ANSI C

Arrays und Pointer, Ueberblick

```
// ***************************************************************************
//
//                              ARRAYS und POINTER
//
// ***************************************************************************



// ---------------------------------------------------------------------------
//                       Variablen-Namen sind Adressen!
// ---------------------------------------------------------------------------

'Z'         // Zeichen/Character (ein Byte)

"ABCDE"     // String-Konstante (= Byteblock):
// hier 6 Bytes gross (inkl. '\0' als Abschluss).
// Als Referenz gilt die Adresse, an der das 1. Element liegt (hier 'A').

char c = 'Z';
// Adresse c enthält ein Byte mit dem Wert 0x5A (Zeichen 'Z').

int i = 12345;
// An der Adresse i und folgenden liegen 2 oder 4 Bytes (je nach Compiler und uC),
// die den Wert 12345d enthalten.



// ***************************************************************************
//
//                             ARRAY oder VEKTOR
//                (= Speicherblock ab konstanter Adresse)
//
// ***************************************************************************
// Ein Array (oder Vektor) ist ein Byte-Block, dessen Name für die Adresse
// des ersten Elements steht.

char szMeldung[] = "ABCDE";
// Ein String-Block angelegt im Speicher ab Adresse szMeldung.
// Ein Null-Byte ('\0') als Abschluss wird vom Compiler hinzugefügt.
// Inhalt  : Block mit Zeichen 'A' ... '\0'
// Adresse : szMeldung + 0 enthält 'A'
//           szMeldung + 1 enthält 'B'
//           ...
//           szMedlung + 4         'E'
//           szMeldung + 5         '\0'
```

```
#define MAX_SIZE  10

unsigned char ucBuffer[MAX_SIZE];
// Speicher-Block mit 10 char-Elementen 0...9 ab Adresse ucBuffer
// Inhalt  : noch undefiniert
// Adresse : ucBuffer + 0
//           ucBuffer + 1
//           ...
//           ucBuffer + 9   // MAX_SIZE

// ---------------------------------------------------------------------
//      a[i]  :  Der Index i ist ein Offset zur konstanten Adresse a
// ---------------------------------------------------------------------

char a[index];   // Array
// index ist immer als Offset zur Adresse (= Array-Name)
// zu betrachten: ((Adresse a) + index).


// AUSNAHME: bei Pointern (Adress-Variablen)
char *pa;        // Pointer
// hier ist pa[index] gleichbedeutend mit ((Inhalt von pa) + index).


// WICHTIG: Der Programmierer ist dafür verantwortlich, dass die Array-
// grenzen nicht über- oder unterschritten werden, gültig sind hier
// Indizes von [0] bis [MAX_SIZE-1].


// ***********************************************************************
//
//                                POINTER
//        (= "Gefäss" für eine Adresse, welche variabel sein kann)
//
// ***********************************************************************
// Ein Pointer (oder Zeiger) ist eine Variable, die eine Adresse enthält.

// WICHTIG: Ein Pointer muss immer initialisiert werden, damit
// er auf ein gültiges Objekt zeigt.

// Beispiele: Pointer initialisieren

int n;
int *pn = &n;
// Adresse zuweisen mit Adressoperator


char *pText;
// Inhalt  : eine noch undefinierte (!) Adresse als Variable
// Adresse : pText


// char-Pointer initialisieren mit Array-Null-Adresse:
// Adresse des Elements 0 zuweisen
//
// Variante 1 (ist zu bevorzugen):
pText = &ucBuffer[0];

// Variante 2:
// abgekürzt ohne Adressoperator (nur bei Array-Namen möglich,
// da diese vom Compiler wie Pointer behandelt werden!)
pText = ucBuffer;
```

```
// jedoch: die Zuweisung
ucBuffer = pText;
// FUNKTIONIERT NICHT, da ucBuffer eine Konstante (Adresse) ist!


char *pMeldung = "ABCDE";
// String-Block angelegt irgendwo im Speicher
// Null-Byte ('\0') als Abschluss wird vom Compiler hinzugefügt.
// Inhalt : die Adresse des Zeichens 'A' (an der Pointer-Adresse pMeldung.)

pMeldung = "FGHIJK";
// Einem Pointer kann eine neue Adresse zugewiesen werden, da er
// eine Variable ist ("Pointer umbiegen").


// ------------------------------------------------------------------------
// Pointer: Dereferenzierung mittels den Operatoren: *  []  ->
// ------------------------------------------------------------------------
// Dereferenzierung bedeutet:
// Zugriff auf das Objekt (die Variable) auf das ein Pointer zeigt.

// Beispiele:

char a[10] = "ABCDE";   // Name a ist Adresse des 0-ten Elements 'A'
char *pa   = &a[0];     // Adresse des 0-ten Elements an pa
char c;

// Dereferenzierung mit den Operatoren * oder []

c = *pa;        /* oder */   c = a[0];   // c enthält 'A'

c = *(pa + 2);  /* oder */   c = a[2];   // c enthält 'C'

c = pa[2];      /* oder */   c = a[2];   // c enthält 'C'


// Dereferenzierung mit dem Operator ->

value_k = ps->k;
// ps ist ein Pointer auf eine struct oder class, k eine Komponente
// auf deren Inhalt zugegriffen wird.


// ------------------------------------------------------------------------
// Beispiel: Uebergabe eines String-Arrays an den printf-Befehl.
// ------------------------------------------------------------------------
// Die Uebergabe erfolgt über den Array-Namen oder einen Pointer.
// Die folgenden 4 Varianten geben "CDE" aus, sind also identisch

char a[] = "ABCDE";
char *pa = &a[0];

printf ("  %s  \r\n", (a + 2));     // Array-Name (= Adresse!) + Offset
printf ("  %s  \r\n", (&a[2]));     // Adress-Operator mit (Array-Name + Index)
printf ("  %s  \r\n", (pa + 2));    // Pointer (Pointer-Inhalt + Offset)
printf ("  %s  \r\n", (&pa[2]));    // Adress-Operator mit (Pointer-Inhalt + Index)
```

```
// ***************************************************************************
//
//                          KURZ: ARRAY vs. POINTER
//
// ***************************************************************************

char aMessage[] = "Nachricht";   // Array (Vektor)
// Speicherblock (hier als String) liegend ab Adresse aMessage.

char *pMessage  = "Nachricht";   // Pointer-Variable
// enthält die Adresse vom Zeichen 'N' des Strings, dieser liegt irgendwo
// im Speicher (meist im ROM/Flash).



// ***************************************************************************
//
//                                BEISPIELE
//
// ***************************************************************************

// ---------------------------------------------------------------------------
// String-Länge bestimmen
int StrLen(const char *s)
{
   int i;
   for (i = 0; *s++; i++);
   return i;
}
// ---------------------------------------------------------------------------
// String src nach dest kopieren
char *StrCopy(char *dest, const char *src)
{
   char *s = dest;
   while (*dest++ = *src++);
   return s;
}
// ---------------------------------------------------------------------------
// Integer-to-String Konversion, Base 10/16
#define LEN 8

char *IntToStr(int val, unsigned char base)
{
    static char buf[LEN];
    int i;

    buf[LEN - 1] = 0;    // String-Ende: Null-Byte
    i = LEN - 2;         // Zeichen-Zähler (rückwärts ab String-Ende)

    if (!val) {
       buf[i] = '0';
       return &buf[i];
    }

    while (val && i) {
       buf[i] = "0123456789ABCDEF"[val % base];
       i--;
       val /= base;
    }
    return &buf[i+1];
}
// ende
// ---------------------------------------------------------------------------
```

***