Kapitel 8

Zeiger und Adressen

Aufgabe

Erstelle eine Funktion, die Variablenwerte des Hauptprogramms tauscht.

Lösungsversuch:

```
void tausche( int a, int b)
    {
    int t;

    t = a;
    a = b;
    b = t;
    }

void main()
    {
    int x = 1;
    int y = 2;

    printf( "Vorher:  %d %d\n", x, y);
    tausche( x, y);
    printf( "Nachher: %d %d\n", x, y);
    }
```

Diese Funktion erhält zwei Parameter, a und b, und tauscht deren Werte.

Wir rufen die Funktion `tausche`, um die Werte von x und y zu tauschen.

```
Vorher:  1 2
Nachher: 1 2
```

Fazit: Die Funktion ist wirkungslos, da sie nur auf Kopien der Variablen `x` und `y` arbeitet. Die Originale sind nicht betroffen.

Adressen

Zur Lösung des Tauschproblems übergeben wir der Funktion `tausche` die Adressen, an denen die zu tauschenden Werte im Speicher stehen. Die Funktion kann dann über die Adressen direkt auf die Originaldaten zugreifen.

> Die (Speicher-)Adresse einer Variablen erhält man, indem man dem Variablenamen den **Adressoperator** & voranstellt.

Mit Verwendung des Adressoperators ergibt sich dann der folgende Funktionsaufruf:

```
void main()
    {
    int x = 1;
    int y = 2;

    printf( "Vorher:  %d %d\n", x, y);
    tausche( &x, &y);
    printf( "Nachher: %d %d\n", x, y);
    }
```

Hier übergeben wir die Adresse der Variablen `x` und die Adresse der Variablen `y`.

Diese Art der Parameterübergabe kennen Sie bereits von `scanf`. Dort kam es ja auch darauf an, durch das Unterprogramm Variablenwerte im Hauptprogramm zu verändern.

Was eine Adresse genau ist, interessiert im Grunde genommen nicht. Wichtig ist nur, dass wir über die Adresse Zugriff auf das an der Adresse im Speicher hinterlegte Datum haben.

Die Schnittstelle der Funktion muss jetzt allerdings geändert werden, da keine `int`-Werte mehr übergeben werden sondern Adressen an denen im Speicher `int`-Werte stehen.

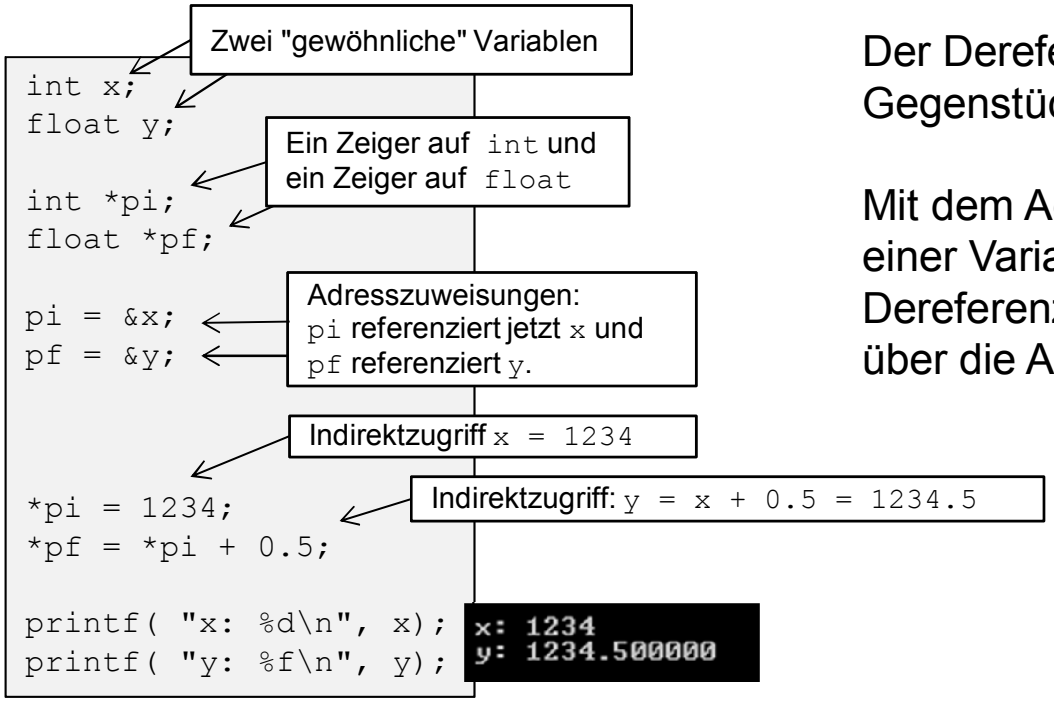
Zeiger

Eine Variable, in der die Adresse einer anderen Variablen gespeichert ist, nennen wir eine **Zeigervariable** oder kurz **Zeiger** bzw. **Pointer**.

