Einführung Pointer

C-Kurs 2013, 2. Vorlesung

Nico nico@freitagsrunde.org
Andy andrew@freitagsrunde.org

`http://wiki.freitagsrunde.org`

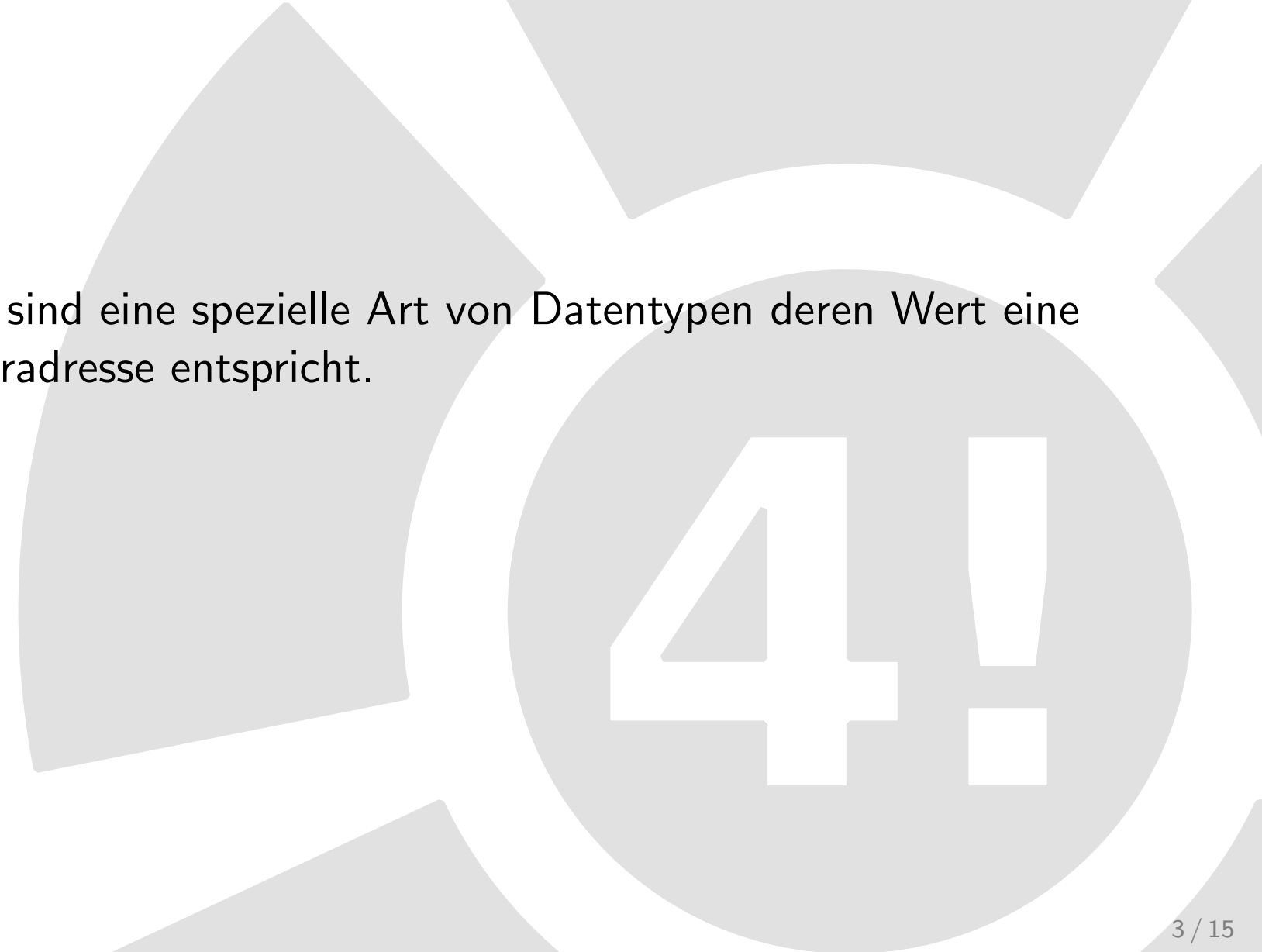
10. September 2013

Speicheadressierung

- Arbeitsspeicher ist in Bereiche unterteilt, die einzeln adressierbar sind
- der Wert einer Variable entspricht einem Wert an einer bestimmten Adresse

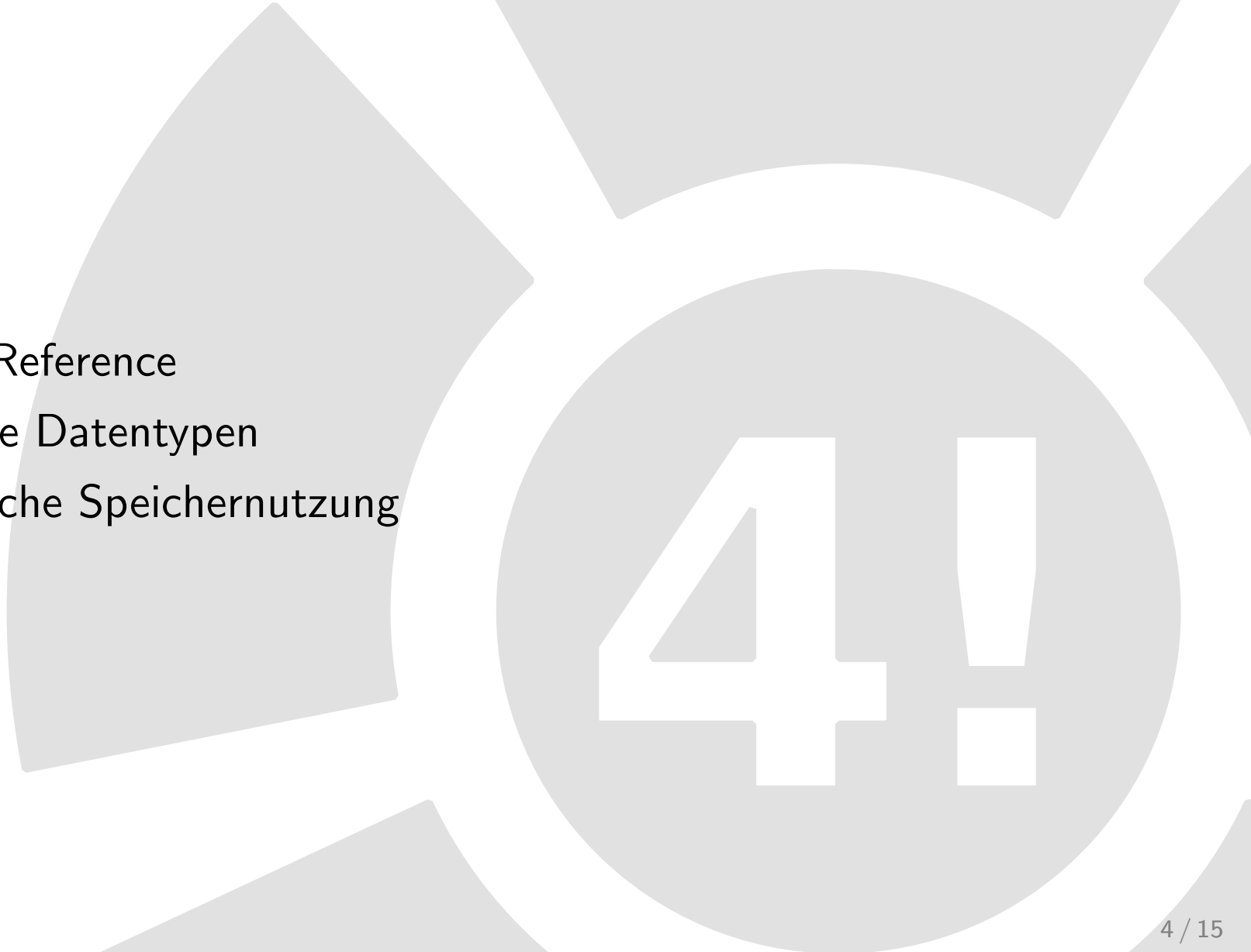
Was sind Pointer?

