

Übung03: Informationstechnik (IT)

Harald Bucher

Institutsleitung
Prof. Dr.-Ing. Dr. h.c. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Institut für Technik der Informationsverarbeitung (ITIV)



Teil 1: Besprechung der Übungsaufgaben 3.01-3.06

KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

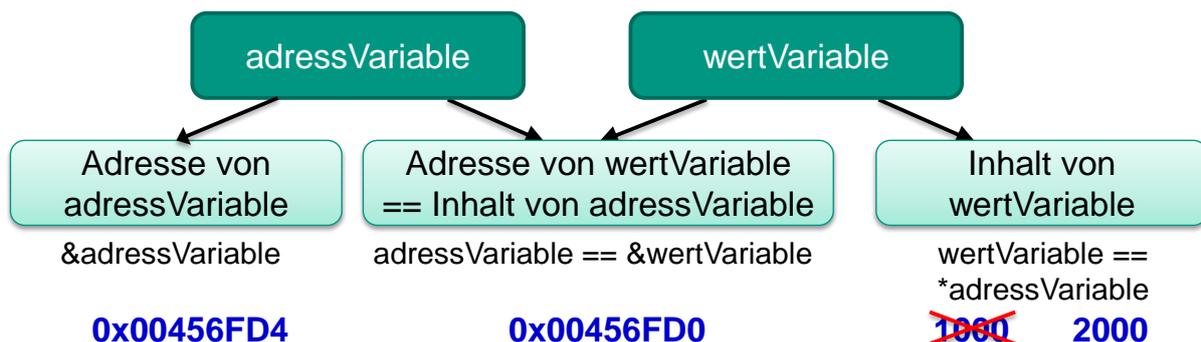
www.kit.edu

Adressvariablen: Zugriff

- Um die Variable zu erhalten, auf welche ein Zeiger verweist, verwendet man den ***** Operator (Dereferenzierung)
- *** vor Adressvariable, wird zum Inhalt, welcher an dieser Adresse gespeichert ist

Beispiel:

```
int wertVariable = 1000;
int* adressVariable = &wertVariable;
*adressVariable = *adressVariable * 2;
```

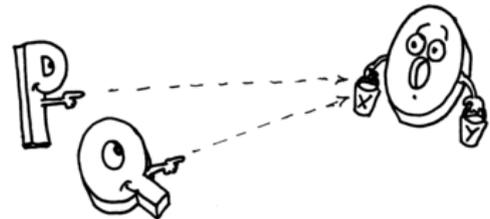


Referenzen

- Problemstellung:
 - Wie kann man auf eine Variable mit mehreren Namen zugreifen?
- Lösung: Referenzen
 - Kann direkt wie die Originalvariable verwendet werden
 - Bei Funktionen sehr nützlich
- Deklaration mit **&** „Ampersand“
- Beispiel: `int wert = 10.7;`
`int& verweis = wert;`



- Keine Variable → belegt keinen Speicher
 - Muss initialisiert werden
 - Später nicht mehr veränderbar



Arrays und Zeiger

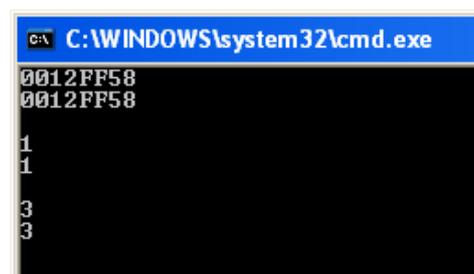
- Arrayname = Zeiger auf das erste Arrayelement
- Beispiel: `int arr[3] = { 1, 2, 3 };`
`int* arrayZeiger = arr; // = &arr[0];`
- `arr` und `arrayZeiger` sind Integer-Zeiger auf `arr[0]`, an dieser Stelle ist der Wert 1 gespeichert

