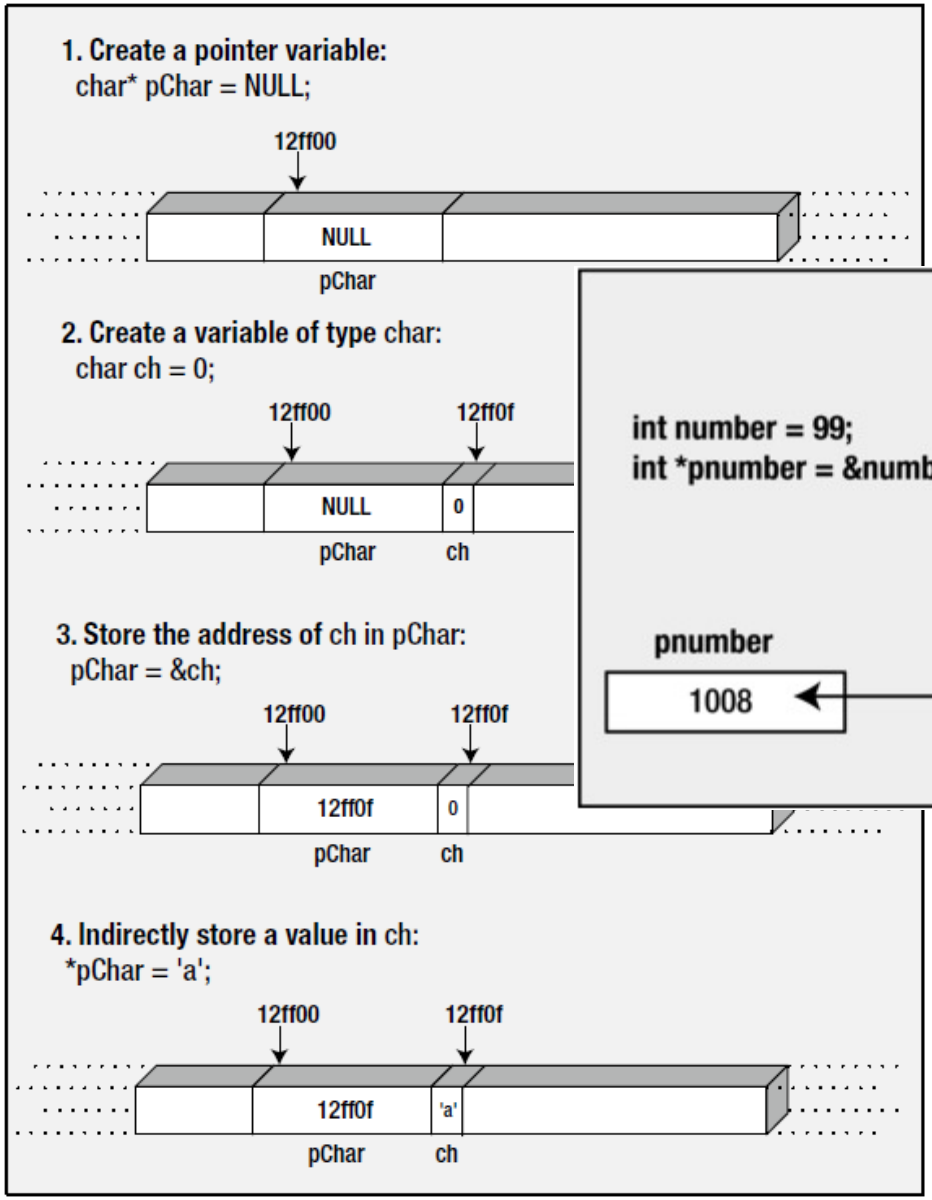
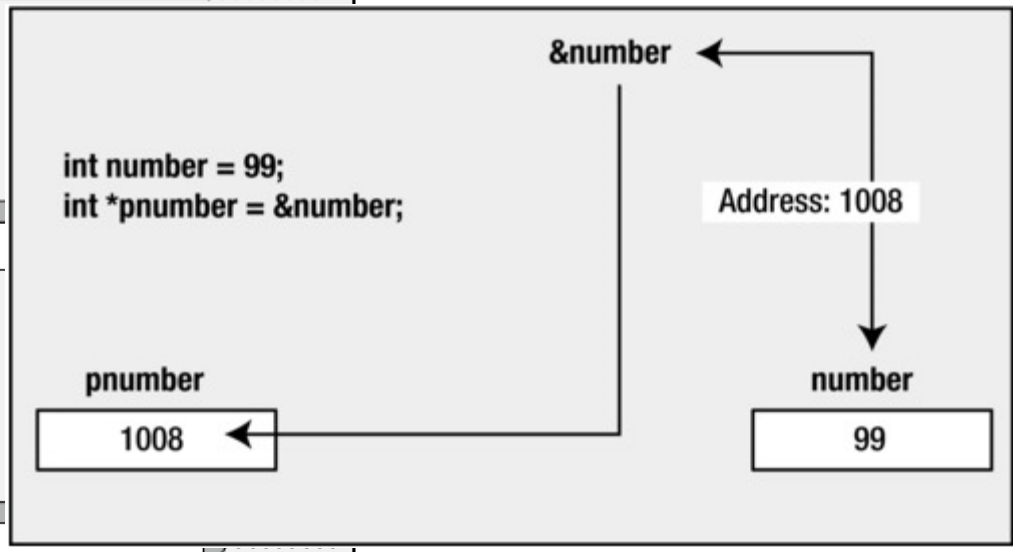




Pointer and Arrays

Chapter Goals

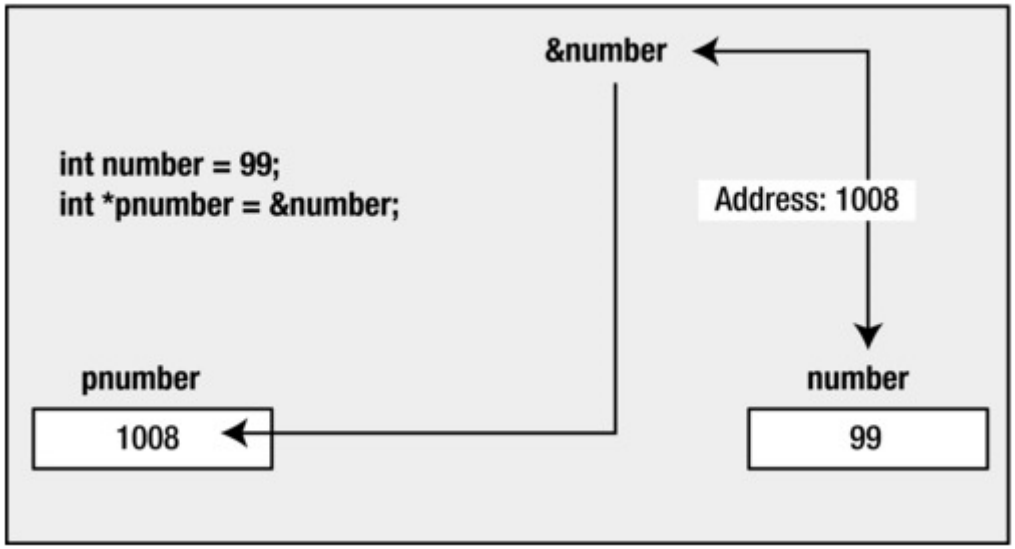
- Pointer Konzept
- Adressoperator
- Deklaration und Benutzung
- const und Pointer

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

Erster Blick auf Pointers

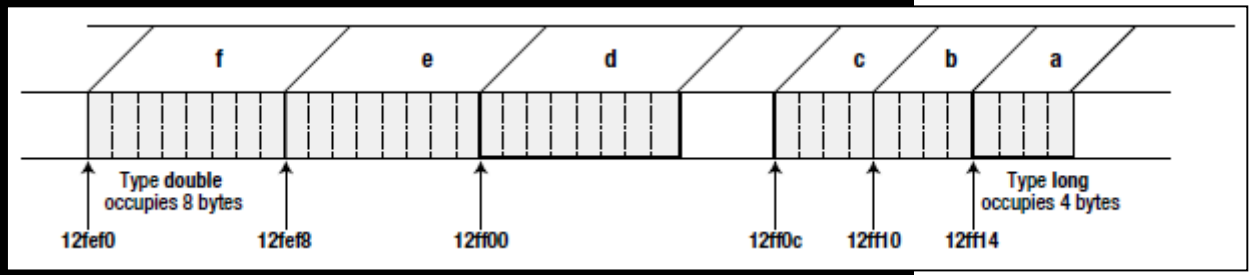
- **Variable Deklaration**

```
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;
```

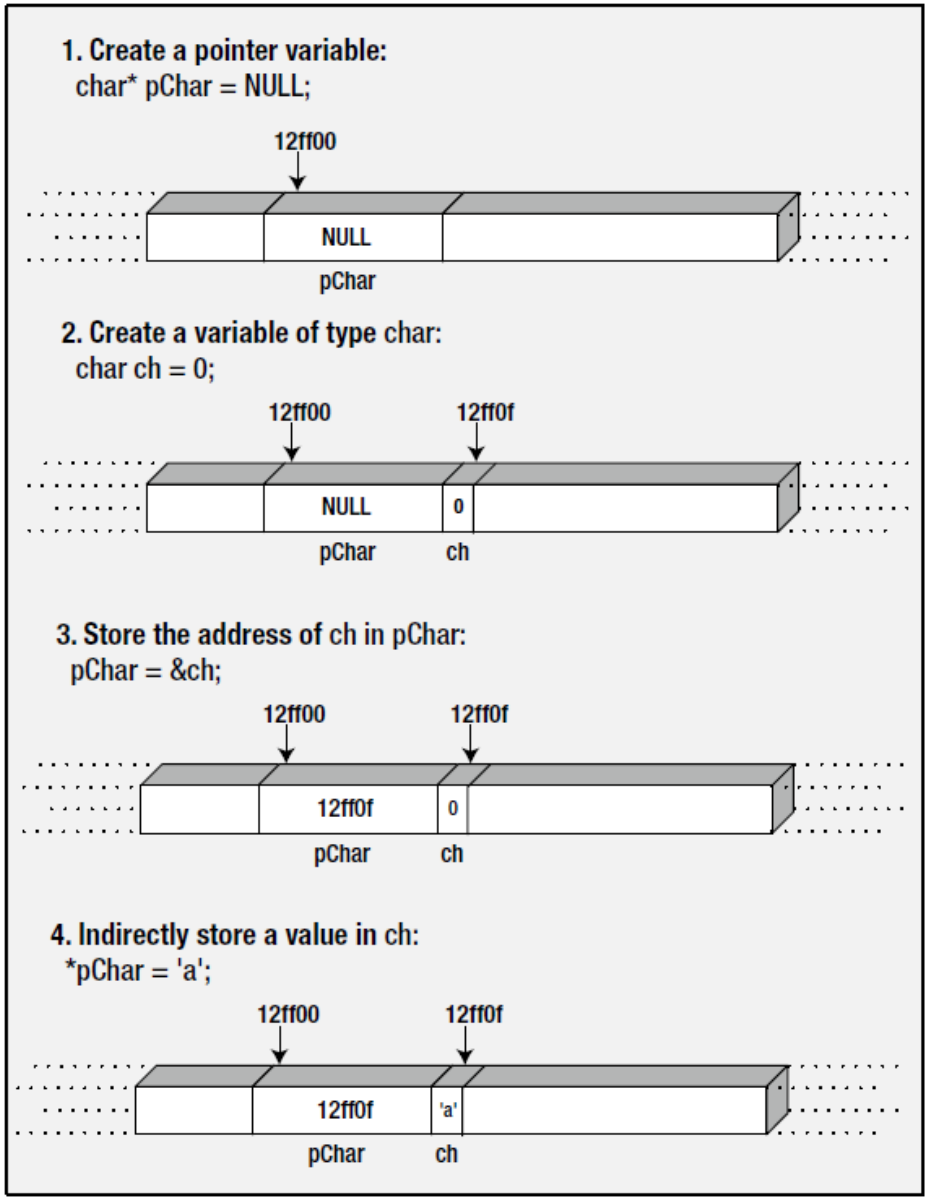
Pointer Deklaration

```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
```

