Pointer und Arrays (2)

- Arrays und Pointer
- Mehrdimensionale Arrays und Pointer

Arrays und Pointer

- **Ein Array ist eine Sammlung von Objekten vom gleichen Typ.**
- **Ein Pointer ist eine Variable, die eine Adresse speichert.**
- **Arrays und Pointer können annähernd gleich verwendet werden:**

```
char single = 0;
scanf_s("%c", &single, sizeof(single));

char multiple[10];
scanf_s("%s", multiple, sizeof(multiple));
```

- **Der Arrayname kann wie ein Pointer verwendet werden.**
- **Der Arrayname zeigt auf das erste Element eines Arrays.**
- **Aber, ein Array ist kein Pointer!**
- **Unterschiede:**
  - **Die Adresse, die vom Arraynamen referenziert wird, kann nicht geändert werden.**
  - **Jedoch kann die Adresse, die in einem Pointer gespeichert ist, geändert werden.**

Bsp. 04 veranschaulicht, dass der Arrayname auf die erste Adresse zeigt

```c
// Program pointer_04.c  Arrays and pointers
#include <stdio.h>

int main(void)
{
  char multiple[] = "My string";

  char *p = &multiple[0];
  printf("The address of the first array element  : %p\n", p);

  p = multiple;
  printf("The address obtained from the array name: %p\n", multiple);
  return 0;
}
```

Output:

```
The address of the first array element  : 0x28cc72
The address obtained from the array name: 0x28cc72
```

Bsp. 05 veranschaulicht, wenn man einem Pointer einen Wert hinzuaddiert

```c
// Program pointer_05.c Incrementing a pointer to an array
#include <stdio.h>
#include <string.h>

int main(void)
{
  char multiple[] = "a string";
  char *p = multiple;

  for(int i = 0 ; i < strnlen(multiple, sizeof(multiple)) ; ++i)
    printf("multiple[%d] = %c  *(p+%d) = %c  &multiple[%d] = %p  p+%d = %p\n",
                      i, multiple[i], i, *(p+i), i, &multiple[i], i, p+i);
  return 0;
}
```

Output:

```
multiple[0] = a  *(p+0) = a  &multiple[0] = 0x28cc6f  p+0 = 0x28cc6f
multiple[1] =    *(p+1) =    &multiple[1] = 0x28cc70  p+1 = 0x28cc70
multiple[2] = s  *(p+2) = s  &multiple[2] = 0x28cc71  p+2 = 0x28cc71
multiple[3] = t  *(p+3) = t  &multiple[3] = 0x28cc72  p+3 = 0x28cc72
multiple[4] = r  *(p+4) = r  &multiple[4] = 0x28cc73  p+4 = 0x28cc73
multiple[5] = i  *(p+5) = i  &multiple[5] = 0x28cc74  p+5 = 0x28cc74
multiple[6] = n  *(p+6) = n  &multiple[6] = 0x28cc75  p+6 = 0x28cc75
multiple[7] = g  *(p+7) = g  &multiple[7] = 0x28cc76  p+7 = 0x28cc76
```

Bsp. 06 gleiches Beispiel aber mit einem Array vom Typ long

```c
// Program pointer_06.c Incrementing a pointer to an array of integers
#include <stdio.h>

int main(void)
{
  long multiple[] = {15L, 25L, 35L, 45L};
  long *p = multiple;

  for(int i = 0 ; i < sizeof(multiple)/sizeof(multiple[0]) ; ++i)
    printf("address p+%d (&multiple[%d]): %llu        *(p+%d)   value: %d\n",
                                    i, i, (unsigned long long)(p+i),  i,  *(p+i));
  printf("\n    Type long occupies: %d bytes\n", (int)sizeof(long));
  return 0;
}
```

Output - Adresse als unsigned long long (64Bit) dargestellt:

```
address p+0 (&multiple[0]): 2673768        *(p+0)   value: 15
address p+1 (&multiple[1]): 2673772        *(p+1)   value: 25
address p+2 (&multiple[2]): 2673776        *(p+2)   value: 35
address p+3 (&multiple[3]): 2673780        *(p+3)   value: 45
```

Bsp. 07 Array-Adressen von multidimensionalen Arrays

```c
// Program pointer_07.c Two-dimensional arrays and pointers
#include <stdio.h>

int main(void)
{
  char board[3][3] = {
                       {'1','2','3'},
                       {'4','5','6'},
                       {'7','8','9'}
                     };

  printf("address of board        : %p\n", board);
  printf("address of board[0][0]  : %p\n", &board[0][0]);
  printf("contents of board[0]    : %p\n", board[0]);
  return 0;
}
```

