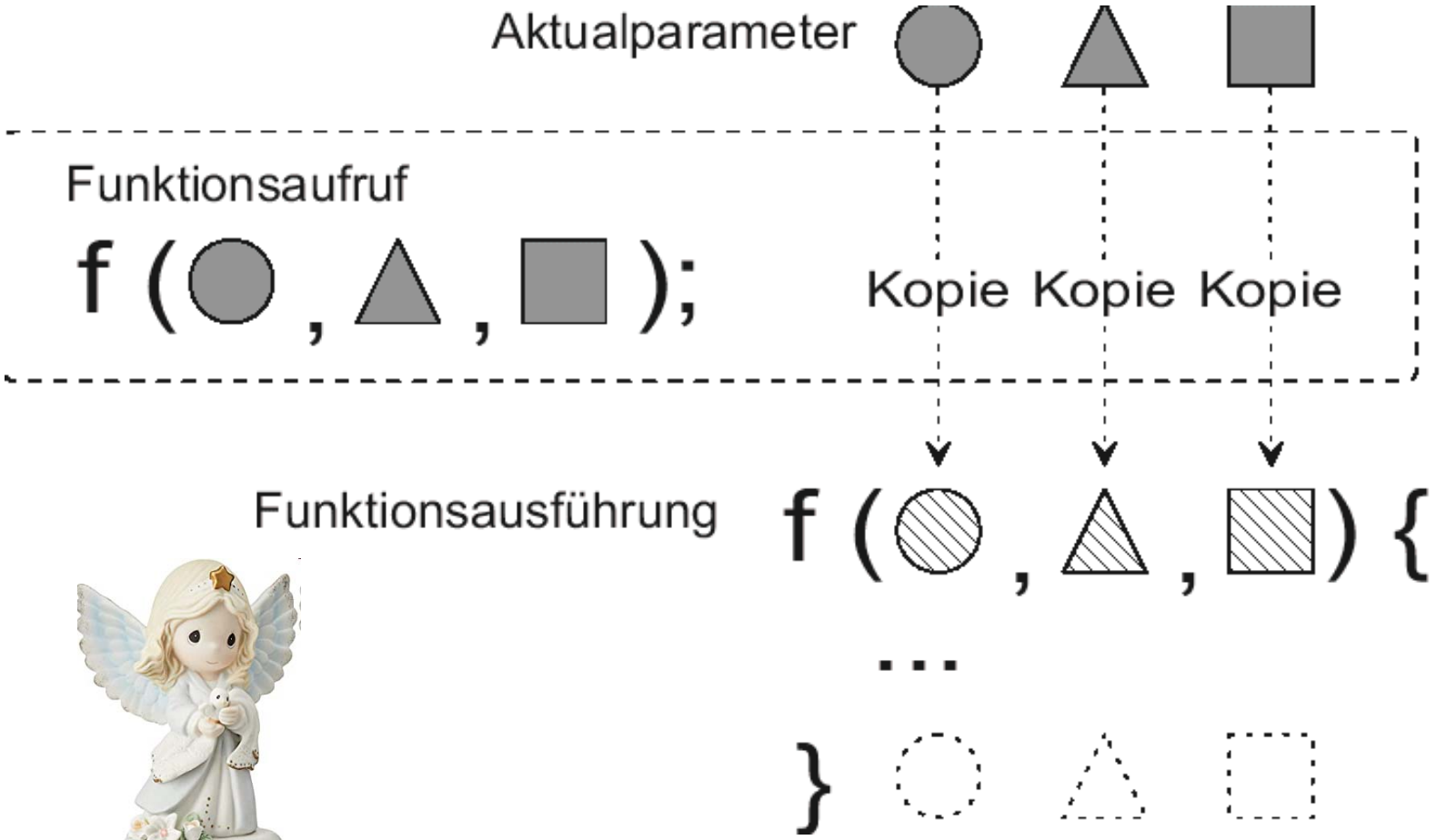
INF1
Mehr zu Funktionen und Strings (Saiten)

