Zeiger in C

Thomas Mechelke

August 2011

Dieser Artikel ist die Ausarbeitung meiner Präsentation über Zeiger in C. Er ist in zwei Teile zerlegt: Der Erste Teil beschäftigt sich mit den Grundlagen zu Zeigern und der Zweite Teil wird Einblick in die häufigsten Anwendungsbereiche geben.

1 Grundlagen

1.1 Motivation

Warum ist es sinnvoll sich mit Zeigern auszukennen wenn man C Programmierung betreiben will? Der Hauptgrund ist, dass Zeiger ein essentieller Bestandteil von C sind. Es gibt Dinge, die sind ohne Zeiger einfach nicht in C zu realisieren. Dazu gehören z.B. komplexere Datenstrukturen oder die Möglichkeit dynamisch Speicher zu verwalten. Es gibt zwar auch viele Stellen an denen man ohne Zeiger auskommt, aber an einigen kann man die Performance des Programmes verbessern wenn man Zeiger benutzt, zum Beispiel bei der Parameterübergabe von größeren Strukturen. Des weiteren gibt es einige Stellen an denen Zeiger „Hinter den Kulissen“ auftreten, Arrays haben beispielsweise sehr viel mit Zeigern gemeinsam. Durch Zeiger bekommt man also viele Möglichkeiten und Freiheiten, allerdings kann man, wenn man unvorsichtig ist, auch einige Fehler mit ihnen Produzieren weshalb man sich gut mit diesem Sprachkonstrukt auskennen sollte, bevor man es einsetzt.

1.2 Definition

Die folgende Begriffserklärung ist nicht die offizielle Definition zu Zeigern, die man im C Standard findet, da diese ziemlich lang und technisch ist.

Ein Zeiger ist eine Variable, die eine Speicheradresse enthält.

Diese Darstellung ist zwar nicht exakt, dafür aber sehr kurz und anschaulich.

Zur Erläuterung zunächst ein Beispiel: Was passiert wenn man eine Integer Variable deklariert? Bei der Ausführung des Programms wird ein kleiner Teil des Hauptspeichers reserviert an dem die Daten der Variablen abgespeichert werden. Auf die Adresse an der sie liegen haben wir keinen Einfluss. Der Bitwert an der Stelle wird im Falle eines Integers dann als Zahl interpretiert. Wenn man eine Zeigervariable deklariert gibt es im Prinzip wenig Unterschiede. Auch diese bekommt ihren kleinen Bereich im Speicher zugewiesen. Der Unterschied ist, dass der Wert als Speicheradresse Interpretiert wird und nicht als Zahl. In Abbildung 1 ist dies einmal Veranschaulicht. Die Grafik soll einen ausschnitt des Hauptspeichers darstellen. Die Adressen sind hier dezimal durchnummeriert. Die Werte an den einzelnen

Stellen sind unbekannt, mit einer Ausnahme an Adresse 3401. Hier wurde eine Zeigervariable abgelegt die den Wert 1003 enthält. Dies ist eine andere Stelle im Speicher, weshalb man auch sagt der Zeiger „zeigt“ an diese Stelle. Falls dort etwas sinnvolles steht wie etwa eine Integervariable, kann man diese mithilfe des Zeigers manipulieren.

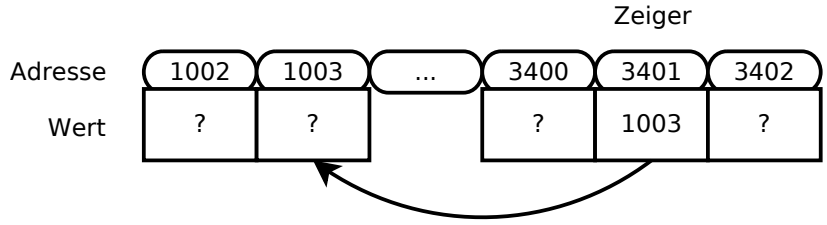


Abbildung 1: Ausschnitt eines Hauptspeichers: Ein Zeiger „zeigt“ an eine andere Stelle.

1.3 Syntax

Die Grundlegende Syntax zu Zeigern in C soll hier in drei Schritten erklärt werden. Als erstes kommt wie bei normalen Variablen die Deklaration im Quelltext. Als nächstes muss man der Zeigervariablen eine sinnvolle Speicheradresse zuweisen. Im dritten Schritt kann man die Daten an dieser Adresse dann mithilfe des Zeigers manipulieren.

