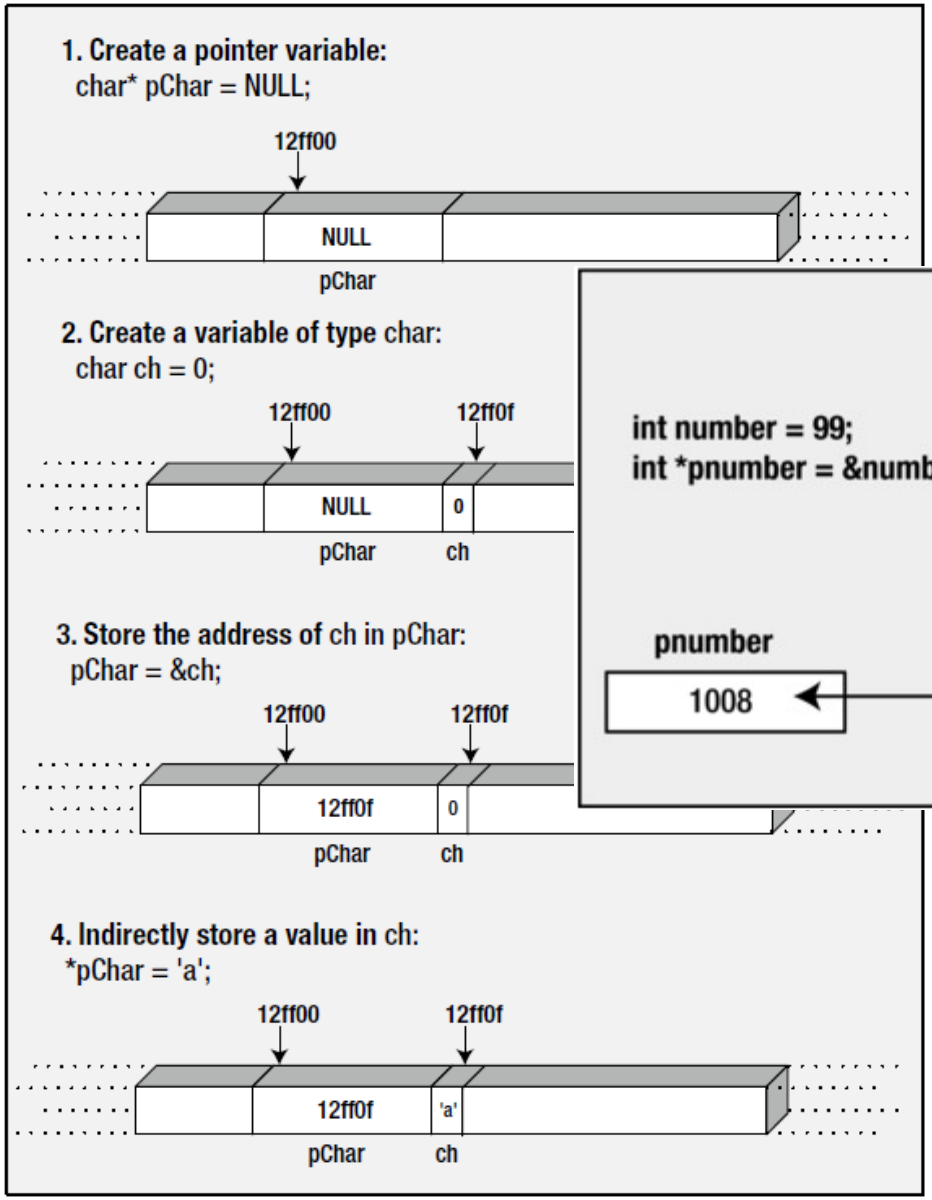
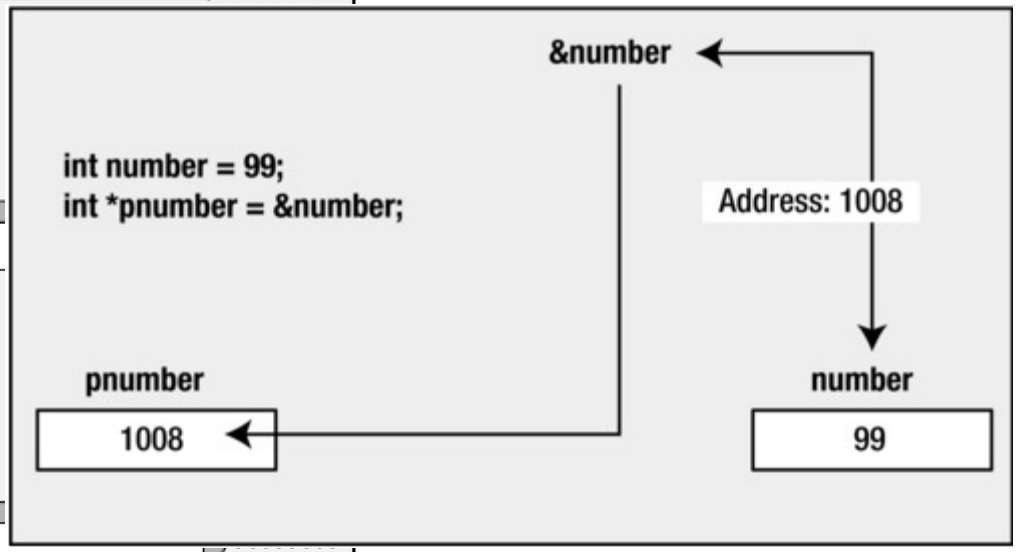