- Pass by Value, structs
- Pass by Reference
- Array Parameter
- Strings und Stringfunktionen

Pass by Value, Structs

Pass by Value

recycled



Parameterübergabe (Pass by Value)

- Die formalen Parameter der Funktion werden mit den aktuellen Parametern initialisiert
- Die formalen Parameter entsprechen lokalen Variablen in der Funktion
- Wenn diese in der Funktion geändert werden, hat dies keine Auswirkungen „nach aussen“
- Man arbeitet in der Funktion quasi *mit Kopien* der übergebenen Daten
- falls Ausdrücke übergeben werden, werden diese zuerst ausgewertet

Parameterliste


```
/**
*/

double dreiecksFlaeche(double a, double b, double c) {
    ...
    return flaeche;
}
```

- Beschreibung, Kommentar
- Typ der Funktion, d.h. der Typ des Rückgabewertes
- Name der Funktion
- Liste der Parameter, d.h. der Übergabewerte
- Anweisungsteil

Datenübergabe mit Hilfe von Parametern

In der Funktion: *Formale Parameter*

Liste von Variablen-definitionen

Entsprechen lokalen Variablen in der Funktion (beim Aufruf der Funktion initialisiert)

Parameterübergabe von struct by Value

- Datenstrukturen werden bei der Parameterübergabe (wie bei der Zuweisung) kopiert
- Die Funktion arbeitet also mit einer **lokalen Kopie**

```
void schreibeZeit(Zeit zeit) {
        ...

}
```

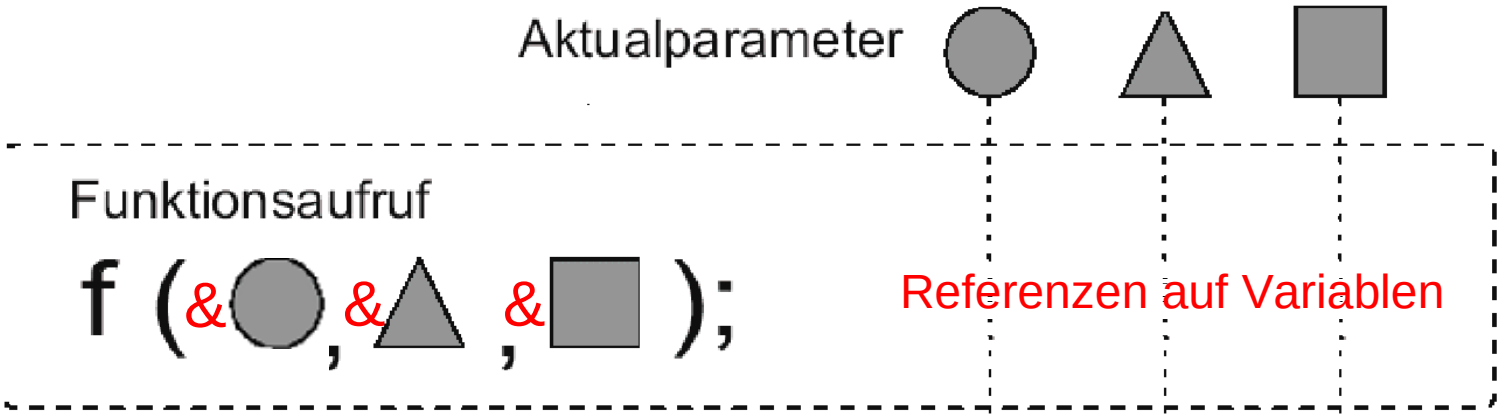
Parameterübergabe von struct by Value

```
typedef struct {int h, min, sec;} Zeit ;

int main (void) {
      Zeit beginn = { 9, 50, 00 };
      printf("%d:%d\n", beginn.h, beginn.min);
      plusViertelstunde(beginn);
      printf("%d:%d\n", beginn.h, beginn.min);
      return 0;
}

void plusViertelstunde(Zeit zeit) {
      if (zeit.min >= 45) zeit.h += 1;
      zeit.min = (zeit.min + 15) % 60;
      printf("%d:%d\n", zeit.h, zeit.min);
}
```

```
Ausgabe:

9:50
10:5
9:50
```