Deklaration:

```
int eineZahl = 0;
int *einZeiger;
```

Die erste Codezeile sollte keine Fragen aufwerfen, es wird eine Integervariable deklariert und initialisiert. In der zweiten Zeile wird eine Variable vom Typ „Zeiger auf Integer“ deklariert. Dies wird durch den Stern vor dem Variablennamen deutlich gemacht. Wenn als Schlüsselwort davor nicht int sondern beispielsweise double stände hätte man eine Variable vom Typ „Zeiger auf Double“ deklariert. Die verschiedenen Zeigertypen sind unterschiedlich und nicht kompatibel zueinander im Bezug auf Zuweisungen. Nach der Deklaration hat die Zeigervariable einen uns unbekannten Wert.

Adresszuweisung:

```
einZeiger = &eineZahl;
```

An dieser Stelle weisen wir dem Zeiger die Adresse der Integervariablen zu. Dies geschieht mithilfe des kaufmännischen Und, dass in C als Adressoperator bezeichnet wird. Dieser einstellige Operator gibt einfach die Adresse einer Variablen zurück, sodass sie in einer Zeigervariablen gespeichert werden kann. Wichtig ist hierbei, dass die Typen zusammenpassen. Hier haben wir einen „Zeiger auf Integer“ der die Adresse eines Integers aufnimmt. Wenn wir beispielsweise den Typen von eineZahl zu float wechseln würden, hätten wir hier einen Syntaxfehler. Die Zeigervariable zeigt jetzt auf eine andere Variable, die wir nun manipulieren können.

Dereferenzierung:

```
*einZeiger = 7;
```

Hier taucht wieder der Stern auf, der eine andere Bedeutung hat als bei der Deklaration. Der Effekt dieser Zeile ist, dass die Variable eineZahl nun den Wert 7 hat.

Der Stern ist in diesem Fall ein einstelliger Operator, der im Prinzip die gespeicherte Adresse des Zeigers auswertet und nun Zugriff auf diese Stelle im Speicher ermöglicht. In unserem Fall ist dies die Variable eineZahl. Man ist bei dieser Schreibweise nicht auf Zuweisungen beschränkt. Wenn sie in einem Ausdruck steht kann so auch einfach der Wert der Integervariablen ausgelesen werden.

Der Unterschied zwischen Schritt zwei und drei ist Wichtig. Man kann über den Wert der Zeigervariablen selbst sprechen, oder über den Wert der Variablen auf die gezeigt wird. Für diejenigen denen dieses Konzept neu ist kann es hilfreich sein mal ein bisschen mit diesen Operatoren herum zuspielen um ein besseres Gefühl für Zeiger zu bekommen.

1.4 Fehlerquellen

> „Writing in C or C++ is like running a chain saw with all the safety guards removed," – Bob Gray

Dieses Zitat sagt etwas überspitzt, dass man mit C (und C++) eine ganze menge Fehler produzieren kann. Das liegt zum Teil auch daran, dass man viele Freiheiten durch Zeiger bekommt, die bei unvorsichtigem umgehen, zu fehlerhaftem Code führen können. Wenn man die Initialisierung einer Zeigervariablen vergisst (schritt zwei der vorherigen Sektion) und den Zeiger dann Dereferenziert können zwei Dinge geschehen. Die Variable hat einen unbekannten Wert und zeigt auf einen ungültigen Speicherbereich. In diesem Fall wird das Programm mit einer Fehlermeldung abstürzen. Es kann aber auch passieren, dass zufällig eine gültige Speicheradresse enthalten ist die in irgendeinen Bereich des Programmes zeigt. Wenn man den Zeiger nun verwendet kann man Daten ändern und sich viel später im Programmablauf über diese Daten wundern. Diese Art von Fehlern sind schwer aufzufinden, da die Entdeckung des Fehlers und die Entstehung möglicherweise sehr weit auseinanderliegen. Wenn man Zeiger benutzt sollte man also immer sicher sein, dass sie tatsächlich an die richtige Stelle zeigen. Eine Möglichkeit sich vor ungewollten Änderungen zu schützen ist sich anzugewöhnen seine Zeiger mit dem Nullzeiger zu initialisieren.

```
int *einZeiger = 0;
```

Die Null ist hier also nicht die Zahl Null. Der Vorteil ist, dass der Nullzeiger in jedem Fall eine ungültige Adresse ist. Wenn man nun vergisst dem Zeiger eine Sinnvolle Adresse zuzuweisen und ihn Dereferenziert, dann wird es sofort zum Programmabsturz kommen und die Gefahr von schleichenden Fehlern ist damit gebannt. Dies ist allerdings auch kein Allheilmittel, denn es gibt Möglichkeiten wie ein Zeiger nach der Zuweisung einer gültigen Adresse ungültig werden kann. Wer sich dafür interessiert sollte sich mit der Dynamischen Speicherverwaltung in C auseinandersetzen.

2 Anwendung

2.1 Zeiger und Funktionen

Zeiger können als Parameter von Funktionen übergeben werden oder als Rückgabewert dienen. Bevor darauf eingegangen wird soll hier einmal der Grundlegende Ablauf eines Funktionsaufrufes erklärt werden.

