor mit vorgegebener Länge:

```
vector<int> v2(10); // Länge 10
```

Konstruktor mit vorgegebener Länge und Füllwert:

```
vector<int> v3(5,1); // Länge 5, alle Werte gleich 1
```

Kopier-Konstruktor:

```
vector<int> v4(v3); // Länge 5, alle Werte gleich 1
```

Zuweisung:

```
vector<int> v5 = v4; // Länge 5, alle Werte gleich 1
```

Ab C++11 lassen sich initiale Werte angeben:

```
vector<int> v6{2,4,6,8,1,3,5,7,9}; // Länge 9
```

(Compiler-Flag `-std=c++11` nicht vergessen!)

Beispiele

- Typische **for**-Schleife zum Durchlaufen aller Elemente eines Vektors `v`:
for (**int** `i=0`; `i<v.size()`; `i++`) ...
- Alternativ in die andere Richtung durchlaufen (von hinten nach vorne):
for (**int** `i=v.size()-1`; `i>=0`; `i--`) ...

```
#include <iostream>
#include <vector>

using namespace std;

void printVector(const vector<int> & v)
{
    for (int i=0; i<v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
}
```

Beispiele (2)

```
int main()
{
    vector<int> v1;
    vector<int> v2(5,2);
    vector<int> v3(5,0);

    cout << "Laengen: ";
    cout << v1.size() << " ";
    cout << v2.size() << " ";
    cout << v3.size() << endl;

    cout << endl << "Vektor 2:" << endl;
    printVector(v2);

    for (int i=0; i<v3.size(); i++)
    {
        v1.push_back(1);
        v2.pop_back();
        v3[i] = 3;

        cout << endl << "i=" << i << ":" << endl;
        printVector(v1);
        printVector(v2);
        printVector(v3);
    }

    return 0;
}
```

Ausgaben des Programms:

Laengen: 0 5 5

Vektor 2:
2 2 2 2 2

i=0:
1
2 2 2 2
3 0 0 0 0

i=1:
1 1
2 2 2
3 3 0 0 0

i=2:
1 1 1
2 2
3 3 3 0 0

i=3:
1 1 1 1
2
3 3 3 3 0

i=4:
1 1 1 1 1
3 3 3 3 3

Inhalt

1 Motivation

2 Vektoren in C++

- Vektoren anlegen und verändern
- Vektoren und Funktionen

3 Mehrdimensionale Felder

Größe von Vektoren verändern

- Funktion `resize`

```
vector<int> v1(5,1); // 1 1 1 1 1
```

```
// Kleinere Anzahl:
```

```
// erste Werte uebernehmen, die restlichen verwerfen
```

```
v1.resize(3); // 1 1 1
```

```
// Groessere Anzahl:
```

```
// erste Werte uebernehmen, zusaetzliche Werte undefiniert
```

```
v1.resize(6); // 1 1 1 0(?) 0(?) 0(?)
```

```
v1.resize(3); // 1 1 1
```

```
// Besser:
```

```
// groessere Anzahl und Fuellwert fuer zusaetzliche Stellen
```

```
v1.resize(6,4); // 1 1 1 4 4 4
```

Größe von Vektoren verändern (2)

- Funktion `assign` (neue Größe und Füllwert für alle Elemente):

```
vector<int> v2(5,1); // 1 1 1 1 1
```

```
v2.assign(10,2); // 2 2 2 2 2 2 2 2 2 2
```

- Funktion `clear` (löscht alle Werte, liefert leeren Vektor)

```
vector<int> v3(5,3); // 3 3 3 3 3
```

```
cout << v3.size() << endl; // 5
```

```
v3.clear(); // keine Funktions-Parameter!
```

```
cout << v3.size() << endl; // 0
```

```
// Danach:
```

```
// Elemente mit push_back hinzufügen
```

```
// oder resize bzw. assign verwenden
```


Vektoren als Parameter für Funktionen

向量类型的字段可以像其他数据类型的变量一样使用
可以使用函数的参数或作为返回值

- Felder vom Typ `vector` können wie Variablen von anderen Datentypen als Parameter von Funktionen oder als Rückgabewerte verwendet werden

```
void set(vector<int> & v, int value = 0)
{
    for (int i=0; i<v.size(); i++)
        v[i] = value;
}
```

```
int main()
{
    vector<int> v1(3);
    vector<int> v2(5);

    set(v1,1); // 1 1 1
    set(v2); // 0 0 0 0 0

    ...
}
```

Beispiele

- Maximaler Wert in einem Vektor:

```
double get_maximum_value(const vector<double> & v)
{
    double maxValue = v[0];

    for (int i=1; i<v.size(); i++)
        if (v[i]>maxValue)
            maxValue = v[i];

    return maxValue;
}
```