Pointer and Arrays

Chapter Goals

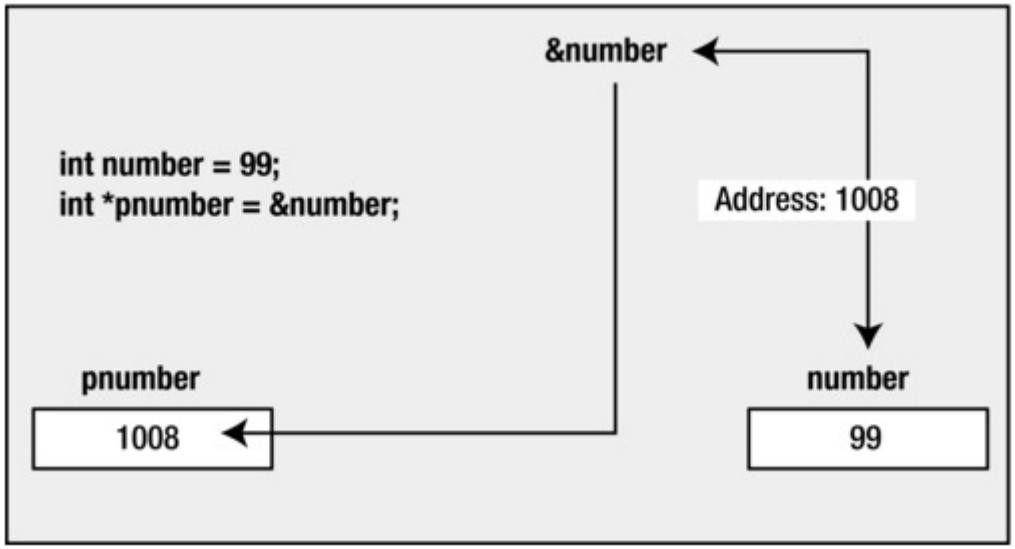
- Pointer Konzept
- Adressoperator
- Deklaration und Benutzung
- const und Pointer

- Array Konzept
- Deklaration und Benutzung
- Mehrdimensionale Arrays
- Initialiesierung

