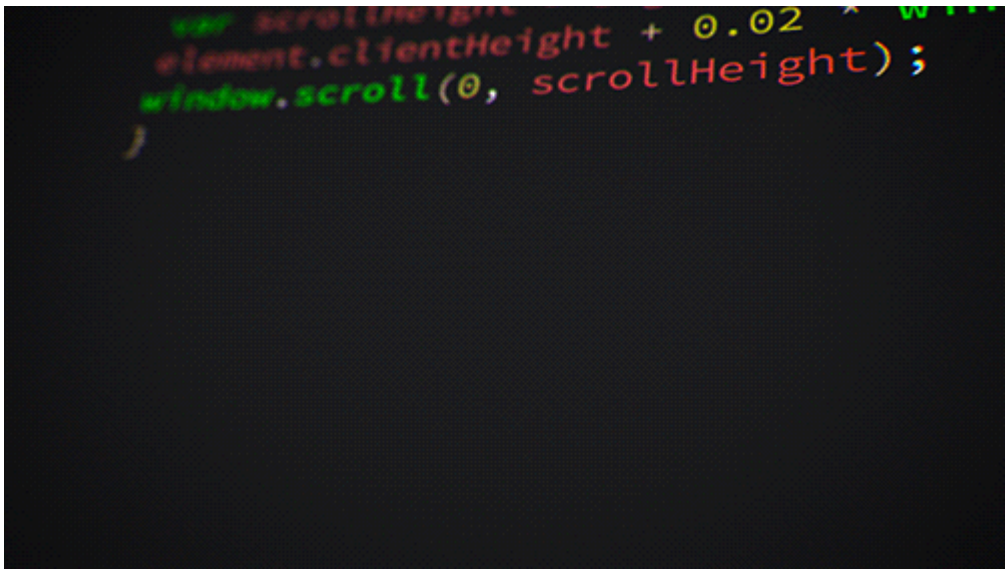
Grundlagen der Sprache C++

Parameter	Kursinformationen
Veranstaltung:	`Prozedurale Programmierung / Einführung in die Informatik`
Semester	`Wintersemester 2022/23`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	`Array, Zeiger und Referenzen`
Link auf Repository:	https://github.com/TUBAF-IfI-LiaScript/VL_ProzeduraleProgrammierung/blob/master/03_ArrayZeigerReferenzen.md
Autoren	Sebastian Zug & André Dietrich & Galina Rudolf

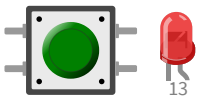


Fragen an die heutige Veranstaltung ...

- 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?

Wie weit waren wir gekommen?

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



ButtonLogic.cpp

```
1  void setup() {
2    pinMode(2, INPUT);
3    pinMode(3, INPUT);
4    pinMode(13, OUTPUT);
5  }
6
7  void loop() {
8    bool a = digitalRead(2);
9    if (a){
10     for (int i = 0; i<10; i++){
11       digitalWrite(13, HIGH);
12       delay(250);
13       digitalWrite(13, LOW);
14       delay(250);
15     }
16   }
17 }
```

```
Sketch uses 1104 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. 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[1][0]	a[1][1]	a[1][2]	a[1][3]
	a[2][0]	a[2][1]	a[2][2]	a[2][3]

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>
using namespace std;

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++)
    cout<<a[i]<<" ";
  cout<<"\nNur zur Info "<< sizeof(a);
  cout<<"\nZahl der Elemente "<< sizeof(a) / sizeof(int);
  return 0;
  }
```

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

C++ (g++ 9.3, C++20 + GNU extensions)

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

int main() {
  int a[3];        // Array aus 3
  a[0] = -2;
  a[1] = 5;
  a[2] = 99;
  for (int i=0; i<3; i++){
    printf("%d ", a[i]);
  }
  cout<<"\nGröße des Arrays "<<si
  cout<<"Zahl der Elemente " << s
  cout<<"Adresse des Arrays " <<
  return 0;
}
```

Edit this code

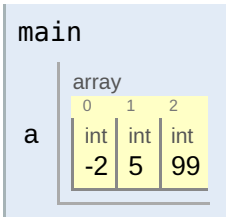
line that just executed

next line to execute

Print output (drag lower right corner to resize)

```
Größe des Arrays 12
Zahl der Elemente 3
Adresse des Arrays 0xfff000bd4
```

Stack Heap



C/C++ details: none [default view]

Wie können Arrays noch initialisiert werden:

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

ArrayExample.cpp

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

int main(void) {
  int a[] = {5, 2, 2, 5, 6};
  float b[5] = {1.0};
  for (int i=0; i<5; i++){
      cout<<a[i]<<" "<< b[i]<<"\n";
    }
  return 0;
}
```

```
5 1
2 0
2 0
5 0
6 0
```

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

ArrayExample.cpp

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

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

```

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

Fehlerquelle Nummer 1 - out of range

ArrayExample.cpp

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

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

```
-2 5 99 49209088
```

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>
using namespace std;

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++)
    b[i]=i;
  for (int i=0; i<10; i++)
    if (a[i]!=b[i])
      cout<<"An Stelle "<<i<<" unterscheiden sich die Vektoren \n";
  return 0;
  }
```

```
An Stelle 3 unterscheiden sich die Vektoren
An Stelle 4 unterscheiden sich die Vektoren
```

Welche Verbesserungsmöglichkeiten sehen Sie bei dem Programm?

Mehrdimensionale Arrays

Deklaration:

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

```
int Matrix[4][5];     /* zweidimensional   4 Zeilen x 5 Spalten */
```

Deklaration mit einer sofortigen Initialisierung aller bzw. einiger Elemente:

```
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 [1]

nDimArray.cpp

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

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++) {
      cout<<brett[i][j]<<" ";
    }
    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
```

[1] Quelle: C-Kurs

Anwendung eines zweidimesionalen Arrays

Elementwese Addition zweier Matrizen

Addition.cpp

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

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++)
      cout<<C[i][j]<<"\t";
    cout<<"\n";
  }
    return 0;
}
```

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

Weiteres Beispiel: Lösung eines Gleichungssystem mit dem Gausschen Elimnationsverfahren Link

Sonderfall Zeichenketten / Strings

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

stringarray.cpp

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

int main(void) {
  cout<<"Diese Form eines Strings haben wir bereits mehrfach benutzt!
  ////////////////////////////////////////////////////////////////////
    /////

  char a[] = "Ich bin ein char Array!";  // Der Compiler fügt das \0
    automatisch ein!
  if (a[23] == '\0'){
    cout<<"char Array Abschluss in a gefunden!";
  }

  cout<<"->"<<a<<"<-\n";
  char b[] = { 'H', 'a', 'l', 'l', 'o', ' ',
               'F', 'r', 'e', 'i', 'b', 'e', 'r', 'g', '\0' };
  cout<<"->"<<b<<"<-\n";
  char c[] = "Noch eine \0Moeglichkeit";
  cout<<"->"<<c<<"<-\n";
  char d[] = { 80, 114, 111, 122, 80, 114, 111, 103, 32, 50, 48,  50,
    0  };
  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 <-
->ProzProg 2022<-
```

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

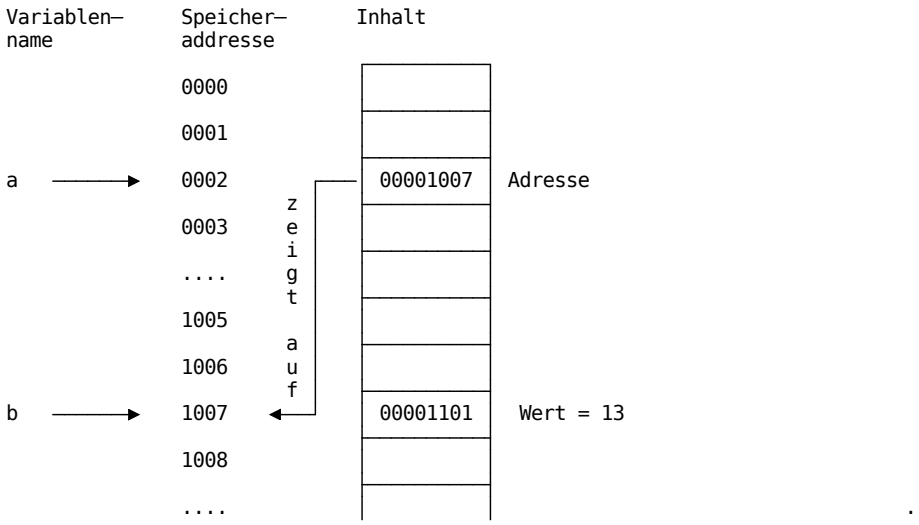
```cpp
#include <iostream>
#include <string>
using namespace std;

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

  int res = anna.compare(hanna);

  if (res == 0)
    cout << "\nBoth the input strings are equal." << endl;
  else if(res < 0)
    cout << "String 1 is smaller as compared to String 2\n.";
    else
    cout<<"String 1 is greater as compared to String 2\n.";

  return EXIT_SUCCESS;
}
```

```
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 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 müssen 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`.

PointerExamples.cpp

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

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 */

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

```
Pointer ptr_a    0x7ffc6358324c
Pointer ptr_feld 0x7ffc63583270
Pointer ptr_b    0x7ffc6358324c
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

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

int main(void)
{
  int a = 15;
  int * ptr_a = &a;
  cout<<"Wert von a                    "<<a<<"\n";
  cout<<"Pointer ptr_a                 "<<ptr_a<<"\n";
  cout<<"Wert hinter dem Pointer ptr_a "<<*ptr_a<<"\n";
  *ptr_a = 10;
  cout<<"Wert von a                    "<<a<<"\n";
  cout<<"Wert hinter dem Pointer ptr_a "<<*ptr_a<<"\n";
  return 0;
}
```

```
Wert von a                    15
Pointer ptr_a                 0x7ffd7558a97c
Wert hinter dem Pointer ptr_a 15
Wert von a                    10
Wert hinter dem Pointer ptr_a 10
```

C (gcc 9.3, C17 + GNU extensions)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6    int a = 15;
7    int * ptr_a = &a;
8    printf("Wert von a
9    printf("Pointer ptr_a
10   printf("Wert hinter dem Pointer
11   *ptr_a = 10;
12   printf("Wert von a
13   printf("Wert hinter dem Pointer
14   return EXIT_SUCCESS;
15 }
```

Edit this code

line that just executed
next line to execute

< Prev | Next >

Print output (drag lower right corner to resize)

Stack | Heap

main
- a: int ?
- ptr_a: pointer to int ?

Note: ? refers to an uninitialized value

C/C++ details: none [default view]

Fehlerquellen

Fehlender Adressoperator bei der Zuweisung

PointerFailuresI.cpp

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

```
main.cpp: In function ‘int main()’:
main.cpp:8:11: error: invalid conversion from ‘int’ to ‘int*’ [-
fpermissive]
    8 |   ptr_a = a;
      |           ^
      |           |
      |           int
```

Fehlender Dereferenzierungsoperator beim Zugriff

PointerFailuresII.cpp

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

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

```
Pointer ptr_a                  0x7ffc5a2bc33c
Wert hinter dem Pointer ptr_a  0x7ffc5a2bc33c
Aus Maus!
```

Uninitialierte Pointer zeigen "irgendwo ins nirgendwo"!

PointerFailuresIII.cpp

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

int main(void)
{
  int * ptr_a;
  *ptr_a = 10;
  // korrekte Initalisierung
  // int * ptr_a = NULL;
  // Prüfung auf gültige Adresse
  // if (ptr_a != NULL) *ptr_a = 10;
  cout<<"Pointer ptr_a                    "<<ptr_a<<"\n";
  cout<<"Wert hinter dem Pointer ptr_a  "<<*ptr_a<<"\n";
  cout<<"Aus Maus!\n";
  return 0;
}
```

```
main.cpp: In function ‘int main()’:
main.cpp:7:10: warning: ‘ptr_a’ is used uninitialized [-Wuninitialized]
    7 |   *ptr_a = 10;
      |   ~~~~~~~^~~~
main.cpp:6:9: note: ‘ptr_a’ was declared here
    6 |   int * ptr_a;
      |         ^~~~~
```

Dynamische Datenobjekte

C++ bietet die Möglichkeit den Speicherplatz für eine Variable zur Laufzeit zur Verfügung zu stellen. Mit `new`-Operator wird der Speicherplatz bereit gestellt und mit `delete`-Operator (`delete[]`) wieder freigegeben.

`new` erkennt die benötigte Speichermenge am angegebenen Datentyp und reserviert für die Variable auf dem Heap die entsperechde Byte-Menge.

new.cpp

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

int main(void)
{
  int * ptr_a;
  ptr_a=new int;
  *ptr_a = 10;
  cout<<"Pointer ptr_a                    "<<ptr_a<<"\n";
  cout<<"Wert hinter dem Pointer ptr_a  "<<*ptr_a<<"\n";
  cout<<"Aus Maus!\n";
  delete ptr_a;
  return 0;
}
```

```
Pointer ptr_a                  0x55729954deb0
Wert hinter dem Pointer ptr_a  10
Aus Maus!
```

newArray.cpp

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

int main(void)
{
  int * ptr_a;
  ptr_a=new int[3];
  ptr_a[0] = ptr_a[1] = ptr_a[2] = 42;
  cout<<"Werte hinter dem Pointer ptr_a:  ";
  for (int i=0;i<3;i++) cout<<ptr_a[i]<<" ";
  cout<<"\n";
  cout<<"Aus Maus!\n";
  delete[] ptr_a;
  return 0;
}
```

```
Werte hinter dem Pointer ptr_a:  42 42 42
Aus Maus!
```

- `delete` daft nur einmal auf ein Objekt angewendet werden
- `delete` daft ausschließlich auf mit new angelegte Objekte oder NULL-Pointer angewandt werden
- Nach der Verwendung von `delete` ist das Objekt *undefiniert* (nicht gleich NULL)

> **Merke:** Die Verwendung von Zeigern kann zur unerwünschten Speicherfragmentierung und die Programmierfehler zu den Programmabstürzen und Speicherlecks führen. *Intelligente* Zeiger stellen sicher, dass Programme frei von Arbeitsspeicher- und Ressourcenverlusten sind.

Referenz

Eine Referenz ist eine Datentyp, der Verweis (Aliasnamen) auf ein Objekt liefert und ist genau wie eine Variable zu benutzen ist. Der Vorteil der Referenzen gegenüber den Zeigern besteht in der einfachen Nutzung:

- Dereferenzierung ist nicht notwendig, der Compiler löst die Referenz selbst auf
- Freigabe ist ebenfalls nicht notwendig

> **Merke:** Auch Referenzen müssen vor der Verwendung initialisiert werden. Eine Referenz bezieht sich immer auf ein existierendes Objekt, sie kann nie NULL sein

referenzen.cpp

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

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

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

```
a: 1 r: 1
a: 0x7fffb8d30ffc r: 0x7fffb8d30ffc
```

Die Referenzen werden verwendet:

- zur "call bei reference"-Parameterübergabe
- zur Optimierung des Programms, um Kopien von Objekten zu vermeiden
- in speziellen Memberfunktionen, wie Copy-Konstruktor und Zuweisungsoperator
- als sogenannte universelle Referenz (engl.: universal reference), die bei Templates einen beliebigen Parametertyp repräsentiert.

Achtung: Zur dynamischen Verwaltung von Speicherbereichen sind Referenzen nicht geeignet.

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>
using namespace std;
#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;
  cout<<"Value left "<<a[i_left]<<" right "<<a[i_right]<<"\n
    ----------------------\n";
  do{
    cout<<"Value left "<<a[i_left]<<" right "<<a[i_right];
    if (a[i_right] + a[i_left] == ZIELWERT){
      cout<<" -> TREFFER";
    }
    cout<<"\n";
    if (a[i_right] + a[i_left] >= ZIELWERT) i_right--;
    else i_left++;
  }while (i_right != i_left);
  return 0;s
}
```

```
main.cpp: In function ‘int main()’:
main.cpp:20:12: error: ‘s’ was not declared in this scope
   20 |   return 0;s
      |            ^
```

<!--START_SKIP_IN_PDF-->

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>
using namespace std;

int main(void) {
  float b[5] = {1.0, 4.8, 1.2, 42.0, 99.0};
  cout << b[2];

  return 0;
}
```

Wie lautet die Ausgabe dieses Programms?

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

int main(){
  int a[5] = {5, 8};
  cout << a[2];
  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>
using namespace std;

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}};
  cout << [_____]
  return 0;
}
```

Wie lautet die Ausgabe dieses Programms?

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

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

  cout << A[1][0] + B[0][1];
  return 0;
}
```

Zeichenketten

Durch welche Sequenz werden Zeichenketten abgeschlossen?

Wie lautet die Ausgabe dieses Programms?

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

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

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>
using namespace std;

int main(){
  int a = 15;
  int *ptr_a = &a;
  cout << *ptr_a;

  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>
using namespace std;

int main(){
  int a = 10;
  int *ptr_a = &a;
  cout << ptr_a;
  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>
using namespace std;

int main(){
  int a = 10;
  int &b = a;
  int *c = b;
  cout << c;
  return 0;
}
```

- ◯ a
- ◯ b
- ◯ c

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

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

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

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

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

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

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

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

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

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

<!--END*SKIP*IN_PDF⟶