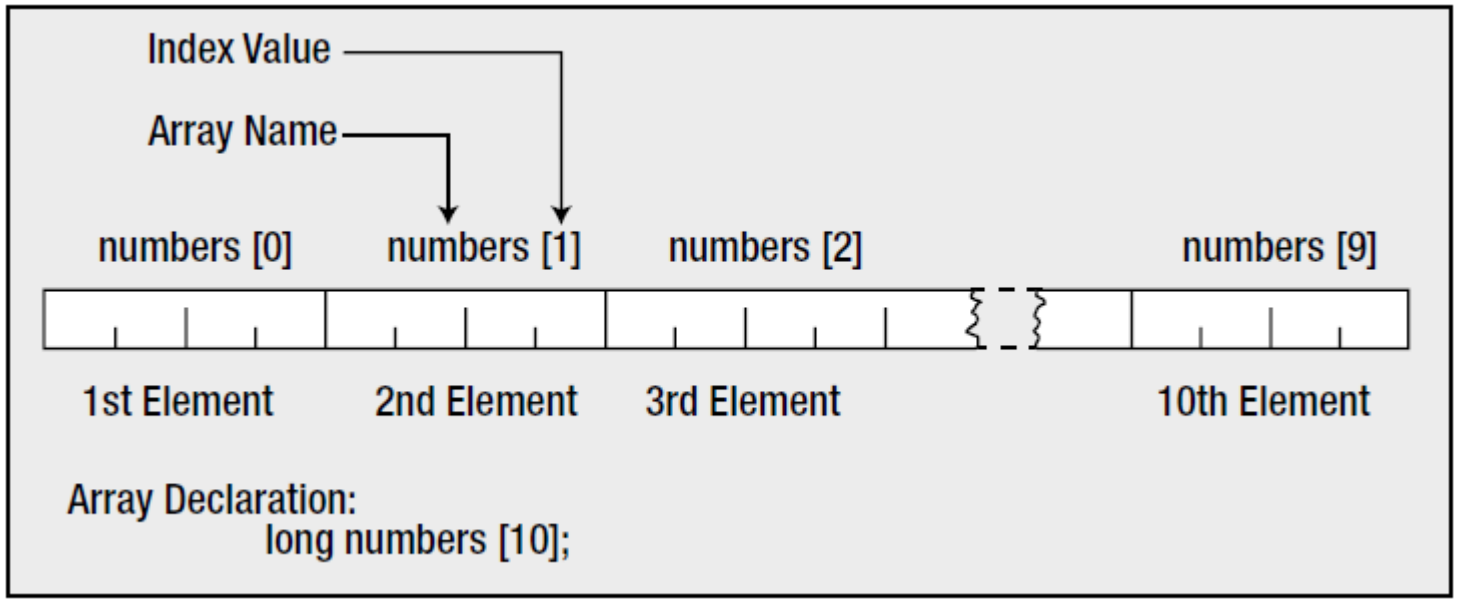
- **const Pointer auf const Variable**

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

Erster Blick auf Arrays

- **Variable Deklaration**

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



Mittelwert mit Arrays

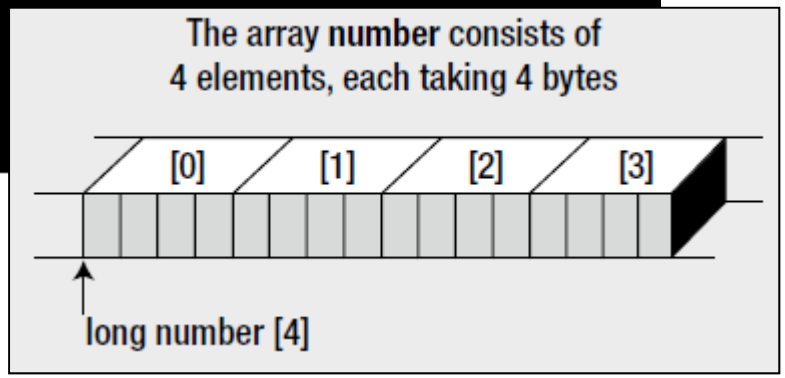
```c
// Program array.3 Averaging ten grades - storing the values the easy way
#include <stdio.h>

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

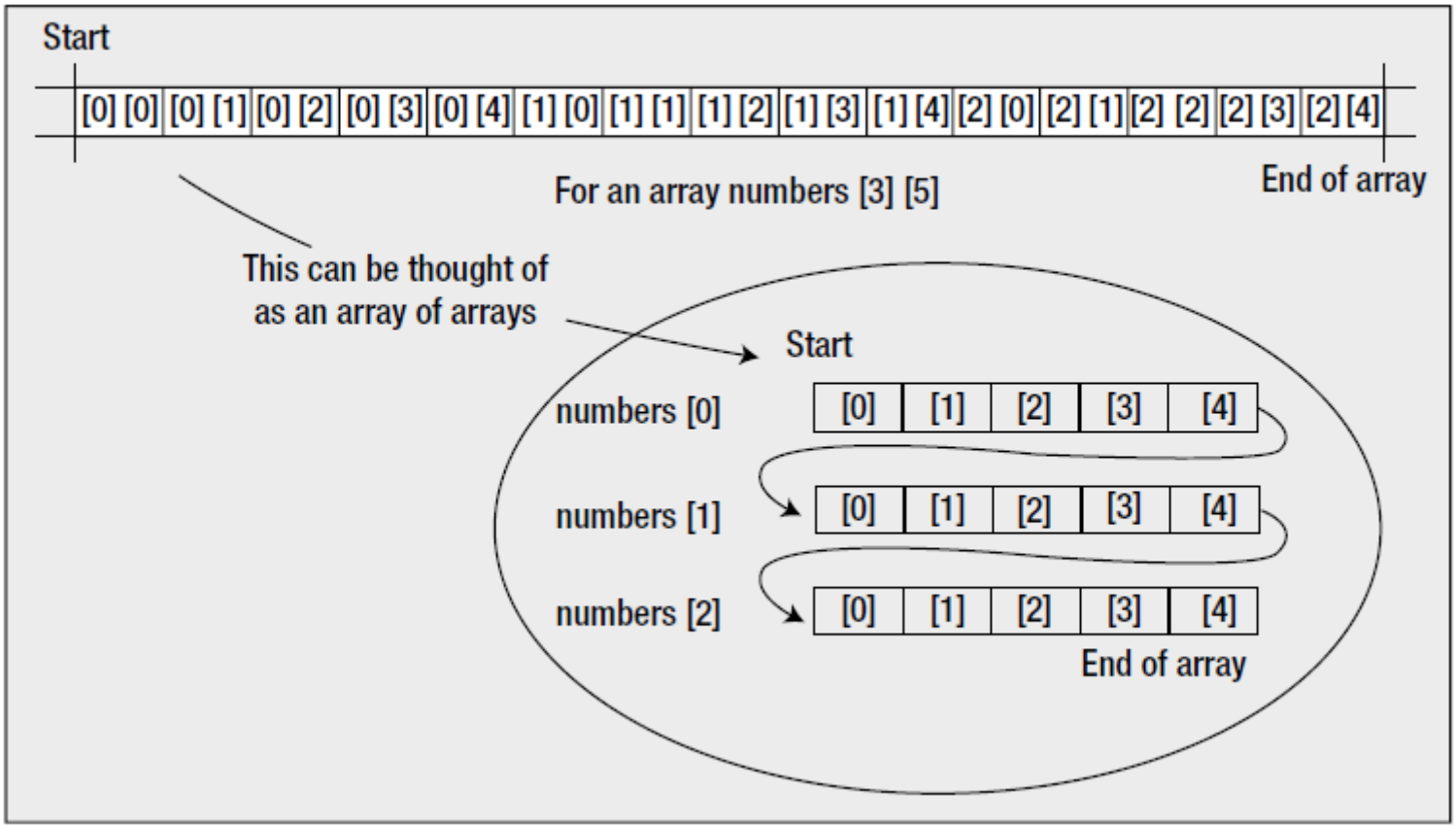
```c
// Program array.5b Find out the size of an Array
#include <stdio.h>

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;
}
```

01-Arrays – Aufgaben

1. Schreiben Sie ein Programm, das fünf Werte vom Typ double von der Tastatur einliest und Sie in einem Array speichert. Berechnen Sie den Kehrwert jedes Wertes (der Kehrwert von x ist 1.0 / x) und speichern Sie diese in einem separaten Array. Geben Sie jeden Kehrwert und die Summe aller Kehrwerte auf der Konsole aus.

2. Definieren Sie ein Array 'data' mit 100 Elementen vom Typ double . Schreiben Sie eine Schleife, die die folgende Sequenz von Werten im Arrays speichert: 1 / (2 * 3 * 4), 1 / (4 * 5 * 6), 1 / (6 * 7 * 8) ... bis zu 1 / (200 * 201 * 202) Schreiben Sie eine weitere Schleife, die das folgende errechnet : data [0] - data [1] + data [2] - data [3] + ... - data [99]. Multiplizieren Sie das Ergebnis mit 4.0, fügen 3.0 und Geben Sie das Ergebnis auf der Konsole aus. Erkennen Sie die Wert, den Sie bekommen ?