Erster Blick auf Pointers

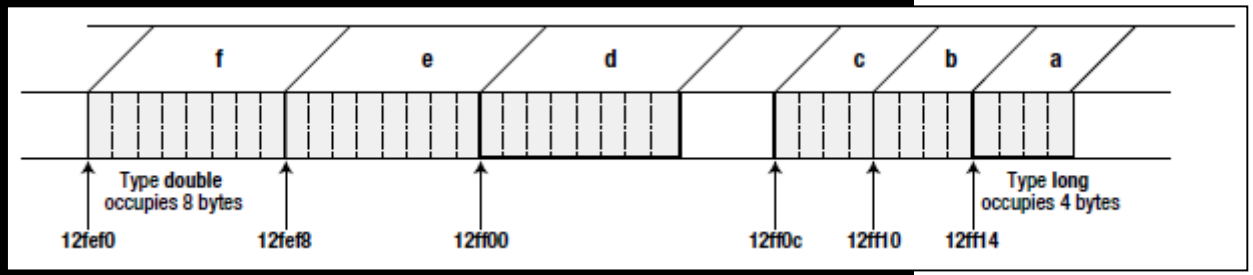
- **Variable Deklaration**

```
int number = 99;
int *pnumber = &number;
```



Der Adressoperator '&'

```c
// Program array.5 Using the & operator
#include<stdio.h>

int main(void)
{
  // Define some integer variables
  long a = 1L;
  long b = 2L;
  long c = 3L;

  // Define some floating-point variables
  double d = 4.0;
  double e = 5.0;
  double f = 6.0;

  printf("A variable of type long occupies %u bytes.", sizeof(long));
  printf("\nHere are the addresses of some variables of type long:");
  printf("\nThe address of a is: %p  The address of b is: %p", &a, &b);
  printf("\nThe address of c is: %p", &c);
  printf("\n\nA variable of type double occupies %u bytes.", sizeof(double));
  printf("\nHere are the addresses of some variables of type double:");
  printf("\nThe address of d is: %p  The address of e is: %p", &d, &e);
  printf("\nThe address of f is: %p\n", &f);
  return 0;
```




```
A variable of type long occupies 4 bytes.
Here are the addresses of some variables of type long:
The address of a is: 000000000012ff14 The address of b is: 000000000012ff10
The address of c is: 000000000012ff0c
A variable of type double occupies 8 bytes.
Here are the addresses of some variables of type double:
The address of d is: 000000000012ff00 The address of e is: 000000000012fef8
The address of f is: 000000000012fef0
```

Pointer Deklaration

- **Int Pointer**

```
int *pnumber;
int* pnumber;
```

- **NULL ist eine Konstante die in der Standard Library definiert ist**

```
int *pnumber = NULL;
```

- **Initialisierung**

```
int number = 99;
int *pnumber = &number;
```

- **Declaration Statements**

```
double value, *pVal, fnum;
int *p, q;
```

- **Indirection Operator**

```
int number = 15;
int *pnumber = &number;
int result = 0;

result = *pointer + 5;
```

