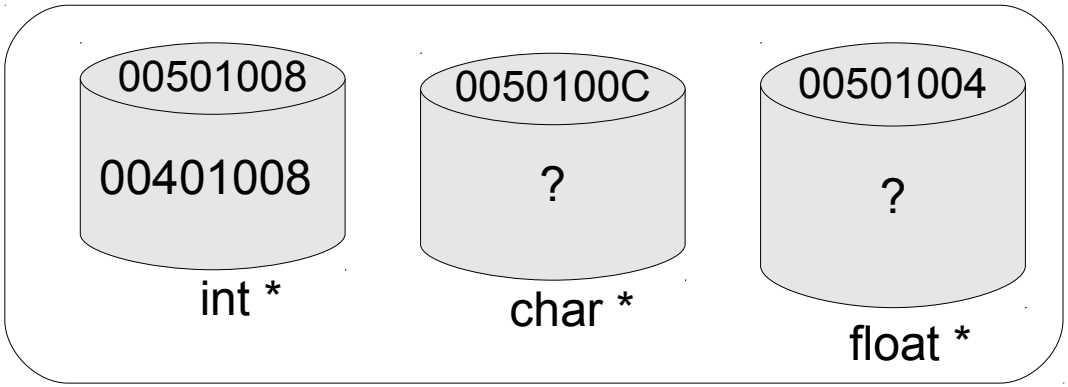
C++ - Einführung in die Programmiersprache

Zeiger und deren Nutzung



Zeiger (Pointer)

- Barcode einer Box.
- Speicherung von Adressen im Speicher.
- Platzhalter für den Beginn eines Speicherbereichs. An der Position ist ein Wert von einem bestimmten Datentyp gespeichert.

Vorteile / Nutzen

- Der Verweis auf eine Speicheradresse benötigt meist weniger Speicher als der Wert, auf den verwiesen wird.
- Manipulation von Werten an verschiedenen Positionen.
- Reservierung und Freigabe von Speicher.
- Implementation von Stacks, Listen etc.

Variablen in C++

```
int value;
int valueExpression;

value = 5;

valueExpression = value * value;
```

Erläuterung

- Speicherung von Werten.
- Die Deklaration einer Variablen legt den Datentyp fest. Welche Werte können in der Variablen abgespeichert werden?
- Mit Hilfe des Gleichheitszeichens wird der Variablen ein Wert zugewiesen. Der Wert kann entsprechend des Datentyps der Variablen interpretiert werden.
- Der zugewiesene Wert kann mit Hilfe eines Ausdrucks berechnet werden.

Zeiger in C++

```
int value;
int valueExpression;
int *pointer;

value = 5;
pointer = &value;
valueExpression = *pointer * value;
```

Erläuterung

- Speicherung von Adressen, an denen beliebige Werte abgelegt wurden.
- Die Deklaration eines Zeigers legt fest, wie groß der Speicherbereich ist, auf den verwiesen werden soll.
- Mit Hilfe des Gleichheitszeichens wird dem Zeiger eine Speicheradresse zugewiesen. Die Adresse ist der Anfang eines Speicherbereiches.
- Der Wert, der an dieser Position hinterlegt ist, kann in Ausdrücken zur Berechnung genutzt werden. Aus der, im Zeiger gespeicherten Adresse kann eine neue berechnet werden.

Deklaration

`int`		`value`	`;`
`int`	`*`	`pointer`	`;`
`Datentyp`		`name`	`;`

- Jede Variablendeklaration beginnt mit der Angabe des Datentyps.
- Dem Datentyp folgt ein Bezeichner. Der Bezeichner ist frei wählbar.
- Zeiger haben ein Sternchen als Präfix des Bezeichners.

Datentypen eines Zeigers

- Wie groß ist der Datenbereich auf den verwiesen wird?
- Auf welche Kategorie von Werten wird verwiesen?
- Zeiger können jeden Datentyp wie `int`, `char` etc. besitzen.

Position des Sternchens

```
int *pointer;
int* zeiger;
int * zeiger
```

- Das Setzen von Leerzeichen vor und nach dem Sternchen ist optional.
- In der ersten Zeile ist wird der Datentyp `int` genutzt. Das Sternchen kennzeichnet den Namen als Zeiger, der vom Datentyp Integer ist.
- In der zweiten Zeile wird der Datentyp `int*` genutzt. Der Zeiger ist vom Datentyp „Zeiger auf ein Integer“.

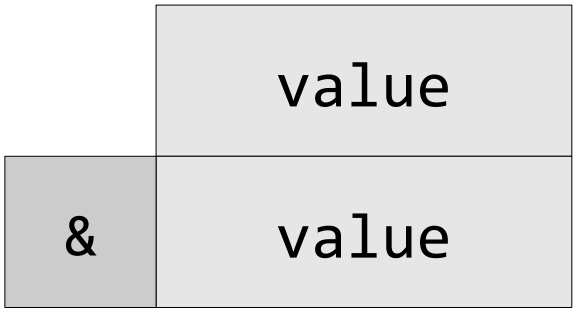
Name eines Zeigers

- Der Name ist ein Etikett für den Anfang eines Speicherbereichs.
- Der Name ist frei wählbar.
- Der Bezeichner sollte den Wert widerspiegeln, auf den dieser verweist.

Bezeichner

- Benutzerdefinierte Namen für Konstanten, Variablen, Zeiger Funktionen etc.
- Die Namen sind in ihrem Codeblock eindeutig.
- Schlüsselwörter der Programmiersprache sind als benutzerdefinierte Namen nicht erlaubt.
- Unterscheidung zwischen Groß- und Kleinschreibung. Die Namen „MINZAHL“ und „MINzAHL“ sind Etiketten für unterschiedliche Speicherplätze in C++.

Adresse einer Variablen



- Der Variablenname ist ein Platzhalter für einen Wert von einem bestimmten Typ.
- Der Wert selber ist irgendwo im Speicher abgelegt.
- Mit Hilfe des Adressoperators (kaufmännisches Und) direkt vor dem Variablennamen wird die Adresse des Ablageortes ermittelt.

… an einen Zeiger übergeben

`value`	`=`		`value`
`pointer`	`=`	`&`	`value`

- Variablen wird ein Wert einer anderen Variablen zugewiesen.
- Zeigern wird die Adresse einer Variablen oder der Beginn eines Speicherbereichs zugewiesen.

Verweis auf den Wert an der Speicherstelle

`value`	`=`		`value`
`value`	`=`	`*`	`pointer`

- Einer Variablen wiederum kann der Wert auf den, der Zeiger zeigt, zugewiesen werden.
- Der Zeiger wird dereferenziert. Der Verweis wird aufgelöst.
- Als Präfix hat der Zeiger ein Sternchen. Das Sternchen wird als Dereferenzierungsoperator genutzt.

Hinweise

- Ob der Zeiger auf eine Speicheradresse zeigt oder nicht, wird nicht automatisiert überprüft.
- Falls der Dereferenzierungsoperator in einer Ausdrucksanweisung vergessen wird, wird eine Fehlermeldung „invalid conversion“ ausgegeben.
- Der Wert, auf den der Zeiger verweist, wird entsprechend der Verarbeitung interpretiert.

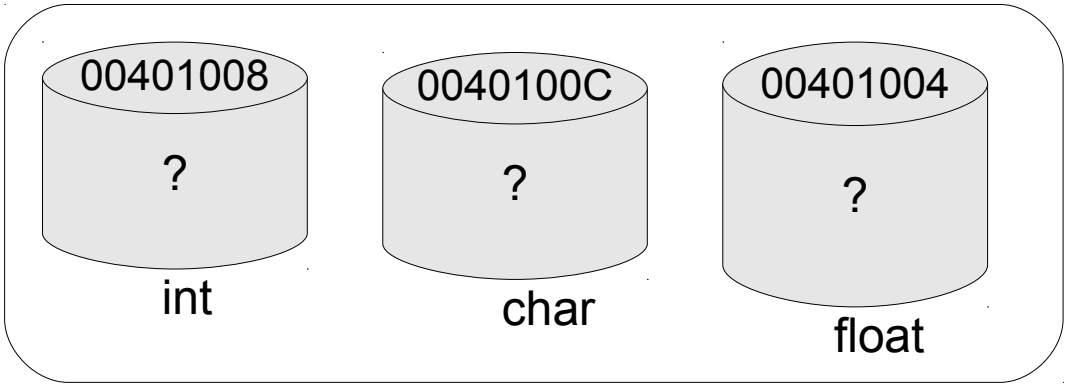
Kennzeichnung eines undefinierten Zeigers

```
int *pointer = 0;
int *pointer = NULL;
int *pointer = nullptr;
```

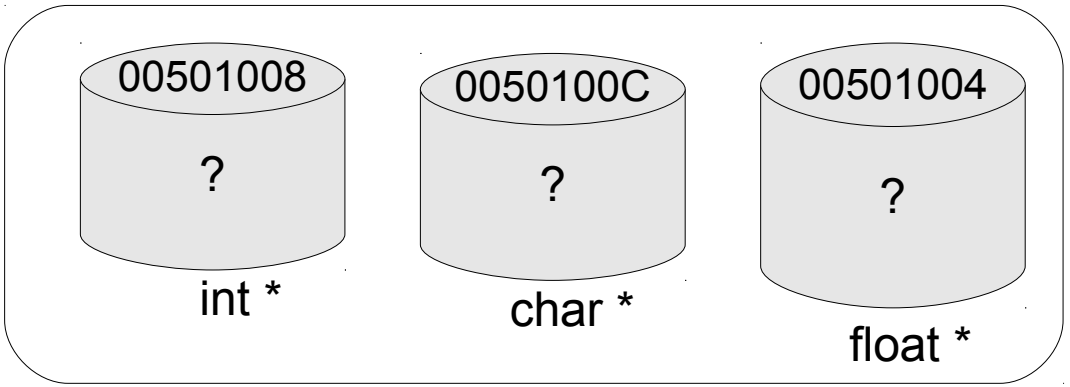
- `NULL` ist zum Beispiel in der Bibliothek `<iostream>` mit einer Konstanten vom Wert 0 definiert.
- Seit C++11 kann durch Zuweisung von `nullptr` ein Zeiger zu einem Null-Pointer werden. Der Null-Pointer ist in der Bibliothek `<cstddef>` definiert und zeigt nie auf valide Daten.

Grafische Darstellung: Deklaration

… der Variablen

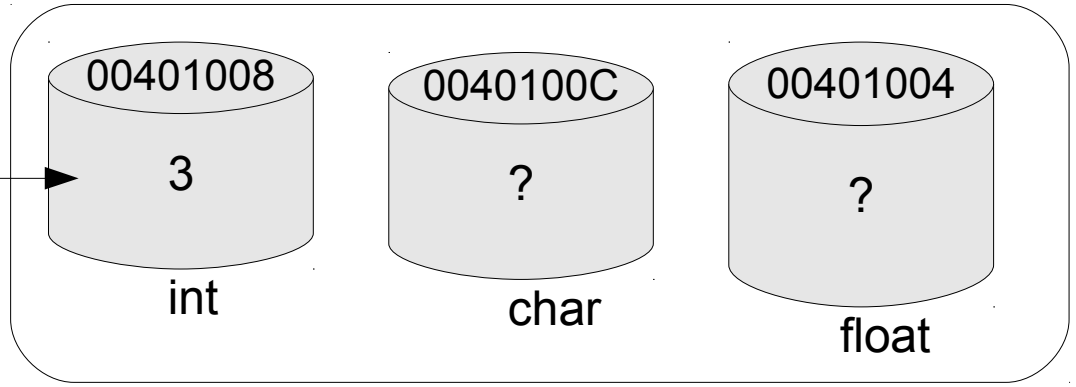


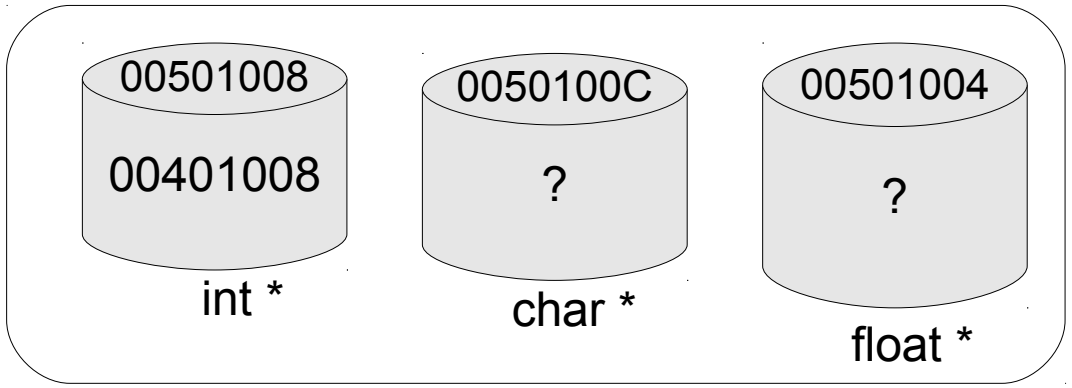
… der Zeiger



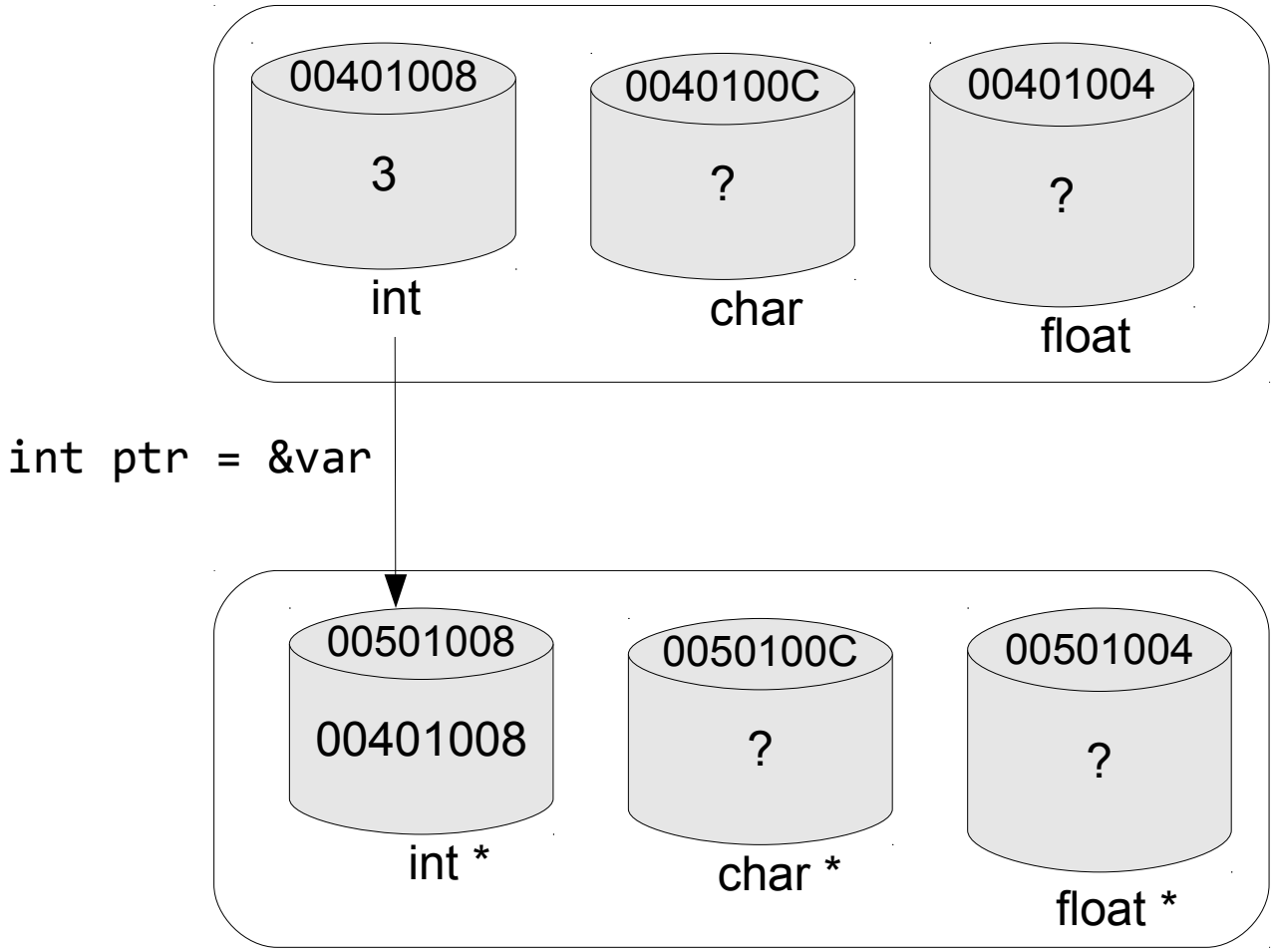
Grafische Darstellung: Initialisierung der Variablen

```
int var = 3
```

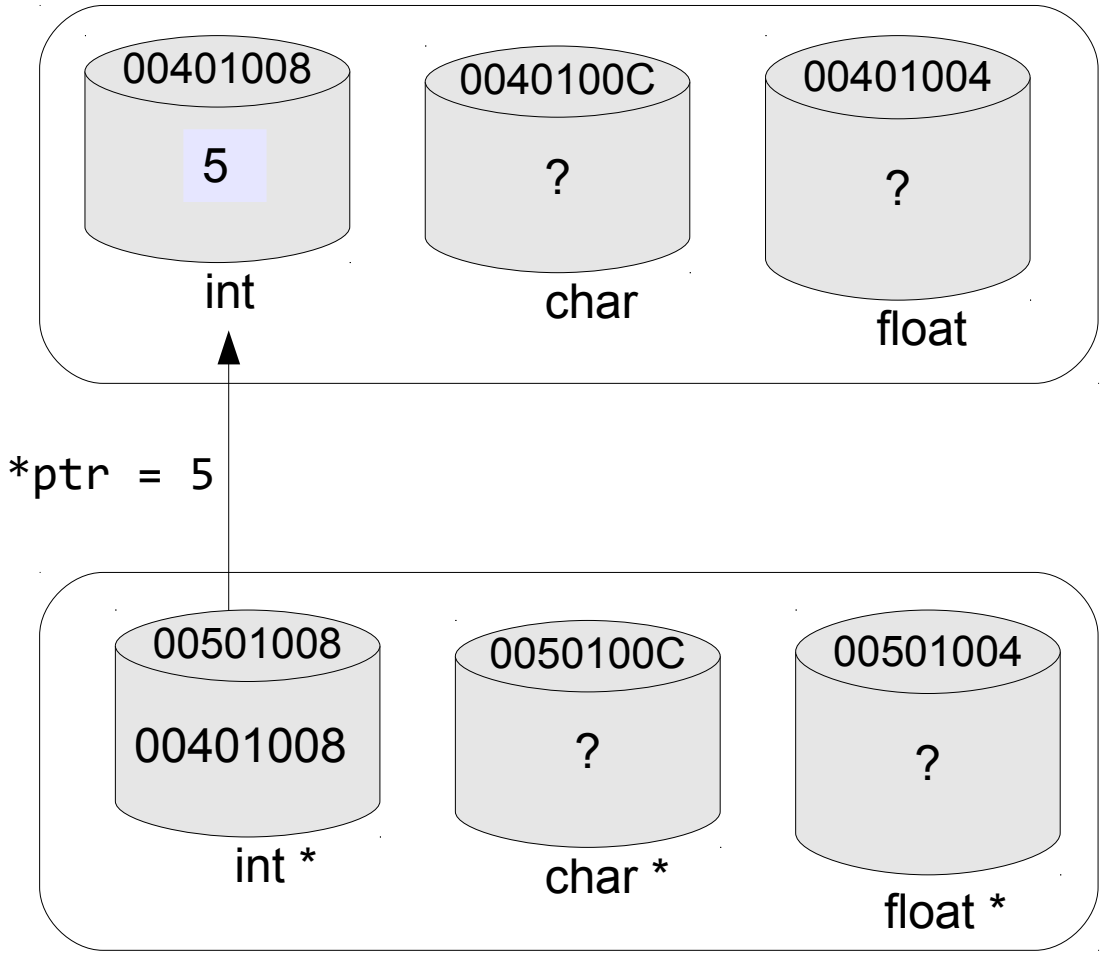




Grafische Darstellung: Initialisierung des Zeigers



Grafische Darstellung: Derefenzierung



Anforderung von Speicher

```
int *pointer = nullptr;

if(!pointer){
    pointer = new int;
};

delete pointer;
pointer = nullptr;
```

Verweis der Zeiger auf eine Speicheradresse?

```
if(!pointer){
```

- Das Ausrufezeichen negiert den Inhalt des Zeigers.
- Wenn der Zeiger auf einen nicht validen Speicher verweist, dann ...

Anforderung von Speicher

```
int *pointer = nullptr;
pointer = new int;
```

- Durch das Schlüsselwort `new` wird Speicher zur Laufzeit des Programms angefordert.
- In diesem Beispiel wird zur Speicherung eines Integer-Wertes Speicherplatz angefordert.

Hinweise

- Für jeden Datentyp int, double, char etc. kann Speicher angefordert werden.
- Der Datentyp des Zeigers sollte passend zur angeforderten Speichergröße gewählt werden.
- Wenn der Zeiger auf einen gültigen Speicherbereich verweist und diesem nochmals Speicher mit Hilfe von `new` zugewiesen wird, wird der Speicherbereich komplett neu überschrieben.

Freigabe von Speicher

```
delete pointer;
pointer = nullptr;
```

- Mit Hilfe von `delete` wird Speicher, auf den der Zeiger verweist wieder freigegeben.
- Hinweis: Speicher, der mit `new` angefordert wurde, muss durch das Schlüsselwort `delete` frei gegeben werden.

Dynamische Arrays

```
int MAX_SIZE = 12;
double *pointer = nullptr;

if(!pointer){
    pointer = new double[MAX_SIZE];
}

for(int index = 0; index < MAX_SIZE; index++){
     pointer[index] = index;
}

delete[] pointer;
pointer = nullptr;
```

Anforderung von Speicher

```
int *pointer = nullptr;
pointer = new double[MAX_SIZE];
```

- Das Schlüsselwort `new` fordert neuen Speicher an.
- Die Größe eines Elements wird durch die Angabe eines Datentyps festgelegt. In diesem Beispiel wird Speicherplatz in der Größe eines `Double` pro Element angefordert.
- In den eckigen Klammern wird die Anzahl der Elemente angegeben. In diesem Beispiel wird die Anzahl durch eine Konstante festgelegt.

Freigabe von Speicher

```
delete[] pointer;
pointer = nullptr;
```

- Mit Hilfe von `delete` wird Speicher, auf den der Zeiger verweist wieder freigegeben.
- Bei Anforderung von mehr als einem Element, müssen dem Schlüsselwort `delete` die leeren eckigen Klammern folgen. Andernfalls entsteht ein Speicherleck.

Elemente des Arrays

```
pointer[index] = index;
```

- Der Zeiger `pointer` verweist auf das erste Element in einem Array.
- Dem Namen des Zeigers folgen die eckigen Klammern. In den eckigen Klammern wird ein Index angegeben.
- Der Index ist eine Ganzzahl, die eindeutig ein Element in einem Array identifiziert.

Andere Möglichkeit

```
index = 2
*(tmpptr + index)  = index;
```

- Der Zeiger verweist auf das erste Element in einem Array.
- Dieser Speicheradresse in diesem Beispiel durch Addition verändert.
- Der Zeiger wird um x Positionen verschoben.

Zeiger auf das erste Element

```
int MAX_SIZE = 12;
double *pointer = nullptr;
double *tmpptr = nullptr

pointer = new double[MAX_SIZE];
tmpptr = pointer;
```

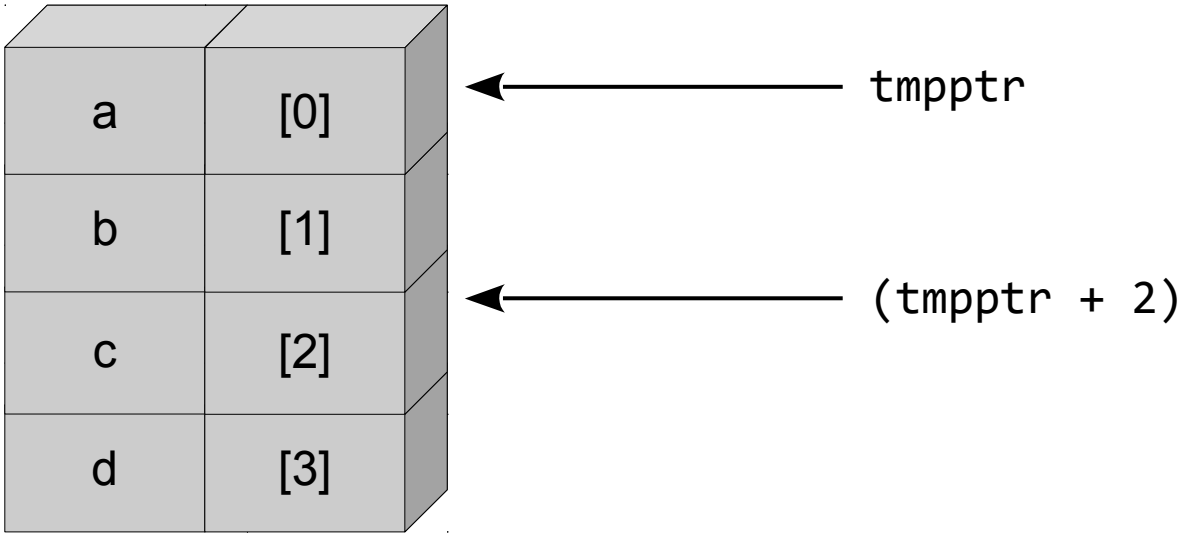
- Das Schlüsselwort `new` gibt den Beginn des angeforderten Speicherbereichs zurück.
- Der Beginn des Speicherbereichs ist die Adresse des ersten Elements in einem Array. Der Zeiger verweist auf das erste Element in einem Array.
- Die Anfangsadresse darf niemals verloren gehen!

Verschiebung des Zeigers

```
index = 2
*(tmpptr + index)  = index;
```

- Der Zeiger wird mit Hilfe eines Ausdrucks verschoben. Der Ausdruck muss geklammert werden.
- Der Dereferenzierungsoperator verweist auf die, mit Hilfe des Ausdrucks berechneten Speicheradresse. An dieser Adresse wird der Wert abgelegt.
- In diesem Beispiel wird die aktuelle Speicheradresse mit (`2 * sizeof(double)`) addiert. Der Zeiger wird nach rechts verschoben.

Grafische Darstellung



Dekrement oder Inkrement nutzen

```
tmpptr = (pointer + (MAX_SIZE – 1));

do{

    cout << "Referenz auf: " << tmpptr << '\n';
    Cout << "… den Wert: " << *tmpptr << '\n';
    tmpptr--;
}while (tmpptr >= pointer);
```

Hinweise

- C++ erlaubt Addition und Subtraktionen von Ganzzahlen zu einem Zeiger.
- Der Zeiger auf die Anfangsadresse sollte nicht überschrieben werden. Andernfalls kann der Speicher nicht mehr freigegeben werden.

Zeiger auf einen konstanten Wert

```
double wert = 5;
const double *constValue = &wert;
```

- Der Wert, auf den der Zeiger verweist ist, konstant.
- Aber die Speicheradresse, die im Zeiger gespeichert ist kann verändert werden.

Neuzuweisung

```
double wert01 = 7.0;
const double * constptr = &wert01;

double wert02 = 5.0;
constptr = &wert02;
```

- Eine Neu-Zuweisung an einen nicht konstanten Zeiger ist möglich.

Fehler: Dereferenzierung des Zeigers

```
double wert01 = 7.0;
const double * constptr = &wert01;

*constptr = *constptr / 2;
```

- Der Wert, auf den der Zeiger verweist darf nicht verändert werden.

Nutzung von const_cast

```
double wert01 = 7.0;
const double * constptr = &wert01;

double *pointer = const_cast <double *>(constptr);
*pointer = *pointer / 2;
```

- Der Operator const_cast ist unabhängig von einem Datentyp definiert.
- Mit Hilfe des Operators kann ein Zeiger auf einen konstanten Wert in einen Zeiger auf einen nicht-konstanten Wert umgewandelt werden.
- Der Schreibschutz auf ein Wert wird aufgehoben.

Der Zeiger … wird umgewandelt

```
double wert01 = 7.0;
const double * constptr = &wert01;

double *pointer = const_cast <double *>(constptr);
*pointer = *pointer / 2;
```

- In runden Klammern wird der umzuwandelnde Zeiger angegeben.
- In diesem Beispiel wird ein Zeiger auf einen konstanten double-Wert umgewandelt.

… in einem Zeiger von dem ...

```
double wert01 = 7.0;
const double * constptr = &wert01;

double *pointer = const_cast <double *>(constptr);
*pointer = *pointer / 2;
```

- In spitzen Klammern wird der Datentyp des Zeigers angegeben, in dem umgewandelt werden soll. In diesem Beispiel wird der Zeiger in einem Zeiger, der auf einen double-Wert verweist umgewandelt.
- Der Datentyp sollte dem Zeiger links vom Gleichheitszeichen entsprechen oder entsprechend interpretiert werden.

Konstanter Zeiger

```
double wert01 = 7.0;
double * const constptr = &wert01;
```

- Der Zeiger ist konstant.
- Die, in dem Zeiger gespeicherte Adresse kann nicht verändert werden.

Dereferenzierung des Zeigers

```
double wert01 = 7.0;
double * const constptr = &wert01;

wert01 = *constptr / 2;
```

- Mit Hilfe des Dereferenzierungsoperators wird auf den Wert an der angegebenen Speicherstelle verwiesen.
- Der Wert, auf den verwiesen wird, kann genutzt und verändert werden.

Fehler: Neuzuweisung

```
double wert01 = 7.0;
double * const constptr = &wert01;

int wert02 = 2;
constptr = &wert02;
```

- Eine Neu-Zuweisung an einen konstanten Zeiger ist nicht möglich. Das Programm bricht mit einem Fehler ab.
- Zeiger auf das erste Element eines Arrays sollten immer als konstant definiert werden.