Der Hauptspeicher der einem Programm zurVerfügung steht wird in zwei unterschiedliche Bereiche aufgeteilt. Der dynamische Speicher (Heap) ist ein Speicherbereich in dem das Programm zur Laufzeit Speicher anfordern und in beliebiger Reihenfolge wieder freigeben kann. Auf dem Stapelspeicher (Stack) kann nur der Speicher freigegeben werden der als letztes reserviert wurde. Der Stack (auch Funktionsstack) wird verwendet um die wichtigen Daten von aufgerufenen Funktionen zu speichern. Dazu gehören die Rücksprungadresse, lokale Variablen und übergebene Parameter. Der letzte Punkt ist wichtig. Übergebene Parameter werden, wenn man keine Zeiger verwendet, einfach kopiert und die aufgerufene Funktion arbeitet mit einer Kopie des Originalwertes. Eine Veranschaulichung dieses Prinzips gibt das folgende Beispiel

```
void eineFunktion(int eineZahl)
{
    eineZahl = 1;
}

int main()
{
    int eineZahl = 0;
    eineFunktion(eineZahl);
    printf("%i", eineZahl);
    exit(EXIT_SUCCESS);
}
```

Welcher Wert wird auf der Konsole ausgegeben, wenn man dieses kleine Programm laufen lässt? Da eineFunktion nur mit einer Kopie Arbeitet hat die Zuweisung keine Auswirkungen auf den Wert der Variablen eineZahl der main Funktion und es wird eine „0“ ausgegeben. Diese Art von Funktionsaufruf wird „call by value“ genannt.

Mit Zeigern wird es möglich, dass die aufgerufene Funktion die Variablen der aufrufenden Funktion manipulieren kann. Einen solchen Funktionsaufruf nennt man „call by referece“. Statt den Wert einer Variablen als Parameter zu übergeben, wird stattdessen die Adresse übergeben an der der Wert steht.

```
void eineFunktion(int *eineZahl)
{
    *eineZahl = 1;
}

int main()
{
    int eineZahl = 0;
    eineFunktion(&eineZahl);
```

```
    printf("%i", eineZahl);
    exit(EXIT_SUCCESS);
}
```

Die Unterschiede zum vorherigen Codebeispiel sind minimal. Statt eines Integers wird nun ein Zeiger auf einen Integer übergeben. Der Wert der auf den Stack kopiert wird, ist die Adresse an der die Variable eineZahl aus der main Methode liegt. Auf der Konsole wird bei diesem Programm eine „1“ ausgegeben.

Die Anwendungen für Funktionen des „call by referece“ Typs sind vielseitig. Man kann dadurch etwa die harte Grenze von nur einem Rückgabewert pro Funktion umgehen und einen Zeigerparameter als zusätzlichen Rückgabewert auffassen. Typischerweise sollten solche Probleme aber durch andere Konstrukte wie Strukturen gelöst werden. Wo wir gerade beim Thema sind: Strukturen werden, wenn sie als Parameter einer Funktion übergeben werden, vollständig auf den Stack kopiert wie andere Variablen auch. Bei großen Strukturen kann das eine Verschwendung von Ressourcen sein. Wenn man stattdessen nur einen Zeiger auf die Struktur als Parameter übergibt, wird auch nur der Zeiger auf den Stack kopiert. In kritischen abschnitten eines Programms die tausendfach durchlaufen werden, wie in tief verschachtelten Schleifen, kann sich das durchaus bemerkbar machen.

2.2 Zeiger und Arrays

Zeiger und Arrays haben in C viel gemeinsam. Was damit gemeint ist soll einmal an den folgenden drei Codeschnipseln erklärt werden, die alle die selbe Funktionalität haben. Es wird jeweils ein Array mit neun Integerfeldern deklariert, die ersten beiden Felder werden mit den zahlen 9 und 8 initialisiert und dann wird das zweite Feld mit printf auf der Konsole ausgegeben.

```
int main()
{
    int einArray[9];
    einArray[0] = 9;
    einArray[1] = 8;

    printf("%i", einArray[1]);
    exit(EXIT_SUCCESS);
}
```

```
int main()
{
    int einArray[9];
    *einArray = 9;
    *(einArray + 1) = 8;

    printf("%i", *(einArray+1));
    exit(EXIT_SUCCESS);
}
```

Im ersten Beispiel wird die gebräuchliche Array-Schreibweise genutzt, die aber hinter den Kulissen in die Zeiger-Schreibweise des zweiten Beispiels überführt wird.