```
// Program array_06.c 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

  /*****************************************************************
   * 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;
}
```

TS-HF
Programmieren 2 (C)
09.03.2017

01-Arrays – Grupppenaufgabe Tic-Tac-Toe

Implementieren Sie das Spiel Tic-Tac-Toe für zwei Spieler:

```
 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9
Player 1, please enter a valid square number for where you want to place
your X:
```

Vorgehen:

1. Konzept erstellen
    a. Analyse (Was?)
       z.B. Spielbrett zeichnen
    b. Umsetzungsvorschlag (Wie?)
       z.B. ein zweidimensionales Array implementiert das Spielbrett.

2. Implementieren sie das Programm Schrittweise.
   z.B. im ersten Schritt den Code für den Game -Loop und das Spielbrett implementieren

- Was ist ein String
- String Operationen
- Analysieren von Strings
- Übung

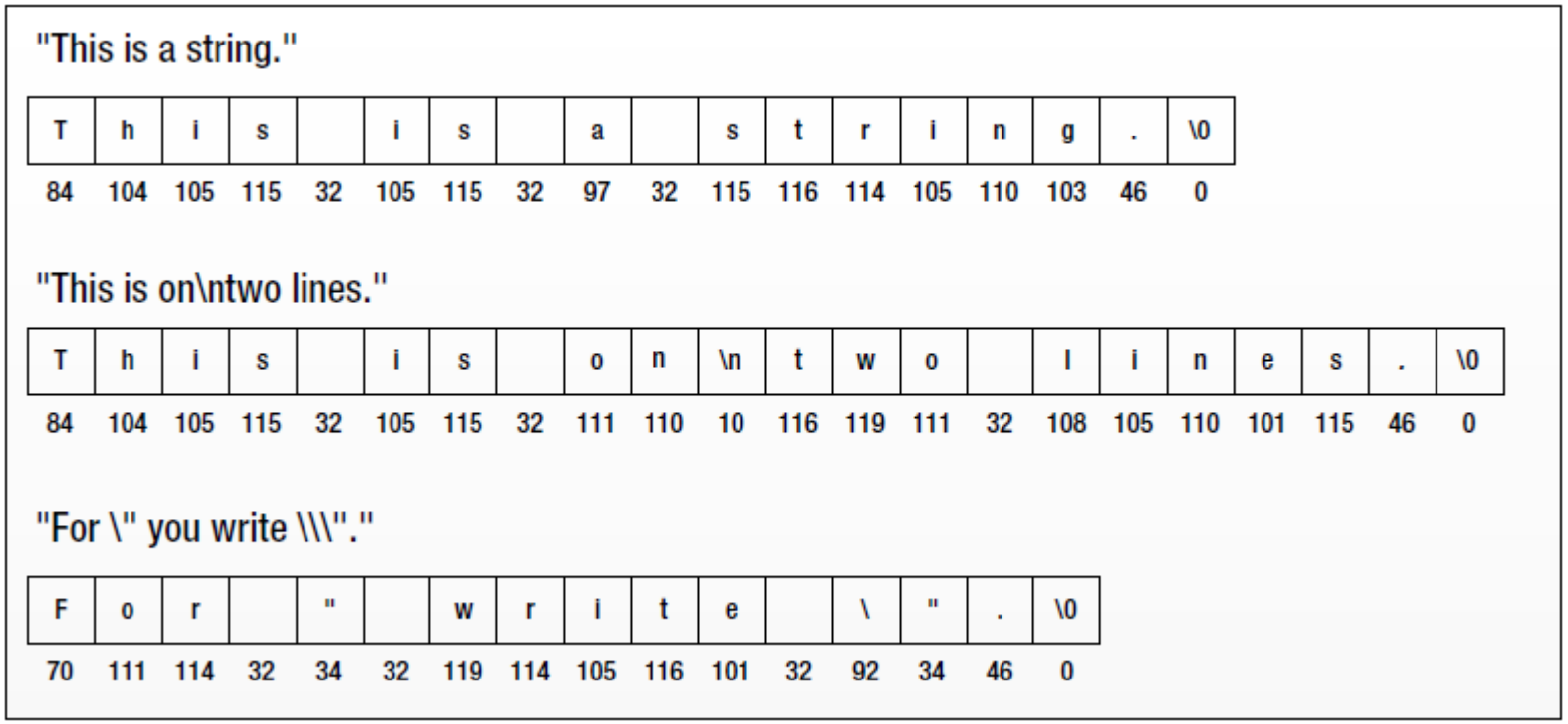
String

```
#include <stdio.h>

void main()
{
  printf("This is a string.");
  printf("This is on\ntwo lines!");
  printf("For \" you write \\\".");

}

>>
This is a string.This is on
two lines!For " you write \".
```

Stringvariablen:

```
char saying[20];
char saying[] = "This is a string.";
char str[40] = "To be";
const char message[] = "The end of the world is nigh.";

printf("\nThe message is: %s", message);
```

String – endet mit \0



Array of Strings

```
char sayings[3][32] = {
   "Manners maketh man.",
   "Many hands make light work.",
   "Too many cooks spoil the broth."
};


char sayings[][32] = {
   "Manners maketh man.",
   "Many hands make light work.",
   "Too many cooks spoil the broth."
};


for(unsigned int i = 0 ; i < sizeof(sayings)/
sizeof(sayings[0]) ; ++i)
   printf("%s\n", sayings[i]);
```

Stringoperationen - strlen

```
#include <stdio.h>
#include <string.h>

void main() {

  char str[][70] = {
     "Computers do what you tell them to do, not what you want them to do.",
     "When you put something in memory, remember where you put it.",
     "Never test for a condition you don't know what to do with.",
  };

  unsigned int strCount = sizeof(str)/sizeof(str[0]); // Number of strings

  for(unsigned int i = 0 ; i < strCount ; ++i) {
        printf("The string:\n \"%s\"\n contains %zu characters.\n",
                  str[i], strlen(str[i]));
     }
}

>>
The string:
 "Computers do what you tell them to do, not what you want them to do."
 contains 68 characters.
The string:
 "When you put something in memory, remember where you put it."
 contains 60 characters.
The string:
 "Never test for a condition you don't know what to do with."
 contains 58 characters.
```

Stringoperationen – strcpy / strncpy

```c
#include <stdio.h>
#include <string.h>

void main()
{
  char source[] = "Only the mediocre are always at their best.";
  char destination[50];
  strcpy(destination, source);
  printf("The copied string is\n%s", destination);

  char source2[] = "Only the mediocre are always at their best.";
  char destination2[50];
  strncpy(destination2, source2, 17);
  destination2[17] = '\0';
  printf("\n\nThe copied string is\n%s", destination2);
}
```

```
>>
The copied string is
Only the mediocre are always at their best.

The copied string is
Only the mediocre
```

Stringoperationen – strcat / strncat

```c
#include <stdio.h>
#include <string.h>

void main()
{
  char str1[50] = "To be, or not to be, ";
  char str2[] = "that is the question.";
  strcat(str1, str2);
  printf("The combined strings:\n%s\n", str1);

  char str12[50] = "To be, or not to be, ";
  char str22[] = "that is the question.";
  strncat(str12,  str22, 4);
  printf("\n\nThe combined strings:\n%s\n", str12);
}

>>
The combined strings:
To be, or not to be, that is the question.

The combined strings:
To be, or not to be, that
```

Stringoperationen – strcmp / strncmp

```c
#include <stdio.h>
#include <string.h>

void main()
{
  char str1[] = "The quick brown fox";
  char str2[] = "The quick black fox";
  if(strcmp(str1, str2) > 0)
     printf("str1 is greater than str2.\n");

  if(strncmp(str1, str2, 10) <= 0)
     printf("\n%s\n%s", str1, str2);
  else
     printf("\n%s\n%s", str2, str1);
}

>>
str1 is greater than str2.

The quick brown fox
The quick black fox
```

Stringoperationen – strchr

```
#include <stdio.h>
#include <string.h>

void main()
{
  char str[] = "Peter piper picked a peck of pickled pepper."; // The string to be searched
  char ch = 'p';                                 // The character we are looking for
  char *pGot_char = str;                         // Pointer initialized to string start
  int count = 0;                                 // Number of times found
  while(pGot_char = strchr(pGot_char, ch)) // As long as NULL is not returned...
     { // ...continue the loop.
        ++count;     // Increment the count
        ++pGot_char; // Move to next character address
     }
  printf("The character '%c' was found %d times in the following string:\n\"%s\"\n",
            ch, count, str);

  // Search str1 for the occurrence of str2
  char str1[] = "This string contains the holy grail.";
  char str2[] = "the holy grail";

  if(strstr(str1, str2))
     printf("\n\"%s\" was found in \"%s\"\n",str2, str1);
}

>>
The character 'p' was found 8 times in the following string:
"Peter piper picked a peck of pickled pepper."

"the holy grail" was found in "This string contains the holy grail."
```

Stringoperationen – strtok

?

Reading Strings – gets / fgets

?

Strings analysieren

```
#include <stdio.h>
#include <ctype.h>

  // islower()  Lowercase letter
  // isupper()  Uppercase letter
  // isalpha()  Uppercase or lowercase letter
  // isalnum()  Uppercase or lowercase letter or a digit
  // iscntrl()  Control character
  // isprint()  Any printing character including space
  // isgraph()  Any printing character except space
  // isdigit()  Decimal digit ('0' to '9')
  // isxdigit() Hexadecimal digit ('0' to '9', 'A' to 'F', 'a' to 'f')
  // isblank()  Standard blank characters (space, '\t')
  // isspace()  Whitespace character (space, '\n', '\t', '\v', '\r', '\f')
  // ispunct()  Printing character for which isspace() and isalnum() return false

void main()
{
  const char message[] = "The quick brown fox. 0123?!";
  int nLetters = 0;
  int nDigits = 0;
  int nPunct = 0;
  size_t i = 0;

  while(message[i])
     {
        if(isalpha(message[i]))
          ++nLetters;
        else if(isdigit(message[i]))
          ++nDigits;
        else if(ispunct(message[i]))
          ++nPunct;
        ++i;
     }
  printf("\nDie Message enthaelt %d Letters, %d Zahlen, %d Satzzeichen", nLetters, nDigits, nPunct);
}

>>
Die Message enthaelt 16 Letters, 4 Zahlen, 3 Satzzeichen
```

Strings umwandeln – toupper / tolower

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
  char text[100];
  char substring[40];
  size_t text_len = sizeof(text);
  size_t substring_len = sizeof(substring);

  printf("Geben sie einen Text ein (weniger als %d Buchstaben):\n", text_len);
  gets(text);
  printf("Geben sie einen Substring ein (wenige als %d Buchstaben):\n", substring_len);
  gets(substring);

  for(int i=0; (text[i] = (char)toupper(text[i])) != '\0'; i++);
  for(int i=0; (substring[i] = (char)toupper(substring[i])) != '\0'; i++);

  printf("Der Substring %s gefunden.\n", ((strstr(text, substring) == NULL) ? "wurde nicht" : '
}

>>
Geben sie einen Text ein (weniger als 100 Buchstaben):
The quick brown fox.
Geben sie einen Substring ein (wenige als 40 Buchstaben):
brown
Der Substring wurde gefunden.
```

Strings umwandeln – atof… / strtod…

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// atof()      Gibt einen Wert vom Typ double von einem String zueuck
// atoi()      Gibt einen Wert vom Typ int von einem String zurueck
// atol()      Gibt einen Wert vom Typ long von einem String zurueck
// atoll()     Gibt einen Wert vom Typ long long von einem String zurueck

