Fachhochschule Köln
Cologne University of Applied Sciences

Angewandte Mathematik und Programmierung

Einführung in das Konzept der objektorientierten Anwendungen zu wissenschaftlichen Rechnens mit C++ und Matlab

SS2013

Fomuso Ekellem

Inhalt

- **Bis jetzt:**
  - Entwicklungsumgebung
  - Datentypen
  - Operatoren
  - Ausdrücke
  - Schleifen
- **Heute:**
  - Strukturen
  - Zeiger
  - Referenzen

Strukturen/Klassen

- Eine Struktur ist eine Sammlung von verwandten Daten in einer einzigen Speichereinheit.
- Wenn Sie beispielsweise eine Adressliste verwalten wollen, können Sie alle Felder, die in einer typischen Adressliste benötigt werden, in einer einzigen Datenvariablen abspeichern.
- Zunächst deklarieren Sie eine Struktur und erzeugen dann später, wenn Sie die Struktur einsetzen wollen, eine Instanz (Ausprägung) dieser Struktur. Strukturen werden mit dem Schlüsselwort struct deklariert.
- Strukturen sind für alle andere Strukturen in einem Projekt sichtbar und stehen frei zu Verfügung für alle Projekt Mitglieder.
- **Siehe Beispiel**
  - Struktur Deklaration
  - Instanzierung von Struct ur(in „main“ oder unmittelbar nach struct deklaration) Sie können die Instanzen einer Struktur auch schon bei ihrer Deklaration erzeugen. Fügen Sie dazu am Ende der Strukturdeklaration zwischen der schließenden Klammer und dem Semikolon, einen (oder mehrere) Variablennamen an.
  - Struct als eigene Header Datei

Zeiger

- **Ein Zeiger (englisch: pointer) ist eine Variable, die die Adresse einer anderen Variable enthält. Da der Zeiger keine direkte Verbindung zu den eigentlichen Daten herstellt, spricht man auch von Indirektion oder Referenzierung.**

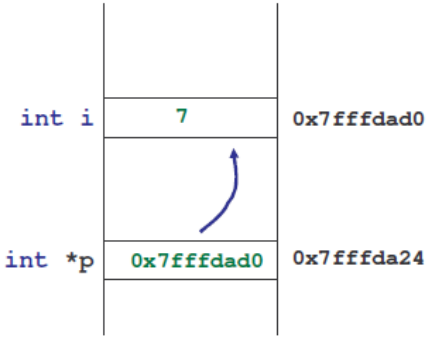
Zeiger Variablen

Erinnerung Variable

- Variable = Name für Speicherbereich = Name für Adresse
- Typ (z.B. int) definiert, wie Bits interpretiert werden sollen, die an dieser Adresse gespeichert sind.
- Jede Variable ist genau einem Adressbereich fest zugeordnet

Zeiger

- Variable, wie alle anderen auch
- Steht irgendwo im Speicher an bestimmter Adresse
- Hat einen Wert.
- Bedeutung des Wertes = Adresse einer anderen Variable



Zeiger Variablen

- Die allgemeine Syntax zur Deklaration eines Zeiger lautet:

```
DatenTyp * variable [ = Wert];
```

```
int a = 42;
int* z1, x1; //Achtung: x1 ist KEIN Zeiger
int* z2 = z1,* x2; //z2 und x2 sind Zeiger
```

- Adress-Operator (&) “ampersand“ Unärer Operator, der die Speicheradresse des Operanden zurückliefert .
- Dereferenzierungs-Operator (*) Unärer Operator, gibt eine Referenz auf das Objekt zurück, auf welches der Operand zeigt.

Achtung: je nach Kontext auch Multiplikation !