Pass by Reference, Pointer

Zeiger als Parameter: Pass by Reference



… Zeiger als Parameter: Pass by Reference

- Es werden nicht die Variablen übergeben, sondern lediglich Referenzen (Adressen, bzw. Zeiger) auf diese
- Die Variablen können dadurch innerhalb der Funktionen verändert werden
- Es ist also eine "Rückgabe" von Resultaten über die Referenz-Parameter möglich

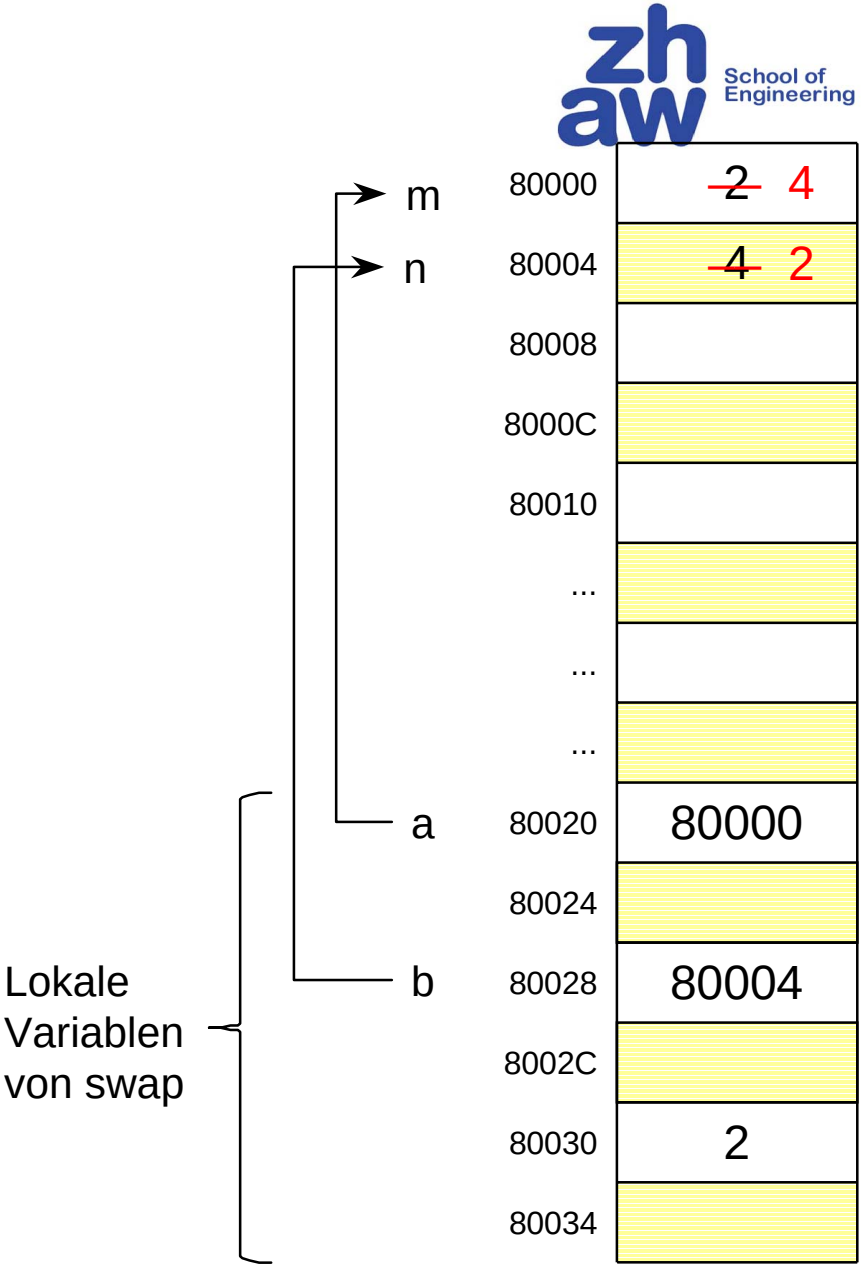
```
void inc(int *val) {
    (*val)++;
}
int main() {
    int i = 4;
    inc(&i);
    printf("%d",i); // 5
}
```

