Einführung in die Programmierung

Bachelor of Science

Prof. Dr. Rethmann

Fachbereich Elektrotechnik und Informatik
Hochschule Niederrhein

WS 2009/10

Übersicht

- Arrays
- Datentypen, Operatoren und Kontrollstrukturen
- Funktionen und Zeiger
- *Zeichenketten und Strukturen*
- Dateioperationen und Standardbibliotheken
- strukturierte Programmierung
- modulare Programmierung

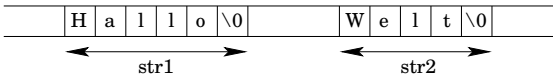
Zeichenketten

Zeichenketten belegen einen zusammenhängenden Speicherbereich und können daher als Array betrachtet werden.

```
#include <stdio.h>

void main(void) {
    char str1[] = {'H','a','l','l','o','\0'};
    char str2[] = "Welt";

    printf("%s, %s!\n", str1, str2);
}
```



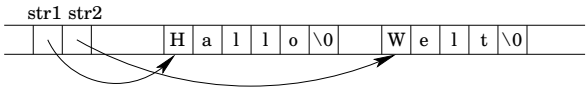
Zeichenketten

Eine Array-Deklaration definiert einen Zeiger auf das erste Element des Arrays.

```
#include <stdio.h>

void main(void) {
    char str1[] = "Hallo";
    char *str2 = "Welt";

    printf("%s, %s!\n", str1, str2);
}
```

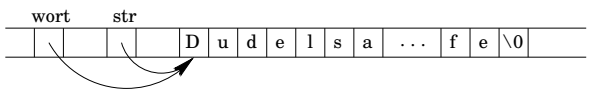


Zeichenketten

```
#include <stdio.h>
int strlength(char *str) {
    int n = 0;

    while (str[n] != '\0')
        n += 1;
    return n;
}
void main(void) {
    char *wort = "Dudelsackpfeife";
    int len = strlength(wort);

    printf("length(%s) = %d\n", wort, len);
}
```



Arrays und Zeichenketten

Ist das folgende Programm korrekt? Kann ein Zeiger wie ein Array benutzt werden?

```
#include <stdio.h>

void main(void) {
    int i;
    char *captain = "Picard";

    printf("captain = %s\n", captain);

    printf("captain = ");
    for (i = 0; i < 6; i++)
        printf("%c", captain[i]);
    printf("\n");
}
```


Arrays und Zeichenketten

Ist das folgende Programm korrekt? Kann ein Zeiger auf einen anderen Speicherbereich zeigen?

```
#include <stdio.h>

void main(void) {
    char *captain = "Picard";

    printf("captain = %s\n", captain);

    captain = "Sisco";
    printf("captain = %s\n", captain);
}
```


Arrays und Zeichenketten

Ist das folgende Programm korrekt?

```
#include <stdio.h>

void main(void) {
    int i;
    char *captain1 = "Picard";
    char *captain2 = "Sisco";

    printf("1. captain = %s\n", captain1);
    printf("2. captain = %s\n", captain2);

    for (i = 0; i < 6; i++)
        captain1[i] = captain2[i];
    printf("1. captain = %s\n", captain1);
    printf("2. captain = %s\n", captain2);
}
```

Arrays und Zeichenketten

Vorsicht: Eindimensionale Arrays können auch als Zeiger deklariert werden, aber

- Zeichenkonstanten liegen im *Programmsegment*,
- Arrays werden im *Datensegment* abgelegt.

Beispiel:

```
char ar[] = "Riker";
char *st = "Worf";

ar[0] = 'X';  /* o.k.! */
st = "Troi";  /* o.k.! */
st[0] = 'X';  /* segmentation fault! */
```


Arrays und Zeichenketten

```
#include <stdio.h>

/* Prototypen */
void output(int *a, int n);
void tausche(int *a, int n);

/* Hauptprogramm */
main() {
    int b[6] = {1, 2, 3, 4, 5, 6};

    output(b, 6);
    tausche(b, 6);
    output(b, 6);
}
```


Arrays und Zeichenketten

```
void output(int *a, int n) {
    int i;

    for (i = 0; i < n; i++)
        printf("%4d\n", a[i]);
}

void tausche(int *a, int n) {
    int i, j;

    for (i = 0, j = n-1; i < j; i++, j--) {
        int t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
}
```


Arrays und Zeichenketten

Vorsicht: Arrays lassen sich nicht mit == vergleichen.

Beispiel:

```
int a[3] = {1, 2, 3};
int b[3] = {1, 2, 3};

if (a == b) ...
```

Beispiel:

```
char c1[10], c2[10];

scanf("%s", c1);
scanf("%s", c2);
if (c1 == c2) ...
```

Beispiel:

```
char *c1 = "Picard";
char *c2 = "Picard";

if (c1 == c2) ...
```


Arrays und Zeichenketten

Bei einem Vektor bedeutet `const`, dass die Elemente nicht verändert werden können.

Beispiel:

```
const int arr[] = {1, 2, 3, 4};
```

Dann ist eine Zuweisung wie `arr[0] = 0;` verboten!

