Notizen AIK ProgTech 3: Referenzen

Klaus Kusche

Man kann zwar auch ganz normale Variablen als Referenz deklarieren, aber in der Praxis werden Referenzen fast nur für *Funktionsparameter* und *Funktionsreturnwerte* verwendet.

Referenz-Parameter

- *Deklaration* des Parameters im Funktionskopf *mit* **&**
- *Argument* im Aufruf *"ganz normal"* wie "by Value", *ohne* **&** oder *****
- *Verwendung* des Parameters in der Funktion *"ganz normal"* wie "by Value", *ohne* *****
- Intern wird ein Pointer übergeben, aber dieser Pointer ist *nie* **NULL** und kann in der Funktion *nicht verändert* werden (zeigt immer auf die Variable im Aufrufer).
- Ist ein Referenz-Parameter **const** deklariert, können beim Aufruf auch *Konstanten* und *Rechnungen* als Argument übergeben werden, nicht nur Variablen.

Arrays als Parameter

C: Nur *"by reference"* als *Pointer* möglich.

C++ früher: Wie in C immer als *Pointer*.
Array-Referenzen kamen so gut wie nie vor, waren praktisch unbekannt!

C++ heute: *Dynamisch angelegte* bzw. *variabel große* Arrays wie bisher als *Pointer*,
Arrays fixer Größe wenn möglich als *Referenz-Parameter*, z.B.:

```
int sum(int (&arr)[size]) { ...
```

Achtung:

1.) Die **( )** rund um **&arr** sind *notwendig*!!!

int &arr[size] wäre keine Referenz auf ein Array,
sondern ein Array von Referenzen (was verboten ist).

2.) Man darf die **[ ]** *nicht leer lassen*!

3.) **size** muss eine *Konstante* sein (Zahl, **#define**, globaler **const int**)!

==› nur für Arrays *fixer Größe* möglich,
nicht für variabel große bzw. dynamisch angelegte Arrays!

Mit *Templates* (lernen wir später) kann dieselbe Funktion
für Arrays *verschiedener fixer Größe* verwendet werden
(der Compiler setzt bei jedem Aufruf für **size** automatisch
die *tatsächliche Größe* des übergebenen Arrays ein):

```
template <int size>
int sum(int (&arr)[size]) { ...
```

4.) Als *Argument beim Aufruf* muss wirklich ein *Array* übergeben werden,
ein *Pointer auf ein* (z.B. mit **new** angelegtes) *Array* ist *verboten*!

Vorteil von Array-Referenzen gegenüber Array-Pointern:

1.) **`sizeof(arr)`** *funktioniert* in der Funktion, wenn **`arr`** eine Referenz ist!
(wenn man das Array als *Pointer* übergibt, funktioniert **`sizeof`** *nicht*)

2.) Der Compiler *prüft*, ob das im Aufruf übergebene Array
die *richtige Größe* **`size`** hat (wenn nicht: Compilerfehler!).

Übersicht über Funktionsparameter

	Der Wert geht in die Funktion ***nur hinein, nicht heraus***		Die Funktion soll den Wert ***im Aufrufer ändern können***	
	reines C	**C++**	**reines C**	**C++**
Einfache Werte	by Value	by Value	**`*a`**	**`&a`**
`struct`, Objekte	**`const *a`** (oder by Value)	**`const &a`** (oder by Value)	**`*a`**	**`&a`**
Arrays fixer Größe	**`const *a`** oder **`const a[]`**	**`const (&a)[s]`** oder **`const *a`** oder **`const a[]`**	**`*a`** oder **`a[]`**	**`(&a)[s]`** oder **`*a`** oder **`a[]`**
Variabel große Arrays	**`const *a`** oder **`const a[]`**	**`const *a`** oder **`const a[]`**	**`*a`** oder **`a[]`**	**`*a`** oder **`a[]`**

Referenzen als Returnwert

- Üblich für *Objekte und Strukturen*
- *Typische Returnwerte:* **`*this`**, Element eines übergebenen Arrays, als Parameter übergebene Referenz, Referenz-Returnwert einer anderen Funktion, ...
- ***Nicht:*** Lokale Variablen, temporäre Objekte
(*verschwinden* beim Return ==› Referenz zeigt danach *"ins Leere"*!)

Im Aufrufer

- ... verwendbar *wie ein "normaler" Returnwert*
- ... aber auch als L-Value, d.h. als *linke Seite einer Zuweisung* (oder eines **>>** usw.):
Der zuzuweisende Wert wird in der Variable / dem Objekt / dem Array-Element gespeichert, auf das die zurückgegebene Referenz zeigt.

  In der Praxis: Such-Funktionen und **`[ ]`** geben eine Referenz auf das gefundene Element zurück, damit kann dieses Element im Aufrufer sowohl gelesen als auch geschrieben werden.