Output:

```
address of board        : 0x28cc77
address of board[0][0]  : 0x28cc77
contents of board[0]    : 0x28cc77
```

Bsp. 07a : Array Werte auslesen mit Hilfe von Pointer Indirektion

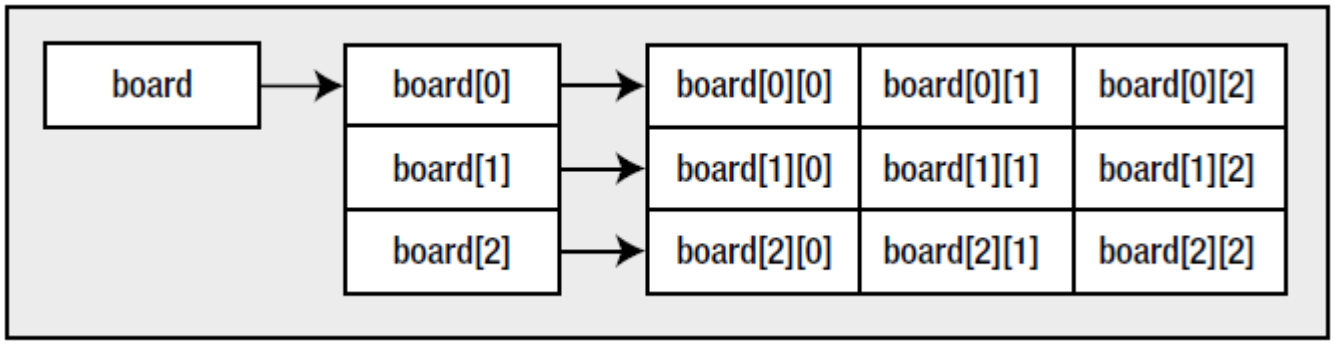
```c
// Program pointer_07a.c Two-dimensional arrays and pointers
#include <stdio.h>

int main(void)
{
  char board[3][3] = {
                       {'1','2','3'},
                       {'4','5','6'},
                       {'7','8','9'}
                     };

  printf("value of board[0][0] : %c\n", board[0][0]);
  printf("value of *board[0]   : %c\n", *board[0]);
  printf("value of **board     : %c\n", **board);
  return 0;
}
```

Output:

```
value of board[0][0] : 1
value of *board[0]   : 1
value of **board     : 1
```

Referenzierung eines Arrays, die Subarrays und die Elementwerte

- **Um auf den ersten Elementwert mit dem Arraynamen zu gelangen braucht es zwei Indirektionen ****
- **Mit einer Indirektion * gelangt man auf das erste Array von "Array auf Array" (ist das gleiche wie Arrayname[0])**



Bsp. 08: veranschaulicht wie ein multidimensionales Array gespeichert wird

```c
// Program pointer_08.c  Getting values in a two-dimensional array
#include <stdio.h>

int main(void)
{
  char board[3][3] = {
                       {'1','2','3'},
                       {'4','5','6'},
                       {'7','8','9'}
                     };

   // List all elements of the array
  for(int i = 0 ; i < 9 ; ++i)
    printf(" board: %c\n", *(*board + i));
  return 0;
}
```

Output:

```
board: 1
board: 2
..
board: 8
board: 9
```

Bsp. 09: Multidimensionales Array mit Pointer auslesen

```c
// Program pointer_09.c  Multidimensional arrays and pointers
#include <stdio.h>

int main(void)
{
  char board[3][3] = {
                       {'1','2','3'},
                       {'4','5','6'},
                       {'7','8','9'}
                     };

  char *pboard = *board;              // A pointer to char
  for(int i = 0 ; i < 9 ; ++i)
    printf(" board: %c\n", *(pboard + i));

  return 0;
}
```

Output:

```
 board: 1
 board: 2
 ..
 board: 9
```

Mit verschiedenen Pointer Indirektionen auf Arrayelemente zugreifen

board	0	1	2
0	board[0][0] *board[0] **board	board[0][1] *(board[0]+1) *(*board+1)	board[0][2] *(board[0]+2) *(*board+2)
1	board[1][0] *(board[0]+3) *board[1] *(*board+3)	board[1][1] *(board[0]+4) *(board[1]+1) *(*board+4)	board[1][2] *(board[0]+5) *(board[1]+2) *(*board+5)
2	board[2][0] *(board[0]+6) *(board[1]+3) *board[2] *(*board+6)	board[2][1] *(board[0]+7) *(board[1]+4) *(board[2]+1) *(*board+7)	board[2][2] *(board[0]+8) *(board[1]+5) *(board[2]+2) *(*board+8)