- Pointer sind eine spezielle Art von Datentypen deren Wert eine Speicheradresse entspricht.



Wozu sind Pointer gut?

- Strings
- Arrays
- Call by Reference
- komplexe Datentypen
- dynamische Speichernutzung



Deklarierung

Pointerdeklarierung

```
1 int *int_pointer;
2 char *char_pointer;
```

Deklarierung

Pointerdeklarierung

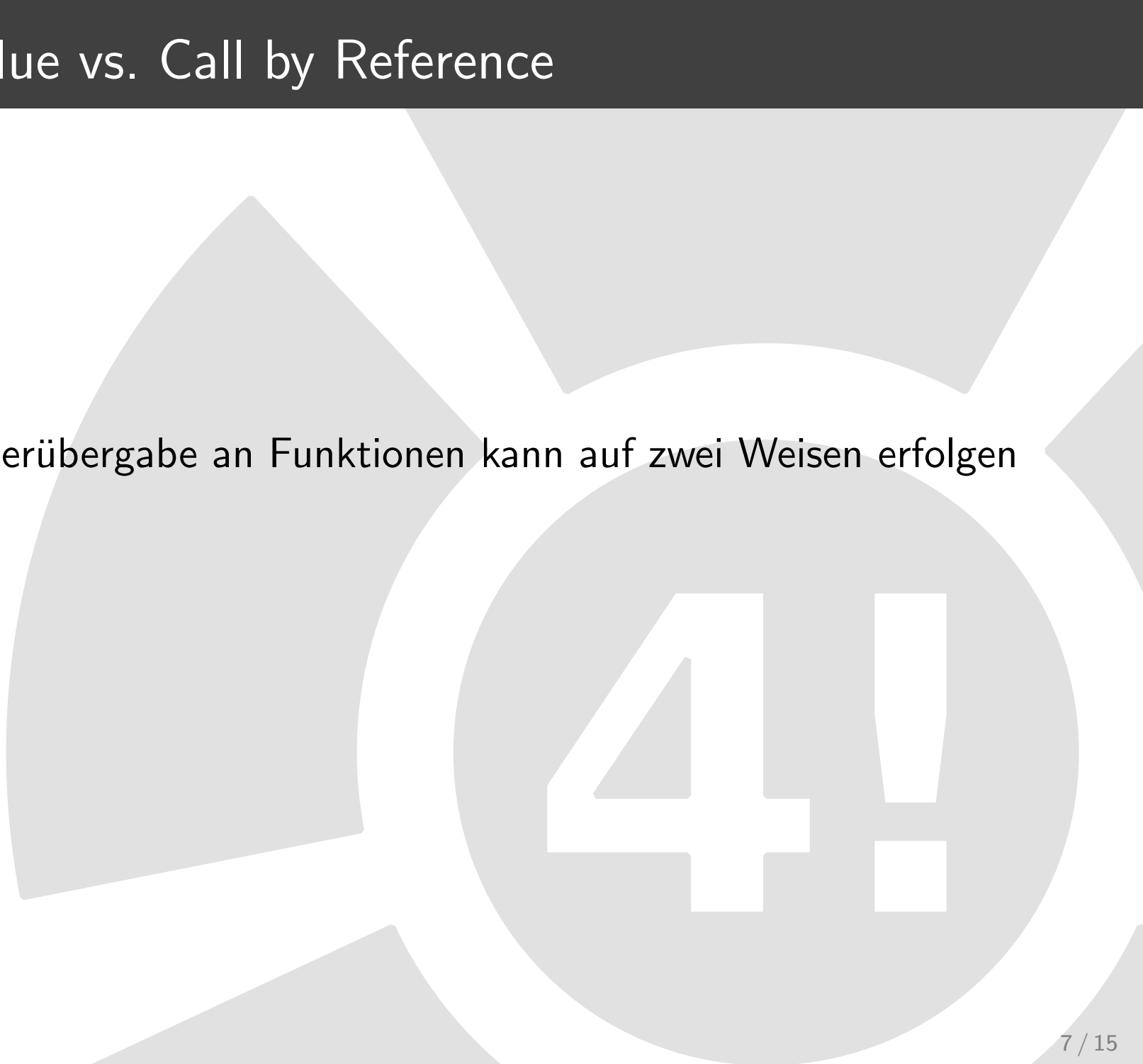
```
1 int *int_pointer;
2 char *char_pointer;
3 TYP *TYP_pointer;
```

Referenzierung und Dereferenzierung

- Referenzierungsoperator: **&**
  - Rückgabe der Adresse das nachfolgenden Ausdrucks
- Dererefenzierungoperator: ***
  - Rückgabe des Wertes an der Adresse des nachfolgenden Pointers
- **Beispielcode Live-Coding**

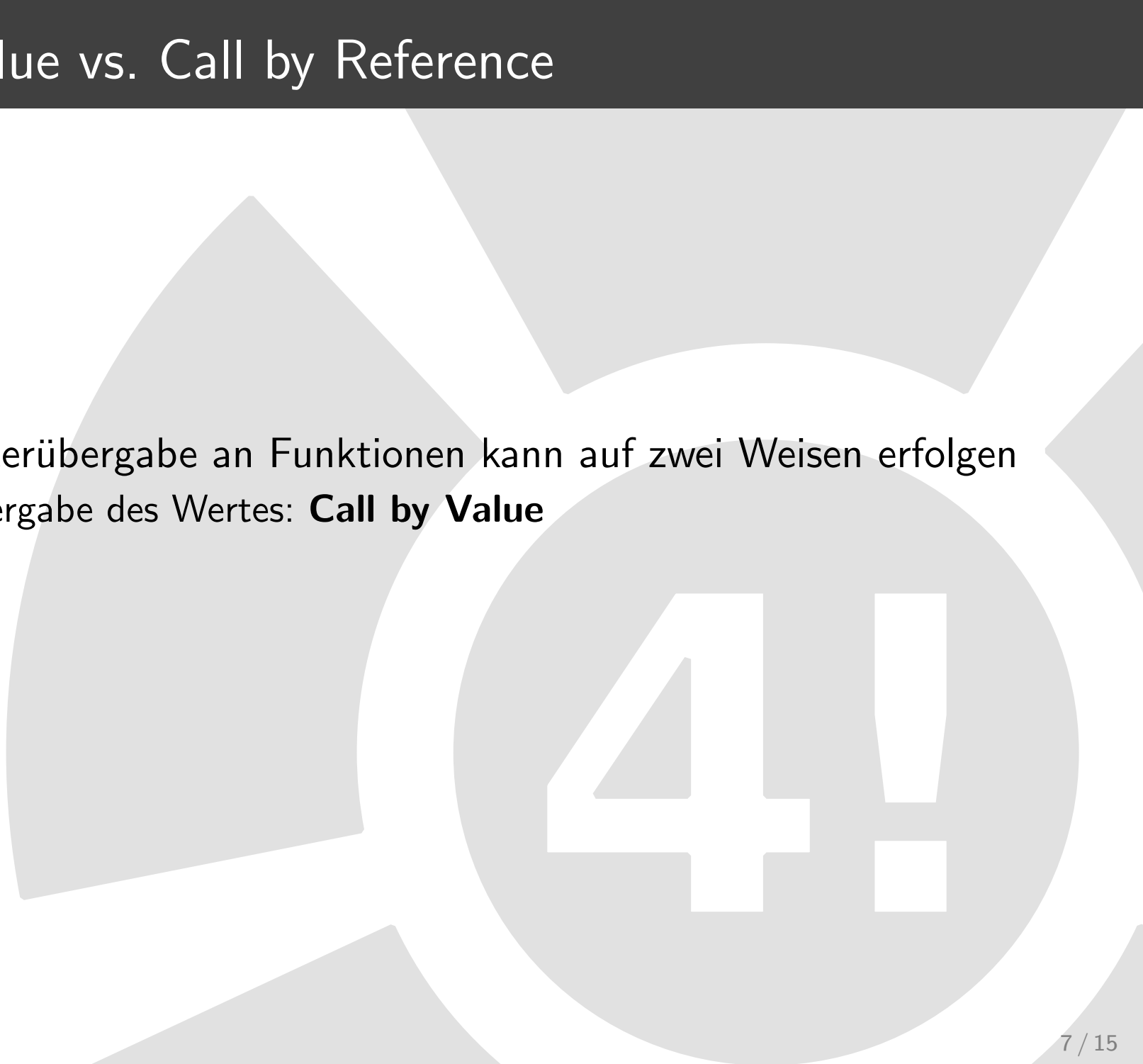
Call by Value vs. Call by Reference

- Parameterübergabe an Funktionen kann auf zwei Weisen erfolgen



Call by Value vs. Call by Reference

- Parameterübergabe an Funktionen kann auf zwei Weisen erfolgen
  - Übergabe des Wertes: **Call by Value**



Call by Value vs. Call by Reference

- Parameterübergabe an Funktionen kann auf zwei Weisen erfolgen
  - Übergabe des Wertes: **Call by Value**
  - Übergabe der Adresse: **Call by Reference**

Arrays

- bereits aus der ersten Vorlesung bekannt:

Arrays in bekannter Syntax

```
1 int numbers[] = {0, 1, 1, 2, 3, 5, 8, 13};
2 int number = numbes[5];
```

Arrays

- bereits aus der ersten Vorlesung bekannt:

Arrays in bekannter Syntax

```
1 int numbers[] = {0, 1, 1, 2, 3, 5, 8, 13};
2 int number = numbes[5];
```

- Arrays sind auch Pointer!

Arrays als Pointer

```
1 int number = *(numbers + 5)
```

Arrays als Pointer

- Ein Array ist ein Pointer auf das erste Element

Arrays als Pointer

```
1 numbers = &numbers[0]
```

Arrays als Pointer

- Ein Array ist ein Pointer auf das erste Element

Arrays als Pointer