Arrays und Zeichenketten

Zeiger sind Adressen, also positive, ganze Zahlen. Daher sind auch Rechenoperationen wie Addition und Subtraktion erlaubt:

```
#include <stdio.h>

void main(void) {
    char *s = "Hallo";
    int i = 0;

    while (*(s+i) != '\0') {
        printf("%c", *(s+i));
        i += 1;
    }
    printf("\n");
}
```

`s[i]` wird vom Compiler übersetzt zu `*(s+i)`!

Arrays und Zeichenketten

```
#include <stdio.h>
void main(void) {
    int values[] = { 6, 5, 4, 3, 2, 1};
    int *p, i = 0;

    p = &values[0];
    for (i = 0; i < 6; i++)
        printf("%d\n", *(p+i));

    p = &values[0];
    for (i = 0; i < 6; i++) {
        printf("%d\n", *p);
        p += 1;
    }
}
```

Sprachdefinition: Zeigt `p` auf ein Element eines Arrays, dann zeigt `p + j` auf das `j`-te Element hinter `p`.

Arrays und Zeichenketten

Vorsicht: Es sind auch negative Werte für `j` zulässig und es findet keine Bereichsüberprüfung statt.

Beispiel:

```
char *cp;
char z[4] = {'1', '2', '3', '4'};

cp = &z[1];       /* cp zeigt auf z[1] */
cp++;             /* cp zeigt auf z[2] */
printf(" z[2] = %c\n", *cp);
printf("z[-8] = %c\n", *(cp-10));
```

Dynamische Speicheranforderung

Arrays: zur Übersetzungszeit muss bekannt sein, wie viele Elemente gespeichert werden sollen (nicht in C99)

In der Regel ist diese Größe nicht vorhersagbar.
→ Speicher wird zur Laufzeit (dynamisch) angefordert

`void * malloc(size_t size)` liefert Zeiger auf Speicherbereich der Größe `size`, oder `NULL`, wenn die Anfrage nicht erfüllt werden kann. Der Bereich ist nicht initialisiert.

Beispiel:

```
int *pa, i, len;
...
pa = (int *) malloc(len * 4);
if (pa != NULL) ...
```


Dynamische Speicheranforderung

Der unäre Operator `sizeof` liefert die Anzahl der Bytes für ein Objekt oder einen Typ.

Syntax:

```
sizeof <object>
sizeof(type)
```

Beispiel:

```
int i, j;
j = sizeof i;
j = sizeof(int);
```

`sizeof` darf nicht auf Operanden vom Typ Funktion, unvollständige Typen (Felder ohne Größenangabe) oder `void`-Typen angewendet werden.

Beispiel:

```
pa = (int *) malloc(len * sizeof(int));
v = malloc(len * sizeof(int []));  /* falsch! */
```


Dynamische Speicheranforderung

Die Funktion `void free(void *p)` gibt den Speicherbereich, auf den `p` zeigt, wieder frei. Der Speicherbereich muss vorher über einen Aufruf von `malloc` allokiert worden sein!

Beispiel:

```
int *array;
...
array = (int *) malloc(len * sizeof(int));
if (array == NULL)
    return -1;
for (i = 0; i < len; i++)
    array[i] = rand() % 256;
...
free(array);
```


Pointer Fun

`www.cs.stanford.edu/cslibrary/PointerFunCppBig.avi`

Aufzählungstyp

Der `enum`-*Typ*: Konstanten werden Integer-Werte zugewiesen, Konstanten sind ansprechbar über Bezeichner.

Beispiel:

```
enum Tag {Mo, Di, Mi, Do, Fr, Sa, So};
```

Die Werte werden intern auf fortlaufende, nicht-negative ganze Zahlen abgebildet, beginnend bei 0.
Im Beispiel: `Mo == 0`, `Di == 1`, `Mi == 2` usw.

Eine solche Abbildung kann auch direkt definiert werden:

```
enum Tag {Mo = 1, Di = 2, Mi = 4, Do = 8,
          Fr = 16, Sa = 32, So = 64};
```


Aufzählungstyp

Jeder Name darf nur einmal verwendet werden, d.h. Namen in verschiedenen Aufzählungen müssen sich unterscheiden.

Die Werte in einer Aufzählung können gleich sein.

Variablen können mit Aufzählungstypen deklariert werden.

- Sie unterliegen nicht notwendigerweise der Typprüfung.
- Eine Aufzählungskonstante des Typs kann als Wert zugewiesen werden.
- Sie können in logischen Ausdrücken verglichen und in arithmetischen Ausdrücken verknüpft werden.

Aufzählungstyp

Beispiel:

```
enum Tag {MO, DI, MI, DO, FR, SA, SO};
enum Monat {JAN, FEB, MAR, APR, MAY, JUN,
        JUL, AUG, SEP, OCT, NOV, DEC};

enum Tag t = MO;
enum Monat m = JAN;

if (t == DI)
    ...
if (t == m)     /* TRUE, denn:  Mo == 0 == Jan */
    ...

t = JAN + 3;    /* kein Fehler, keine Warnung! */
```


Strukturen

Motivation: Oft sind Daten aus einfacheren Daten zusammengesetzt:

Buch: Autor, Titel, Verlag, Jahr, ISBN
Adresse: Name, Vorname, Straße, HNr, PLZ, Ort

Um diese Zusammengehörigkeit auszudrücken, können wir Daten mittels `struct` strukturieren.

