4.4 Von C nach C++/Referenzen

4.4.1 Idee

Der Umgang mit Zeigern in C ist unübersichtlich und kann fehleranfällig sein (wenn der Programmierer Fehler macht).

Deshalb gibt es in C++ eine neue Technik zum Ersetzen von Zeiger-Variablen und Inhalts- und Adressoperatoren.

4.4.2 Referenz als Datenstruktur

Eine Referenz ist neuer Name für bestehende Variable.

Das scheint zunächst sinnlos. Warum soll eine Variable zwei oder mehr Namen haben? Interessant wird das erst im Zusammenhang mit Funktionen.

Die Deklaration und Initialisierung einer Referenz funktioniert wie folgt:

```
1 /* 1 */ int n;          // eine Variable
2 /* 2 */ int& rn = n;    // ein zweiter Name fuer: n
3 /* 3 */ rn=3;           // Zuweisung an n
4 /* 4 */ cout << n;      // ergibt 3
```

In Zeile 2 ist die Datentypangabe gefordert; der Datentyp muss gleich dem Datentyp der referenzierten Variable sein. Eigentlich ist das eine Redundanz, denn der Datentyp ist ja bekannt. Es dient nur der Einheitlichkeit der Programmiersprache (s. u.: Parameter-Übergabe bei Funktionen und Wertrückgabe von Funktionen).

Das Gleichheitszeichen in Zeile 2 ist eine Initialisierung und sagt, zu welchem Namen (zu welcher Adresse) `rn` gehören soll.

Das Gleichheitszeichen in Zeile 3 ist eine Zuweisung: Der Inhalt von `n` bzw. `rn` wird verändert (Tabelle 1).

nach Zeile 1:	nach Zeile 2:	nach Zeile 3:
int n	int rn und n	int rn und n
Adresse: 628	Adresse: 628	Adresse: 628
Inhalt: ?	Inhalt: ?	Inhalt: 3

Tabelle 1: Speicher-Diagramm zum angegebenen Beispiel

Das Referenz-`&`-Zeichen ist nicht gleich dem Adress-Operator-`&`:

Bei einer *Vereinbarung* ist das `&`-Zeichen Symbol einer Referenz und gehört zum Datentyp.

Bei einer *Anweisung* dagegen handelt es sich bei einem `&`-Zeichen um einen Adressoperator, er gehört zu einem Ausdruck: scanf("%i", &n);

4.4.3 Referenz als Parameter

Man kann eine Referenz als Parameter übergeben. Hier ein Beispiel:

makebetrag.cpp

```
1 #include <iostream>
2 using namespace std;
3 void makebetrag(int&);
4
5 int main(void)
6 {
7     int a=-4;
8     cout << "Wert von a: " << a << endl;
9
```

```
10      cout << "Rufe makebetrag auf..." << endl;
11      makebetrag(a);
12
13      cout << "Neuer Wert von a: " << a << endl;
14      return 0;
15  }
16  //-------------------------------------------------
17  void makebetrag(int& x)
18  {
19      if(x<0)
20          x=-x;
21  }
```

Wenn man eine Referenz als Parameter übergibt, dann ist der formale Parameter (hier `x`), den die Funktion erhält, *nur ein neuer Name* für den aktuellen Parameter (hier `a`) des Funktionsaufrufs.

Das bedeutet: An die Funktion wird nicht eine Kopie von `a` übergeben, sondern die Adresse des Originals `a`. In der Funktion wird damit auf das Original zugegriffen.

Es funktioniert also wie bisher in C (dort hieß es `int *pa`), aber der Zugriff ist hier ohne den Adressoperator möglich:

a) beim Aufruf der Funktion

b) bei Benutzung der Parameter innerhalb der Funktion

Intern arbeitet der Compiler also mit Zeigern, gegenüber dem Programmierer wird das aber versteckt.

Das nennt man: *Call-By-Reference*. In den Sprachen PASCAL, Modula-2, ALGOL, ADA (und vielen anderen) gibt es das schon seit jeher, in C dagegen ist es nur durch Zeiger zu emulieren.