```
1 numbers = &numbers[0]
2
3 numbers[k] = *(numbers + k)
```

Strings

- Strings sind Arrays von **char**s

String Initialisierung

```
1 char str[] = "EIN_STRING";
2 char *str_pointer   = "EIN_ANDERER_STRING";
3 char *str_pointer_2 = "EIN_ANDERER_STRING";
```

Strings

- Strings sind Arrays von **char**s

String Initialisierung

```
1 char str[] = "EIN_STRING";
2 char *str_pointer   = "EIN_ANDERER_STRING";
3 char *str_pointer_2 = "EIN_ANDERER_STRING";
```

- Besonderheiten
  - char-Pointer sind nicht veränderbar
  - str_pointer_2 zeigt auf die selbe Adresse wie str_pointer
  - sizeof(str) = 11 (#chars + '\0')

sizeof

- Größe eines Arrays lässt sich ermitteln mit **sizeof**

sizeof Syntax

```
1 sizeof(VARIABLE) = von einer Variable belegter Speicher
2 sizeof(TYP) = von einer Instanz eines Typs belegter Speicher
```

sizeof

- Größe eines Arrays lässt sich ermitteln mit **sizeof**

sizeof Syntax

```
sizeof(VARIABLE) = von einer Variable belegter Speicher
sizeof(TYP) = von einer Instanz eines Typs belegter Speicher
```

sizeof Beispiel

```
printf("char: %lu\n", sizeof(char));
printf("int:  %lu\n", sizeof(int));
printf("C-Kurs:  %lu\n", sizeof("C-Kurs"));

int numbers[] = {0, 1, 1, 2, 3, 5, 8, 13};
printf("numbers: %lu\n", sizeof(numbers));
printf("#Elemente in numbers: %lu\n",
       sizeof(numbers)/sizeof(int));
```

Pointerarithmetik

- arithmetische Operatoren lassen sich auch auf Pointer anwenden

Pointerarithmetik

```
1  int *int_pointer   = (int  *)0x100;
2  char *char_pointer = (char *)0x100;
3  void *void_pointer = (void *)0x100;
4
5  int_pointer  += 1;
6  char_pointer += 1;
7  void_pointer += 1;
8
9  printf("Speicherplatz der Datentypen:",
10        sizeof(int), sizeof(char), sizeof(void));
11 printf("Pointer zeigen auf: \%p \%p \%p\n",
12         int_pointer, char_pointer, void_pointer);
```

Pointerarithmetik

- arithmetische Operatoren lassen sich auch auf Pointer anwenden

Pointerarithmetik

```
1  int *int_pointer   = (int  *)0x100;
2  char *char_pointer = (char *)0x100;
3  void *void_pointer = (void *)0x100;
4
5  int_pointer  += 1;
6  char_pointer += 1;
7  void_pointer += 1;
8
9  printf("Speicherplatz der Datentypen:",
10        sizeof(int), sizeof(char), sizeof(void));
11 printf("Pointer zeigen auf: \%p \%p \%p\n",
12         int_pointer, char_pointer, void_pointer);
```

Speicherplatz der Datentypen: 4, 1, 1
Pointer zeigen auf: 0x104 0x101 0x101

Pointerarithmetik

- arithmetische Operatoren lassen sich auch auf Pointer anwenden

Pointerarithmetik

```
1  int *int_pointer   = (int  *)0x100;
2  char *char_pointer = (char *)0x100;
3  void *void_pointer = (void *)0x100;
4
5  int_pointer  += 1;
6  char_pointer += 1;
7  void_pointer += 1;
8
9  printf("Speicherplatz der Datentypen:",
10        sizeof(int), sizeof(char), sizeof(void));
11 printf("Pointer zeigen auf: \%p \%p \%p\n",
12         int_pointer, char_pointer, void_pointer);
```

Speicherplatz der Datentypen: 4, 1, 1
Pointer zeigen auf: 0x104 0x101 0x101

Ergebnis der arith. Operation variiert je nach Größe des Datentyps

ptr → ptr

- Pointer lassen sich auch referenzieren:

ptr von ptr

```
1 int  *ptr;
2 int  **ptr = &ptr;
```

- Damit kann man z.B. Arrays aus Pointern erzeugen.
- Mehrdimensionale Arrays sind ein Anwendungsbeispiel für Pointern auf Pointer

Mehrdimensionale Arrays

Mehrdimensionale Array

mehrdimensionale Arrays

```
1 int main() {
2     int matrix[][] = { {0, 1, 2},
3                        {3, 4, 5} };
4     return 0;
5 }
```

Zugriff auf Mehrdimensionale Arrays

Zugriff auf mehrdimensionale Arrays

```
1     matrix[1][2] = 4;
```

const correctness

const ist ein Schlüsselwort das Schreibzugriff auf eine Variable verbietet:

```
const int const var = 5;
   A          B
```

- A und B haben den gleichen Effekt, Wert von var ist nicht änderbar

const correctness

const ist ein Schlüsselwort das Schreibzugriff auf eine Variable verbietet:

```
const int const var = 5;
  A         B
```

const kann auch bei Pointern verwendet werden:

```
const int const * const var;
  A         B       C
```

- A und B haben den gleichen Effekt, Wert von var ist nicht änderbar
- C verhindert Änderungen des Pointers, Zielwert kann geändert werden