Zeiger

Benjamin Kuffel Juli 2013

Grundlagen

Ein Zeiger ist eine Variable, welche die Speicheradresse einer anderen Variablen enthält.
Zeiger werden in C häufig verwendet, weil dies oftmals die einzige Möglichkeit ist, um ein Programm zu realisieren und weil sie zu kompakteren und effizienteren Programmen führen als andere Methoden. Durch die Verwendung von Zeigern haben Programmierer mehr Möglichkeiten und Freiheiten ihr Programm zu realisieren.

Ein kleiner Überblick über die Anwendungsbereiche von Zeigern:

- Speicherbereiche können dynamisch reserviert, verwaltet und wieder gelöscht werden.

- Mit Zeigern können Datenobjekte direkt (call-by-reference) an Funktionen übergeben werden.

- Mit Zeigern lassen sich Funktionen als Argumente an andere Funktionen übergeben.

- Rekursive Datenstrukturen, wie Listen und Bäume, lassen sich fast nur mit Zeigern erstellen.

- Es lässt sich ein typenloser Zeiger (void*) definieren, womit Datenobjekte beliebigen Typs verarbeitet werden können.

Zeigerdeklaration, -operationen und Adressoperator

Der Adressoperator „&“ liefert die Adresse eines Objektes und kann nur auf Objekte im Speicher angewendet werden, d.h. auf Variablen und Vektorelemente.

Wird der „*“-Operator (nicht zu verwechseln mit dem „*“-Operator für Zahlen, d.h. Multiplikation) auf einen Zeiger angewendet, so greift er auf das Objekt zu, auf welches der Zeiger verweist.

```
int  x = 1, y = 2, z[10];

int *p;          /* p ist ein Zeiger auf int */

p = &x;          /* p zeigt nun auf x */

 y = *p;         /* y ist nun 1 */

*p = 0;          /* x ist nun 0 */
```

p = &z[0] /* p zeigt nun auf z[0] */

Die „&“- und „*“-Operatoren haben gegenüber den arithmetische Operatoren eine höhere Priorität.

*p = *p + 1 /* erhöht *p um 1 */

p += 1 / erhöht *p um 1 */

++*p /* erhöht *p um 1 */

(*p)++ /* erhöht *p um 1 */

Die Deklaration „int** q“ ist zu lesen wie „(int*)* q“, d.h. „q“ ist ein Zeiger auf „int*“. In diesem Fall verweist also ein Zeiger auf einen anderen Zeiger, der wiederum auf int zeigt

Außerdem ist es unzulässig, den „&“-Operator (nicht zu verwechseln mit der logischen UND-Verknüpfung „&“) zweimal hintereinander („&&x“) auf ein Objekt x anzuwenden, denn eine Adresse „&x“ ist selbst kein Objekt mehr.

Mehrfache „indirection“, wie z.B. „***p“, ist dagegen zulässig, vorausgesetzt, dass der Ausdruck „**p“ ein Zeiger ist.

Typ cast mit Zeigern

Zeiger sind im Prinzip identisch mit (positiven) Integer-Zahlen, daher können Zeiger und int ineinander umgewandelt werden:

p = (int*)x;

x = *p; /* lädt x mit dem Inhalt von Speicherstelle */

Die Identität von Zeigern und int-Zahlen (genauer gesagt: unsigned int, denn die Adressen sind immer positiv) resultiert daraus, dass Zahlen und Zeiger im Speicher die gleiche Darstellung haben. Die Wertebereiche müssen jedoch nicht immer gleich sein, abhängig von der zugrunde liegenden Computerarchitektur. So könnten unsigned int-Zahlen bspw. im Wertebereich [0,65536] liegen (bei 16-Bit-Darstellung), wogegen der Speicher 32-Bit-Adressen (d.h. [0, 232]) haben kann.

Heap-Speicher

Der Heapspeicher, auch „dynamischer Speicher“ genannt, ist ein Speicherbereich, aus dem zur Laufzeit eines Programms Speicherplatz angefordert und wieder freigegeben werden kann.

Da die an malloc übergebene Blockgröße variabel ist, kann der Heap-Speicher immer passend angefordert werden („int *p = (int*)malloc(sizeof(100)“). Nach der Verwendung sollte der Heap-Block mithilfe von „free(p)“ wieder freigegeben werden, um den Speicherplatz für andere Zwecke erneut verwenden zu können.

Arrays

Benachbarte Arrayelemente befinden sich stets auch benachbart im Speicher und werden unter „v[i]“ gespeichert. Der Index i bezeichnet den Abstand (offset) des i-ten Elementes von der Basisadresse V (=&V[0]).

Der Operator „[x]“ entspricht der Addition von Array-Basisadresse und Index. „p = &V[0]“ oder „p = V“ lässt p auf den Anfang von V zeigen und „V[3] =7“, p[3] = 7“ und „*(p+3) =7“ sind äquivalent.

Wie gerade gezeigt, besteht ein enger Zusammenhang zwischen Arrays und Zeigern in der Sprache C.

void-Zeiger

Ein Zeiger auf void ist ein typenloser Zeiger. Wenn der Datentyp des Zeigers noch nicht feststeht, wird der void-Zeiger verwendet. Der Vorteil von void-Zeigern liegt darin, dass man ihnen eine beliebige Adresse zuweisen kann.

Außerdem kann ein void-Zeiger durch eine explizite Typumwandlung in jeden anderen, beliebigen Datentyp umgewandelt werden.

Null-Zeiger

Der Nullzeiger ist ein Zeiger mit einem speziellen, dafür reservierten Wert, der anzeigt, dass auf nichts verwiesen wird. Dieser Wert wird auch Nullwert genannt, womit jedoch nicht zwingend numerisch 0 gemeint ist. Zum Beispiel wird eine einfach verkettete Liste meist so implementiert, dass das letzte Element auf den Nullzeiger als Folgeelement verweist. Er kennzeichnet in diesem Fall also das Ende der Liste.

Funktionszeiger

In C ist eine Funktion selbst keine Variable, aber man kann Zeiger auf Funktionen definieren , die unter anderem zugewiesen, in Vektoren eingetragen, an Funktionen übergeben und als Resultatwert von Funktionen geliefert werden können.

Call by Value und Call by Reference

call-by-value

Bei CBV werden die an eine Funktion übergebenen Daten kopiert. Es besteht dann keine Verbindung mehr zwischen den Daten beim Aufrufund den Daten in der Funktion.

call-by-reference

Anstatt die Daten zu kopieren, werden Referenzen auf die Daten übergeben. Man kann sich das als Zeiger vorstellen: Beim Aufrufzeigt eine Variable auf einen bestimmten Wert, in der Funktion zeigt eine andere Variable auf denselben Wert. Methodenaufrufe an einem so übergebenen Objekt arbeiten also auf demselben Objekt, das auch außerhalb sichtbar ist.