FH Salzburg
Informationstechnik &
System-Management

Objektorientierte Programmierung LB

* **DI (FH) DI Roland J. Graf MSc**
  roland.graf@fh-Salzburg.ac.at
* **Maximilian E. Tschuchnig BSc**
  maximilian.tschuchnig@fh-salzburg.ac.at

LB01

Technik
Gesundheit
Medien

Themen LB01

- char, auto, string

- arrays vs. vectors
- new & delete, heap vs. stack
- return values, scope
- reference vs. pointer

- cin, cout & cerr
- iostream

Unicode-Typen

Codierung	Datentyp	String-Literal
UTF-8	char	u8”UTF-8 kodierter Text”
UTF-16	char16_t	u”UTF-16 kodierter Text”
UTF-32	char32_t	U”UTF-16 kodierter Text”


```
// 01-01: variable initialization
#include "stdafx.h"

int main() {
    int ivar = 10;
    float fvar { 123.456f };
    char c8var { 'A' };
    char16_t c16val{ u'Ω' }; // (mind.) 16 Bit breites Zeichen
    char32_t c32val{ U'Ω' }; // (mind.) 32 Bit breites Zeichen
    // do something
    return 0;
}
```

Initialisierung mit {…} im C++ 11 Stil

string literal

Siehe C++ Reference:

- Fundamental types
- String literal
- Numeric limits

auto specifier (seit C++11)

- Compiler(!) wählt automatisch die passende Typisierung
- dies ist also keine dynamische Typisierung, sondern eine Typinferenz

```
// 01-02: auto specifier
#include "pch.h"

int main() {
    auto aval1 = 1234;     // = int
    auto aval2 = 1234L;    // = long
    auto aval3 = 12.34;    // = double
    auto aval4 = 12.34F;   // = float
    auto aval5 = 'A';      // = char
    auto aval6 = u'A';     // = char16_t
    // do something
    return 0;
}
```

Precompiled header in Visual Studio C++

auto specifier

Ausdruck (rvalue) bestimmt Typ der Variablen

Siehe C++ Reference:
- auto specifier
- Typinferenz

Achtung:
Das Schlüsselwort *auto* ist nicht abwärtkompatibel, da es vor C++11 zur Beschreibung der Speicherklasse verwendet werden konnte.

auto function (seit C++ 11)

- Der return type einer Funktion bestimmt den Typ der Funktion.

```
#define _USE_MATH_DEFINES  // for C++
#include <cmath>

auto foo(double d, int i)
{
    return i / d;
}

int main() {
    auto aval1 = std::tan( M_PI_2 );
    auto aval2 = foo(3.1415926f, 2);
    // do something
    return 0;
}
```

auto specifier

Ausdruck bzw. Returntyp bestimmt den Typ der Funktion

Siehe C++ Reference:
- auto specifier

auto function -> return type (C++ 14)

- auto-Funktionen können Typ auch erzwingen

```
#define _USE_MATH_DEFINES  // for C++
#include <cmath>

auto foo2(float f, int i) -> double
{
    return i / f;
}

int main() {
    auto aval1 = std::tan( M_PI_2 );
    auto aval2 = foo2(3.1415926f, 2);
    // do something
    return 0;
}
```

Angabe des Typs der Funktion

Type der Funktionen bestimmen Typ der auto Variablen

Siehe C++ Reference:
- auto specifier

String

- String erlaubt komfortablen Umgang mit Zeichenketten

```
#include <string>

int main() {
    std::string str1 { "Text 1" };
    std::string str2 = str1;
    std::string str3 { };

    auto len = str2.length();
    str2 = "Text 2";
    str2.append("!");

    return 0;
}
```