mit *val++ wird Pointer erhöht, Änderung geht nach Aufruf verloren: Pointer selber wird als "Wert" übergeben

```
void incp(int **ppint) {
    (*ppint)++;
}

int a[2] = {1,2};
int* pint = a;
incp(&pint);
printf("%d\n",*pint);
```

… Zeiger als Parameter

```
// Parameter sind Zeiger
void swap (int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Aufruf mit Adressoperator
int m=2, n=4;
swap(&m, &n);
```



… Zeiger als Parameter

- Bestimme Zeit differenz mit Struct

```
Zeit differenz(Zeit t1, Zeit t2){
  int d = abs((t1.sec+60*t1.min+3600*t1.h)
              -(t2.sec+60*t2.min+3600*t2.h));
  Zeit t = {d/3600,d/60%60,d%60};
  return t;
}
```

- Struct können auch zurückgegeben werden

```
Zeit differenz(Zeit* t1, Zeit* t2){
  int d = abs((t1->sec+60*t1->min+3600*t1->h)
              -(t2->sec+60*t2->min+3600*t2->h));
  Zeit t = {d/3600,d/60%60,d%60};
  return t;
}
```

Zeiger als Rückgabewert: Beispiel

- Zeiger können auch zurückgegeben werden

```
Zeit* besser(Zeit* t1, Zeit* t2){
  if(t1->sec+60*t1->min+3600*t1->h
              < t2->sec+60*t2->min+3600*t2->h)){
      return t1;
  }
  else {
      return t2;
  }
}
```

… Zeiger als Rückgabewert: Dangling References

- Probleme entstehen, wenn eine Funktion einen Zeiger auf eine **lokale Variable** zurückgibt
- Dieser Speicher **kann** nach Beenden der Funktion bereits anderweitig genutzt werden

```
Zeit* differenz(Zeit* t1, Zeit* t2){
  Zeit t;
  int d = abs((t1->sec+60*t1->min+3600*t1->h)
              -(t2->sec+60*t2->min+3600*t2->h));
  t = {d/3600,d/60%60,d%60};
  return &t;
}
```

```
dangling.c:3: warning: function returns address of local variable
```

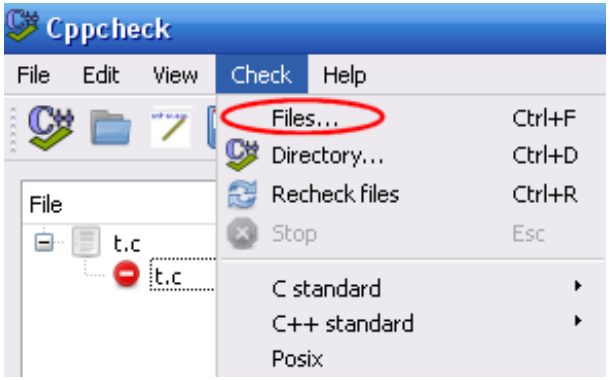
- **Sehr schwer zu findende Fehler! Compiler-Warnungen nicht ignorieren!**

Verwenden von Werkzeugen: CppCheck

outlook

- Es gibt eine Reihe von Werkzeugen um solche Fehler zu entdecken
- sog. statische Code Analyse, wie z.B. Lint und CppCheck

```
Zeit* differenz(Zeit* t1, Zeit* t2){
  Zeit t;
  int d = abs((t1->sec+60*t1->min+3600*t1->h)
            -(t2->sec+60*t2->min+3600*t2->h));
  t = {d/3600,d/60%60,d%60};
  return &t;
}
```



… Zeiger als Parameter

School of Engineering

Vorteile

- Werte in der aufgerufenen Funktion gesetzt und verändert werden.
- Effizienter für grosse Datenstrukturen (Pointer:4/8 Bytes statt Wert)

Nachteile

- Parameter müssen in der Funktion dereferenziert werden
- Bei jedem Aufruf muss für jeden Parameter die Übergabeart berücksichtigt und ggf. der Adresse-Operator eingesetzt werden
- "dangling" References auf Stack Variable

Array Parameter

Array-Parameter immer als Zeiger

- Bei der Übergabe von Arrays an Funktionen wird keine Kopie übergeben, sondern immer ein Zeiger auf das erste Array-Element
- Das bedeutet, dass in der Funktion keine lokale Kopie des Arrays zur Verfügung steht
- Änderungen das Arrays in der Funktion wirken sich auf das Originalarray aus

Array-Parameter: Beispiel

```
void function(int feld[]) {
    feld[3] = 33;
}

int main(void) {
    int feld[] = {1, 2, 3, 4, 5};
    printf(" %d", feld[3]);
    function(feld);
    printf(" %d", feld[3]);
    return 0;
}
```

Ausgabe?

Ausgabe?

Array-Parameter: Beispiel

```
void function(int *feld) {
    feld[3] = 33;
}

int main(void) {
    int feld[] = {1, 2, 3, 4, 5};
    printf(" %d", feld[3]);
    function(feld);
    printf(" %d", feld[3]);
    return 0;
}
```

gleich wie:

function(int feld[])

Array-Parameter: Beispiel

- Da die Grösse in der Funktion nicht bekannt ist, muss sie als Parameter übergeben werden

```
int vergleiche(Zeit t1, Zeit t2){
  return (t1.sec+60*t1.min+3600*t1.h)-(t2.sec+60*t2.min+3600*t2.h);
}

Zeit bestZeit(int teilnehmer, Zeit zeiten[]) {
    Zeit best = zeiten[0];
    int i;
    for (i = 1; i < teilnehmer; i++) {
        if (vergleiche(zeiten[i],best) < 0) {
            best = zeiten[i];
       }
    }
    return best;
}
```

Strings und Stringfunktionen

Strings in C

- Import. `#include <string.h>`
- Bereits häufig verwendet: Stringliterale
  "Dies ist ein Text"
- Tatsächlich handelt es sich dabei um ein
  Array von char-Werten
- Das Ende des Strings wird durch den Wert 0 markiert
  Als char Wert: '\0'
- Typ und Initialisierung:

```
char meldung[100];          // Platz für String bis zu 99 Zeichen
char fehler[] = "Error";    // char-Array der Laenge 6
```

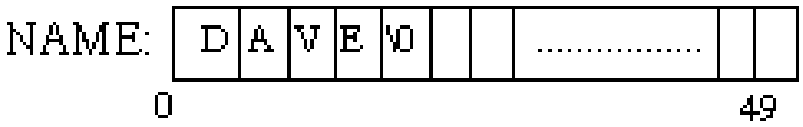
Char Konstanten

- Zeichen Konstanten wie in Java mit `''`
- Werte **`char`** 7 Bit, ASCII
- folgende Spezialzeichen sind definiert

end of string	NUL	\0	0
audible alert (bell)	BEL	\a	7
backspace	BS	\b	8
horizontal tab	HT	\t	9
newline	LF	\n	10
vertical tab	VT	\v	11
formfeed	FF	\f	12
carriage return	CR	\r	13
double quote	"	\"	34
single quote	'	\'	39
question mark	?	\?	63
backslash	\	\\	92

Arrays und Strings

- Grösse implizit durch Initialisierung bestimmen
  - `int i[] = {1,2,3,4,5};`



- Strings
  - Strings sind einfach Arrays of `char`
  - letzte Position immer ein '\0' Zeichen

  - Beispiel
  - `char hallo[10];`
  - `char hallo[] = {'H','a','l','l','o','\0'};`
  - `char hallo[] = "Hallo";`
  - `char hallo[10] = "Hallo";`