Zeiger

Eigenschaften von Zeiger

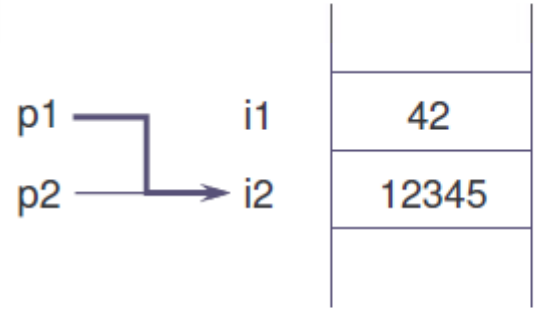
- Auf Wert einer anderen Variable zugreifen, ohne deren Name zu verwenden (oder kennen)!
- Ansonsten fast alle Fähigkeiten der normalen Variablen
  - Arithmetik, Zuweisen, Vergleichen
- Direkter Zugriff auf den Speicher des Computers.
- Für schnelle maschinennahe Programmierung
- Keine Überprüfung auf korrekte Handhabung
- Leider teilweise unverzichtbar.
- Sehr "gewöhnungsbedürftige" Syntax

```
float a = 42;
std::cout << "Variable a steht im Speicher ab Adresse " << &a << std::endl;
```

Ausgabe: Variable a steht im Speicher ab Adresse 0xbffff588

Zeiger

```
int i1 = 0, i2 = 100;
int *p1 = &i1, *p2 = &i2;

*p1 = 42;  //     i1=42
*p2 = *p1; //     i2=i1
p1 = p2;
*p1 = 12345;
int* p3;
```



p3 == NULL besonderen Wert 0x00000000

Dynamische Variablen

- Variablen die zur Laufzeit (auf dem sog. *Heap)* angelegt werden
- Mit Hilfe von Zeigern verwaltet man in C++ dynamische Variablen
- Speicheranforderung für dyn. Variablen mittels *new-Operator.*
- Speicherfreigabe von dyn. Variablen mittels *delete-Operator.*
- Achtung: in C++ sind *new und delete reservierte* Schlüsselworte.

Zeiger, **new-Operator**

- Syntax, um mit *new Variablen anzulegen*

```
// Anlegen einer einzigen Variable von „Type“
new Type;
// Anlegen eines Feldes von Variablen von „Type“
new Type[intexpr];
new class(arglist); // Anlegen einer Instanz
```

- Mit *new werden neue Instanzen von* Elementartypen oder Klassen angelegt.
- Bei Klassen kann man den zu verwendenden Konstruktor angeben.

Zeiger, **new-Operator**

```
// Anlegen einer einzigen Variable von „Type“
new Type;
// Anlegen eines Feldes von Variablen von „Type“
new Type[intexpr];
```

- intexpr kann jeder Ausdruck sein, der einen positiven int zurückliefert
- *new legt den benötigten Speicher konsekutiv an* (intexpr*sizeof(Type)) Bytes und liefert die Adresse des ersten Elementes zurück
- Initialisiert mit 0 bzw. Default-Konstruktor

Zeiger, **delete-Operator**

- Syntax um dynamische Variablen freizugeben

```
// Freigabe einer einzigen dyn. Variable
delete zeiger;
// Freigabe eines dyn. Feldes
delete[] zeiger;
```

- Bei Freigabe von Klassenvariablen werden deren Destruktoren aufgerufen
- Achtung: der Inhalt der Zeigers wird durch *delete* nicht verändert, aber der Speicher auf den er verweist ist danach „ungültig“

Zeiger

	*	&

als **Modifier**	`int* p;`
	`void function( int* p);`
	Deklaration eines Zeigers
		`int& a = b;`
		`void function( int& a);`
		Deklaration einer Referenz

als **unärer** Operator	`std::cout << *p;` **Dereferenzierung:** Der Speicherinhalt ab Adresse p wird dem Zeigertyp entsprechend interpretiert und ausgegeben.	`std::cout << &a;` **Adress-Operator:** Gibt Adresse aus, an der Variable a im Speicher gespeichert ist.
als **binärer** Operator	`std::cout << a * b;` **Muliplikation** von a und b	`std::cout << (a & b);` **Bitweises logisches "Und"** von a und b

Zeiger

```
int a = 42;
int* b = &a;
*b = 27;
std::cout << a << std::endl;
```

Ergebnis? 27

```
int a = 42, b = 137;
int* c = &a;
int* d = &b;
c = d;
std::cout << a << "\n" << *c
          << std::endl;
```