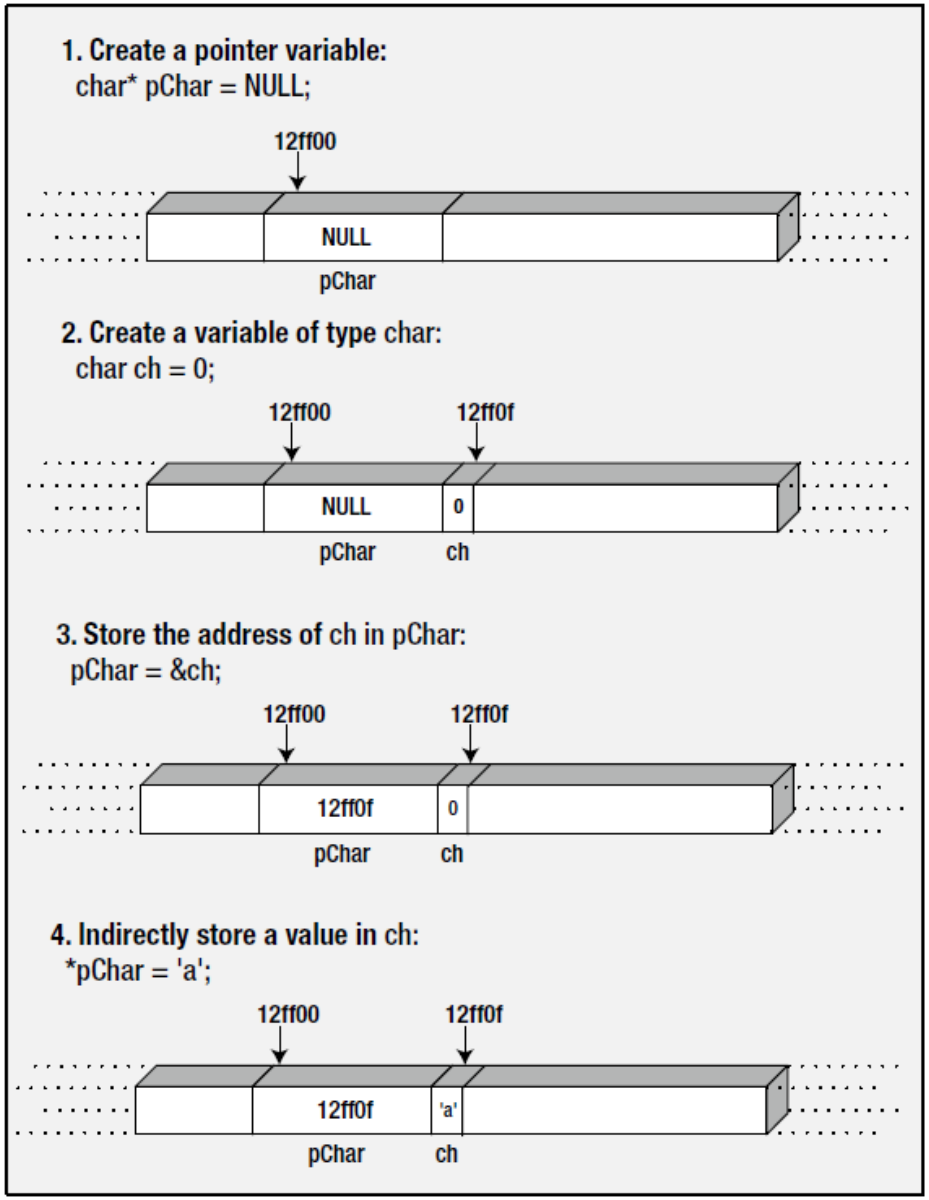
Pointer Deklaration

```c
// Program pointer.1 A simple program using pointers
#include <stdio.h>

int main(void)
{
  int number = 0;                                  // A variable of type int initialized to 0
  int *pnumber = NULL;                             // A pointer that can point to type int

  number = 10;
  printf("number's address: %p\n", &number);       // Output the address
  printf("number's value: %d\n\n", number);        // Output the value

  pnumber = &number;                               // Store the address of number in pnumber

  printf("pnumber's address: %p\n", (void*)&pnumber);    // Output the address
  printf("pnumber's size: %d bytes\n", sizeof(pnumber)); // Output the size
  printf("pnumber's value: %p\n", pnumber);              // Output the value (an address)
  printf("value pointed to: %d\n", *pnumber);            // Value at the address
  return 0;
}
```

Output:

```
number's address: 000000000012ff0c
number's value: 10
pnumber's address: 000000000012ff00
pnumber's size: 8 bytes
pnumber's value: 000000000012ff0c
value pointed to: 10
```

Benutzen von Pointer



Benutzen von Pointer (2)

- **Dereferenzieren**

```
int number = 99;
int *pnumber = &number;
*pnumber += 25;
```

- **Inhalt ändern**

```
int value = 999;
pnumber = $value;
*pnumber += 25;
```

Benutzen von Pointer

```c
// Program pointer.2  What's the pointer of it all
#include <stdio.h>

int main(void)
{
  long num1 = 0L;
  long num2 = 0L;
  long *pnum = NULL;

  pnum = &num1;                                  // Get address of num1
  *pnum = 2L;                                    // Set num1 to 2
  ++num2;                                        // Increment num2
  num2 += *pnum;                                 // Add num1 to num2

  pnum = &num2;                                  // Get address of num2
  ++*pnum;                                       // Increment num2 indirectly

  printf("num1 = %ld  num2 = %ld  *pnum = %ld  *pnum + num2 = %ld\n",
                                    num1, num2, *pnum, *pnum + num2);
  return 0;
}
```

