

Programmiersprache 1 (C++)

Prof. Dr. Stefan Enderle
NTA Isny

8. Datentypen

8.1 Einfache Datentypen

- Wiederholung: Einfache Datentypen:
 - Logische Werte: `bool`
 - Zeichen: `char`
 - Ganze Zahl: `int`
 - Gleitkomma Zahl: `float, double`
 - Ohne Wert: `void`
- Beispiele:
 - `bool keyPressed;`
 - `float value;`

Modifikatoren

- Typ-Modifikatoren:
 - Mit oder ohne Vorzeichen: `signed / unsigned`
 - Länge: `short / long`
- Beispiele:
 - `unsigned int anzahlProdukte; // int ohne VZ`
 - `short int kleineMenge; // 2 Byte`
 - `long int grosseMenge; // 4 Byte`
- Datentypen ohne `signed/unsigned` sind immer „signed“!

sizeof()

- sizeof():
 - Mit der Funktion sizeof() kann die Länge eines Datentyps in byte abgefragt werden
- Beispiele:
 - `sizeof(char);` // = 1 (byte)
 - `sizeof(float);` // = 4 (byte)
 - `sizeof(double);` // = 8 (byte)
- Auch eigene Datentypen sind möglich.

Wertebereiche int

- Anzahl Bytes und Wertebereiche für Ganze Zahlen:

Datentyp	Bytes	Wertebereich
signed char	1	-128 ... 127
unsigned char	1	0 ... 255
signed short int	2	$-2^{15} \dots 2^{15}-1$
unsigned short int	2	0 ... $2^{16}-1$
signed int	4	$-2^{31} \dots 2^{31}-1$
unsigned int	4	0 ... 2^{32}
signed long int	4	$-2^{31} \dots 2^{31}-1$
unsigned long int	4	0 ... 2^{32}
signed long long int	8	$-2^{63} \dots 2^{63}-1$
unsigned long long int	8	0 ... 2^{64}

Wertebereiche float

- Anzahl Bytes und Wertebereiche für Realzahlen:

Datentyp	Bit (Mant/Exp)	Wertebereich	Dez.stellen
float	32 (23/8)	$3.4 \cdot 10^{-38} \dots 3.4 \cdot 10^{38}$	7
double	64 (52/11)	$1.7 \cdot 10^{-308} \dots 1.7 \cdot 10^{308}$	15
long double	80 (64/15)	$1.2 \cdot 10^{-4932} \dots 1.2 \cdot 10^{4932}$	19

const

- Qualifizierer `const` :
 - `const` wird vor eine „Variable“ geschrieben, wenn sich deren Wert nicht verändert.
 - Vorteil:
 - Der Compiler kann dies prüfen.
 - Evtl. effizientere Ausführung
- Beispiele:
 - `const float PI=3.14;`
 - `const string defaultLand = "Deutschland";`

Typumwandlung

- Typumwandlung
 - Implizit:
 - bool (false/true) → int (0/1)
 - int (0/n) → bool (false/true)
 - char → int ASCII-Wert
 - int → float oder double
 - float oder double → int Abschn. der Dez.Stellen
 - unsigned → signed
 - Explizit:
 - `int i = (int) 'A'; // i=65`
 - `int i = int('A'); // i=65`

8.2 Aufzählungen

- Aufzählungen definieren eigene Typen, die nur eine (kleine) Anzahl möglicher Werte enthalten sollen.
- Definition einer Aufzählung:

```
enum Farbe { rot, gruen, blau };
```
- Variable des Aufzählungstyps:

```
Farbe f;
```

Aufzählungen

- Benutzung von Aufzählungstypen:
- Beispiel:

```
Farbe f;
```

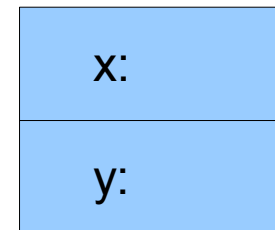
```
f = rot;           // Zuweisung  
if (f==blau) ...  // Abfrage
```

```
void outputFarbe (Farbe color)  
{  
    if (color==rot)  cout << "rot";  
    ...  
}
```

8.3 Strukturen

- Strukturen (auch Verbunde) dienen der Speicherung von zusammengehörigen Daten gleichen oder unterschiedlichen Typs.
- Beispiel: Definition einer Struktur:

```
struct Punkt {  
    int x;  
    int y;  
};
```



Strukturen

- **Beispiele: Variablen anlegen**

```
Punkt ursprung;  
Punkt posQuadrat;
```

- **Beispiel: Zugriff auf Elemente**

```
ursprung.x = 0;  
ursprung.y = 0;  
posQuadrat.x = 20;  
posQuadrat.y = 15;
```

Strukturen

- Direkte Initialisierung:

```
Punkt ursprung = {0, 0};
```

- Bemerkung:

Die Elemente einer Struktur stehen direkt hintereinander im Speicher.

8.4 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 Nummerierung von 0 bis n-1 !!

Arrays

- Zugriff auf ein Element des Array:

- Zuweisung:

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

- Auslesen:

```
int zahl = i[0];  
cout << c[43] << 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 Arrays 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; a++)  j[a]=i[a];
```

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]
```

Arrays: Call by Reference !

- Vorsicht bei der Parameter-Übergabe:
Ganze Felder können NICHT als Parameter übergeben oder zurückgeliefert werden!
- Es wird nur der Zeiger darauf übergeben
→ Calls by reference !
- Beispiel:

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

```
void change(int i[3])  
{  
    i[0] = 15;  
}
```

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

Arrays: Call by Reference !

- Durch `const` kann die Referenz als konstant definiert werden.
- Beispiel:

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

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

```
{
```

```
    i[0] = 15;           // Fehler wegen const !
```

```
}
```

```
change(i);              // i darf sich nicht ändern
```