Ergebnis? 42 137

```
int a = 42;
int b = 137;
int*   c = &a;
int** d = &c;
std::cout << *d << std::endl;
```

Ergebnis? 0x???????? Adresse von c

Zeiger

```
int   a = 42;
int* p;
p = &a;

std::cout << "an der Adresse " << p
          << " steht (als integer) "
          << *p << std::endl;

float* q;
q = p;
q = reinterpret_cast<float*>(p);

std::cout << "an der Adresse " << q
          << " steht (als float) "
          << *q << std::endl;
```

p enthält eine Adresse. Der Speicherinhalt dort soll als **Integer** interpretiert werden

p enthält jetzt die Adresse, wo die Variable a im Speicher liegt.

q enthält eine Adresse. Der Speicherinhalt dort soll als **float** interpretiert werden

hier meckert der Compiler – zurecht!

das muß der Compiler akzeptieren

```
an der Adresse 0xbffff578 steht (als integer) 42
an der Adresse 0xbffff578 steht (als float) 5.88545e-44
```

Zeigerarithmetik

Zeiger sind Zahlen, d.h. wir können mit ihnen rechnen:

```
int * p = &a;
```

p *(aa8200)*	*aa8192*
b (aa8196)	9
a (aa8192)	16

```
*p = 200;
```

p *(aa8200)*	*aa8192*
b (aa8196)	9
a (aa8192)	200

```
*(p+1) = 300;
```

p *(aa8200)*	*aa8192*
b (aa8196)	300
a (aa8192)	200

Zeiger p weist auf ein int, d.h. durch Addition von 1 erhöhen wir die Adresse um die Größe eines int. Der C/C++ Ausdruck dafür lautet *sizeof*(int).

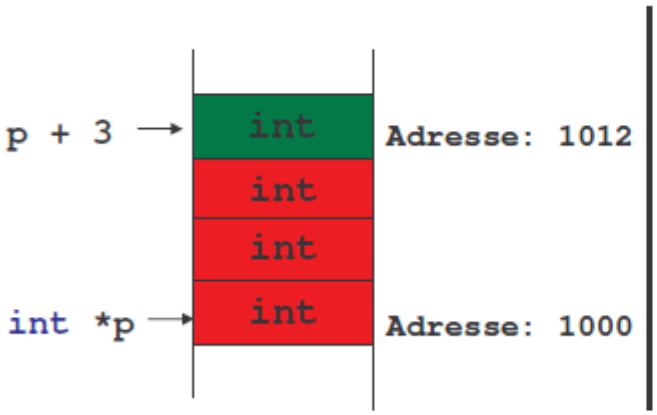
Zeigerarithmetik

- Ist die low-level-Methode, mit Arrays zu arbeiten
- Ausdruck der Form (auch – statt +)

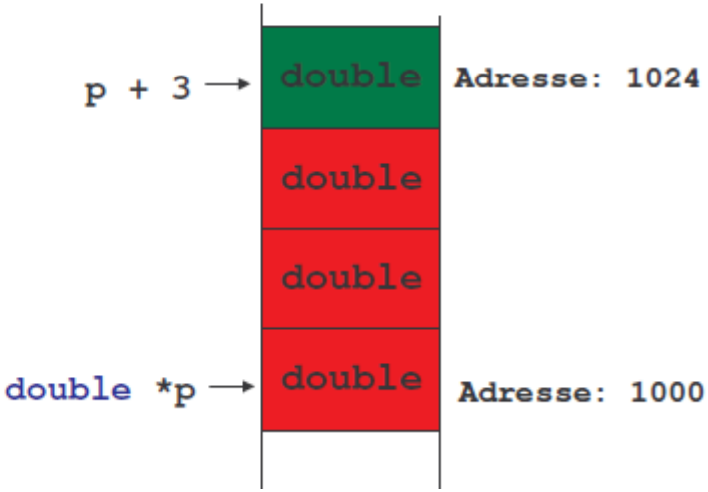
  `pointer + integer`

  Achtung! Bedeutung ist **nicht:** `Adresse + k`
- Sondern: `Adresse + k*sizeof(T)`