Deklaration über Headerdatei <string>

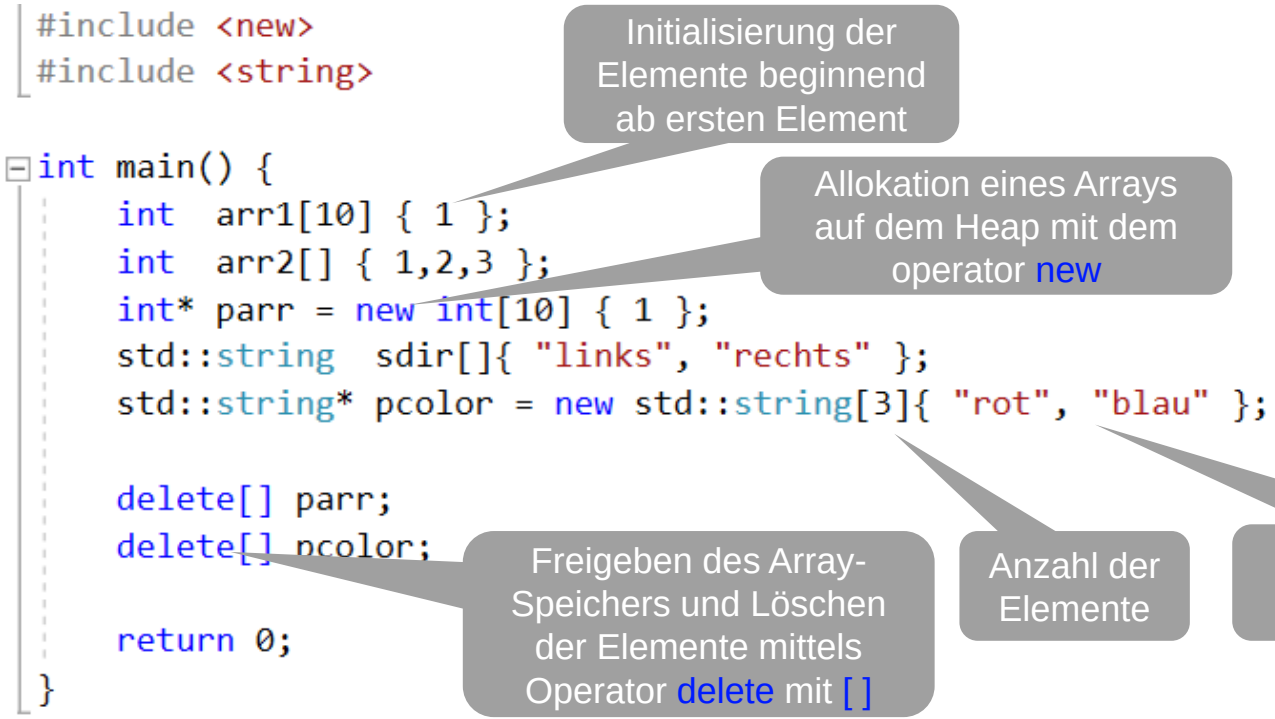
Initialisierung über Zuweisungen möglich

Aufruf einfacher Methoden über object.methodname(…)

Siehe C++ Reference:
- String

String Typ	Codierung
string	UTF-8
u16string	UTF-16
u32string	UTF-32
u.e.m	

C-style Arrays in C++



Siehe C++ Reference:

- Array declaration
- Operator new
- Operator delete
- Arrays – Tutorial
- Dynamic memory

std::array – Statisches Array


```
#include <string>
#include <array>

std::array<int, 10> garr0{};

int main() {
    std::array<int, 10> arr0;
    std::array<int, 10> arr1{};
    std::array<int, 10> arr2{ 1,2,3 };
    std::array <std::string, 2> sdir{ "links", "rechts" };
    std::array<int, 10>* parr = new std::array<int, 10>{ 1, 2, 3 };
    int size = arr0.size();
    int sof = sizeof(arr0);
    // do other stuff
    delete parr;
    return 0;
}
```

Typ und Anzahl der Elemente

Initialisierung der Elemente beginnend ab dem Element [0]

Array auf Heap
Vorsicht beim Zugriff auf die Elemente!