// strtod()    Gibt einen oder mehrere Werte vom Typ double von einem String zurueck
// strtof()    Gibt einen oder mehrere Werte vom Typ float von einem String zurueck
// strtold()   Gibt einen oder mehrere Werte vom Typ long double von einem String zurueck
// strtoll()   Gibt einen oder mehrere Werte vom Typ long long von einem String zurueck
// strtol()    Gibt einen oder mehrere Werte vom Typ long von einem String zurueck
// strtoull()  Gibt einen oder mehrere Werte vom Typ unsigned long long von einem String zurueck

void main() {
  double value = 0;
  char str[] = "3.5 2.5 1.26";          // The string to be converted
  char *pstr = str;                     // Pointer to the string to be converted
  char *ptr = NULL;                     // Pointer to character position after conversion
  while(true)
     {
        value = strtod(pstr, &ptr); // Convert starting at pstr
        if(pstr == ptr)                 // pstr stored if no conversion...
          break;                        // ...so we are done
        else
          {
             printf(" %f", value);      // Output the resultant value
             pstr = ptr;                // Store start for next conversion
          }
     }
}

>>
 3.500000 2.500000 1.260000
```

Übung:

- Schreiben Sie ein Programm, das ein Text von der Tastatur einliest und die Häufigkeit aller Wörter ausgibt.
  - Die Gross- und Kleinschreibung soll nicht beachtet werden.
  - Der eingegeben Text kann innerhalb einer maximalen Länge, beliebig gross sein.

Uebung zum Stoff "String":

Es soll ein Programm entwickelt werden, das einen Text von der Tastatur einliest und die Häufigkeit der Wörter ausgibt. Zum Beispiel würde der Text "Hallo Walter, hallo Erich" die folgende Auswertung ausgeben:

```
>>
hallo       2
walter      1
erich       1
```

* Weitere Anforderungen:

Gross- und Kleinschreibung wird nicht beachtet. Wörter können entweder in Kleinbuchstaben oder in Grossbuchstaben konvetiert werden.

Die maximale Textgrösse ist festgelegt und darf bei der Eingabe nicht überschritten werden. Es wird also kein dynamische Memory alloziert, wenn der eingegeben Text grösser ist, als der für den Text zur Verfügung stehende Speicher, z.B ein char array[LENGTH].

Der Text soll solange eingelesen werden, bis der Benutzer eine Leerzeile eingibt.

* Konzept:

Mit fgets wird der Text eingelesen. Und zwar so lange, bis der String "\n" (Leerzeile) erkannt wird.

Der eingegeben Text wird zunächst in einem separaten Array gespeichert wie z.B. 'text[TEXT_LENGTH]'

Danach erfolgt das herauslesen der einzelnen Wörter.

Die verschiedenen Trennzeichen (Delimiters) wie z.B. ' ', '.', ',', ':', '!', '?', '(', ')'... sollen sicherstellen, dass nur Wörter in der Statistik ausgewertet werden.

Die einzelnen Wörter werden in einem Array von Strings wie z.B. 'char words[TOTAL_WORDS][WORD_LENGT]' eingelesen. Und zwar wird jedes Wort nur einmal darin gespeichert.

Für die Statistik muss die Häufigkeit jedes Wortes gespeichert werden. Die Häufigkeit wird in einem separaten Array gespeichert wie z.B. 'int word_count[TOTAL_WORDS]'

* Umsetzung Version 01:

In der ersten Version wird das Einlesen des Textes und das Abspeichern in einem separatem Array implementiert.

Ein Text kann mit mehreren Eingaben (New Line) eingegben werden. Alle Text-Teile werden zunächst im Buffer 'char buf[BUF_LENGTH]' gespeichert.

```
// version 01 ----------------------------------------------------
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>

#define BUF_LENGTH 10
#define TXT_LENGTH 1000

void main()
{
  char text[TXT_LENGTH];
  char buf[BUF_LENGTH];
  bool end = false;

  do {
        fgets(buf, BUF_LENGTH, stdin);
        if (strcmp(buf, "\n")== 0)
              {
                end = true;     // Leerzeile, bedeutet das Ende der Texteingabe
              }
        else
              {
                for(int i=0; (buf[i] = (char)tolower(buf[i])) != '\0'; i++);  // konvertieren zu Kleinbuchstaben
                strcat(text, buf);
              }
  } while(!end);
  printf("\n%s", text);

}
```

Frage(n):
Der Buffer 'buf' hat nur eine Grösse von 10 Zeichen. Das Programm funktioniert aber auch, wenn eine Text länger als 10 Zeichen ist. Weshalb?

Uebung zum Stoff "String":

Es soll ein Programm entwickelt werden, das einen Text von der Tastatur einliest und die Häufigkeit der Wörter ausgibt. Zum Beispiel würde der Text "Hallo Walter, hallo Erich" die folgende Auswertung ausgeben:

```
>>
hallo       2
walter      1
erich       1
```

* Weitere Anforderungen:

Gross- und Kleinschreibung wird nicht beachtet. Wörter können entweder in Kleinbuchstaben oder in Grossbuchstaben konvetiert werden.

Die maximale Textgrösse ist festgelegt und darf bei der Eingabe nicht überschritten werden. Es wird also kein dynamische Memory alloziert, wenn der eingegeben Text grösser ist, als der für den Text zur Verfügung stehende Speicher, z.B ein char array[LENGTH].

Der Text soll solange eingelesen werden, bis der Benutzer eine Leerzeile eingibt.

* Konzept:

Mit fgets wird der Text eingelesen. Und zwar so lange, bis der String "\n" (Leerzeile) erkannt wird.

Der eingegeben Text wird zunächst in einem separaten Array gespeichert wie z.B. 'text[TEXT_LENGTH]'

Danach erfolgt das herauslesen der einzelnen Wörter.

Die verschiedenen Trennzeichen (Delimiters) wie z.B. ' ', '.', ',', ':', '!', '?', '(', ')'... sollen sicherstellen, dass nur Wörter in der Statistik ausgewertet werden.

Die einzelnen Wörter werden in einem Array von Strings wie z.B. 'char words[TOTAL_WORDS][WORD_LENGT]' eingelesen. Und zwar wird jedes Wort nur einmal darin gespeichert.

Für die Statistik muss die Häufigkeit jedes Wortes gespeichert werden. Die Häufigkeit wird in einem separaten Array gespeichert wie z.B. 'int word_count[TOTAL_WORDS]'

* Umsetzung Version 01:

In der ersten Version wird das Einlesen des Textes und das Abspeichern in einem separatem Array implementiert.

Ein Text kann mit mehreren Eingaben (New Line) eingegben werden. Alle Text-Teile werden zunächst im Buffer 'char buf[BUF_LENGTH]' gespeichert.

```
// version 01 ------------------------------------------------------
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>

#define BUF_LENGTH 10
#define TXT_LENGTH 1000

void main()
{
  char text[TXT_LENGTH];
  char buf[BUF_LENGTH];
  bool end = false;

  do {
        fgets(buf, BUF_LENGTH, stdin);
        if (strcmp(buf, "\n")== 0)
              {
                end = true;    // Leerzeile, bedeutet das Ende der Texteingabe
              }
        else
              {
                for(int i=0; (buf[i] = (char)tolower(buf[i])) != '\0'; i++);  // konvertieren zu Kleinbuchstaben
                strcat(text, buf);
              }
  } while(!end);
  printf("\n%s", text);

}
```

Frage(n):
Der Buffer 'buf' hat nur eine Grösse von 10 Zeichen. Das Programm funktioniert aber auch, wenn ein Text länger als 10 Zeichen ist. Weshalb?

* Umsetzung Version 02:

In der zweiten Version werden die Wörter vom eingelesen Text mit strtok herausgelesen und im Array von Strings 'words' gespeichert.

Die Wörter werden aber nur einmal in 'words' gespeichert.

Am Ende des Programms werden die in 'words' enthaltenen Wörter ausgegeben.

```
// version 02 ------------------------------------------------------
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>

#define BUF_LENGTH 10
#define TXT_LENGTH 1000
#define TOTAL_WORDS 400
#define WORD_LENGTH 20
```

```
void main()
{
  char text[TXT_LENGTH];
  char buf[BUF_LENGTH];
  bool end = false;
  char delimiters[] = " .,:;()!?\n\"";
  char *pWord;
  char words[TOTAL_WORDS][WORD_LENGTH];
  int word_cnt = 0;
  bool word_fnd;


  do {
        fgets(buf, BUF_LENGTH, stdin);
        if (strcmp(buf, "\n")== 0)
              {
                end = true;    // Leerzeile, bedeutet das Ende der Texteingabe
              }
        else
              {
                for(int i=0; (buf[i] = (char)tolower(buf[i])) != '\0'; i++);  // konvertieren zu Kleinbuchstaben
                strcat(text, buf);
              }
  } while(!end);
  //printf("\n%s", text);

  // get the first word
  pWord = strtok(text, delimiters);
  if (pWord != NULL)
        {
              strcpy(words[0], pWord);
              word_cnt++;
        }

  while(pWord)
        {
              pWord = strtok(NULL, delimiters);
              if (pWord)
                {
                      // check if the word already exists
                      word_fnd = false;
                      for(int i=0; i<word_cnt; i++)
                            {
                              if (strcmp(words[i], pWord) == 0)
                                    {
                                          word_fnd = true;
                                    }
                            }
                      if (!word_fnd)
                            {
                              // a new word found, insert the word
                              strcpy(words[word_cnt], pWord);
                              word_cnt++;
                            }
                }
        }

  // print all words
  for (int i=0; i<word_cnt; i++)
        {
              printf("\n%s", words[i]);
```

```
        }
}
```

Frage(n):
Für die Statistik soll die Anzahl der gleichen Wörter, welche im Text enthalten sind, gezählt werden. Was wäre ein geeigneter Datentyp dafür?

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

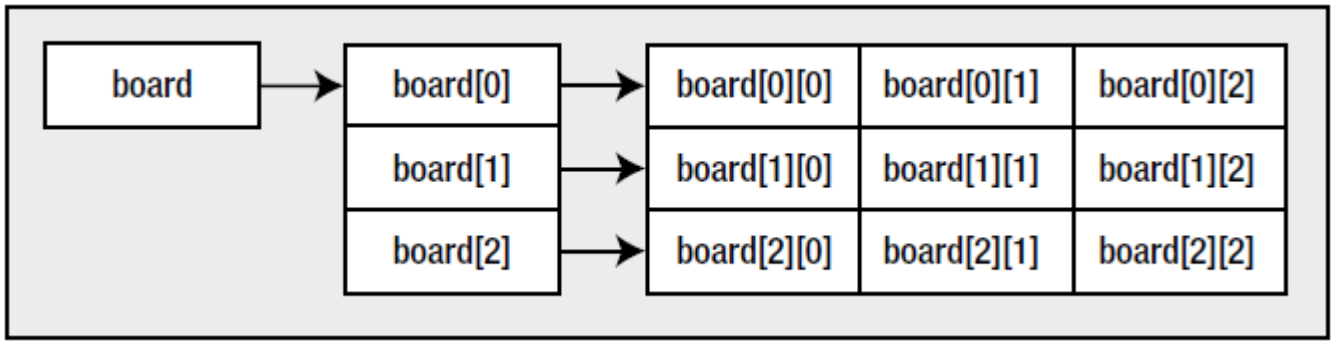
```c
// Program pointer_07a.c Two-dimensional arrays and pointers
#include <stdio.h>

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)

TS-HF
Programmieren 2 (C)
27.04.2017

Tag04 - Pointer und Array – Aufgaben

1. Schreiben Sie das "Hat-Size" Programm "array_06.c" neu (siehe Stoff von Tag02). Verwenden Sie Pointer und Pointer Indirektionen anstatt Arraynamen.