`sizeof(int) = 4`



`sizeof(double) = 8`



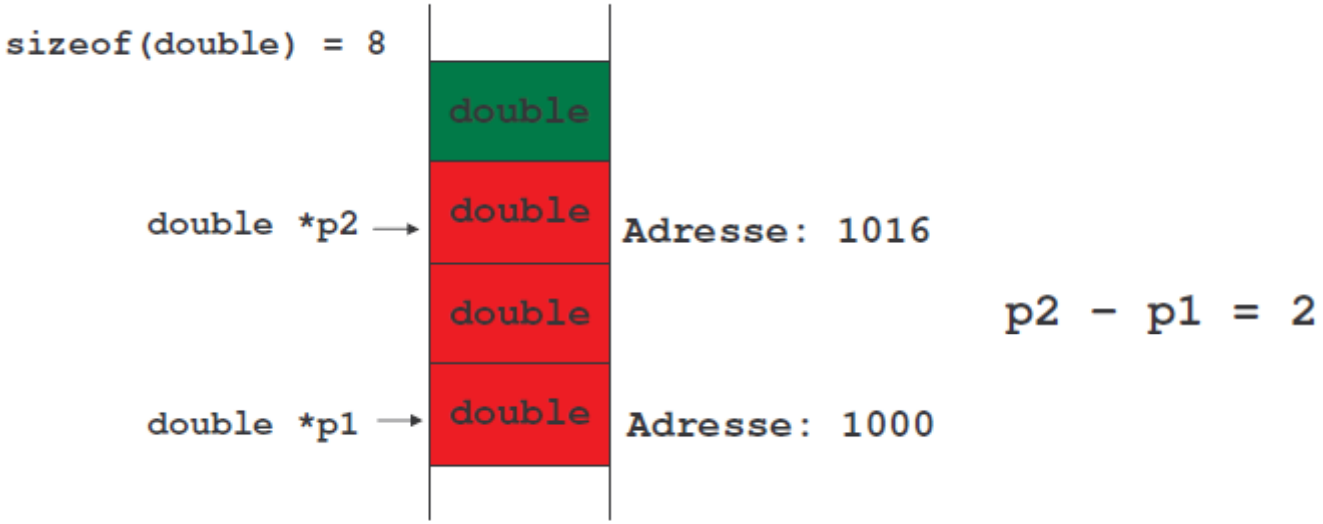
Zeigerarithmetik

- Subtraktion von Zeigern:

```
pointer1 - pointer2
```

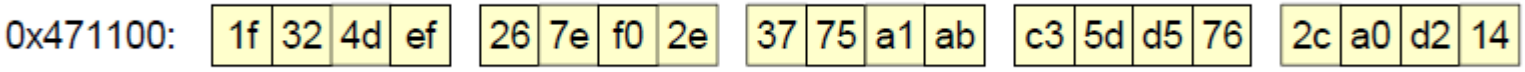
- Bedeutung:

```
(Adresse1 - Adresse2) / sizeof(T)
```

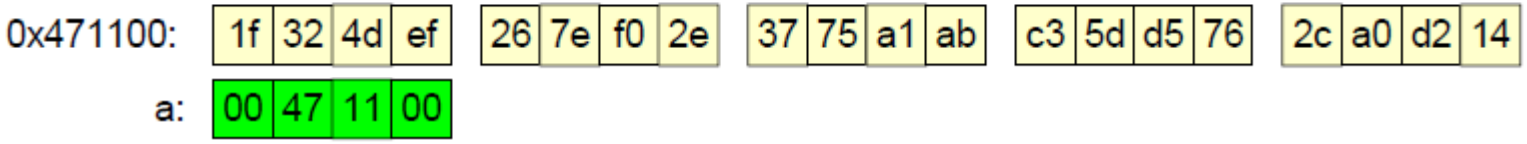


Zeiger und Felder

```
int* a = new int[5];// reserviere einen Speicherbereich, der
                    // 5 integers aufnehmen kann.
```