Zeiger und Arrays sind an fast allen Stellen austauschbar. Man kann beispielsweise auch einen Zeiger mit Arrayschreibweise verwenden

```
int main()
{
    int einArray[9];
    int *einZeiger = &einArray[0];

    einZeiger[0] = 9;
    einZeiger[1] = 8;

    printf("%i", einZeiger[1]);
    exit(EXIT_SUCCESS);
}
```

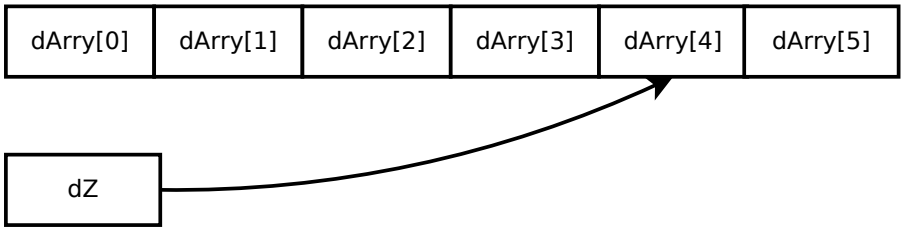
Ein Array ist also im Prinzip ein Zeiger, das auf das erste Element zeigt. Es gibt allerdings auch Unterschiede zwischen Zeigern und Arrays. Bei der Deklaration eines Zeigers wird nur Speicherplatz für die Zeigervariable selbst reserviert, bei Arrays wird zusätzlich der Speicherplatz für die Felder Reserviert. Außerdem kann man die Adresse auf die ein Zeiger zeigt während der Laufzeit verändern, bei einem Array ist das nicht möglich. Einmal Deklariert zeigt eine Arrayvariable immer auf den Anfang des reservierten Speicherbereiches und ähnelt damit mehr einem konstanten Zeiger. Der dritte Unterschied tritt beim verwenden von sizeof() auf. Bei einem Zeiger wird die Größe des Zeigers zurückgegeben, auf ein Array angewendet wird dagegen die Größe des reservierten Speicherbereiches zurückgegeben. In der Regel wird man die Ähnlichkeiten zwischen Arrays und Zeigern nicht so künstlich ausnutzen wie in den drei Beispielen weiter oben und die jeweils vorgesehene Syntax der beiden Konstrukte verwenden um Verwirrungen zu vermeiden. Der Nutzen dieser Gemeinsamkeiten hält sich in Grenzen, da die Ähnlichkeiten eher eine Folge der grundlegenden Architektur von C sind. Abgesehen von der Zeigerarithmetik kann man bei einigen Funktionsaufrufen statt eines Zeigers möglicherweise ein Array übergeben, oder umgekehrt.

2.2.1 Zeigerarithmetik

Zeiger und Arrays wirken in C zusammen um so Zeigerarithmetik möglich zu machen. Man betrachte einmal die folgenden Variablendeklarationen:

```
int *dZ, dArry[6];
dZ = &dArry[4];
```

Die Auswirkungen dieser beiden Zeilen im Speicher kann man sich wie folgt visualisieren:



Die Zeigervariable zeigt nach der Zuweisung auf das fünfte Element im Array. Wichtig ist, dass die Elemente des Arrays im Speicher alle nebeneinander liegen, weshalb die folgende Zeile Sinn macht.

```
dZ++;
```

Diese Schreibweise kennt man von Integertypen und erhöht dort den Wert der Variablen um eins. Im Kontext von Zeigern und Arrays, wird vom Compiler die Größe des zugrunde liegenden Types des Arrays berücksichtigt und die Zeigervariable gerade um so viel erhöht, dass sie auf das nächste Element im Array verweist. In unserem Fall also auf das sechste und damit letzte Element von dArry. Mit

```
dZ--;
```

kann man den Zeiger in die andere Richtung des Arrays verschieben. Statt den Kurzschreibweisen fürs inkrementieren/dekrementieren kann man natürlich auch folgende Syntax verwenden

```
dZ -= 2;
dZ = dZ + 3;
```

und ist somit nicht auf Einzelschritte beschränkt. Hierbei muss man nur aufpassen, dass man den Rand des Arrays nicht überschreitet, was zu Fehlern führen kann wie in der Sektion Fehlerquellen erklärt wurde. Die Möglichkeit Zeiger in einem Array zu verschieben führt zu einer in C Typischen Variante der for Schleife. Hier werden die Felder des Arrays auf übliche Art und weise initialisiert.

```
for (i = 0; i < 6; i++)
    dArry[i] = 0;
```

Und hier wird die Zeigervariante vorgestellt:

```
for(dZ = &dArry[0]; dZ < &dArry[6]; dZ++)
    *dZ = 0;
```

Die beiden Varianten haben dieselben Auswirkungen.

3 Quellen

„The C Book“ - Banahan, Brady, Doran http://publications.gbdirect.co.uk/c_book/ [05.05.11]

„C Wikibook“ http://de.wikibooks.org/wiki/C-Programmierung [05.05.11]

„C Standart ISO/IEC 9899“ http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf