

Programmiertechnik (C++)

Prof. Dr. Stefan Enderle
NTA Isny

1. Wiederholung

Arrays

- Arrays (Felder) fassen Objekte gleichen Typs zusammen.
- Deklaration von Arrays:

```
int i[100];  
char c[50];  
double d[1000];
```
- Speicherlayout: (Bild)
- Achtung: Es werden n Elemente angelegt, aber Numerierung von 0 bis n-1 !!

Arrays

- Zugriff auf ein Segment des Array:
 - Zuweisung:

```
i[0] = 15;  
c[43] = 'A';
```

- Auslesen:

```
cout << i[100] << endl;
```

- Achtung: Feldgrenzen werden nicht überprüft!!
D.h. ein falscher Index schreibt irgendwo in den Speicher!

Arrays

- Initialisierung eines Array bei der Deklaration:

```
int i[10] = { 1, 5, 4, 7, 2, 3, 9, 8, 0, 12 };
```

- Zu wenige Initialisierer sind erlaubt:

```
int i[3] = { 1 }; // i[1]=i[2]=0
```

- Zu viele Initialisierer ergeben einen Fehler:

```
int i[3] = {1, 2, 3, 4, 5}; // Fehler
```

Arrays

- Zuweisung von Array ist nicht erlaubt:

```
int i[3] = { 1, 5, 4 };  
int j[3];  
j=i; // Fehler!
```

- Man muss einzelne Werte kopieren:

```
for (a=0; a<3; <++) j[a]=i[a];
```

Arrays

- Vorsicht bei der Parameter-Übergabe:
Felder können NICHT als Parameter übergeben
bzw. zurückgeliefert werden!
(Nur die Zeiger darauf!)

- Beispiel:

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

```
void change(int i[3])
```

```
{
```

```
    i[0] = 15;
```

```
}
```

```
change(i); // i={15,2,3}
```

Mehrdimensionale Arrays

- Felder können „mehrdimensional sein“, d.h. mehrere Indices benutzen.

- Beispiel Matrix:

```
int m[3][3]; // 3x3 Matrix
```

- Zugriff:

```
m[1][2] = 15;  
cout << m[0][2] << endl;
```

Mehrdimensionale Arrays

- Initialisierung eines mehrdimensionalen Arrays trotzdem „eindimensional“:

```
int m[3][3] = { 1,2,3,4,5,6,7,8,9 };
```

- Zugriff:

```
m[1][2] = 15;  
cout << m[0][2] << endl;
```

- Häufiger Fehler:

```
m[1,2]; // liefert „Unterfeld“  
// korrekt: m[1][2]
```

Strings – 1. C-Strings

- In C gibt es keinen Datentyp „String“
- Stattdessen wird als String ein Array von Zeichen benutzt:

```
char name[100];
```
- Bei der Deklaration wird ein Feld mit 100 Zeichen Platz allokiert. `name` enthält dann lediglich den Zeiger darauf.
(Bild)
- D.h., der Typ von `name` ist „Zeiger auf ein Zeichen“:

```
char* name;
```

Strings – 1. C-Strings

- Initialisierung eines „C-Strings“ funktioniert:

```
char name[100]="Anna";
```

- Sogar ohne Angabe der Länge automatisch:

```
char name[]="Anna";
```

(Hier werden 5 (!) Zeichen allokiert, nämlich „Anna“ mit abschließender 0.)

Strings – 1. C-Strings

- Problem: Da `name` nur ein Zeiger ist, kann man ihm keinen Wert (also String) zuweisen!

- Beispiel:

```
char name1[100]="Anna";  
char name2[100]="Bernd";
```

```
name1=name2; // Was passiert hier?
```

- In C zeigt nun der Zeiger `anna1` ebenfalls auf die Speicherstelle, wo „Bernd“ steht.
- In C++ ist die Zuweisung von Array-Pointern nicht erlaubt und gibt einen Fehler.

Strings – 1. C-Strings

- Für String-Operationen in C nutzt man die Funktionen der Bibliothek `<string.h>`:
 - `char *strcpy(s, ct)`
Zeichenkette `ct` nach `s` kopieren, inklusive `'\0'`.
 - `char *strcat(s, ct)`
Zeichenkette `ct` hinten an die Zeichenkette `s` anfügen.
 - `int strcmp(cs, ct)`
Zeichenketten `cs` und `ct` vergleichen.
Liefert `<0` wenn `cs<ct`, `0` wenn `cs==ct`, oder `>0`, wenn `cs>ct`.
 - `char *strchr(cs, c)`
Liefert Zeiger auf das erste `c` in `cs` oder `NULL`, falls nicht vorhanden.
 - `char *strstr(cs, ct)`
Liefert Zeiger auf erste Kopie von `ct` in `cs` oder `NULL`, falls nicht vorhanden.
 - `size_t strlen(cs)`
Liefert Länge von `cs` (ohne `'\0'`).

Strings – 2. C++-Strings

- In C++ gibt es den Datentyp `string`
- Er kann wie andere interne Datentypen benutzt werden:
 - Deklaration: `string name;`
 - Initialisierung: `string name="Anna";`
 - Auslesen: `cout << name << endl;`

Strings – 2. C++-Strings

- Auf C++-Strings sind weitere Operationen definiert:
 - Zuweisung:

```
string name1="Anna";  
string name2;  
name2 = name1; // name2 enthält Kopie!
```
 - Verkettung: `name = name1 + name2;`
 - Vergleiche:
 - `name1 == name2`
 - `name1 != name2`
 - `name1 > name2`
 - `name1 < name2`

Strings – 2. C++-Strings

- Länge eines Strings:
`len = name.size();`
`len = name.length();`
- Verlängern/Verkürzen:
`name.resize(n, '*');`