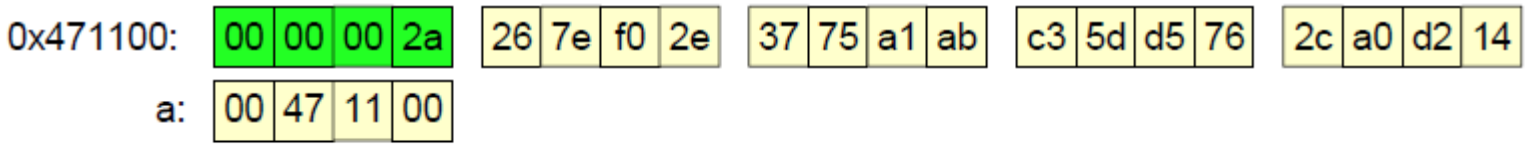



```
int* a = new int[5];// reserviere einen Speicherbereich, der
                    // 5 integers aufnehmen kann. In a wird die
                    // Anfangsadresse von diesem Bereich gespeichert
```



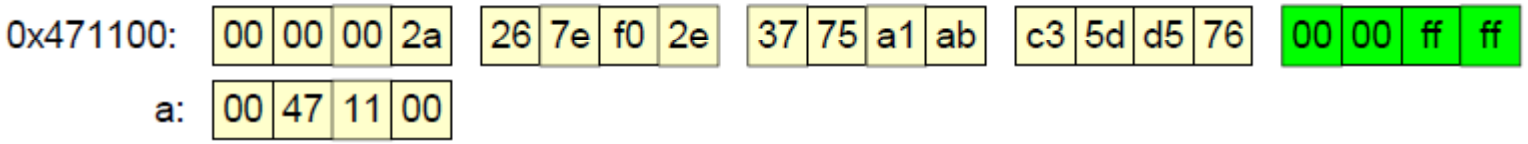
Zeiger und Felder

```
int* a = new int[5];// reserviere einen Speicherbereich, der
                    // 5 integers aufnehmen kann. In a wird die
                    // Anfangsadresse von diesem Bereich gespeichert

a[0] = 42;          // schreibe 42 in das erste Element
```



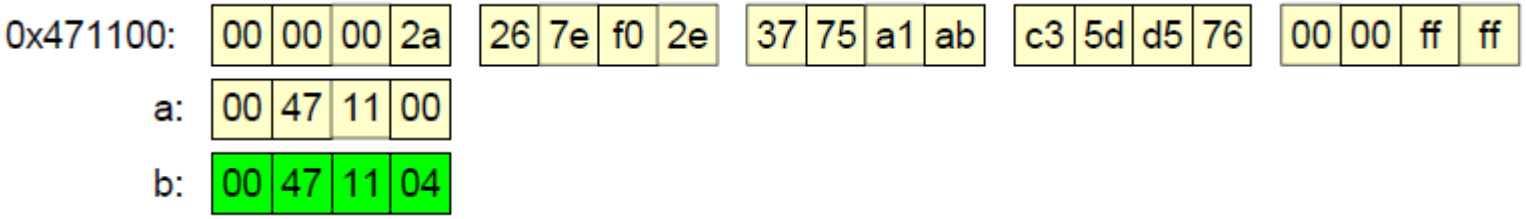
Zeiger und Felder

```
int* a = new int[5];// reserviere einen Speicherbereich, der
                    // 5 integers aufnehmen kann. In a wird die
                    // Anfangsadresse von diesem Bereich gespeichert

a[0] = 42;          // schreibe 42 in das erste Element

a[4] = 65535;       // schreibe 65535 in das letzte Element
```