Freigeben des std::array-Speichers

Siehe C++ Reference:

- std::array
- Speichereffizienz von std::array

std::vector – Dynamisches Array (=Vektor)


```
#include <string>
#include <vector>

int main() {
    std::vector<int>  arr0(100);
    std::vector<int>  arr1(10, 1);
    std::vector<int>  arr2{ 1,2,3 };
    std::vector <std::string>  sdir{ "links", "rechts" };
    std::vector<int>* parr = new std::vector<int>{ 0,1,2,3,4,5,6,7,8,9 };
    // do something
    delete parr;
    return 0;
}
```

Typ

Anzahl der Elemente

Initialisierungs-wert

Initialisierung der Elemente beginnend ab ersten Element

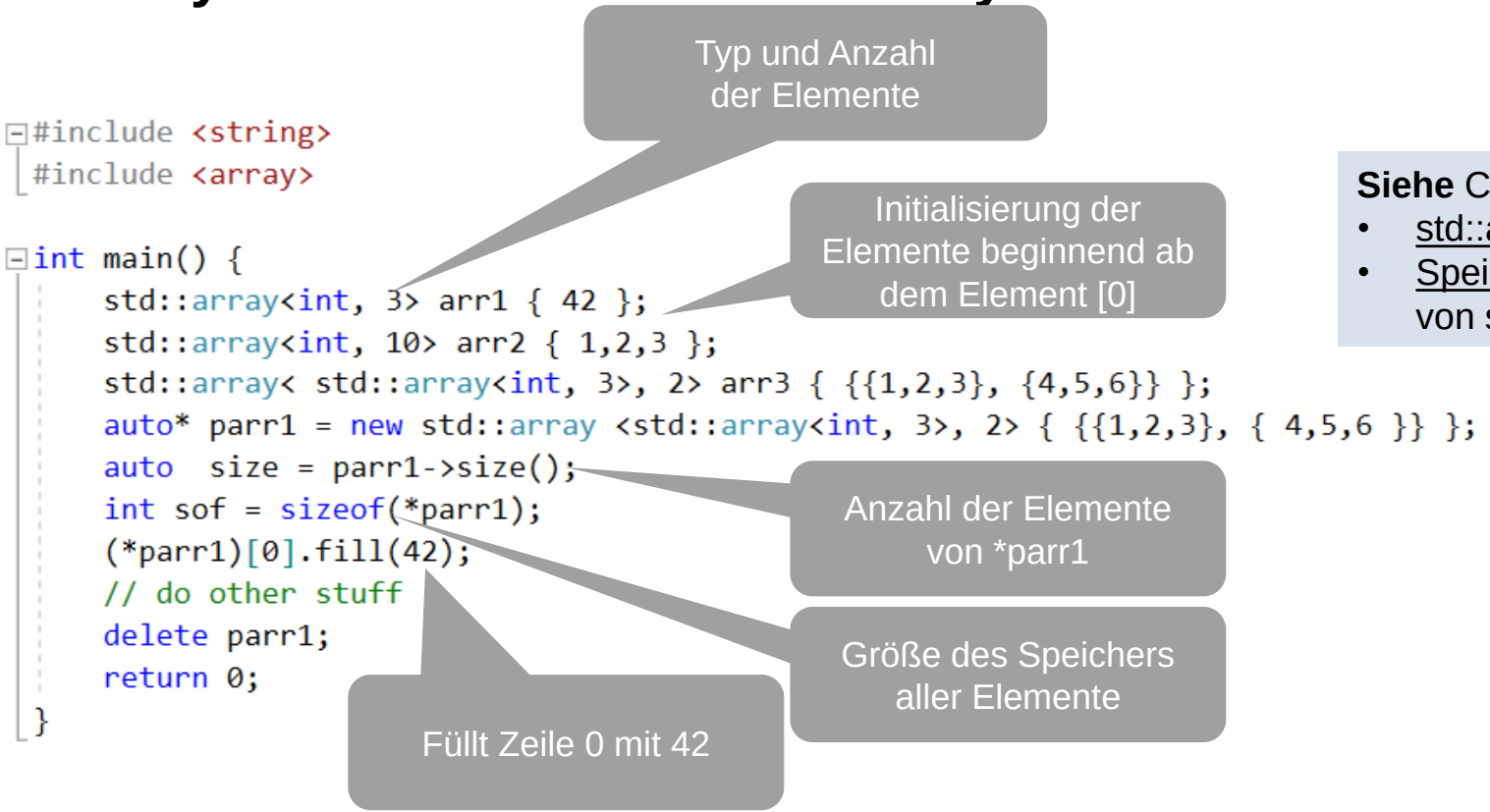
Freigeben des Vektors und Löschen aller Elemente