Output:

```
num1 = 2 num2 = 4 *pnum = 4 *pnum + num2 = 8
```

Benutzen von Pointer für Input Daten

```c
// Program pointer.3  Using pointer arguments to scanf
#include <stdio.h>

int main(void)
{
  int value = 0;
  int *pvalue = &value;                  // Set pointer to refer to value

  printf ("Input an integer: ");
  scanf(" %d", pvalue);                  // Read into value via the pointer

  printf("You entered %d.\n", value);      // Output the value entered
  return 0;
}
```

Output:

```
Input an integer: 10
You entered 10
```

Testen auf NULL Pointer

```
int *pvalue = NULL;
```

ist gleich wie

```
int *pvalue = 0;
```

auf NULL testen

```
if ( !pvalue ) {
  // the Pointer ist NULL
}
```

oder

```
if ( pvalue == NULL ) {
  // the Pointer ist NULL
}
```

const und Pointer

- **Pointer auf const**

```
long value = 9999L;
const long *pvalue = &value;
*pValue = 8888L   // Error attempt to change const
pValue = &number; // Ok; changing the addresss in pValue
```

- **const Pointer auf Variable**

```
int count = 43;
int *const pcount = &count;
item = 34;
pcount = &item;   // Error attempt to change a constant pointer
*pcount = 345;    // OK; changes the value of count
```

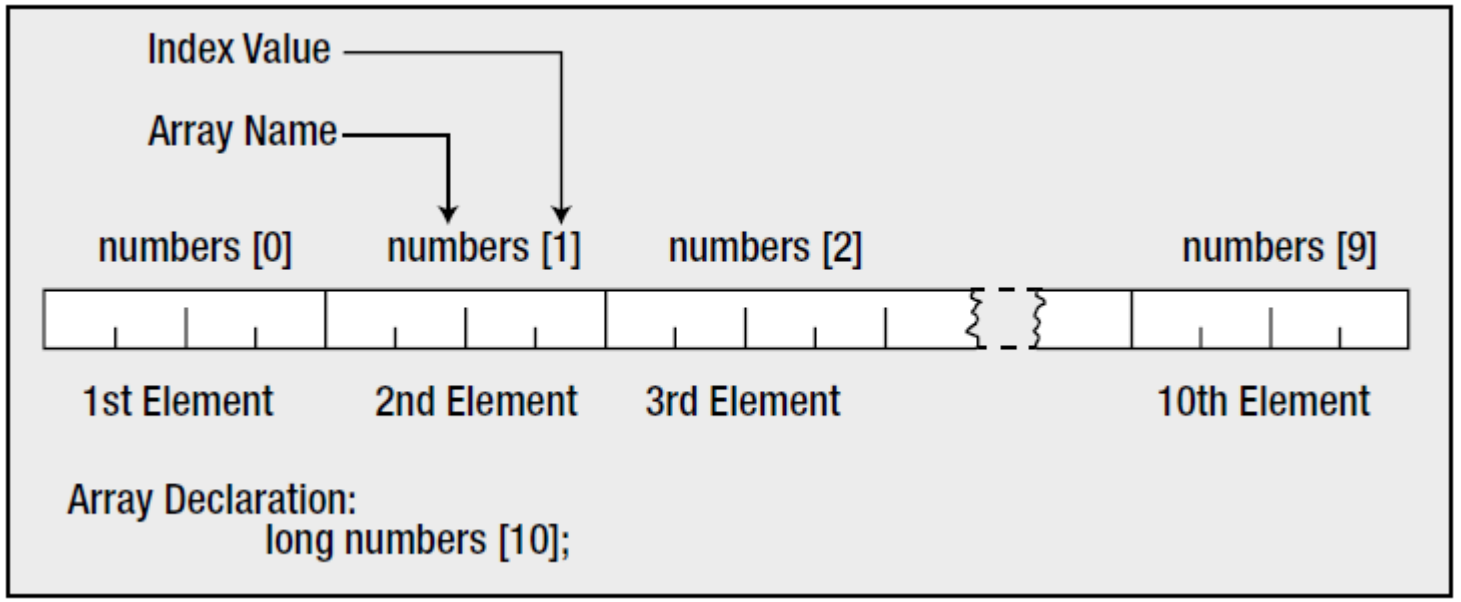
- **const Pointer auf const Variable**

```
int item = 25;
const int *const pitem = &item;
```

Erster Blick auf Arrays

- **Variable Deklaration**

```
long numbers[10];
```



Mittelwert mit Arrays

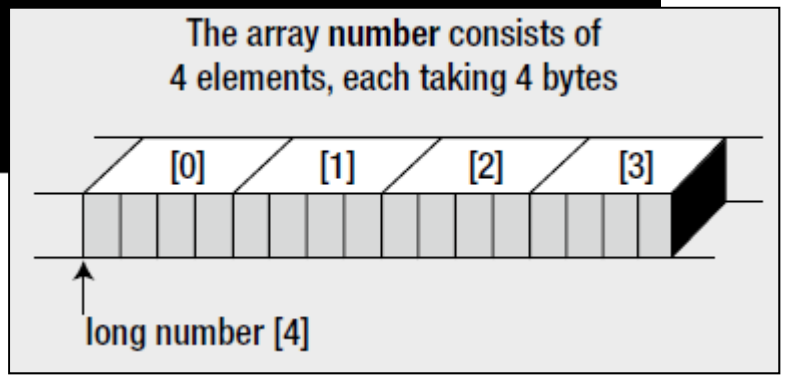
```c
// Program array.3 Averaging ten grades - storing the values the easy way
#include <stdio.h>

int main(void)
{
  int grades[10];                          // Array storing 10 values
  unsigned int count = 10;                 // Number of values to be read
  long sum = 0L;                           // Sum of the numbers
  float average = 0.0f;                    // Average of the numbers

  printf("\nEnter the 10 grades:\n");      // Prompt for the input

  // Read the ten numbers to be averaged
  for(unsigned int i = 0 ; i < count ; ++i)
  {
    printf("%2u> ",i + 1);
    scanf("%d", &grades[i]);               // Read a grade
    sum += grades[i];                      // Add it to sum
  }

  average = (float)sum/count;              // Calculate the average

  printf("\nAverage of the ten grades entered is: %f\n", average);
  return 0;
}
```

Output:

```
Enter the ten grades:
1> 450
2> 765
3> 562
4> 700
5> 598
6> 635
7> 501
8> 720
9> 689
10> 527
Average of the ten grades
entered is: 614.70
```

Arrays und Adressen

```c
// Program array.5a Using the & operator
#include<stdio.h>

int main(void)
{
  // Define a long array
  int data[5];

  for(unsigned int i=0; i < 5; i++)
  {
    data[i] = 12*(i+1);
    printf("data[%d] Address: %p  Contents: %d\n", i, &data[i], data[i]);
  }

  return 0;
}
```



Output:

```
data[0] Address: 000000000012fee4 Contents: 12
data[1] Address: 000000000012fee8 Contents: 24
data[2] Address: 000000000012feec Contents: 36
data[3] Address: 000000000012fef0 Contents: 48
data[4] Address: 000000000012fef4 Contents: 60
```

Initialisieren eines Arrays

- **vollständige Initialisierung**
  ```
  double values[5] = { 1.5, 2.5, 3.5, 4.5, 5.5 };
  ```
- **nicht-vollständige Initialisierung**
  ```
  double values[5] = { 1.5, 2.5, 3.5 };
  ```
  die nicht-initialisierten Elemente werden auf 0 gesetzt!
- `double values[5] = {0.0};`
  initialisiert alle Elemente mit 0

- `int primes[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29};`
  die Anzahl der Initialwerte bestimmt die Länge des Arrays

Mit sizeof() die Grösse eines Arrays bestimmen

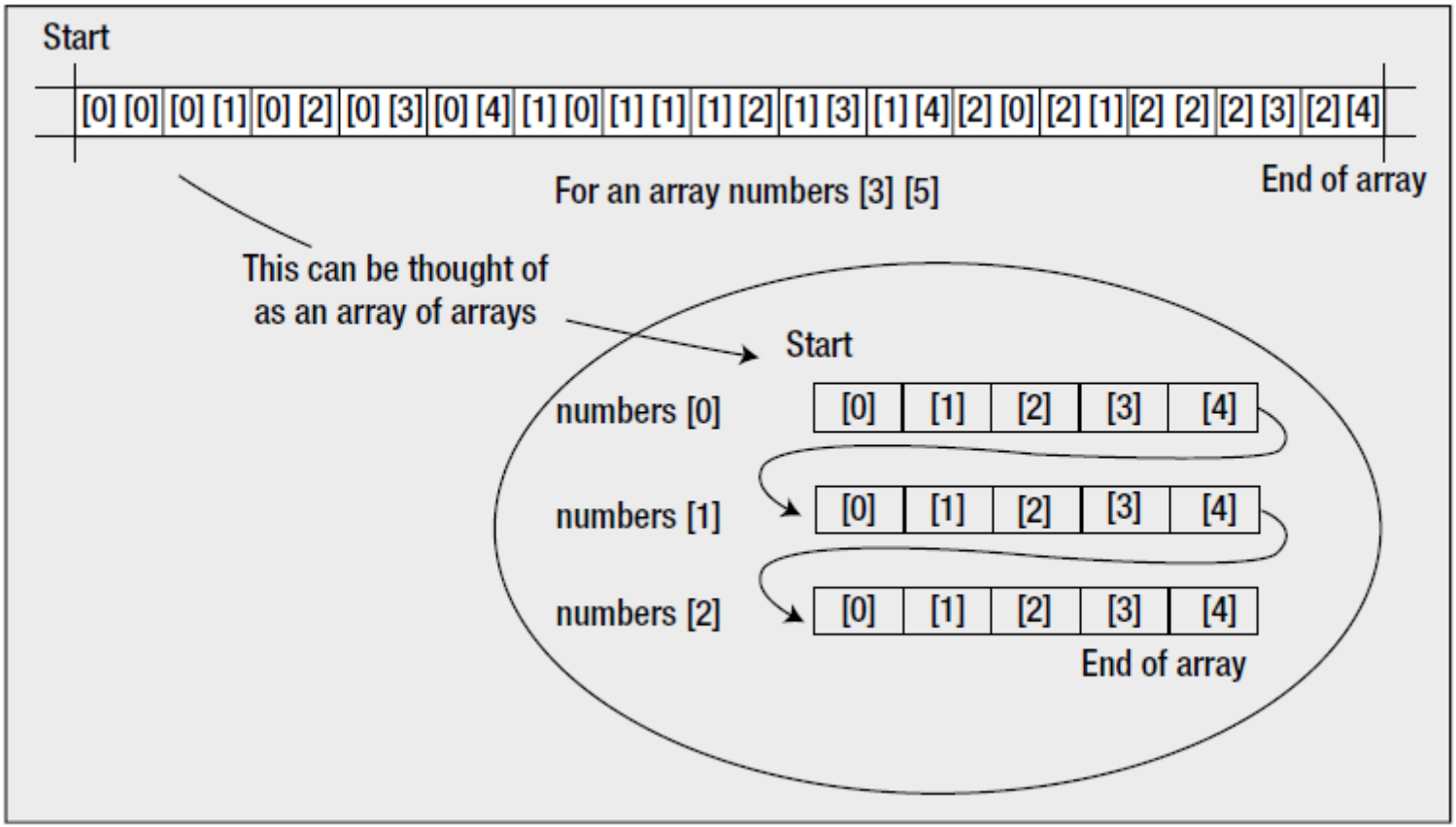
```c
// Program array.5b Find out the size of an Array
#include <stdio.h>

int main() {

  double values[5] = { 1.5, 2.5, 3.5, 4.5, 5.5 };

  // findout the size of the array values with the sizeof operator
  printf("The size of the array, values, is %zu bytes.\n", sizeof values);

  // findout the number of elements in the array values
  size_t element_count = sizeof(values)/sizeof(values[0]);
  printf("The size of the array is %zu bytes ", sizeof(values));
  printf("and there are %u elements of %zu bytes each\n", element_count, sizeof(values[0]));

  return 0;
}
```

Output:

```
The size of the array, values, is 40 bytes
The size of the array is 40 bytes and there are 5 elements of 8 bytes each
```

Mehrdimensionale Arrays

- `float numbers[3][5];`



Initialisieren von mehrdimensionalen Arrays

```
int numbers[3][4] = {
    { 10, 20, 30, 40 }, // Values for first row
    { 15, 25, 35, 45 }, // Values for second row
    { 47, 48, 49, 50 }  // Values for third row                    };
```

```
int numbers[3][4] = {0}; // all values set to 0
```

```
int numbers[2][3][4] = {
    { // First block of 3 rows
      { 10, 20, 30, 40 },
      { 15, 25, 35, 45 },
      { 47, 48, 49, 50 }
    },
    { // Second block of 3 rows
      { 10, 20, 30, 40 },
      { 15, 25, 35, 45 },
      { 47, 48, 49, 50 }
    }
                        };
```

Initialisieren von mehrdimensionalen Arrays (2)

```
int numbers[2][3][4];

// each loop iterates for one row
for(int i = 0 ; i < sizeof(numbers)/sizeof(numbers[0]) ; ++i)
{
  for(int j = 0 ; j < sizeof(numbers[0])/sizeof(numbers[0][0]) ; ++j)
  {
    for(int k=0 ; k<sizeof(numbers[0][0])/sizeof(numbers[0][0][0]); ++k)
    {
      sum += numbers[i][j][k];
    }
  }
}
```

Mehrdimensionales Array - Beispielanwendung

```c
// Program array.6 Know your hat size - if you dare...
#include <stdio.h>
#include <stdbool.h>

int main(void)

{
   /*******************************************************
    * The size array stores hat sizes from 6 1/2 to 7 7/8 *
    * Each row defines one character of a size value so   *
    * a size is selected by using the same index for each *
    * the three rows. e.g. Index 2 selects 6 3/4.         *
    *******************************************************/
   char size[3][12] = {           // Hat sizes as characters
      {'6', '6', '6', '6', '7', '7', '7', '7', '7', '7', '7', '7'},
      {'1', '5', '3', '7', ' ', '1', '1', '3', '1', '5', '3', '7'},
      {'2', '8', '4', '8', ' ', '8', '4', '8', '2', '8', '4', '8'}
                    };

  int headsize[12] =                  // Values in 1/8 inches
      {164,166,169,172,175,178,181,184,188,191,194,197};

  float cranium = 0.0;                // Head circumference in decimal inches
  int your_head = 0;                  // Headsize in whole eighths
  bool hat_found = false;             // Indicates when a hat is found to fit

  // Get the circumference of the head
  printf("\nEnter the circumference of your head above your eyebrows "
       "in inches as a decimal value: ");
  scanf(" %f", &cranium);


  your_head = (int)(8.0*cranium);      // Convert to whole eighths of an inch
```

Mehrdimensionales Array – Beispielanwendung (2)

```
/*****************************************************************
 * Search for a hat size:                                        *
 * Either your head corresponds to the 1st head_size element or  *
 * a fit is when your_head is greater that one headsize element  *
 * and less than or equal to the next.                           *
 * In this case the size is the second headsize value.           *
 *****************************************************************/
unsigned int i = 0;                 // Loop counter
if(your_head == headsize[i])  // Check for min size fit
  hat_found = true;
else
  for (i = 1 ; i < 12 ; ++i)
    // Find head size in the headsize array
    if(your_head > headsize[i - 1] && your_head <= headsize[i])
    {
      hat_found = true;
       break;
    }

if(hat_found)
  printf("\nYour hat size is %c %c%c%c\n",
         size[0][i], size[1][i],
         (size[1][i]==' ') ? ' ' : '/', size[2][i]);
// If no hat was found, the head is too small, or too large
else
{
  if(your_head < headsize[0])        // check for too small
    printf("\nYou are the proverbial pinhead. No hat for"
                                            " you I'm afraid.\n");
  else                               // It must be too large
    printf("\nYou, in technical parlance, are a fathead."
                                " No hat for you, I'm afraid.\n");
}
return 0;
}
```