- Beispiel:

```
cout << arrayZeiger << endl;
cout << arr << endl << endl;

cout << *arrayZeiger << endl;
cout << *arr << endl << endl;

cout << *(arrayZeiger + 2) << endl;
cout << *(arr + 2) << endl << endl;
```



```
C:\WINDOWS\system32\cmd.exe
0012FF58
0012FF58
1
1
3
3
```

Arrays und Zeiger – Rechnen mit Zeigern

- Zugriff auf das Array auch über Zeiger möglich (schreiben / lesen)
 - `arr[2]` ist gleichbedeutend mit `*(arr + 2)`
 - Es werden auch hier **keine** Grenzen überprüft
 - Größe der Rechenschritte im Speicher wird entsprechend dem Typ des Zeigers automatisch angepasst

■ Beispiel: `int arr[3] = { 1, 2, 3 };`

`int* arrayZeiger = arr;`

`arrayZeiger = arrayZeiger + 2;`

Variable	Inhalt der Variablen	Adresse (hex)
<code>arr[0]</code>	1	0x0012FF50
<code>arr[1]</code>	2	0x0012FF54
<code>arr[2]</code>	3	0x0012FF58
<code>arrayZeiger</code>	0x0012FF50 0x0012FF58	0x0012FF40

Zeigerarithmetik und Read-Only Zeiger

- Arithmetische Operationen und Vergleiche sind mit Zeigern möglich
 - Operatoren `++`, `--`, `+=`, `-=`, ...
 - Auf Operatorenreihenfolge achten (oder Klammern setzen)
- Subtraktion von Zeigern zur Bestimmung des Index in Arrays
 - Addition von Zeigern nicht zulässig, da auch nicht sinnvoll

■ Beispiel:

`int index = arrayzeiger - arr;`

index = 2

Header-Dateien und Linking

- Aufteilung des Codes in übersichtliche Module
 - Aufteilung in mehrere Quellcode-Dateien

- Zugriff über Prototypen in Header-Dateien
 - Compiler übersetzt einzelne Dateien – Linker bindet sie zusammen
 - Header-Dateien werden in die anderen Quelldateien inkludiert

- Schutz vor mehrfachen Einbinden der Dateien
 - Verwendung von Präprozessor-Direktiven

```

#ifndef _MYHEADER_
#define _MYHEADER_

[...]
```

#endif

Wird vom Compiler / Präprozessor verarbeitet

Inhalt der Header-Datei

Funktionen und Header-Dateien

```

// Deklaration
// von cin, cout,
// ...
```

Header-Datei
iostream

Kopie

Quelldatei
main.cpp

```

#include <iostream>
#include "myheader.h"

using namespace std;

int main() {
    int a;
    ...
    cin >> a;
    cout << myfunc( a );
    ...
    return 0;
}
```

```

// Deklaration
// eigener
// Funktionen
// und Klassen
long myfunc( int );
```

Header-Datei
myheader.h

Kopie

Quelldatei
myheader.cpp

```

// Definition
// eigener Funktionen
// und Klassen
long myfunc( int a ) {
    ...
}
```

Inhalt: Übung03 – Teil 1

• Aufgabe 3.01: Verständnisfragen

• Aufgabe 3.02: Funktionen – Deklaration, Prototypen, Aufruf

• Aufgabe 3.03: Funktionen und Header-Dateien

• Aufgabe 3.04: Programmstrukturen

• Aufgabe 3.05: Referenzen und Zeiger

• Aufgabe 3.06: Arrays und Zeigerarithmetik

Aufgabe 3.01: Verständnisfragen **Lsg. (1)**

- a) Ein Funktionsaufruf ist ein Ausdruck, dessen Typ bestimmt ist durch
- die an die Funktion übergebenen Argumente. Falsch
 - die im Funktionskopf deklarierten Parameter. Falsch
 - den Return-Wert der Funktion. Richtig
- b) Der Prototyp einer Funktion stellt dem Compiler Informationen über
- den Return-Typ der Funktion bereit. Richtig
 - die Namen der Parameter bereit. Falsch
 - den Typ jedes Parameters bereit. Richtig
- c) Ein Compiler erkennt eine falsche Anzahl von Argumenten nicht.
Falsch