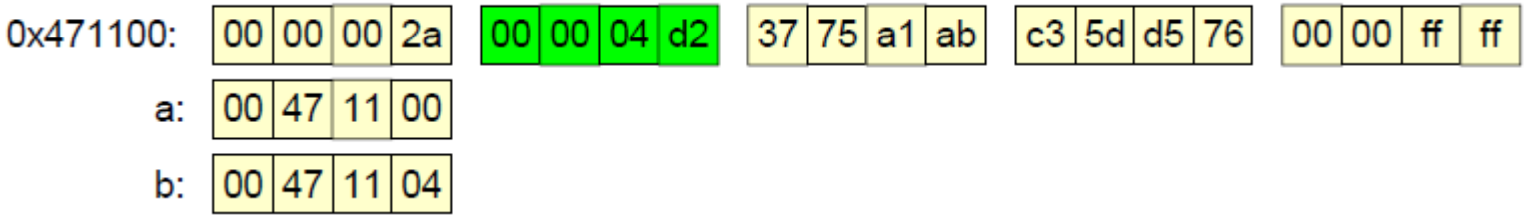
Zeiger und Felder

```
int* a = new int[5];// reserviere einen Speicherbereich, der
                    // 5 integers aufnehmen kann. In a wird die
                    // Anfangsadresse von diesem Bereich gespeichert

a[0] = 42;          // schreibe 42 in das erste Element

a[4] = 65535;       // schreibe 65535 in das letzte Element

int* b = a + 1;     // Rechenoperationen bei Zeigern berücksichtigen
                    // deren Datentyp! Also zeigt b 4 Bytes hinter a
```



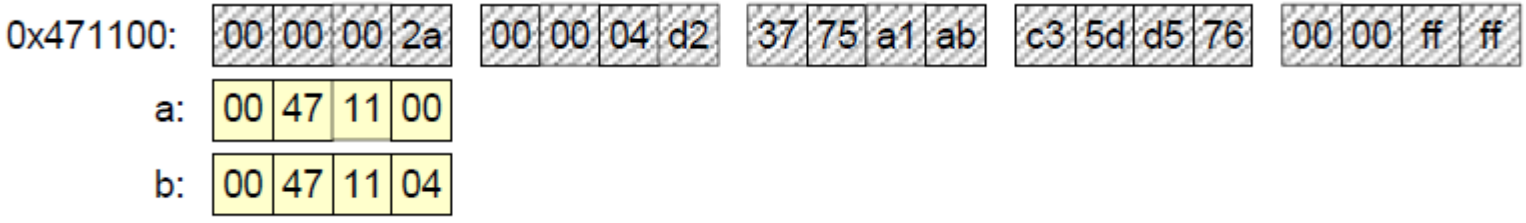
Zeiger und Felder

```
int* a = new int[5];// reserviere einen Speicherbereich, der
                    // 5 integers aufnehmen kann. In a wird die
                    // Anfangsadresse von diesem Bereich gespeichert

a[0] = 42;          // schreibe 42 in das erste Element

a[4] = 65535;       // schreibe 65535 in das letzte Element

int* b = a + 1;     // Rechenoperationen bei Zeigern berücksichtigen
                    // deren Datentyp! Also zeigt b 4 Bytes hinter a

*b = 1234;          // schreibe 1234 in das zweite Element des Feldes
```



Zeiger und Felder

```
int* a = new int[5];// reserviere einen Speicherbereich, der
                    // 5 integers aufnehmen kann. In a wird die
                    // Anfangsadresse von diesem Bereich gespeichert

a[0] = 42;          // schreibe 42 in das erste Element

a[4] = 65535;       // schreibe 65535 in das letzte Element

int* b = a + 1;     // Rechenoperationen bei Zeigern berücksichtigen
                    // deren Datentyp! Also zeigt b 4 Bytes hinter a

*b = 1234;          // schreibe 1234 in das zweite Element des Feldes

delete[] a;         // gib den Speicherbereich wieder frei.
```



Beispiel

- **Siehe Beispiel**
  - Dereferenzierung von Zeigern
  - Zugriffsmöglichkeiten bei Zeigern
  - Zeiger auf Felder

Referenzen

- Eine Referenz ist eine besondere Art von Zeiger, die es ermöglicht, einen Zeiger wie ein reguläres Objekt (als alias) zu behandeln. Referenzen werden mit dem Referenzierungsoperator (&) deklariert.
- **Siehe Beispiel**
  - Referenzierungsoperator (&)