Strukturen

Zusammenfassung von Daten unterschiedlichen Typs.

Syntax:

```
struct <name> {
    <member_1>;
    ....
    <member_n>;
}
```

Beispiel:

```
struct adresse {
    char *str, *ort;
    int hnr, plz;
    long tel;
}
```

Semantik:

- <name> ist der *Typname der Struktur* (kann entfallen)
- <member_i> ist eine Variablendeklaration
- die Variablen der Struktur heißen *Komponenten* oder auch *Attribute*
- ok: gleiche Komponente in verschiedenen Strukturen

Strukturen

Strukturen ermöglichen die *Organisation von Daten*.

Eine `struct`-Vereinbarung definiert einen Datentyp.

Hinter der Komponentenliste kann eine Liste von Variablen stehen, genau wie bei einem elementaren Datentyp:

Beispiel:

```
int a, b, c;
```

```
struct datum {
    int tag, monat, jahr;
} x, y, z;
```

Beide Definitionen vereinbaren Variablen des angegebenen Typs und reservieren Speicherplatz.

Strukturen

Die Komponenten einer Struktur werden mit dem Punktoperator . angesprochen.

- linker Operand: *Strukturvariable*
- rechter Operand: *Komponentenname*

Beispiel:

```
struct datum {
    int tag, monat, jahr;
} x, y, z;

x.tag = 23;
x.monat = 10;
x.jahr = 2008;
```


Strukturen

Strukturen dürfen andere Strukturen enthalten:

```
struct mitarbeiter {
    char *name, *vorname, *persnr;
    struct adresse adresse;
    int gehalt;
}
```

Strukturen können in Vektoren gespeichert werden:

```
struct mitarbeiter personal[20];
...
for (i = 0; i < 20; i++)
    printf("Name[%d] = %s\n", i,
           personal[i].name);
```

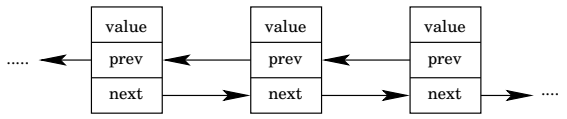

Strukturen

Vorsicht: Strukturen dürfen sich nicht selbst enthalten!
→ wieviel Speicherplatz soll bereitgestellt werden?

Rekursive Strukturen: In der Strukturdeklaration wird ein Zeiger auf die Struktur selbst vereinbart.

Beispiel:

```
struct liste {
    struct mitarbeiter value;
    struct liste *prev, *next;
} fischer, meier, schulze;
```



Strukturen

Abkürzende Schreibweise für Zeiger auf Strukturen:
`(*name).komponente` entspricht `name->komponente`

Beispiel:

```
fischer.next = &meier;
fischer.prev = NULL;
fischer.value.name = "Fischer";
...
schulze.next = NULL;
schulze.prev = &meier;
schulze.value.name = "Schulze";

for (l = &fischer; l != NULL; l = l->next)
    printf("%s\n", l->value.name);
```


Strukturen

wird mittels `typedef` einer Struktur ein *Synonym* zugewiesen, dann kann das Synonym wie ein einfacher Datentyp genutzt werden

Beispiel:

```
typedef struct {
    char *str, *ort;
    int hnr, plz;
    long tel;
} adresse_t;

adresse_t adr;
adr.str = "Waldweg";
...
```


Strukturen

eine Synonym-Vereinbarung kann andere bereits definierte Synonyme enthalten

Beispiel:

```
typedef struct {
    char *name, *vorname, *persnr;
    adresse_t adresse;
    int gehalt;
} mitarbeiter_t;

mitarbeiter_t arb;
arb.name = "Maier";
...
```

Soll einer rekursiven Struktur mittels typedef ein Synonym zugewiesen werden, darf der Strukturname nicht entfallen.

Beispiel:

```
typedef struct liste {
    mitarbeiter_t value;
    struct liste *prev, *next;
} liste_t;

liste_t l;
mitarbeiter_t fischer, meier, schulze;
```


Initialisierung: Der Struktur-Definition folgt eine Liste von konstanten Ausdrücken für die Komponenten.

Beispiele:

```
adresse_t a = {"Am Bach", "Aldrup", 4, 12345,
               4711};

mitarbeiter_t m = {"Fischer", "Anna", "08F42W",
  {"Am Bach", "Aldrup", 4, 12345}, 2500};

liste_t l = {{"Fischer", "Anna", "08F42W",
  {"Am Bach", "Aldrup", 4, 12345, 4711}}, 0, 0};
```

Anmerkungen:

- Die Reihenfolge der Komponenten ist zu beachten!
- Es müssen nicht alle Komponenten initialisiert werden.