Aufgabe 3.01: Verständnisfragen Lsg. (2)

- d) In C++ ist eine Standard-Header-Datei
- eine Objektdatei. Falsch
 - eine ausführbare Datei. Falsch
 - eine Textdatei. Richtig
- e) Zur Ausführung der Anweisung `cout << "Himm...!" << endl;` genügt es, die Header-Datei `iostream` in Ihrem Programm zu inkludieren. Falsch
- f) Eine außerhalb einer Funktion definierte Variable wird als global bezeichnet.
- g) Eine Funktion kann innerhalb einer anderen Funktion definiert werden. Falsch

Aufgabe 3.02: Funktionen - ... Lsg. (1)

- a) Bestimmen Sie die Fehler in folgenden Prototypen:
- i. `double calculate double x, double y;`
Lösung: `double calculate(double x, double y);`
- ii. `void myFunc(int n, m);`
Lösung: `void myFunc(int n, int m);`
- iii. `int your-Func();`
Lösung: `int your_Func();`
- iv. `Bool test(void);`
Lösung: `bool test(void);`

Aufgabe 3.02: Funktionen - ... Lsg. (2)

b) Welche der folgenden Funktionsaufrufe sind korrekt? Wenn der Funktionsaufruf nicht korrekt ist beschreiben Sie den Fehler.

i. `int max(int a, int b, int c);`
`int result = max(7, 12);`

Lösung: Die Anzahl der Parameter im Prototyp stimmt nicht mit der Anzahl Argumente beim Aufruf der Funktion überein.

ii. `double square(double wert);`
`double x = 2.1;`
`cout << square(x);`

Lösung: Richtig

Aufgabe 3.02: Funktionen - ... Lsg. (3)

b) Welche der folgenden Funktionsaufrufe sind korrekt? Wenn der Funktionsaufruf nicht korrekt ist beschreiben Sie den Fehler.

iii. `int random(void);`
`random(1);`

Lösung: Unzulässig. Die Funktion `random()` erwartet keine Argumente.

iv. `int random(int a);`
`random(1);`

Lösung: Richtig. Auch wenn die Funktion einen Rückgabewert hat, muss dieser nicht verwendet / zugewiesen werden.

Aufgabe 3.03: Funktionen und Header-Dateien

- Erstellen Sie ein Programm, das die Oberfläche und das Volumen einer Kugel mit dem Radius r berechnet. Dazu soll der Radius über die Tastatur eingelesen werden und später das Ergebnis auf dem Bildschirm ausgegeben werden.
- Die Konstante π ($= 3,1415927$) und die zwei Funktionen, die jeweils die Oberfläche und das Volumen berechnen, sollen in eine Header-Datei implementiert werden. Die Header-Datei wiederum soll in die Hauptdatei eingebunden werden.

```
C:\Dokumente und Einstellungen\Admin\Eigene Dateien\Cp
Geben Sie einen Radius (in cm) ein: 2
Oberflaeche der Kugel: 50.2655 cm^2
Volumen der Kugel: 33.5103 cm^3
```

$$O(r) = 4 \cdot \pi \cdot r^2$$

$$V(r) = \frac{4}{3} \cdot \pi \cdot r^3$$

Aufgabe 3.03: Funktionen + Header Lsg. (1)

```
#include <iostream>
#include "kugel.h"
```

main.cpp

```
using namespace std;
```

```
int main() {
    double r = 0;

    cout << "Geben Sie einen Radius (in cm) ein: ";
    cin >> r;

    if( cin == false ) {
        cout << "Ihre Eingabe ist nicht korrekt!" << endl;
    } else if( r < 0 ) {
        cout << "Sie haben einen negativen Wert eingegeben!" << endl;
    } else {
        cout << "Oberflaeche ist: " << oberflaeche( r ) << " cm2" << endl;
        cout << "Volumen ist: " << volumen( r ) << " cm3" << endl;
    }
    return 0;
}
```

