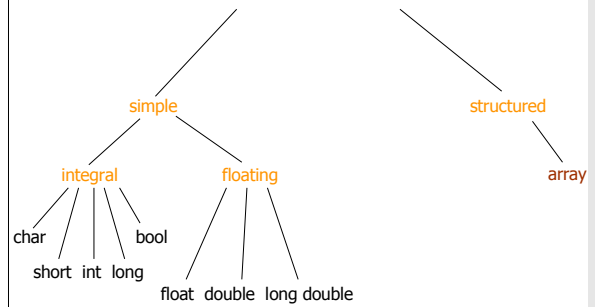


Grundlagen der Programmierung in C++

Arrays und Strings, Teil 1

Wintersemester 2005/2006
G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

Das C++ Typsystem



Problem

- Meßwertreihe einlesen, Mittelwert berechnen, von allen Werten abziehen, mittelwertbereinigte Reihe wieder rausschreiben
- Programm muß Reihe 2x durchgehen
- Lösung: Daten intern zunächst abspeichern, dann über diese 2x laufen
- Benötigt: Datenstruktur zur einfachen Verwaltung vieler Daten mit identischem Typ
- Lösung: neues Sprachkonstrukt

Arrays

- Wahrscheinlich häufigste Datenstruktur ☺
- *Zusammenhängender* Speicherblock, der Folge von *gleichartigen* Elementen enthält (Bsp. Vektor)
- Deklaration:
***Typ* *arrayname*[*n*];**
wobei ***Typ*** ein bekannter Typ ist, ***n*** Konstante.
- Terminologie:
 - *Arrayname* heißt "Array von Elementen des Typs T"
 - Kurz "T-Array" (Bsp. "float-Array")
 - T selbst kann einfach oder wieder zusammengesetzt sein
 - T heißt auch "Basistyp"

Beispiel:

```
int a[8]; // 8 int's
int a[2*4]; // dito
```

Danach sieht a im Speicher so aus:

a[0]	a[1]	a[2]						a[7]
??	??	??	??	??	??	??	??	??

0x1000 0x1020

Speicher ist also reserviert ("allocated"), aber **nicht initialisiert!**

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 5

Zugriff:

`arrayname[int-expr]`
wobei *int-expr* ein Ausdruck ist, der ein `int` liefert.

Achtung

- Index läuft ab 0!
D.h., die Elemente von `int a[8]` laufen von `a[0]`, ... , `a[7]`.
- Kein Bounds-Checking!
- Zugriff auf `a[17]` und `a[-1]` liefert höchstens ein Warnung vom Compiler!
- Aber: Zugriff zur Laufzeit liefert evtl. Absturz!
- I.A. erfolgt Zugriff über nicht-konstante *int-expr*, deren Wert man nicht kennt

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 6

Beispiel:

```
int a[8]; // int-array mit 8 Elementen
for ( int k = 0; k < 8; k ++ )
    a[k] = k; // Initialisierung
```

a[0]	a[1]	a[2]						a[7]
00	01	02	03	04	05	06	07	

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 7

Mehrdimensionale Arrays

Als Basistyp kann man wieder Array schreiben

Beispiel:

```
float a[6][4];
// matrix mit 6 Zeilen und 4 Spalten
```

Allgemein:

`T arrayname[n1][n2]...[nd]`
deklariert d-dimensionales T-Array.

Speicherlayout:

- Ein-dimensionales Array (logisch, da Hauptspeicher lineares Array ist)
- Die ersten n_d Elemente entsprechen `a[0]...[0][0], ..., a[0]...[0][nd-1]`
- Die nächsten n_d Elemente entsprechen `a[0]...[1][0], ..., a[0]...[1][nd-1]`
- Heißt "row-major ordering"

a[0][0]	a[0][1]	...	a[0][n _d -1]	a[1][0]	...
---------	---------	-----	-------------------------	---------	-----

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 8

Statische Initialisierung

- Syntax:

```
int smallPrimes[7] = { 2, 3, 5, 7, 11, 13, 17};
```

```
float rotmatrix[2][2] =
{
  { cos(a), sin(a) },
  { -sin(a), cos(a) }
};
```

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 9

Strings

- 2 Arten:
 - "C-Strings"
 - "C++ Strings"

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 11

C-Strings

- Ist ein 0-terminiertes Char-Array

```
char s[6];
s[0] = 'h';
...
s[4] = 'o';
s[5] = '\0';
```

h	e	l	l	o	\0
---	---	---	---	---	----

- Initialisierung von Char-Arrays

```
char s[] = "hello";
```

- Compiler erzeugt automatisch Array passender Größe
- String-Konstante "hello" ist selbst 0-terminiert, also wird diese 0 mit ins Array kopiert.

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 12

Standard-Library-Funktionen zu C-Strings

- Bekommt man durch
 - #include <string.h>

<code>char *strcpy(char *s1, const char *s2);</code>	Copies the string s2 into the character array s1. The value of s1 is returned.
<code>char *strcat(char *s1, const char *s2);</code>	Appends the string s2 to the string s1. The first character of s2 overwrites the terminating null character of s1. The value of s1 is returned.
<code>int strcmp(const char *s1, const char *s2);</code>	Compares the string s1 with the string s2. The function returns a value of zero, less than zero or greater than zero if s1 is equal to, less than or greater than s2, respectively.
<code>size_t strlen(const char *s);</code>	Determines the length of string s. The number of characters preceding the terminating null character is returned.

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 13

- Achtung: Kein Bounds-Checks!
- Verwende immer die `strn...`-Variante, wenn möglich!
- Beispiel:

<pre>char *strncpy(char *s1, const char *s2, size_t n);</pre>	Copies at most n characters of the string s2 into the character array s1. The value of s1 is returned.
---	--

- Weitere Funktionen: siehe `man string`
- Zeichenklassen:

<pre>int isalpha(char c); isalnum, isupper, isprint, isspace,</pre>	Test, ob Zeichen einer bestimmten Klasse angehört.
---	--

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 14

- Umwandlungen:

<code>atoi, atol, strtol, atof, strtod</code>	Umwandlung von String nach Zahl.
<code>sprintf(char *str, const char *format, ...);</code>	Umwandlung der built-in types nach String.
<code>int toupper(char c);</code> <code>int tolower(char c);</code>	Konvertierung zwischen lower-case und upper-case (gemäß dem aktuellen Locale)

- Einlesen / Ausgeben:

<code>printf(), puts, fprintf, fputs</code>	Ausgeben auf stdout bzw. einem anderen File
<code>gets, fgets, getchar,getc</code>	String bzw. Einzelnes Zeichen lesen.

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Arrays und Strings, Teil 1 15