- Alle geraden Zahlen finden:

```
vector<int> get_even_numbers(const vector<int> & v)
{
    vector<int> evenNumbers; // leerer Vektor

    for (int i=0; i<v.size(); i++)
        if (v[i] % 2 == 0)
            evenNumbers.push_back(v[i]);

    return evenNumbers;
}
```

Beispiele (2)

- Effizientere Variante für zweites Beispiel:

```
void get_even_numbers2(const vector<int> & v, vector<int> & evenNumbers)
{
    evenNumbers.clear(); // leerer Vektor sicherstellen

    for (int i=0; i<v.size(); i++)
        if (v[i] % 2 == 0)
            evenNumbers.push_back(v[i]);
}

int main()
{
    // nach C++11-Standard
    vector<int> inputV{4,7,9,12,3,1,8,17,16};
    vector<int> outputV;

    //Variante 1: ineffizient
    outputV = get_even_numbers(inputV);

    //Variante 2: effizienter
    get_even_numbers2(inputV, outputV);

    ... // weitere Anweisungen

    return 0;
}
```

Range-based for-loop

- Seit Standard C++11 (Compiler-Flag: `-std=c++11`)
- Nützlich, wenn man alle Elemente eines Vektors "ansprechen" will

```
vector<double> v{1.0, 2.0, 3.0, 4.0, 5.0};  
  
for (double d : v)  
    cout << d << " "; // Gibt alle Elemente aus
```

- Variable `d` vom Typ **double** wie bei Funktionsparametern
- Elemente des Vektors werden sequentiell in die lokale Variable `d` kopiert
- Zur Modifikation der Elemente sind Referenzen notwendig (wie bei Funktionsparametern):

```
vector<double> v{1.0, 2.0, 3.0, 4.0, 5.0};  
  
for (double & d : v)  
    d=2*d; // Verdoppelt alle Elemente
```

Inhalt

1 Motivation

2 Vektoren in C++

- Vektoren anlegen und verändern
- Vektoren und Funktionen

3 Mehrdimensionale Felder

Zweidimensionale Vektoren (Matrix)

矩阵的每一行都是一个向量
 向量的向量可服务于 代表矩阵
 向量<double>>矩阵(3, 向量<double>(3));
 长度为3的带有初始值的外部向量, 该向量在每个点上都具有一个
 长度为3的向量的类型为double
 元素访问: 矩阵[1][2] = 4.0; //第二行, 第三列
 行的第一个索引(外部向量), 列的第二个索引(内部向量)
 矩形矩阵当然也是可能的:

- Jede Zeile einer Matrix ist ein Vektor
 整数 rows = 4;
 整数 cols = 6;
 向量<double>>矩阵(行 向量<double>(cols))
- Vektoren von Vektoren dienen zur Repräsentation einer Matrix
- `vector< vector<double> > matrix(3,vector<double>(3));`
- Äußerer Vektor der Länge 3 mit initialem Wert, der (an jeder Stelle) auch ein Vektor der Länge 3 vom Typ **double** ist
- Elementzugriff: `matrix[1][2] = 4.0;` // 2. Zeile, 3. Spalte
- Erster Index für Zeile (äußerer Vektor), Zweiter Index für Spalte (innerer Vektor)
- Rechteckige Matrizen natürlich auch möglich:


```

int rows = 4;
int cols = 6;
vector< vector<double> > matrix (rows, vector<double>(cols));
      
```

Beispiele

```
vector< vector<int> > matrix(4, vector<int>(2));  
  
for (int r=0; r<matrix.size(); r++)  
    for (int c=0; c<matrix[r].size(); c++)  
        matrix[r][c] = 2*r+c;
```

- Vektoren von Vektoren müssen keine Matrizen sein:

```
vector< vector<int> > m(3);  
m[0].assign(4,1);  
m[1].assign(3,2);  
m[2].assign(7,3);
```

- Das kann auch bei der Arbeit mit Matrizen passieren:

```
matrix[1].resize(5,0); // ebenso: matrix[1].assign(5,0);
```

Mehrdimensionale Vektoren (Tensor)

- Weitere (tiefere) Verschachtelungen von Vektoren möglich
- Dreidimensionale Tensoren (z.B. Farbbilder):

```
int zeilen = 480;  
int spalten = 640;  
int farben = 3;
```

```
vector< vector< vector<int> > > farbbild(farben);  
vector< vector<int> > farbkanal(zeilen, vector<int>(spalten));  
farbbild[0] = farbkanal;  
farbbild[1] = farbkanal;  
farbbild[2] = farbkanal;
```

- Vierdimensionale Tensoren (z.B. Videos):
int einzelbilder = 1800;

Gibt es Fragen?

(Es gibt keine dummen Fragen!)