Nicht alles was möglich ist, ist auch immer sinnvoll ☺

Siehe C++ Reference:

- std::vector
- Automat. Speicher-management mit Containern

std::array – Mehrdim. statisches Array




```cpp
#include <string>
#include <array>

int main() {
    std::array<int, 3> arr1 { 42 };
    std::array<int, 10> arr2 { 1,2,3 };
    std::array< std::array<int, 3>, 2> arr3 { {{1,2,3}, {4,5,6}} };
    auto* parr1 = new std::array <std::array<int, 3>, 2> { {{1,2,3}, { 4,5,6 }} };
    auto  size = parr1->size();
    int sof = sizeof(*parr1);
    (*parr1)[0].fill(42);
    // do other stuff
    delete parr1;
    return 0;
}
```

Siehe C++ Reference:

- std::array
- Speichereffizienz von std::array

std::vector – Mehrdim. Vektor


```cpp
#include <string>
#include <vector>

int main() {
    std::vector<int>  arr0(100, 42);
    std::vector< std::vector<int> > arr1(10, std::vector<int>(3, 99));
    std::vector< std::vector<int> > arr2{ { 1,2,3 },{4,5,6} };
    arr2[1][1] = arr0[64];
    int myints[] = { 55,66,77 };
    arr2[0].assign( myints, myints + _countof(myints));
    arr2[1].assign( { 88,89,90 } );
    arr1[0].assign( arr0.begin(), arr0.begin()+arr1[0].size() );

    return 0;
}
```

10x3-dim. int-Vector initialisiert mit 99

Siehe C++ Reference:

- std::vector
- vector::assign

Beispiele für die Zuweisung neuer Werte

Reference vs. Pointer

- Referenz ist Synonym für eine Variable (technisch realisiert wie ein Zeiger)
- Referenz muss bei Deklaration initialisiert bzw. mit einem Objekt gebunden werden und ist danach nicht mehr änderbar

```
int main()
{
    int  var = 1000;
    int& rvar = var;          // Referenz auf var
    const int& rvar2 = rvar;  // Kopie der Referenz rvar

    var = 200;          // Wert in var auf 200 ändern
    rvar = 999;         // ändert über Referent Wert in var auf 999
    rvar2 = 1800;       // Fehler: Zuweisung nicht möglich da const
}
```

Siehe

- Reference declaration
- Reference initialization
- Why does C++ pointers & references?
- Call-by-value or call-by-reference?

cin, cout & cerr

- cin, cout, cerr und clog => Standard IO Streams

```
#include <iostream>

int main(int argc, const char* argv[] )
{
    double d;
    std::cout << "Geben Sie einen Wert ein: ";
    std::cin >> d;
    if (std::cin.fail()) {
        std::cerr << "Eingabe fehlgeschlagen. Programmabbruch!\n";
        return -1;
    }
    std::cout << "Der Wert den Sie eingegeben haben ist " << d << "\n"
              << "Das Dreifache davon ist " << d*3 << "." << std::endl;
    return 0;
}
```

Siehe

- Basic Input/Output
- IO Manipulatoren (Formatierung)

ofstream, ifstream & fstream

Funktion, die Textdatei mit ifstream zeilenweise liest und den Text in einem Vektor zurückgibt

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

std::vector<std::string> readFile(std::string fpath) {
    std::vector<std::string> lines;
    std::ifstream myfile(fpath);            // Öffnet Datei
    if (myfile.is_open()) {                 // Prüft ob Öffnen ok
        std::string line;
        while (std::getline(myfile, line))  // Zeilenweises Lesen
            lines.push_back(line);          // neue Zeile hinzufügen
        myfile.close();                     // Datei schließen
    }
    return lines;                           // Zeilen in Vektor zurück
}
```

Siehe

- Input/output with files
- Input/output library