2. Schreiben Sie ein Rechnerprogramm mit den folgenden Eigenschaften:
   - es unterstützt positive und negative ganze Zahlen und Gleitkommazahlen
   - erlaubt Eingaben von mehrfachen Operationen wie z.B. 1.5 + 4.1 -7/7
   - mit dem ^ Operator werden Exponentialrechnungen ausgeführt z.B. 2 ^ 3 gibt 8
   - ein vorhergehendes Resultat kann mit = am Anfang des Input Strings in die Rechnung einfliessen, z.B. wenn ein früheres Resultat 3 war gibt =*2 + 7 gleich 15
   - die Eingabe "2 + 2*4 – 4*-2.22" wird als ((2 + 2)*4 -4) *(-2.22) gerechnet.

   Verwenden Sie Pointer-Notationen wo möglich.

   Konzeptvorschlag:
   1. Zuerst wird ein Input String eingelesen, welcher vom Benutzer eingegeben wurde.
   2. Das Programm wird beendet, wenn der Benutzer 'ende' eingab.
   3. Alle Spaces werden zuerst im Input String entfernt.
   4. das erste Zeichen wird auf '=' geprüft und falls es zutrifft, wird das frühere Resultat in die Rechnung einbezogen.

   …
   …

Uebung zum Stoff Array und Pointer:

Es soll ein Rechnerprogramm entwickelt werden mit den folgenden Eigenschaften:
- unterstützt positive und negative ganze Zahlen und Gleitkommazahlen
- erlaubt Eingaben von mehrfachen Operationen wie z.B. '1.5 + 4.1 -7/7'
- mit dem ^ Operator werden Exponentialrechnungen ausgeführt z.B. '2 ^ 3' gibt 8
- ein vorhergehendes Resultat kann mit '=' am Anfang des Input Strings in die Rechnung einfliessen, z.B. wenn ein früheres Resultat 3 war dann gibt '=*2 + 7' gleich 15
- die Eingabe '2 + 2*4 – 4*-2.22' wird als ((2 + 2)*4 -4) *(-2.22) gerechnet.

Wo möglich soll anstelle von Array-Notation Pointer-Notation verwendet werden: arr[i] -> *(arr + i)

Konzept:
---
1. Zuerst wird ein Input String eingelesen, welcher vom Benutzer eingegeben wurde.
2. Das Programm wird beendet, wenn der Benutzer 'ende' eingibt.
3. Alle Spaces im Input String werden vor der Verarbeitung entfernt.
4. Das erste Zeichen wird auf '=' geprüft und falls das zutrifft, dann wird das vorangegangene Resultat in die Rechnung einbezogen.

Implementation 1:

Zuerst wird die Rechnung eingelesen. Die Groesse der Rechnung ist auf 200 (siehe RECHNUNGLEN) begrenzt. Die fgets Funktion wird mehrmals aufgerufen, wenn der input_data Buffer kleiner (siehe BUFFLEN) als 200 ist. Nach dem Einlesen werden alle Leerzeichen entfernt.

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>


#define BUFFLEN 20
#define RECHNUNGLEN 200

void main()
{

  char input_data[BUFFLEN];
  char rechnung[RECHNUNGLEN] = {0};;
  bool rechnung_gelesen = false;


  // Die Rechnung einlesen
  printf("Gib die Rechnung ein (max. 200 Zeichen)\n");
  do
        {
              fgets(input_data, BUFFLEN, stdin);
              if ('\n' == *(input_data + strlen(input_data)-1))
                {
                      // eine Rechnung wurde erkannt ('\n' am Ende von input_data) -> entferne '\n'
                      *(input_data + strlen(input_data)-1) = '\0';
                      strcat(rechnung, input_data);
                      rechnung_gelesen = true;
                }
              else
```

```
            {
                // die eingelesene Rechnung ist noch nicht vollst   ändig
                strcat(rechnung, input_data);
            }
        }
  while (!rechnung_gelesen);
  printf("Rechnung: %s\n", rechnung);

  // Leerzeichen entfernen
  char *pch;
  while ((pch=strrchr(rechnung, ' ')) != 0 )
        {
            // letztes Leerzeichen in rechnung mit '\0' ersetzen und
            // den Rest von rechnung (ohne Leerzeichen) wieder anfuegen
            *pch = '\0';
            strcat(rechnung, pch+1);
        }
  printf("Rechung ohne Leerzeichen: %s\n", rechnung);
}
```

Implementation 2:

Im naechsten Schritt sollen die Operanden (-4.5, +3, ..) und die Operatoren (+, -, *, ..)
aus dem String (rechnung[]) extrahiert werden. Um Zahlen aus Zeichenketten zu extrahieren
gibt es in stdlib.h die String Umwandlungsfunktionen atof.../ strtod... (siehe Stoff: String)

```
// Program array_06.c 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

  /*****************************************************************
   * 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  */
       if(your_head == *(headsize+i))  // mit Pointer Noation
    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]) */
      if(your_head > *(headsize + (i - 1)) && your_head <= *(headsize + i))  // mit Pointer Notation
      {
        hat_found = true;
         break;
      }

  if(hat_found)
    printf("\nYour hat size is %c %c%c%c\n",
```

```
/*            size[0][i], size[1][i], */
          *(size[0]+i), *(size[1]+i),  // mit * Indirektion
/*            *(*size + i), *(*size + 12 + i), */
/*            (size[1][i]==' ') ? ' ' : '/', size[2][i]); // mit ** Indirektion */
          (*(size[1]+i)==' ') ? ' ' : '/', *(size[2]+i));  // mit * Indirektion
/*            (*(*size + 12 + i)==' ') ? ' ' : '/', *(*size + 24 + i)); mit ** Indirektion */
  // If no hat was found, the head is too small, or too large
  else
  {
/*     if(your_head < headsize[0])        // check for too small */
    if(your_head < *headsize)        // mit Pointer Notation
      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;
}
```

Pointer und Arrays and Malloc (3)

- malloc, calloc, free and realloc
- String Handling mit Pointer
- Handling Array of Pointers
- Pointers und Array Notation

Dynamische Speicher Allokation

- Erlaubt zur Laufzeit zusätzliches Memory anzufordern
- Dafür sind Pointer notwendig
- Die meisten Programme benutzen dynamische Memory
  z.B. Email Client
- Dynamisches Memory wird auf auf dem Heap gespeichert
  während lokale Variablen und Parameter auf dem Stack gespeichert werden
- Beim Beenden wird der Speicher auf dem Heap gelöscht

malloc, free, calloc and realloc

100 Byte allozieren:

`stdlib.h`

```
int *pNumber = (int*)malloc(100);
```

oder besser

```
int *pNumber = (int*)malloc(25*sizeof(int));
```

der cast ist nicht zwingend, wird von Compiler hinzugefügt

Speicher prüfen

```
int *pNumber = (int*)malloc(25*sizeof(int));
if (!pNumber){
  // memory allocation failure…
}
```

malloc, free, calloc and realloc

Speicher freigeben:

```
free(pNumber);
pNumber = NULL;
```

Schreibe ein Programm, welches eine unbestimmte Anzahl von Primzahlen berechnet:

1. 2, 3 und 5 sind die ersten drei Primzahlen
2. Primzahlen sind ungerade
3. Berechne die nächste Primzahl indem du die letzte Primzahl mit 2 addierst und mit allen bereits gefundenen Primzahlen testest .
4. Implementiere das Programm mit Pointers und dynamischen Memory
    1. Speichere die ersten drei bekannten Primzahlen gerade zu Beginn ab.
    2. Finde die weiteren Primzahlen
    3. Gib die gefunden Primzahlen aus.

malloc, free, calloc and realloc

```
int *pNumber = (int*)calloc(75, sizeof(int));
```

Die `calloc` Funktion benötigt zwei Parameter:

1. Anzahl
2. Grösse einer Speichereinheit `size_t`

Die `calloc` Funktion initialisiert alle Bytes mit 0

```
int *pPrimes =
     calloc(25, sizeof(unsigned long long));


if (pPrimes == NULL){
  printf("Not enough memory.");
}
```

malloc, free, calloc and realloc

Speicher erweitern (verkleinern oder vergrössern)

- Bei einer Vergrösserung hat der Inhalt des neuen Speicherinhalt zufällige Werte

```
int *pPrimes = calloc(25, sizeof(unsigned long long));
if (pPrimes == NULL) {
  printf(“Not enough memory.”);
}

...
...
pTemp = realloc(pPrimes, 100*sizeof(unsigned long long));
if (pTemp == NULL) {
  printf(“No memory reallocation.”);
  free(pPrimes);
  pPrimes = NULL;
}
pPrimes = pTemp;
...
```

malloc, free, calloc and realloc

Probiere es aus:

Ändere das Primzahlen Programm so ab, dass die max. Anzahl Primzahlen, welche zu berechnen sind, nicht mehr angegeben werden muss.

String Handling mit Pointer

```
char *pString = NULL;
```

Ein `char *Pointer` alloziert noch kein Memory für den Inhalt des Strings.

```
const size_t BUF_SIZE = 100;          // Input buffer size
char buffer[BUF_SIZE];                // A 100 byte input buffer
scanf_s("%s", buffer, BUF_SIZE);      // Read a string
// Allocate space for the string
size_t length = strnlen_s(buffer, BUF_SIZE) + 1;
char *pString = malloc(length);
if(!pString)
{
  printf("Memory allocation failed.\n");
return 1; }
strcpy_s(pString, length, buffer);  // Copy string to new memory
printf("%s", pString);
free(pString);
pString = NULL;
```

Handling Array of Pointers

```
char *pS[10] = NULL;
```

```
#define STR_COUNT  10                          // Number of string pointers
const size_t BUF_SIZE = 100;                   // Input buffer size
char buffer[BUF_SIZE];                         // A 100 byte input buffer
char *pS[STR_COUNT] = {NULL};                  // Array of pointers
size_t str_size = 0;
for(size_t i = 0 ; i < STR_COUNT ; ++i)
{
  scanf_s("%s", buffer, BUF_SIZE);             // Read a string
  str_size = strnlen_s(buffer, BUF_SIZE) + 1;  // Bytes required
  pS[i] = malloc(str_size);                    // Allocate space for the string
  if(!pS[i]) return 1;                         // Allocation failed so end
  strcpy_s(pS[i], str_size, buffer);           // Copy string to new memory
}
// Do things with the strings...
// Release the heap memory
for(size_t i = 0 ; i < STR_COUNT ; ++i)
{
  free(pS[i]);
  pS[i] = NULL;
}
```

Handling Array of Pointers

Probiere es aus:

- Gegeben ist das Programm “TextAnalyser”. Es durchsucht ein eingegebener Text nach unterschiedlichen Wörtern und gibt die verschiedenen Wörter zusammen mit der Häufigkeit aus.
- Ändere das Programm so ab, dass der untersuchte Text und die Wörter auf dem Heap gespeichert werden. Die Länge des eingegebenen Textes ist unbestimmt. Verwende ein Array of Pointer anstelle eines mehrdimensionalen Arrays.

Pointers und Array Notation

Für einen Pointer kann eine Array-Notation verwendet werden

```
int count = 100;
double* data = calloc(count, sizeof(double));
...
for(int i = 0 ; i < count ; ++i)
  data[i] = (double)(i + 1)*(i + 1);
```

Die Array Notation ist besser lesbar!

Probiere es aus:

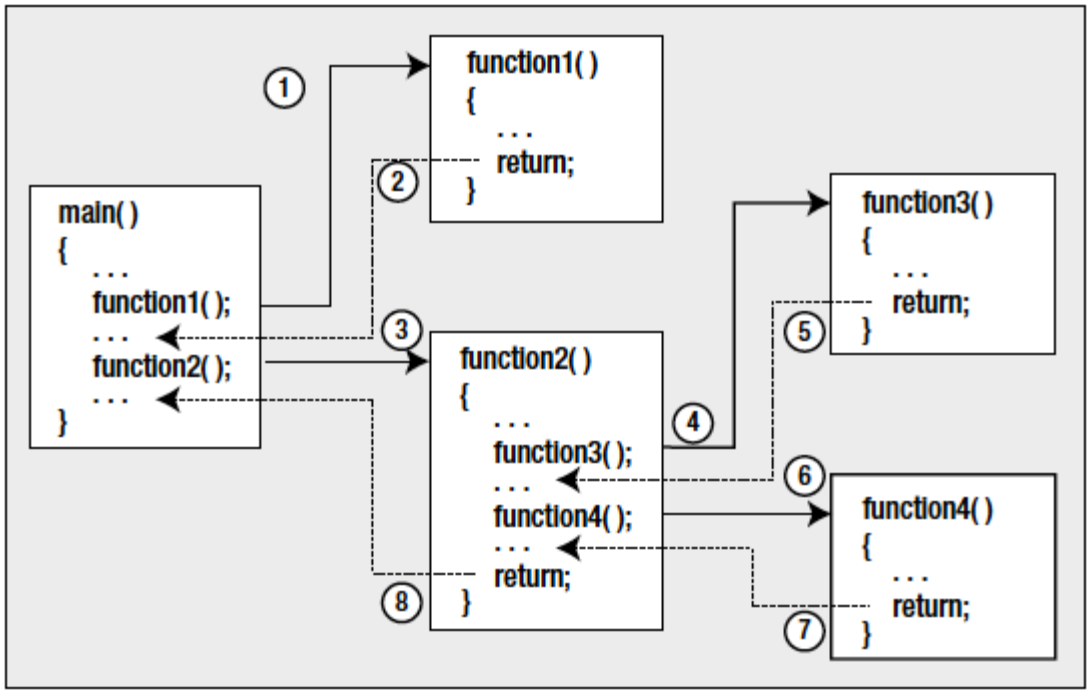
- Schreibe ein Programm, das mehrere eingegebene Sätze alphabetisch sortiert.

```
// Program sentence_sorter
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#define BUF_LEN  256
#define INIT_NSTR  2
#define NSTR_INCR  2

char* str_in();
void str_sort(const char**, size_t);
void swap(const char**, const char**);
void str_out(const char* const*, size_t);
void free_memory(char**, size_t);


int main(void)
{
  size_t pS_size = INIT_NSTR;
  char **pS = calloc(pS_size, sizeof(char*));
  if(!pS)
  {
    printf("Failed to allocate memory for string pointers.\n");
    exit(1);
  }

  char **pTemp = NULL;

  size_t str_count = 0;
  char *pStr = NULL;
  printf("Enter one string per line. Press Enter to end:\n");
  while((pStr = str_in()) != NULL)
  {
    if(str_count == pS_size)
    {
      pS_size += NSTR_INCR;
      if(!(pTemp = realloc(pS, pS_size*sizeof(char*))))
      {
        printf("Memory allocation for array of strings failed.\n");
        return 2;
      }
      pS = pTemp;
    }
    pS[str_count++] = pStr;
  }

  str_sort(pS, str_count);
  str_out(pS, str_count);
  free_memory(pS, str_count);
  return 0;
}



char* str_in(void)
{
```

```
    ... todo
}



void str_sort(const char **p, size_t n)
{
    ... todo
}


void swap(const char** p1, const char** p2)
{
    ... todo
}

void str_out(const char* const* pStr, size_t n)
{
    ... todo
}

void free_memory(char **pS, size_t n)
{
    ... todo
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>

#define MAX_BUF 5        // maximale Buffergroesse fuer den Satz. Wenn der Satz,
                         // dann muss mit realloc zusaetzlicher Speicher angefordert werden.

#define INC_BUF 5        // Bestimmt die Groesse fuer den zusaetzlichen Speicher.

#define MAX_TEXT 50      // Bestimmt die groesse fuer die Laenge des Satzes. Muss genuegend
                         // gross gewaehlt werden, da kein dynamisches Memory fur diesen Buffer
                         // implementiert ist.

int main()
{
  char text[MAX_TEXT];                    // Buffer fuer den Satz mit fgets einlesen

  char **sentence=malloc(MAX_BUF*sizeof(char*));    // Speichert die Saetze. Es koennen beliebig viele Saetze
gespeichert werden.

  size_t max_sentence=MAX_BUF;                  // Bestimmt die Anzahl Saetze die abgespeichert werden koennen.
Wenn diese
                                  // erreicht wird, dann muss wieder zusaetzlichen Speicher angefordert werden.

  size_t counter_sentence=0;                    // Bestimmt die Anzahl der gepeicherten Saetze

  bool sorted;                            // Flag fuer den Bubble-Sort. Wenn z.B. Beispiel das Array bereits sortiert
                                  // vorliegt, dann wird der Vergleich nach bereits nach einem Run beendet.

  while (true)
       {
            printf("Gib einen Satz ein oder enter fuer ende:\n");
            fgets(text, MAX_TEXT, stdin);
            if (text[0] == '\n') break;

            // entferne das New Line Zeichen
            text[strlen(text)-1] = '\0';

            // pruefe ob das sentence zusaetzliches Memory benoetigt
            if(counter_sentence == max_sentence)
              {
                   printf("more memory needed...\n");
                   max_sentence += INC_BUF;
                   sentence = realloc(sentence, max_sentence*sizeof(char*));
                   if (sentence == NULL)
                        {
                          return 2;          // 2 means memory error
                        }
              }
            *(sentence + counter_sentence) = calloc(strlen(text), sizeof(char));
            strcpy(*(sentence + counter_sentence++), text);
       }

  printf("Es werden %d Saetze sortiert...\n", counter_sentence);
```

```
  // Bubble-Sort
  sorted = false;
  int k=counter_sentence;
  char *tmpChr;
  while (!sorted)
        {
             sorted=true;
             for(int i=0; i<k-1; i++)
               {
                    printf("\ncompare sentences %s with %s", sentence[i], sentence[i+1]);
                    if(strcmp(sentence[i], sentence[i+1]) > 0)
                         {
            printf(" --> Swap sentences %s <-> %s", sentence[i], sentence[i+1]);
                          tmpChr = sentence[i];         // Swap
                          sentence[i] = sentence[i+1];  // Swap
                          sentence[i+1] = tmpChr;       // Swap
                          sorted = false;               // Das Array ist noch nicht sortiert.
                                // Es braucht min. einen zusaetzlichen Run.
                         }
               }
        }


  // Ausgabe der sortierten Saetze
  printf("\n\nSortierte Reihenfolge:\n");
  for(int i=0; i<counter_sentence; i++)
        {
             printf("\n%s", *(sentence + i));
        }
  printf("\n");


  // free all the memory
  for(int i=0; i<counter_sentence; i++)
        {
             free(*(sentence + i));   // gibt den Speicher fuer den Satz frei
             *(sentence +i) = NULL;   // Reset den Pointer auf NULL
        }
  free(sentence);              // gibt den Speicher fuer das Satz-Array frei
  sentence = NULL;             // Reset das Satz-Array auf NULL

  return 0;
}
```



Programm Struktur

Programm Struktur

- Variable Scope und Lifetime
- Variable Sope und Functions
- Functions Parameter
- Pointer als Parameter und Return Types

Variable Scope und Lifetime

```
{
int a = 0; // ...
// Reference to a is ? here
// Reference to b is ? here
{
int b = 10; // ...
// Reference to a and b is ? here
} // ...
// Reference to b is ? here
// Reference to a is ? here
}
```

- Variablen innerhalb von Blocks werden auch als “automatische” Variablen bezeichnet. Da automatisch angelegt und zerstört.

Variable Scope und Lifetime (2)

```c
int main(void)
{
    int count = 0; // Declared in outer block
    do
    {
        int count = 0; // This is another variable called count
        ++count; // this applies to inner count
        printf("count = %d\n", count);
    }
    while( ++count <= 5); // This works with outer count

    printf("count = %d\n", count); // Inner count is dead, this is outer count
    return 0;
}
```

- Was wird ausgegeben?
- Begründe deine Antwort

```
count = 1
count = 1
count = 1
count = 1
count = 1
count = 1
count = 6
```

Variable Scope und Functions

- Der Body von jeder Function ist ein Block
- Die automatischen Variablen sind local und nur innerhalb der Function gültig.
- Die Variable Deklaration einer Function ist unabhängig von anderen Variablen in anderen Functions

Functions

- Bestehend aus
  - Function Header
  - Function Body

```
Return_type Function_name ( Parameters –
separated by commas)
{
  // Statements…
}
```

Functions **Parameters**

Die Namen der Parameter sind lokal nur innerhalb der Funktion sichtbar.

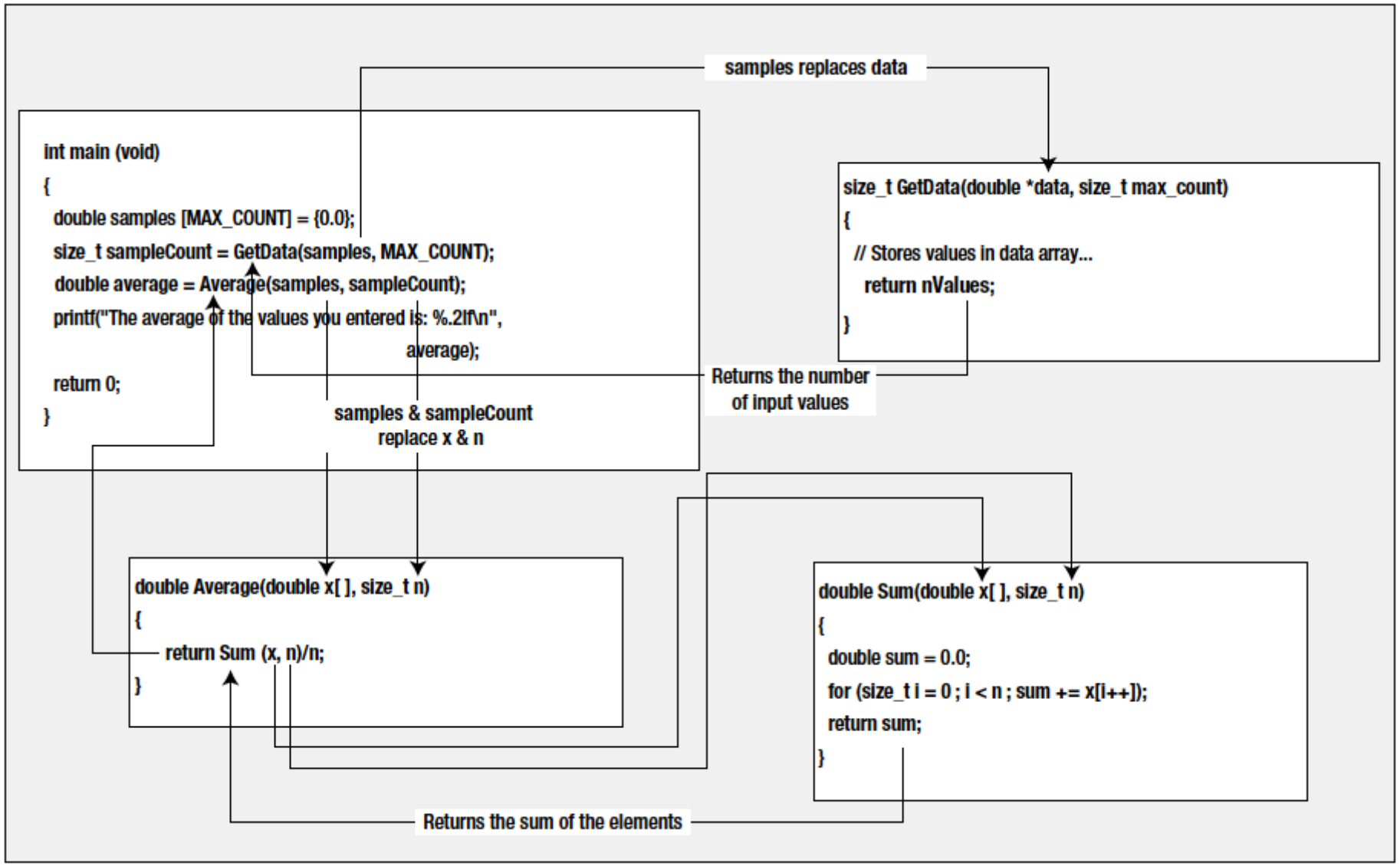
Beispiele:

```
bool SendMessage(char *text)

void PrintData(double *data, int count)

int SumIt(int x[], size_t n)

char* GetMessage(void)
```

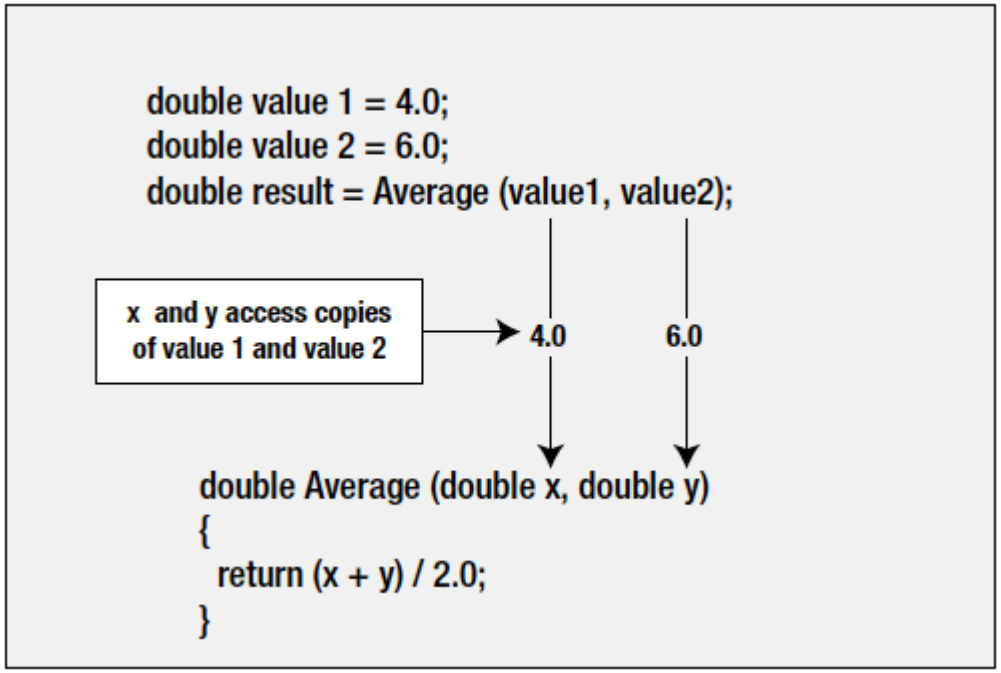
Beispiel - Programm Struktur

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#define MAX_COUNT 50

// Function to calculate the sum of array elements
// n is the number of elements in array x

double Sum(double x[], size_t n)
{
    double sum = 0.0;
    for(size_t i = 0 ; i < n ; ++i)
    sum += x[i];
    return sum;
}

// Function to calculate the average of array elements
double Average(double x[], size_t n)
{
    return Sum(x, n)/n;
}

// Function to read in data items and store in data array
// The function returns the number of items stored
size_t GetData(double *data, size_t max_coun t)
{
    size_t nValues = 0;

    printf("How many values do you want to enter (Maximum %zd)? ", max_count);

    scanf_s("%zd", &nValues);
    if(nValues > max_count)
    {
        printf("Maximum count exceeded. %zd items will be read.", max_count);
        nValues = max_count;
    }

    for( size_t i = 0 ; i < nValues ; ++i)
        scanf_s("%lf", &data[i]);

    return nValues;
}

// main program - execution always starts here
int main(void)
{
    double samples[MAX_COUNT] = {0.0};
    size_t sampleCount = GetData(samples, MAX_COUNT);
    double average = Average(samples, sampleCount);
    printf("The average of the values you entered is: %.2lf\n", average);

    return 0;
}
```

Beispiel - Programm Struktur



Pass-By-Value versus Pass-By-Reference




```
double Sum(double x[], size_t n)
{
double sum = 0.0;
for(size_t i = 0 ; i < n ; ++i)
sum += *(x++);
return sum;
}
```

Functions Prototypes

```
// #include & #define directives...

// Function prototypes
double Average(double data_values[], size_t count);
double Sum(double *x, size_t n);
size_t GetData(double*, size_t);

int main(void)
{
    // Code in main() ...
}

double Average(double x[], size_t n)
{
    return Sum(x, n)/n;
}

double Sum(double x[], size_t n)
{
    // Statements...
}

size_t GetData(double *data, size_t max_count)
{
    // Statements...
}
```

Pointer als Parameter und Return Types

- const Parameter

```
bool SendMessage(const char *text)
// text kann nicht verändert werden!
```

- const Parameter ist nur sinnvoll bei Pointer Parameter
  - Call by Value
- Vorteil
  - Compiler Prüfung!
    - Error wenn const -Daten verändert werden
    - Der Übergebene Pointer muss auf const zeigen
  - Impliziert, dass die Funktion keine Änderungen vornimmt

Pointer als Parameter und Return Types (2)

- Probiere es aus:
  - Analysiere das Programm `sentencesorter.c`
    - Kommentiere die Funktionsheader und die definierten Parametertypen
    - Implementiere die fehlenden Funktionsblöcke
    - Kommentiere das Programm vollständig

Gefahren bei Pointer Rückgabe (Returning Pointer)

- Ein numerischen Wert wird als Kopie zurückgeben
- Mit Hilfe von Pointer können mehrere Werte zurückgegeben werden (wobei auch der Pointer als Kopie zurückgeben wird)
  - z.B. String
- Aber, es gibt Gefahren beim Zurückgeben von Pointer

Gefahren bei Pointer Rückgabe (Returning Pointer) (2)

```c
#include <stdio.h>

long *IncomePlus(long* pPay); // Prototype for increase pay function

int main(void)
{
    long your_pay = 30000L; // Starting salary
    long *pold_pay = &your_pay; // Pointer to pay value
    long *pnew_pay = NULL; // Pointer to hold return value

    pnew_pay = IncomePlus(pold_pay);

    printf("Old pay = $%ld\n", *pold_pay);
    printf(" New pay = $%ld\n", *pnew_pay);

    return 0;
}

// Definition of function to increment pay
long* IncomePlus(long *pPay)
{
    *pPay += 10000L; // Increment the value for pay

    return pPay; // Return the address
}
```

Was ist der Output?

```
Old pay = $40000
  New pay = $40000
```

Gefahren bei Pointer Rückgabe (Returning Pointer) (2)

```c
#include <stdio.h>

long *IncomePlus(long* pPay); // Prototype for increase pay function

int main(void)
{
    long your_pay = 30000L; // Starting salary
    long *pold_pay = &your_pay; // Pointer to pay value
    long *pnew_pay = NULL; // Pointer to hold return value

    pnew_pay = IncomePlus(pold_pay);

    printf("Old pay = $%ld\n", *pold_pay);
    printf(" New pay = $%ld\n", *pnew_pay);

    return 0;
}

// Definition of function to increment pay
long *IncomePlus(long *pPay)
{
    long pay = 0; // Local variable for the result
    pay = *pPay + 10000; // Increment the value for pay

    return &pay; // Return the address of the new pay
}
```

Was ist der Output?

```
Old pay = $30000
  New pay = $27467656
```

```
// Program sentence_sorter
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#define BUF_LEN  256
#define INIT_NSTR  2
#define NSTR_INCR  2

char* str_in();
void str_sort(const char**, size_t);
void swap(const char**, const char**);
void str_out(const char* const*, size_t);
void free_memory(char**, size_t);


int main(void)
{
  size_t pS_size = INIT_NSTR;
  char **pS = calloc(pS_size, sizeof(char*));
  if(!pS)
  {
    printf("Failed to allocate memory for string pointers.\n");
    exit(1);
  }

  char **pTemp = NULL;

  size_t str_count = 0;
  char *pStr = NULL;
  printf("Enter one string per line. Press Enter to end:\n");
  while((pStr = str_in()) != NULL)
  {
    if(str_count == pS_size)
    {
      pS_size += NSTR_INCR;
      if(!(pTemp = realloc(pS, pS_size*sizeof(char*))))
      {
        printf("Memory allocation for array of strings failed.\n");
        return 2;
      }
      pS = pTemp;
    }
    pS[str_count++] = pStr;
  }

  str_sort(pS, str_count);
  str_out(pS, str_count);
  free_memory(pS, str_count);
  return 0;
}



char* str_in(void)
{
```

```
    ... todo
}



void str_sort(const char **p, size_t n)
{
    ... todo
}


void swap(const char** p1, const char** p2)
{
    ... todo
}

void str_out(const char* const* pStr, size_t n)
{
    ... todo
}

void free_memory(char **pS, size_t n)
{
    ... todo
}
```

Pointer auf Funktionen

- Eine Funktion hat eine Adresse im Memory
- Die Deklaration eines Funktionspointers braucht zusätzliche Informationen wie z.B.
  - Anzahl Parameter
  - Parametertyp
  - Return Typ

  ```
  int (*pfunction) (int);
  ```
- Die Deklaration deklariert einen Funktionspointer mit dem Namen "`pfunction`" mit einem Parameter vom Typ `int` und einem Return Typ vom Typ `int`

Pointer auf Funktionen: Beispiele

```
int sum(int a, int b);     // Calculates a+b
int (*pfun)(int, int) = sum;
int result = pfun(55, 65);
..
int product(int a, int b); // Calculates a*b
pfun = product;
Result = pfun(6, 15);
```

Pointer auf Funktionen: Beispiele

```c
#include <stdio.h>

// Function prototypes
int sum(int, int);
int product(int, int);
int difference(int, int);
int main(void)
{
  int a = 10; // Initial value for a
  int b = 5; // Initial value for b
  int result = 0; // Storage for results
  int (*pfun)(int, int); // Function pointer declaration

  pfun = sum; // Points to function sum()
  result = pfun(a, b); // Call sum() through pointer
  printf("pfun = sum result = %2d\n", result);

  pfun = product; // Points to function product()
  result = pfun(a, b); // Call product() through pointer
  printf("pfun = product result = %2d\n", result);

  pfun = difference; // Points to function difference()
  result = pfun(a, b); // Call difference() through pointer
  printf("pfun = difference result = %2d\n", result);

  return 0;
}
```

Array mit Pointer auf Funktionen

```
int (*pfunctions[10]) (int);
```

```
// Function prototypes
int sum(int, int);
int product(int, int);
int difference(int, int);

int main(void)
{
  int a = 10;                          // Initial value for a
  int b = 5;                           // Initial value for b
  int result = 0;                      // Storage for results
  int (*pfun[3])(int, int);            // Function pointer array declaration

  // Initialize pointers
  pfun[0] = sum;
  pfun[1] = product;
  pfun[2] = difference;

  // Execute each function pointed to
  for(int i = 0 ; i < 3 ; ++i)
  {
    result = pfun[i](a, b);            // Call the function through a pointer
    printf("result = %2d\n", result);  // Display the result
  }
}

int sum(int x, int y)
{
```

Funktionsparameter als Funktionspointer

```c
// Function prototypes
int sum(int,int);
int product(int,int);
int any_function(int(*pfun)(int, int), int x, int y);

int main(void)
{
  int a = 10;                       // Initial value for a
  int b = 5;                        // Initial value for b
  int result = 0;                   // Storage for results
  int (*pf)(int, int) = sum;        // Pointer to function

  // Passing a pointer to a function
  result = any_function(pf, a, b);
  printf("result = %2d\n", result );

  // Passing the address of a function
  result = any_function(product,a, b);
  printf("result = %2d\n", result );
}

// Definition of a function to call a function
int any_function(int(*pfun)(int, int), int x, int y)
{
  return pfun(x, y);
}
```

static Variablen in Funktionen

```c
// Function prototypes
void test1(void);
void test2(void);

int main(void)
{
  for(int i = 0 ; i < 5 ; ++i)
  {
    test1();
    test2();
  }
  return 0;
}

// Function test1 with an automatic variable
void test1(void)
{
  int count = 0;
  printf("test1   count = %d\n", ++count );
}

// Function test2 with a static variable
void test2(void)
{
  static int count = 0;
  printf("test2   count = %d\n", ++count );
}
```

```
test1 count = 1
test2 count = 1
test1 count = 1
test2 count = 2
test1 count = 1
test2 count = 3
test1 count = 1
test2 count = 4
test1 count = 1
test2 count = 5
```

Rekursion

- Eine Funktion ruft sich selber auf

```
void looper(void)
{
  printf(“Looper function called.\n”);
  Looper();
}
```

Führt zu einer undefinierter Anzahl von Printouts

Rekursion (2) – Beispiel: Grösster gemeinsamer Teiler

- *Rekursiv*:

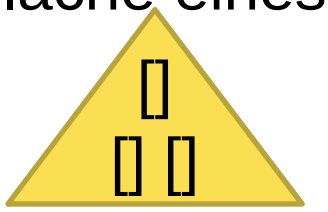
```
int greatest_common_divisor(int x, int y)
{
  return y == 0 ? X : greatest_common_divisor(y, x % y);
}
```

- *Iterativ*:

```
int greatest_common_divisor(int x, int y)
{
  while (true)
  {
    if (y == 0) break;
    int remainder = x % y;
    x = y;
    y = remainder;
  }
  return x;
}
```

Rekursion (3) – Vor und -Nachteile

- *Nachteile*
  - Weniger effizient
  - Gefahr von Stack Overflows
  - Funktionsparameter, Variablen und Return Adressen werden auf dem Stack gespeichert
- *Vorteile*:
  - Rekursive Funktionen sind in der Regel kürzer und eleganter
  - Es gibt Aglorithmen wie z.B. die Berechnung der Permutation die nur sehr schwer iterativ zu programmiern sind.
- Probiere es aus: Berechne die Fläche eines Dreiecks anhand einer gegebenen Breite.

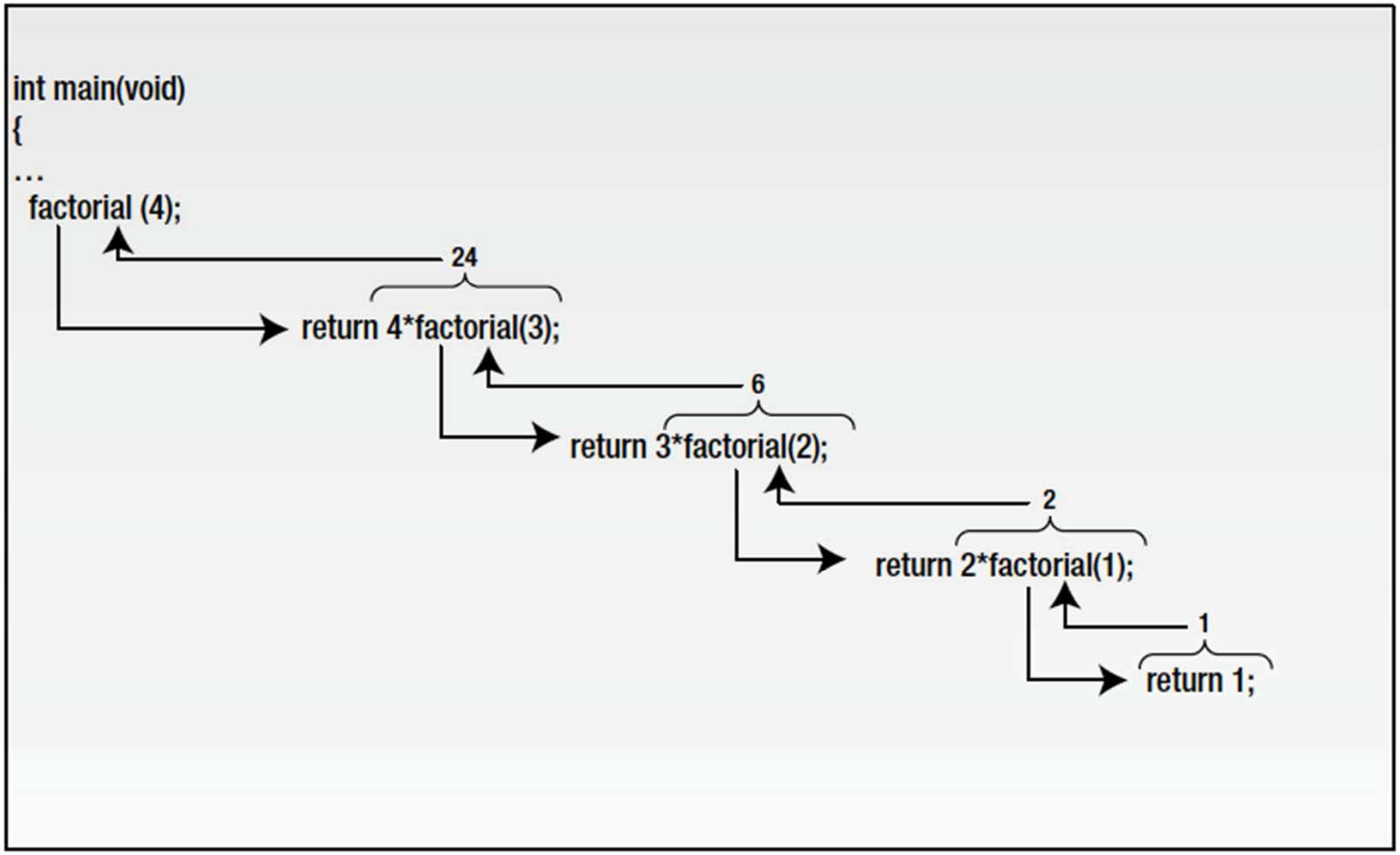


Breite = 2
Fläche = 3

Rekursion (4) – Beispiel: Fakultät (n!)

```c
#include <stdio.h>

unsigned long long factorial(unsigned long long);

int main()
{
    unsigned long long number = 0LL;
    printf("Enter an integer value: ");
    scanf("%llu", &number);
    printf("The factorial of %llu is %llu\n", number, factorial(number));
}

unsigned long long factorial(unsigned long long n)
{
    if (n < 2LL)
        return n;

    return n*factorial(n-1LL);
}
```

Rekursion (5) – Beispiel: Fakultät (n!)



Aufgabe: Othello (Spiel)

- Mache dich mit dem Spiel vertraut: https://de.wikipedia.org/wiki/Othello_(Spiel)
- *Analyse*
  - Bestimme die Programmstruktur
  - Definiere die Funktionen (Name, Parameter, Rückgabewerte)
  - Erstelle ein Flussdiagramm
- *Design*
  - Erstelle den Code
  - Dokumentiere den Code