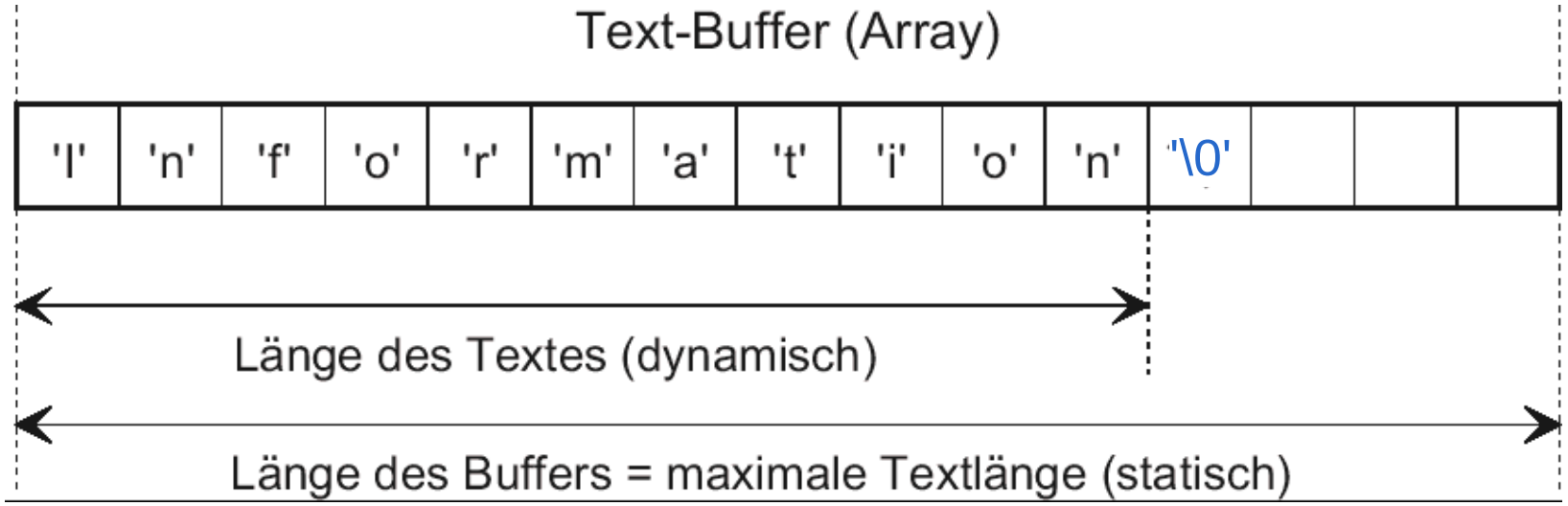
'\0' Zeichen für Stringabschluss

Grösse für Speicherallokation: muss gross genug sein, kann aber auch wesentlich grösser sein

- Achtung:Array muss immer **1 grösser** sein als String

Array von Zeichen, Strings

- Die String-Länge muss nicht immer Array-Länge minus 1 sein
- Häufig wird ein grösseres Array angelegt, das nach Bedarf mit einem String der Länge < Array-Länge belegt werden kann



Array von Zeichen, Strings

School of Engineering

- Array von Zeichen:

```
char c[] = {'0','1', '2', '3', '4', '5', '6', '7', '8', '9'};
```

- String:

```
char d[]= "0123456789";
```

- Worin unterscheiden sich c und d?
- Wie gross sind die Arrays jeweils?

String Funktionen

```
int strlen (char strg[]) {
    int index;      /* index into the string */
    /* Loop until we reach the end of string character */
    for (index = 0; strg[index] != '\0'; index+=1);
    return index;
}
```

Die Übergabe von Strings an eine Funktion entspricht der Übergabe von Arrays

Konstante Parameter

outlook

```
int strlen (const char strg[]) {
    int index;      /* index into the string */
    /* Loop until we reach the end of string character */
    for (index = 0; strg[index] != '\0'; index+=1);
    return index;
}
```

Parameter, die als *const* deklariert werden, können in einer Funktion nicht geändert werden

String Bibliothek

- C bietet in der String-Bibliothek *string.h* einige Funktionen zum Umgang mit Strings an
- Beispiel: strcpy zum Kopieren eines Strings in ein char-Array
  - direkte Zuweisung von Strings ist nicht möglich

```
#include <string.h>

int main (void) {
      char name[4]
      strcpy(name, "Sam");
      ...
      return 0;
}
```

String Bibliothek

- String kopieren **strcpy**

```
char hallo[12] = "Hallo";
char welt[12];
strcpy(welt, hallo);   // welt == "Hallo"
```

- Länge eines Strings: **strlen**

```
int i = strlen(str)
```

- String zusammenfügen **strcat**

```
char welt[] = " Welt";
strcat(hallo, welt); // hallo == "Hallo Welt"
```

- Gefährlich, da Speicher überschrieben wird, wenn zweiter String zu gross

  Lösung: breche nach n-1 Zeichen ab

```
strncpy(hallo,welt,12-1); hallo[11] = '\0';
strncat(hallo,welt,12-strlen(hallo)-1); // strncat fügt noch '\0' an
```

ANSI C String Bibliothek

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

`int strlen (s)`	Returns the **number of characters** in s, excluding the terminating null.
`char *strcpy (s1, s2)`	**Copies** s2 to s1, returning s1.
`char *strncpy (s1, s2, n)`	Like **strcpy**, except at most `n' characters are copied.
`char *strcat (s1, s2)`	**Concatenates** s2 onto s1. null-terminates s1. Returns s1.
`char *strncat (s1, s2, n)`	**Concatenates** s2 onto s1, stopping after `n' characters or the end of s2, whichever occurs first. Returns s1.
`int strstr (s1, s2)`	**Searches** for the string s2 in s1. Returns a pointer if found, otherwise the null-pointer.

in der Praxis meist diese Funktionen

… ANSI C String Bibliothek

`int strcmp (s1, s2)`	**Compares** strings s1 and s2. Returns an integer **that is less than 0 is s1 < s2; zero if s1 = s2; and greater than zero is s1 s2.**
`int strncmp (s1, s2, n)`	Like **strcmp**, except at most `n' characters are compared.
`char *strchr (s, c)`	**Searches** string s for character c. Returns a pointer to the first occurence, or null pointer if not.
`char *strrchr (s, c)`	Like **strchr**, except searches from the end of the string.
`int sprintf(str,format,a1,...)`	same as printf but the output will be stored in str
`int atoi(s1)`	Converts String to int
`double atod(s1)`	Converts String to double

sollte bekannt vorkommen

http://www.cplusplus.com/reference/clibrary/cstring/
http://openbook.galileocomputing.de/c_von_a_bis_z/030_c_anhang_b_020.htm

Übungen

- folgende Definition sei gegeben

```
char h1[] = "Hello ";
char h2[] = " World!!"
```

- Aufgabe
  - bestimmen Sie die Länge der Strings h1 und h2
  - prüfen Sie die beiden Strings auf Gleichheit
  - welches ist der letzte gültige Index-Positionen von h1 und h2
  - erstellen Sie einen String s, der den (zusammengehängten) Inhalt von h1 und h2 enthält (aufpassen wegen der Länge).

Array von Zeichen, Strings

- Was ist der Fehler?

```
char t[4] = "abcd";
```

- Der Zuweisungsoperator ist für Arrays und damit auch für Strings nicht definiert

```
char name[4];
name = "Sam";              /* Fehler !!
char neu[4] = "Sam"        /* zulässig  */
```

Einlesen von Strings in Console

- Einlesen mit scanf

```
char word[20];
scanf("%s",word) ;
```

- aber:
  - kein Check auf Überlauf
  - Abbruch bei Leerzeichen
- Deshalb besser

```
fgets(word, sizeof(word), stdin);
```

- oder

aber heikel; funktionert nur unter Windows; __fpurge unter Linux

scanf(" ") überliest alle Leerzeichen

```
fflush(stdin); // losche Tastatur Buffer
scanf("%19[^\n]s", pPerson->name); // Einlesen
```

- oder

Blank an erster Stelle um führende Leerzeichen zu überlesen

```
scanf(" %19[^\n]s", pPerson->name); // Einlesen
```

https://stackoverflow.com/questions/13542055/how-to-do-scanf-for-single-char-in-c

Strings in Datenstrukturen

- Strings kommen auch häufig in Datenstrukturen vor

```
typedef struct {
    char  name[20];
    int   jahrgang;
} Person ;

Person   person;
Person  *pPerson = &person; // Zeiger auf p
scanf(" %19[^\n]s", pPerson->name); // Einlesen
printf("%s", pPerson->name);  // Namen ausgeben
```

Einlesen von String bis cr und max 19 Zeichen

kein & da schon Pointer

Funktionszeiger

Funktionszeiger

outlook

- Der Name einer Funktion ist (wie ein Array-Name) eine Adresskonstante
- Es können deshalb Zeiger auf Funktionen definiert werden
  - **So kann man auch Funktionen als Parameter an Funktionen übergeben**
- Zeiger *fptr* auf eine Funktion, die *Typ* zurückgibt:

```
typedef int (*fptr)(Argumente);
```

Achtung: Klammern und * bei der Deklaration notwendig

Funktionszeiger, Anwendung Callback

outlook

- Einer Funktion A wird eine andere Funktion B als Parameter mitgegeben
- An einer geeigneten Stelle wird dann innerhalb der Funktion A die Funktion B "zurückgerufen"
  - z.B. um einen Wert zu bestimmen, dessen Berechnung man nicht in Funktion A "hard codieren" möchte

- Definition des Funktionstyps

```
typedef int (*functionB)();
```

- Als Parameter der Funktion A

```
functionA(functionB funcB)
```

- Auf-/Rückruf

```
functionA(functionB funcB) {
   …
   i = funcB();
```

- Übergabe der Funktion als Parameter

```
void foo() {
…
}

functionA(&foo())
```

Beispiel

outlook

- **Bestimmen der Bestzeit**

```
int vergleiche(Zeit t1, Zeit t2){
  return (t1.sec+60*t1.min+3600*t1.h)-(t2.sec+60*t2.min+3600*t2.h);
}

Zeit bestZeit(int teilnehmer, Zeit zeiten[]) {
    Zeit best = zeiten[0];
    int i;
    for (i = 1; i < teilnehmer; i++) {
        if (vergleiche(zeiten[i],best) < 0) {
                best = zeiten[i];
       }
    }
    return best;
}

Zeit best = bestZeit(teilnemer, zeiten);
```

… Beispiel

outlook

Signatur der Vergleichsfunktion

```
typedef int (*Comparator)(Zeit, Zeit);
int vergleiche(Zeit t1, Zeit t2){
  return (t1.sec+60*t1.min+3600*t1.h)-(t2.sec+60*t2.min+3600*t2.h);
}
Zeit bestZeit(int teilnehmer, Zeit zeiten[], Comparator compare) {
    Zeit best = zeiten[0];
    int i;
    for (i = 1; i < teilnehmer; i++) {
        if (compare(zeiten[i],best) < 0) {
            best = zeiten[i];
        }
    }
    return best;
}
Zeit best = bestZeit(teilnemer, zeiten, &vergleiche);
```

Funktionsparameter

Aufruf der Funktion

Übergabe der vergleiche Funktion

… Beispiel

outlook

- Es kann nun ein anderes Kriterium für Bestimmung der Bestzeit verwendet werden
  - **z.B. maximale Zeit**

```
typedef int (*Comparator)(Zeit, Zeit);

int vergleiche2(Zeit t1, Zeit t2){
  return -((t1.sec+60*t1.min+3600*t1.h)-(t2.sec+60*t2.min+3600*t2.h));
}

Zeit best = bestZeit(teilnemer, zeiten, &vergleiche2);
```

Vorzeichenwechsel

Noch Fragen?