Radius abfragen und einlesen

Eingaben überprüfen – ungültig / negativ
Funktionen aufrufen / Ergebnisse ausgeben

Aufgabe 3.03: Funktionen + Header Lsg. (2)

```
#ifndef _KUGEL_
#define _KUGEL_
```

kugel.h

```
#include <cmath>
```

```
const double PI = 3.1415927;
```

Konstante

```
double oberflaeche( double a );
```

```
double volumen( double b );
```

Prototypen

```
#endif
```

```
#include "kugel.h"
```

kugel.cpp

```
double oberflaeche( double a ) {
    double erg = 0;
    erg = 4 * PI * pow( a, 2.0 );
    return erg;
}
```

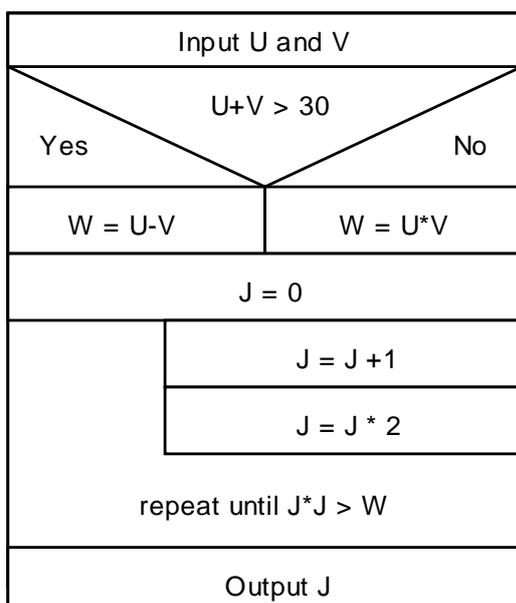
Implementierung

Funktion in <cmath> für x^y

```
double volumen( double b ) {
    double erg = 0;
    erg = 4.0 / 3 * PI * pow( b, 3.0 );
    return erg;
}
```

Aufgabe 3.04: Programmstrukturen Lsg.

- Ein Algorithmus ist festgelegt durch das folgende Nassi-Shneiderman-Diagramm



Bestimmen Sie den Ausgang J für die folgenden Eingänge (es soll kein Programm geschrieben werden):

- U = 20, V = 5
Lösung: 14
- U = 30, V = 30
Lösung: 2
- U = 19, V = 2
Lösung: 14

Aufgabe 3.05: Referenzen und Zeiger

- Beschreiben Sie jeweils die Aufgabe der entsprechenden Zeilen des unten stehenden Programms.
- Was erzeugt das Programm für eine Ausgabe? Bitte überlegen Sie bevor das Programm direkt kompilieren und sich die Ergebnisse anzeigen lassen.
- Welche zwei Besonderheiten fallen Ihnen in Bezug auf die letzten beiden Zeilen `++ptr` und Folgende auf.

Aufgabe 3.05: Referenzen & Zeiger Lsg. (1)

- Beschreiben Sie jeweils die Aufgabe der entsprechenden Zeilen des unten stehenden Programms.

```
int var = 256;
int& ref = var;
int* ptr = &var;
```

```
//Referenzen
```

```
++ref;
```

Referenz um eins inkrementieren

```
cout << var << "\t\t" << ref << endl;
```

Inhalt der Variablen und Referenz ausgeben

```
var += 64;
```

Variable um 64 erhöhen

```
cout << var << "\t\t" << ref << endl;
```

Inhalt der Variablen und Referenz ausgeben

```
cout << &var << "\t" << &ref << endl << endl;
```

Adresse der Variablen und Referenz ausgeben

Aufgabe 3.05: Referenzen & Zeiger Lsg. (2)

- a) Beschreiben Sie jeweils die Aufgabe der entsprechenden Zeilen des unten stehenden Programms.

```
//int var = 321;
//int& ref = var;
//int* ptr = &var;
```

Variable um eins inkrementieren

Inhalt der Variablen ausgeben

Inhalt der Adressvariablen ausgeben

Inhalt des Ziels der Adressvar. ausgeben

```
//Zeiger
++var;
```

```
cout << var << "\t\t" << ptr << "\t" << *ptr << endl;
```

```
*ptr += 12;
```

Worauf die Adressvar. zeigt um 12 erhöhen

```
cout << var << "\t\t" << ptr << "\t" << *ptr << endl;
```

Adresse der Variablen und Adressvariablen ausgeben

```
cout << &var << "\t" << &ptr << endl << endl;
```

```
ptr++;
```

Adressvariable um eins inkrementieren

```
cout << var << "\t\t" << ptr << "\t" << *ptr << endl;
```

Aufgabe 3.05: Referenzen & Zeiger Lsg. (3)

- b) Was erzeugt das Programm für eine Ausgabe? Bitte überlegen Sie bevor das Programm direkt kompilieren und sich die Ergebnisse anzeigen lassen.

```
int var = 256;
int& ref = var;
int* ptr = &var;
```

```
//Referenzen
```

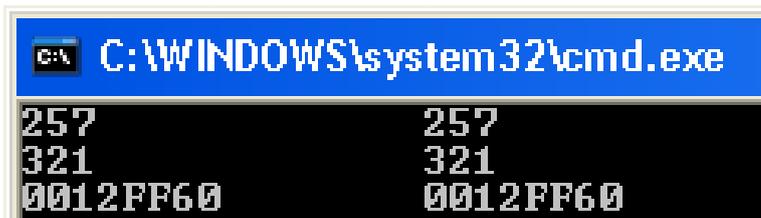
```
++ref;
```

```
cout << var << "\t\t" << ref << endl;
```

```
var += 64;
```

```
cout << var << "\t\t" << ref << endl;
```

```
cout << &var << "\t" << &ref << endl << endl;
```



```
C:\WINDOWS\system32\cmd.exe
257      257
321      321
0012FF60 0012FF60
```

Eine Referenz ist nur ein anderer Name für die Variable,
daher sind alle Werte gleich !

Aufgabe 3.05: Referenzen & Zeiger Lsg. (4)

- b) Was erzeugt das Programm für eine Ausgabe? Bitte überlegen Sie bevor das Programm direkt kompilieren und sich die Ergebnisse anzeigen lassen.

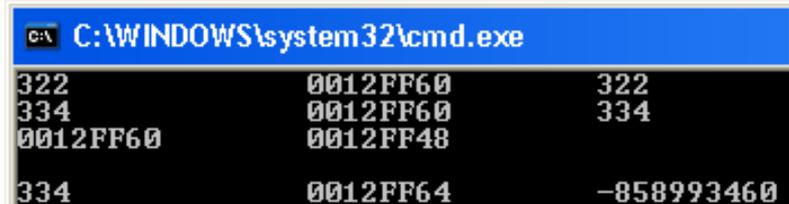
```
//int var = 321;
//int& ref = var;
//int* ptr = &var;

//Zeiger
++var;
cout << var << "\t\t" << ptr << "\t" << *ptr << endl;

*ptr += 12;
cout << var << "\t\t" << ptr << "\t" << *ptr << endl;

cout << &var << "\t" << &ptr << endl << endl;

ptr++;
cout << var << "\t\t" << ptr << "\t" << *ptr << endl;
```



322	0012FF60	322
334	0012FF60	334
0012FF60	0012FF48	
334	0012FF64	-858993460

Die Adressvariable ist eigenständig und erreicht die Variable nur über den Verweisoperator !

Aufgabe 3.05: Referenzen & Zeiger Lsg. (5)

- c) Welche zwei Besonderheiten fallen Ihnen in Bezug auf die letzten beiden Zeilen `++ptr` und Folgende auf.

Antwort:

`++ptr` inkrementiert den Inhalt der Adressvariablen um 4, was vier Byte entspricht. Der Grund hierfür besteht darin, dass `++ptr` ein Zeiger auf `int` ist und Integer bei einem PC 4 Bytes an Speicher einnehmen.

Da die Adressvariable nun auf eine Adresse 4 Byte weiter zeigt und niemand in diesem Fall weiß, was dort gespeichert ist, kann man die Ausgabe nicht vorhersagen.

Das Programm gibt in diesem Fall keine Fehlermeldung aus. Es geht einfach an die entsprechende Speicherstelle, liest die nächsten 4 Byte (da Integer), interpretiert diese als Integer und gibt das erhaltene Ergebnis aus. → **Führt zu Fehlverhalten**

Aufgabe 3.06: Arrays und Zeigerarithmetik

- Was erzeugt das folgende Programm für eine Ausgabe auf dem Bildschirm?

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

int main() {
    char arr[] = "Informationstechnikbuch";
    char* p = arr;
    char* q = arr + 6;

    cout << q - p - 3 << *q;
    p = q++ + 8;

    while( p < q + 9 ) {
        cout << *p++;
    }

    cout << *q << *( p - 3 ) << *( arr + 4 );

    p = arr + 19;
    cout << *p << *--q << *( arr + sizeof( arr ) - 2 ) << "nen";

    return 0;
}
```

Aufgabe 3.06: Arrays und Zeigerarithmetik Lsg.

(1)

- Was erzeugt das folgende Programm für eine Ausgabe auf dem Bildschirm?

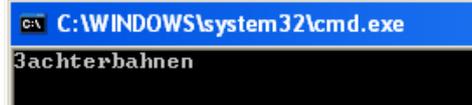
```
#include <iostream>
using namespace std;

int main() {
    char arr[] = "Informationstechnikbuch";
    char* p = arr;           //p = &arr[0]
    char* q = arr + 6;      //q = &arr[6]

    cout << q - p - 3 << *q;
    p = q++ + 8;           //p = &arr[6+8]
                          //hier ist q = &arr[7]

    while( p < q + 9 ) {   //&arr[14] < &arr[7+9]
        cout << *p++;      //cout << arr[14]; p = &arr[15];
    }

    cout << *q << *( p - 3 ) << *( arr + 4 );
    //== cout << arr[7] << arr[16-3] << arr[4];
    p = arr + 19;
    cout << *p << *--q << *( arr + sizeof( arr ) - 2 ) << "nen";
    //== cout << arr[19] << arr[6] << arr[22] << "nen";
    return 0;
}
```



Schleife wird wiederholt,
solange $p < \&arr[16]$

sizeof() gibt Anzahl der Bytes zurück bzw. bei
char-Array, Anzahl der Zeichen (inkl. '\0')

Aufgabe 3.06: Arrays und Zeigerarithmetik Lsg.

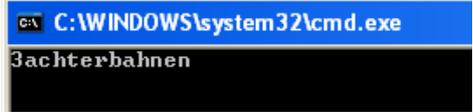
(2)



Was erzeugt das folgende Programm für eine Ausgabe auf dem Bildschirm?

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

C-String! Letztes Zeichen '\\0'!



```
int main() {
    char arr[] = "Informationstechnikbuch";
    char* p = arr;
    char* q = arr + 6;

    cout << q - p - 3 << *q;
    p = q++ + 8;

    while( p < q + 9 ) {
        cout << *p++;
    }

    cout << *q << *( p - 3 ) << *( arr + 4 );

    p = arr + 19;
    cout << *p << *--q << *( arr + sizeof( arr ) - 2 ) << "nen";

    return 0;
}
```

- 3a
- ch
- t
- e
- r
- b
- a
- arr[24-2] = h
- nen

Referenz



- Kompodium: Kapitel 2 - Variablen, Zeiger und Arrays (ab 2.9)
Kapitel 4 – Funktionen
- Tutorium: Aufgabe 5 – 10