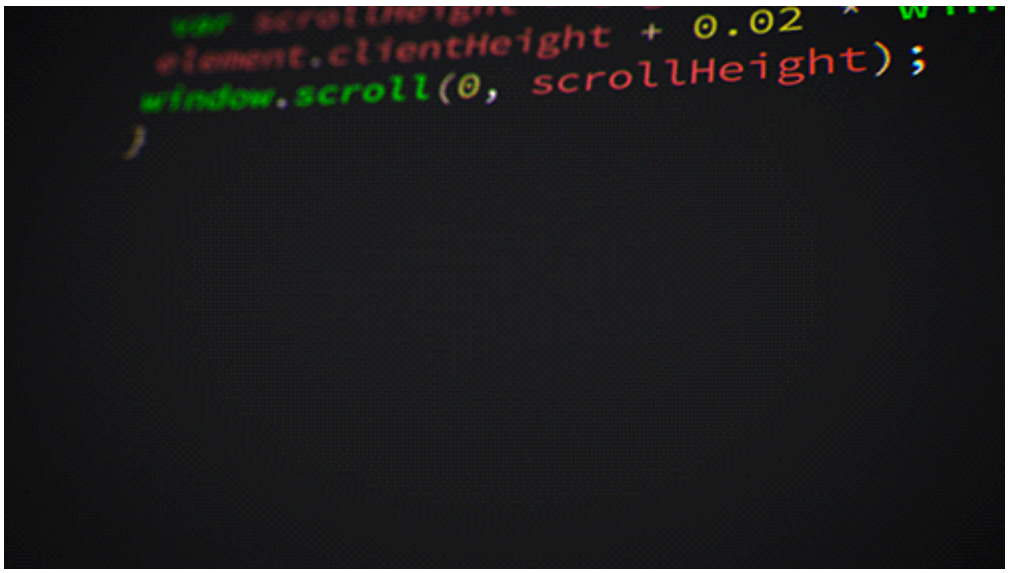
Grundlagen der Sprache C++

Parameter	Kursinformationen
Veranstaltung:	`Prozedurale Programmierung / Einführung in die Informatik / Erhebung, Analyse und Visualisierung digitaler Daten`
Semester	`Wintersemester 2025/26`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	`Array, Zeiger und Referenzen`
Link auf Repository:	https://github.com/TUBAF-IfI-LiaScript/VL_EAVD/blob/master/03_ArrayZeigerReferenzen.md
Autoren	Sebastian Zug & André Dietrich & Galina Rudolf & Copilot & Anna



Fragen an die heutige Veranstaltung ...

- Was ist ein Array?
- Wie können zwei Arrays verglichen werden?
- Erklären Sie die Idee des Zeigers in der Programmiersprache C/C++.
- Welche Gefahr besteht bei der Initialisierung von Zeigern?
- Was ist ein `NULL`-Zeiger und wozu wird er verwendet?

Reflexion Ihrer Fragen / Rückmeldungen

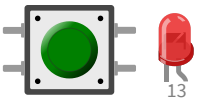
Zur Erinnerung ... Wettstreit zur partizipativen Materialentwicklung mit den Informatikern ...

Preis für das aktivste Auditorium

Format	Informatik Studierende
Verbesserungsvorschlag	2
Fragen	2
generelle Hinweise	0

Wie weit waren wir gekommen?

Aufgabe: Die LED blinkt im Beispiel 10 mal. Integrieren Sie eine Abbruchbedingung für diese Schleife, wenn der grüne Button gedrückt wird. Welches Problem sehen Sie?



2

13

ButtonLogic.cpp

```
1  void setup() {
2    pinMode(2, INPUT);       // Button grün
3    pinMode(13, OUTPUT);
4
5    while (digitalRead(2) == LOW) {     // Kopfgesteuerte Schleife für
6      delay(10);                        // warten auf den Buttondruck
7    }
8
9    for (int i = 0; i<10; i++) {        // 10 mal LED blinken - Wiederh
10     digitalWrite(13, HIGH);           // -schleife
11     delay(500);
12     digitalWrite(13, LOW);
13     delay(500);
14   }
15 }
16
17 void loop() {
18 }
```

```
Sketch uses 1110 bytes (3%) of program storage space. Maximum is 32256
bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes
for local variables. Maximum is 2048 bytes.


Sketch uses 1110 bytes (3%) of program storage space. Maximum is 32256
bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes
for local variables. Maximum is 2048 bytes.
```

Arrays

Bisher umfassten unsere Variablen einzelne Skalare. Arrays erweitern das Spektrum um Folgen von Werten, die in n-Dimensionen aufgestellt werden können. Ein Array ist eine geordnete Folge von Werten des gleichen Datentyps. Die Deklaration erfolgt in folgender Anweisung:

```
Datentyp Variablenname[Anzahl_der_Elemente];
```

```
int a[6];
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

```
Datentyp Variablenname[Anzahl_der_Elemente_Dim0][Anzahl_der_Elemente_Dim1];
```

```
int a[3][5];
```

	Spalten				
Zeilen	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

Achtung 1: Im hier beschriebenen Format muss zum Zeitpunkt der Übersetzung die Größe des Arrays (Anzahl_der_Elemente) bekannt sein.

Achtung 2: Der Variablenname steht nunmehr nicht für einen Wert sondern für die Speicheradresse (Pointer) des ersten Elementes!

Deklaration, Definition, Initialisierung, Zugriff

Initialisierung und genereller Zugriff auf die einzelnen Elemente des Arrays sind über einen Index möglich.

ArrayExample.cpp

```cpp
#include <iostream>

int main(void) {
  int a[3];       // Array aus 3 int Werten
  a[0] = -2;
  a[1] = 5;
  a[2] = 99;
  for (int i=0; i<3; i++){
    std::cout << a[i] << " ";
  }
  std::cout << "\nNur zur Info " << sizeof(a) << "\n";
  std::cout << "Zahl der Elemente " << sizeof(a) / sizeof(int) << "\n
  return 0;
}
```

```
-2 5 99
Nur zur Info 12
Zahl der Elemente 3
-2 5 99
Nur zur Info 12
Zahl der Elemente 3
```

Schauen wir uns das Ganze noch in einer Animation an: PythonTutor

Wie können Arrays noch initialisiert werden:

- vollständig (alle Elemente werden mit einem spezifischen Wert belegt)
- anteilig (einzelne Elemente werden mit spezifischen Werten gefüllt, der rest mit 0)

ArrayExample.cpp

```cpp
#include <iostream>
#include <iomanip>

int main(void) {
  int a[] = {5, 2, 2, 5, 6};
  float b[5] = {1.01};
  for (int i=0; i<5; i++) {
    std::cout << i << "  "  << a[i] << " / " << std::fixed << b[i] <<
  }
  return 0;
}
```

```
0  5 / 1.010000
1  2 / 0.000000
2  2 / 0.000000
3  5 / 0.000000
4  6 / 0.000000
0  5 / 1.010000
1  2 / 0.000000
2  2 / 0.000000
3  5 / 0.000000
4  6 / 0.000000
```

Und wie bestimme ich den erforderlichen Speicherbedarf bzw. die Größe des Arrays?

ArrayExample.cpp

```cpp
#include <iostream>

int main(void) {
  int a[3];
  std::cout << "Speicherplatz [Byte] " << sizeof(a) << "\n";
  std::cout << "Zahl der Elemente " << sizeof(a)/sizeof(int) << "\n";
  return 0;
}
```

```
Speicherplatz [Byte] 12
Zahl der Elemente 3
Speicherplatz [Byte] 12
Zahl der Elemente 3
```

Fehlerquelle Nummer 1 - out of range

ArrayExample.cpp

```cpp
#include <iostream>

int main(void) {
  int a[] = {-2, 5, 99};
  for (int i=0; i<=3; i++)
    std::cout << a[i] << " ";
  std::cout << "\n";
  return 0;
}
```

```
-2 5 99 1348062464
-2 5 99 -501577216
```

Anwendung eines eindimesionalen Arrays

Schreiben Sie ein Programm, das zwei Vektoren miteinander vergleicht. Warum ist die intuitive Lösung `a == b` nicht korrekt, wenn `a` und `b` arrays sind?

ArrayExample.cpp

```cpp
#include <iostream>

int main(void) {
  int a[] = {0, 1, 2, 4, 3, 5, 6, 7, 8, 9};
  int b[10];
  for (int i=0; i<10; i++)  // "Befüllen" des Arrays b
    b[i] = i;

  // Wir haben jetzt zwei Arrays a und b vorzuliegen und wolle diese
  // parweise vergleichen
  for (int i=0; i<10; i++)
    if (a[i] != b[i])
      std::cout << "An Stelle " << i << " unterscheiden sich die Arra
        ;
  return 0;
  }
```

```
An Stelle 3 unterscheiden sich die Arrays!
An Stelle 4 unterscheiden sich die Arrays!
An Stelle 3 unterscheiden sich die Arrays!
An Stelle 4 unterscheiden sich die Arrays!
```

Welche Verbesserungsmöglichkeiten sehen Sie bei dem Programm?

Mehrdimensionale Arrays

Deklaration:

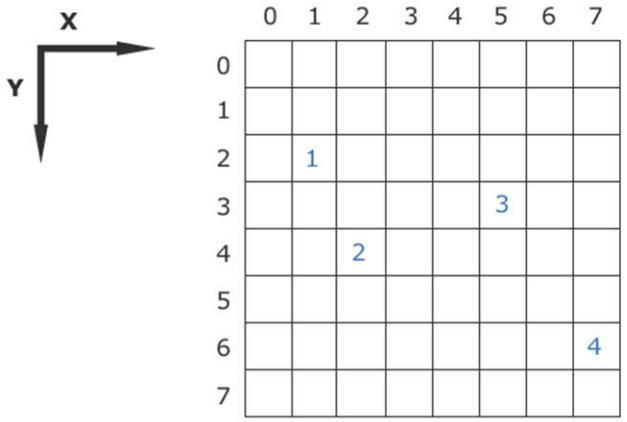
```cpp
int Matrix[4][5];    /* Zweidimensional - 4 Zeilen x 5 Spalten */
```

Deklaration mit einer sofortigen Initialisierung aller bzw. einiger Elemente:

```cpp
int Matrix[4][5] = {{1,  2,  3,  4,  5},
                    {6,  7,  8,  9,  10},
```

```
                    {11, 12, 13, 14, 15},
                    {16, 17, 18, 19, 20}};

int Matrix[4][4] = {{1,},
                    {1, 1},
                    {1, 1, 1},
                    {1, 1, 1, 1}};

int Matrix[4][4] = {1,2,3,4,5,6,7,8};
```

Initialisierung eines n-dimensionalen Arrays:



Darstellung der Matrixinhalte für das nachfolgende Codebeispiel - Aufgabe aus [C-Kurs](http://www.c-howto.de/tutorial/arrays-felder/zweidimensionale-felder/

Praxisbezug: Solche zweidimensionalen Arrays werden oft genutzt, um Karten oder Spielfelder zu modellieren – zum Beispiel für ein Labyrinth, ein Brettspiel oder eine Landkarte. Die Zahlen im Array stehen dann für verschiedene Objekte oder Besonderheiten: - 0 = freies Feld - 1 = Startpunkt - 2 = Hindernis - 3 = Schatz - 4 = Ziel So kann man das Feld auswerten, anzeigen oder für eine Spiellogik weiterverarbeiten.

nDimArray.cpp

```cpp
#include <iostream>

int main(void) {
  // Initiallisierung
  int brett[8][8] = {0};
  // Zuweisung
  brett[2][1] = 1;
  brett[4][2] = 2;
  brett[3][5] = 3;
  brett[6][7] = 4;
  // Ausgabe
  int i, j;
  // Schleife fuer Zeilen, Y-Achse
  for (i=0; i<8; i++) {
    // Schleife fuer Spalten, X-Achse
    for (j=0; j<8; j++) {
      std::cout << brett[i][j] << " ";
    }
    std::cout << "\n";
  }
  return 0;
}
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 3 0 0
0 0 2 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 4
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 3 0 0
0 0 2 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 4
0 0 0 0 0 0 0 0
```

Anwendung eines zweidimesionalen Arrays

Elementwese Addition zweier Matrizen

Addition.cpp

```cpp
#include <iostream>

int main(void)
{
  int A[2][3] = {{1, 2, 3}, {4, 5, 6}};
  int B[2][3] = {{10, 20, 30}, {40, 50, 60}};
  int C[2][3];
  int i, j;
  for (i=0; i<2; i++)
    for (j=0; j<3; j++)
      C[i][j] = A[i][j] + B[i][j];
  for (i=0; i<2; i++)
  {
    for (j=0; j<3; j++)
      std::cout << C[i][j] << "\t";
    std::cout << "\n";
  }
  return 0;
}
```

```
11      22      33
44      55      66
11      22      33
44      55      66
```

Weiteres Beispiel: Lösung eines Gleichungssystem mit dem Gausschen Elimnationsverfahren Link

Merke: Größere Daten in Arrays abzulegen ist in der Regel effizienter als einzelne Variablen zu verwenden. Die Verwendung von Arrays ist aber nicht immer die beste Lösung. Prüfen Sie höherabstraktere Formate wie Listen oder Vektoren!

Sonderfall Zeichenketten / Strings

Folgen von Zeichen werden in C/C++ durch Arrays mit Elementen vom Datentyp `char` repräsentiert. Die Zeichenfolgen werden mit `\0` abgeschlossen.

stringarray.cpp

```cpp
#include <iostream>

int main(void) {
  std::cout << "Diese Form eines Strings haben wir bereits mehrfach benutzt!\n";
  ////////////////////////////////////////////////////////////////////////////
  char a[] = "Ich bin ein char Array!";  // Der Compiler fügt das \0 automatisch ein!
  if (a[23] == '\0') {
    std::cout << "char Array Abschluss in a gefunden!\n";
  }

  std::cout << "->" << a << "<-\n";
  char b[] = { 'H', 'a', 'l', 'l', 'o', ' ',
               'F', 'r', 'e', 'i', 'b', 'e', 'r', 'g', '\0' };
  std::cout << "->" << b << "<-\n";
  char c[] = "Noch eine \0Moeglichkeit";
  std::cout << "->" << c << "<-\n";
  char d[] = { 69, 65, 86, 68, 32, 50, 48, 50, 52, 0 };
  std::cout << "->" << d << "<-\n";

  return 0;
}
```

```
Diese Form eines Strings haben wir bereits mehrfach benutzt!
char Array Abschluss in a gefunden!
->Ich bin ein char Array!<-
->Hallo Freiberg<-
->Noch eine <-
->EAVD 2024<-
Diese Form eines Strings haben wir bereits mehrfach benutzt!
char Array Abschluss in a gefunden!
->Ich bin ein char Array!<-
->Hallo Freiberg<-
->Noch eine <-
->EAVD 2024<-
```

C++ implementiert einen separaten string-Datentyp (Klasse), die ein deutliche komfortableren Umgang mit Texten erlaubt. Beim Anlegen eines solchen muss nicht angegeben werden, wie viele Zeichen reserviert werden sollen. Zudem könenn Strings einfach zuweisen und vergleichen werden, wie es für andere Datentypen üblich ist. Die C const char * Mechanismen funktionieren aber auch hier.

stringarray.cpp

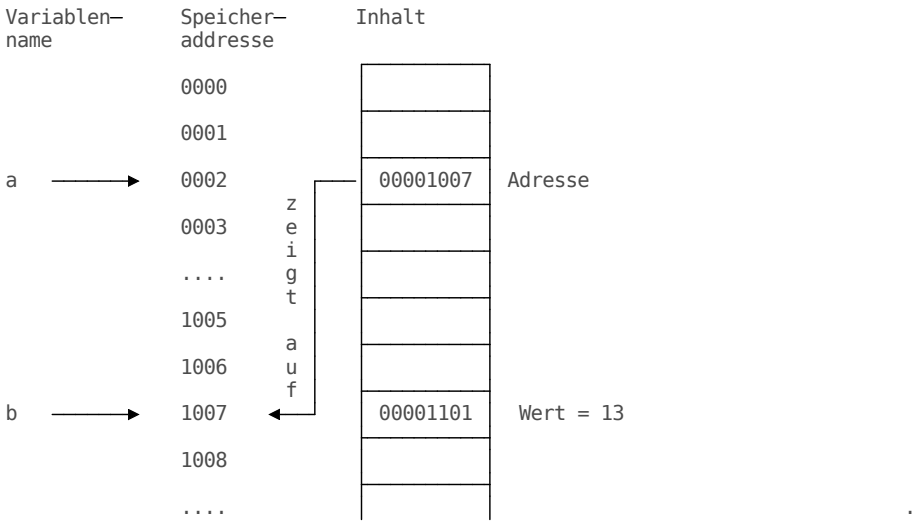
```cpp
#include <iostream>
#include <string>      // Header für string Klasse

int main(void) {
  std::string hanna = "Hanna";
  std::string anna = "Anna";
  std::string alleBeide = anna + " + " + hanna;
  std::cout << "Hallo: " << alleBeide << "\n";

  int res = anna.compare(hanna);
  if (res == 0)
    std::cout << "Both input strings are equal.\n";
  else if (res < 0)
    std::cout << "String 1 is smaller as compared to String 2\n.";
  else
    std::cout << "String 1 is greater as compared to String 2\n.";

  return 0;
}
```

```
Hallo: Anna + Hanna
String 1 is smaller as compared to String 2
.Hallo: Anna + Hanna
String 1 is smaller as compared to String 2
.
```

Grundkonzept Zeiger

Bisher umfassten unserer Variablen als Datencontainer Zahlen oder Buchstaben. Das Konzept des Zeigers (englisch Pointer) erweitert das Spektrum der Inhalte auf Adressen.

An dieser Adresse können entweder Daten, wie Variablen oder Objekte, aber auch Programmcodes (Anweisungen) stehen. Durch Dereferenzierung des Zeigers ist es möglich, auf die Daten oder den Code zuzugreifen.



Welche Vorteile ergeben sich aus der Nutzung von Zeigern, bzw. welche Programmiertechniken lassen sich realisieren:

- dynamische Verwaltung von Speicherbereichen,
- Übergabe von Datenobjekten an Funktionen via "call-by-reference",
- Übergabe von Funktionen als Argumente an andere Funktionen,
- Umsetzung rekursiver Datenstrukturen wie Listen und Bäume.

> An dieser Stelle sei bereits erwähnt, dass die Übergabe der "call-by-reference"-Parameter via Reference ist ebenfalls möglich und einfacher in der Handhabung.

Definition von Zeigern

Die Definition eines Zeigers besteht aus dem Datentyp des Zeigers und dem gewünschten Zeigernamen. Der Datentyp eines Zeigers besteht wiederum aus dem Datentyp des Werts auf den gezeigt wird sowie aus einem Asterisk. Ein Datentyp eines Zeigers wäre also z. B. `double*`.

```
/* kann eine Adresse aufnehmen, die auf einen Wert vom Typ Integer zeigt */
int* zeiger1;
/* das Leerzeichen kann sich vor oder nach dem Stern befinden */
float *zeiger2;
/* ebenfalls möglich */
char * zeiger3;
/* Definition von zwei Zeigern */
int *zeiger4, *zeiger5;
/* Definition eines Zeigers und einer Variablen vom Typ Integer */
int *zeiger6, ganzzahl;
```

Initialisierung

> **Merke:** Zeiger sollten vor der Verwendung initialisiert werden.

Der Zeiger kann initialisiert werden durch die Zuweisung:

- der Adresse einer Variable, wobei die Adresse mit Hilfe des Adressoperators `&` ermittelt wird,
- eines Arrays,
- eines weiteren Zeigers oder
- des Wertes von `NULL`.

<!--- TODO: Sollten wir im C++ Kontext nicht mind. auf `nullptr` verweisen? -->

PointerExamples.cpp

```cpp
#include <iostream>

int main(void) {
  int a = 0;
  int * ptr_a = &a;        /* mit Adressoperator */

  int feld[10];
  int * ptr_feld = feld;   /* mit Array */

  int * ptr_b = ptr_a;     /* mit weiterem Zeiger */

  int * ptr_Null = NULL;   /* mit NULL */

  std::cout << "Pointer ptr_a    " << ptr_a << "\n";
  std::cout << "Pointer ptr_feld " << ptr_feld << "\n";
  std::cout << "Pointer ptr_b    " << ptr_b << "\n";
  std::cout << "Pointer ptr_Null " << ptr_Null << "\n";
  return 0;
}
```

```
Pointer ptr_a    0x7ffd7937bc8c
Pointer ptr_feld 0x7ffd7937bcb0
Pointer ptr_b    0x7ffd7937bc8c
Pointer ptr_Null 0
Pointer ptr_a    0x7ffecfad749c
Pointer ptr_feld 0x7ffecfad74c0
Pointer ptr_b    0x7ffecfad749c
Pointer ptr_Null 0
```

Die konkrete Zuordnung einer Variablen im Speicher wird durch den Compiler und das Betriebssystem bestimmt. Entsprechend kann die Adresse einer Variablen nicht durch den Programmierer festgelegt werden. Ohne Manipulationen ist die Adresse einer Variablen über die gesamte Laufzeit des Programms unveränderlich, ist aber bei mehrmaligen Programmstarts unterschiedlich.

In den Ausgaben von Pointer wird dann eine hexadezimale Adresse ausgegeben.

Zeiger können mit dem "Wert" `NULL` als ungültig markiert werden. Eine Dereferenzierung führt dann meistens zu einem Laufzeitfehler nebst Programmabbruch. NULL ist ein Macro und wird in mehreren Header-Dateien definiert (mindestens in `<cstddef>` (`stddef.h`)). Die Definition ist vom Standard implementierungsabhängig vorgegeben und vom Compilerhersteller passend implementiert, z. B.

```
#define NULL 0
#define NULL 0L
#define NULL (void *) 0
```

Und umgekehrt, wie erhalten wir den Wert, auf den der Pointer zeigt? Hierfür benötigen wir den *Inhaltsoperator* `*` .

DereferencingPointers.cpp

```
1  #include <iostream>
2
3  int main(void) {
4    int a = 15;
5    int * ptr_a = &a;
6    std::cout << "Wert von a                    " << a << "\n";
7    std::cout << "Pointer ptr_a                 " << ptr_a << "\n";
8    std::cout << "Wert hinter dem Pointer ptr_a  " << *ptr_a << "\n";
9    *ptr_a = 10;
10   std::cout << "Wert von a                    " << a << "\n";
11   std::cout << "Wert hinter dem Pointer ptr_a  " << *ptr_a << "\n";
12   return 0;
13 }
```

```
Wert von a                    15
Pointer ptr_a                 0x7fff5999869c
Wert hinter dem Pointer ptr_a  15
Wert von a                    10
Wert hinter dem Pointer ptr_a  10
Wert von a                    15
Pointer ptr_a                 0x7ffd89513f9c
Wert hinter dem Pointer ptr_a  15
Wert von a                    10
Wert hinter dem Pointer ptr_a  10
```

Schauen wir wiederum auf eine grafische Darstellung PythonTutor

Fehlerquellen

Fehlender Adressoperator bei der Zuweisung

PointerFailuresI.cpp

```cpp
#include <iostream>

int main(void) {
  int a = 5;
  int * ptr_a;
  ptr_a = a;
  std::cout << "Pointer ptr_a                    " << ptr_a << "\n";
  std::cout << "Wert hinter dem Pointer ptr_a  " << *ptr_a << "\n";
  std::cout << "Aus Maus!\n";
  return 0;
}
```

```
main.cpp: In function ‘int main()’:
main.cpp:6:11: error: invalid conversion from ‘int’ to ‘int*’ [-
fpermissive]
    6 |   ptr_a = a;
      |           ^
      |           |
      |           int
main.cpp: In function ‘int main()’:
main.cpp:6:11: error: invalid conversion from ‘int’ to ‘int*’ [-
fpermissive]
    6 |   ptr_a = a;
      |           ^
      |           |
      |           int
```

Fehlender Dereferenzierungsoperator beim Zugriff

PointerFailuresII.cpp

```cpp
#include <iostream>

int main(void)
{
  int a = 5;
  int * ptr_a = &a;
  std::cout << "Pointer ptr_a                    " << ptr_a << "\n";
  std::cout << "Wert hinter dem Pointer ptr_a  " << ptr_a << "\n";
  std::cout << "Aus Maus!\n";
  return 0;
}
```

```
Pointer ptr_a                  0x7ffff7a0a07c
Wert hinter dem Pointer ptr_a  0x7ffff7a0a07c
Aus Maus!
Pointer ptr_a                  0x7ffddefceb9c
Wert hinter dem Pointer ptr_a  0x7ffddefceb9c
Aus Maus!
```

Uninitialierte Pointer zeigen "irgendwo ins nirgendwo"!

PointerFailuresIII.cpp

```cpp
#include <iostream>

int main(void) {
  int * ptr_a;
  // korrekte Initalisierung
  // int * ptr_a = NULL;

  // ... hier passiert irgendwas

  *ptr_a = 10;
  // Prüfung auf gültige Adresse
  // if (ptr_a != NULL) *ptr_a = 10;
  std::cout << "Pointer ptr_a                    " << ptr_a << "\n";
  std::cout << "Wert hinter dem Pointer ptr_a  " << *ptr_a << "\n";
  std::cout << "Aus Maus!\n";
  return 0;
}
```

```
main.cpp: In function ‘int main()’:
main.cpp:10:10: warning: ‘ptr_a’ is used uninitialized [-
Wuninitialized]
   10 |    *ptr_a = 10;
      |    ~~~~~~~^~~~

main.cpp: In function ‘int main()’:
main.cpp:10:10: warning: ‘ptr_a’ is used uninitialized [-
Wuninitialized]
   10 |    *ptr_a = 10;
      |    ~~~~~~~^~~~
```

Referenz

Eine Referenz ist wie ein zweiter Name für eine Variable. Sie zeigt immer auf ein existierendes Objekt und kann nicht „ins Leere“ zeigen.

> Referenzen existieren nur in C++. In C gibt es nur Zeiger.

Im Gegensatz zu Zeigern (Pointer) ist die Handhabung für Einsteiger viel einfacher:

	Zeiger (Pointer)	**Referenz**
Kann NULL sein	Ja	Nein
Muss initialisiert werden	Nein	Ja
Dereferenzierung nötig?	Ja (`*p`)	Nein
Nachträglich umsetzbar?	Ja (kann auf andere Variable zeigen)	Nein (bleibt immer auf das gleiche Objekt)
Speicherfreigabe nötig?	Ja (bei `new`)	Nein

Beispiel:

// referenzen.cpp

```cpp
#include <iostream>

int main(void) {
  int  a = 1;       // Variable
  int &r = a;       // Referenz auf a
  int *p = &a;      // Zeiger auf a

  std::cout << "a: " << a << " r: " << r << " *p: " << *p << std::end
  r = 5;            // Ändert auch a
  std::cout << "a: " << a << " r: " << r << std::endl;
  *p = 10;          // Ändert auch a
  std::cout << "a: " << a << " r: " << r << std::endl;
}
```

```
a: 1 r: 1 *p: 1
a: 5 r: 5
a: 10 r: 10
```

Merke: - Referenzen sind ideal für Funktionsparameter, wenn man Werte verändern möchte, ohne Kopien zu erzeugen. - Zeiger braucht man, wenn man „ins Leere“ zeigen oder dynamisch Speicher verwalten will.

Typische Anwendungen für Referenzen: - Übergabe an Funktionen („call by reference“) - Vermeidung von Kopien großer Objekte - Spezielle Memberfunktionen (z. B. Copy-Konstruktor)

Achtung: Referenzen sind nicht geeignet, um dynamisch Speicher zu verwalten oder auf „nichts“ zu zeigen.

Beispiel der Woche

Gegeben ist ein Array, das eine sortierte Reihung von Ganzzahlen umfasst. Geben Sie alle Paare von Einträgen zurück, die in der Summe 18 ergeben.

Die intuitive Lösung entwirft einen kreuzweisen Vergleich aller sinnvollen Kombinationen der n Einträge im Array. Dafür müssen wir $(n-1)^2/2$ Kombinationen bilden.

	1	**2**	**5**	**7**	**9**	**10**	**12**	**13**	**16**	**17**	**18**	**21**	**25**
1	x									18			
2	x	x							18				
5	x	x	x					18					
7	x	x	x	x									
9	x	x	x	x	x								
10	x	x	x	x	x	x							
12	x	x	x	x	x	x	x						
13	x	x	x	x	x	x	x	x					
16	x	x	x	x	x	x	x	x	x				
17	x	x	x	x	x	x	x	x	x	x			
18	x	x	x	x	x	x	x	x	x	x	x		
21	x	x	x	x	x	x	x	x	x	x	x	x	
25	x	x	x	x	x	x	x	x	x	x	x	x	x

Haben Sie eine bessere Idee?

Pairing.cpp

```cpp
#include <iostream>
#define ZIELWERT 18

int main(void) {
  int a[] = {1, 2, 5, 7, 9, 10, 12, 13, 16, 17, 18, 21, 25};
  int i_left = 0;
  int i_right = 12;

  std::cout << "Value left " << a[i_left] << " right " << a[i_right]
    ----------------------\n";
  do {
    std::cout << "Value left " << a[i_left] << " right " << a[i_right
    if (a[i_right] + a[i_left] == ZIELWERT)
      std::cout << " -> TREFFER";
    std::cout << "\n";

    if (a[i_right] + a[i_left] >= ZIELWERT)
      i_right--;
    else
      i_left++;
  } while (i_right != i_left);

  return 0;
}
```

```
Value left 1 right 25
----------------------
Value left 1 right 25
Value left 1 right 21
Value left 1 right 18
Value left 1 right 17 -> TREFFER
Value left 1 right 16
Value left 2 right 16 -> TREFFER
Value left 2 right 13
Value left 5 right 13 -> TREFFER
Value left 5 right 12
Value left 7 right 12
Value left 7 right 10
Value left 9 right 10
Value left 1 right 25
----------------------
Value left 1 right 25
Value left 1 right 21
Value left 1 right 18
Value left 1 right 17 -> TREFFER
Value left 1 right 16
Value left 2 right 16 -> TREFFER
Value left 2 right 13
Value left 5 right 13 -> TREFFER
Value left 5 right 12
Value left 7 right 12
Value left 7 right 10
Value left 9 right 10
```

Quiz

Arrays

Erstellen Sie ein eindimensionales Array namens `arr`, das 7 Elemente vom Typ `int` enthält.

Erstellen Sie ein zweidimensionales Array namens `arr`, das 3*4 Elemente vom Typ `int` enthält.

Zugriff

Wie lautet die Ausgabe dieses Programms?

```
#include <iostream>

int main(void) {
  float b[5] = {1.0, 4.8, 1.2, 42.0, 99.0};
  std::cout << b[2] << "\n";
  return 0;
}
```

Wie lautet die Ausgabe dieses Programms?

```
#include <iostream>

int main() {
  int a[5] = {5, 8};
  std::cout << a[2] << "\n";
  return 0;
}
```

Mehrdimensionale Arrays

Es existiert ein Array `int A[2][5];`. Setzen Sie `[_____]` gleich 1.

	Spalten			
Zeilen	`a[0][0]`	`a[0][1]`	...	
			`[_____]`	

Durch was muss `[_____]` ersetzt werden damit die Zahl `19` aus `m[4][5]` ausgegeben wird?

```
#include <iostream>

int main() {
  int m[4][5] = {{1, 2, 3, 4, 5},
                 {6, 7, 8, 9, 10},
                 {11,12,13,14,15},
                 {16,17,18,19,20}};
  std::cout << [_____] << "\n";
  return 0;
}
```

?

Wie lautet die Ausgabe dieses Programms?

```
#include <iostream>

int main() {
  int A[2][3] = {{1, 2, 3}, {4, 5, 6}};
  int B[2][3] = {{10, 20, 30}, {40, 50, 60}};

  std::cout << A[1][0] + B[0][1] << "\n";
  return 0;
}
```

Zeichenketten

Durch welche Sequenz werden Zeichenketten abgeschlossen?

Wie lautet die Ausgabe dieses Programms?

```
#include <iostream>

int main() {
  char c[] = "Peter wohnt irgendwo\0 in Freiberg.";
  std::cout << c << "\n";
}
```

Zeiger

Worauf zeigen Zeiger?

- ◯ `chars`
- ◯ Referenzen
- ◯ Speicheraddressen

Definition

Welche der folgenden Definitionen sind möglich?

- [ ] int* z1;
- [ ] float * z2;
- [ ] char *z3;
- [ ] int *z4, *z5;
- [ ] int z6*;
- [ ] int *z7, i;

Initialisierung

Durch welches Zeichen werden Addressen ermittelt?

Wie lautet die Ausgabe dieses Programms?

```
#include <iostream>

int main() {
  int a = 15;
  int *ptr_a = &a;
  std::cout << *ptr_a << "\n";

  return 0;
}
```

- ( ) Die Addresse von `a`
- ( ) 15
- ( ) `NULL`

Dynamische Datenobjekte

Wie häufig kann `delete` auf ein Objekt angewendet werden?

- ◯ 0
- ◯ 1
- ◯ 42
- ◯ Beliebig oft

Wie lautet die Aussage dieses Programms?

```
#include <iostream>

int main() {
  int a = 10;
  int *ptr_a = &a;
  std::cout << ptr_a  << "\n";
  delete ptr_a;
  return 0;
}
```

- ◯ 10
- ◯ Die Addresse von `a`
- ◯ Die Addresse des Zeigers `*ptr_a`
- ◯ Es gibt einen Error

Referenz

Welche der im Beispiel benutzten Variablen ist eine Referenz?

```
#include <iostream>

int main() {
  int a = 10;
  int &b = a;
  int *c = b;
  std::cout << c << "\n";
  return 0;
}
```

- a
- b
- c

Hier ist ein Programm mit Ausgabe vorgegeben. Was müsste statt `[_____]` ausgegeben werden?

```
#include <iostream>

int main(void) {
  int  a = 1;
  int &r = a;

  std::cout << "a: " << &a << " r: " << &r << endl;
}
```

```
a: [_____] r: 0x7ffdddd212fc
```

Hier ist ein Programm mit Ausgabe vorgegeben. Was müsste statt `[_____]` ausgegeben werden?

```
#include <iostream>

int main(void) {
  int  a = 1;
  int &r = a;

  std::cout << "a: " << a << " r: " << r << "\n";
}
```

```
a: 1 r: [_____]
```