4.4.4 Referenz im Vergleich zum Zeiger

Nach der Bearbeitung durch den Compiler passiert bei einer Referenz das gleiche wie bei einem Zeiger.

Der Vorteil von `int&` gegenüber `int*` besteht darin, dass die Referenz für den Nutzer der Funktion einfacher zu handhaben ist.

Der Nachteil besteht darin, dass man aus dem Aufruf nicht entnehmen kann, ob der übergebene Parameter in der Funktion verändert werden kann oder nicht. Dazu müsste man sich erst die Funktionsdeklaration ansehen.

4.4.5 Referenz im Vergleich zur Wertübergabe

Bei großen Datenmengen, die an eine Funktion übergeben werden sollen, ist `xyztyp &ra` schneller als `xyztyp a`. da bei der ersten Version nur die Adresse übergeben wird.

Der Nachteil liegt darin, dass in der Funktion versehentlich `ra` verändert werden könnte. Das kann man aber durch Verwendung des Schlüsselwortes `const` verhindern:

```
1  funktionxyz(const xyztyp& ra)
```

4.4.6 Referenz als Rückgabewert

Der Funktionswert ist nur ein neuer Name für das, was zurückgegeben wird. Intern bedeutet das: Die Adresse, die der Aufrufer im Funktionswert erhält, ist die Adresse der Variablen, die hinter `return` angegeben wurde. Deshalb darf hinter `return` in dem Fall keine Konstante stehen!

Im folgenden Beispiel wird eine globale Variable zum leichteren Verständnis benutzt: In einem Not-Betriebsprogramm für ein Kraftwerk sollen alle Maschinen gleichmäßig hochgefahren werden. Dazu muss nach und nach die jeweils langsamste Maschine „angeschoben“ werden:

maschinen.cpp

```
#include <iostream>
using namespace std;

double maschgeschw[4]={70.0,20.0,50.0,180.0};
double& langsamste(void)
{
    int min=0;
    for(int i=0; i<4; ++i)
        if(maschgeschw[i]<maschgeschw[min])
            min=i;
    return maschgeschw[min];
}

int main(void)
{
    while(langsamste()<200.0) // langsamste() wird
    {                         // abgefragt
        cout << "Maschinen: 0:" << maschgeschw[0]
             << " 1:" << maschgeschw[1];
        cout << " 2:" << maschgeschw[2]
             << " 3:" << maschgeschw[3]
             << endl;
        ++langsamste();       // Inhalt (!) von
                              // langsamste() wird
                              // um eins erhoeht!!!
    }
    return 0;
}
```

Merkhilfe (s.o.): Intern arbeitet der Compiler mit Zeigern, gegenüber dem Anwender wird das versteckt (siehe Quelltext: `maschinen_realitaet.cpp`).

Anders als bei Zeigern kann man hier allerdings nur auf den Inhalt zugreifen; die Adresse (Referenz selbst) ist konstant! `++(*pa)` geht, `++pa` geht hier nicht. Man kann also nicht mit einer Referenz durch ein Array laufen.

4.4.7 Kommentar

C++ ist im Vergleich zu C durch das gleichzeitige Vorhandensein von Zeigern und Referenzen netter, aber unübersichtlicher geworden.

Es ist zusätzliche Redundanz in die Sprache eingeführt worden: Verschiedene Programmcodes führen zu gleichem Objektcode.

Für das Symbolisieren der Referenz in einer Vereinbarung wird *ausgerechnet* das `&`-Zeichen verwendet; in einer Anweisung steht dieses Zeichen dagegen für den Adressoperator, den man im Zusammenhang mit Zeigern benutzt. Das ist vor allem für Anfänger verwirrend.

Aber C++ hat immerhin im Vergleich zu Java und C# den Vorteil, dass man in C++ bei der Vereinbarung deutlich sieht, ob man eine Variable `(int x)`, die Referenz auf eine Variable `(int &rx)` oder einen Zeiger auf die Variable `(int *px)` übergibt.