Die Variable, deren Adresse im Zeiger gespeichert ist, bezeichnen wir als die durch den Zeiger **referenzierte** oder **adressierte Variable**.

Über einen Zeiger kann auf die Daten der referenzierten Variablen zugegriffen werden. Wir nennen dies **Indirektzugriff** oder auch **Dereferenzierung**. Zum Zugriff auf die referenzierte Variable verwendet man den Operator * (**Dereferenzierungsoperator)**.

Ist p ein Zeiger, so ist *p die referenzierte Variable.

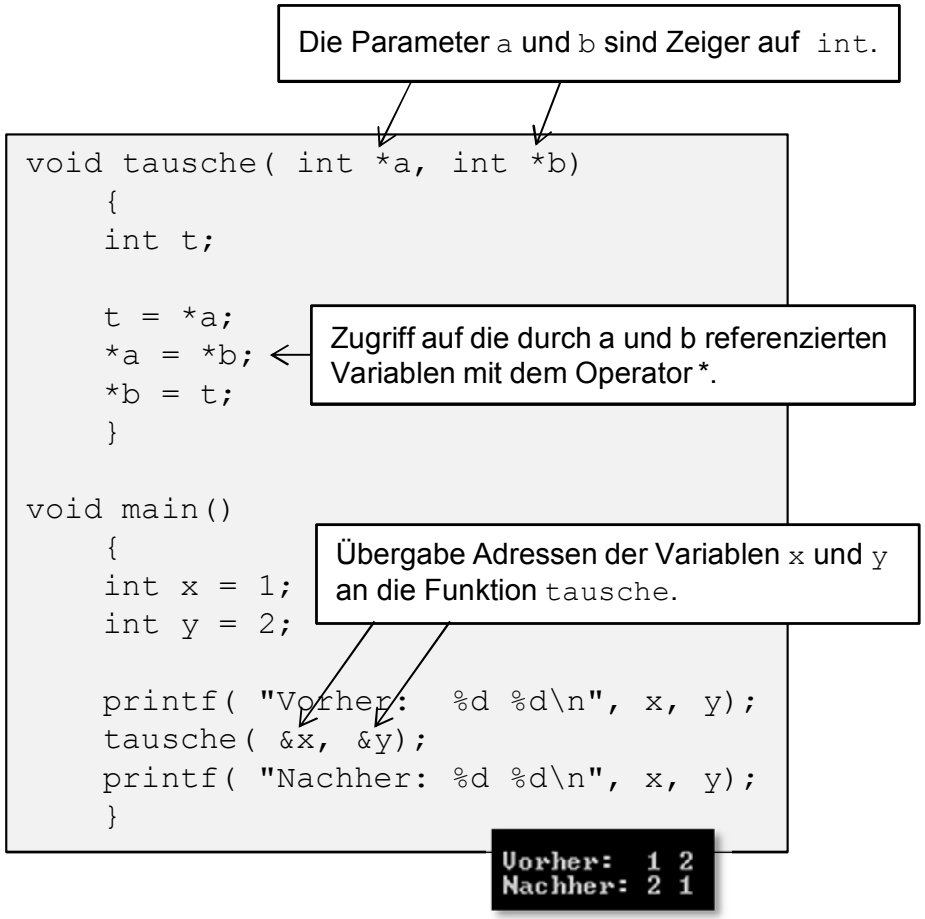


Der Dereferenzierungsoperator (*) ist das Gegenstück zum Adressoperator (&).

Mit dem Adressoperator kommen wir von einer Variablen zu ihrer Adresse, mit dem Dereferenzierungsoperator kommen wir über die Adresse zur Variablen.

Implementierung der Tauschfunktion mit Zeigern

Mit Adress- und Dereferenzierungsoperator kann man das Tauschproblem lösen:




```
void tausche( int *a, int *b)
    {
    int t;

    t = *a;
    *a = *b;
    *b = t;
    }

void main()
    {
    int x = 1;
    int y = 2;

    printf( "Vorher:  %d %d\n", x, y);
    tausche( &x, &y);
    printf( "Nachher: %d %d\n", x, y);
    }
```

Rückgabe von Werten über Zeiger

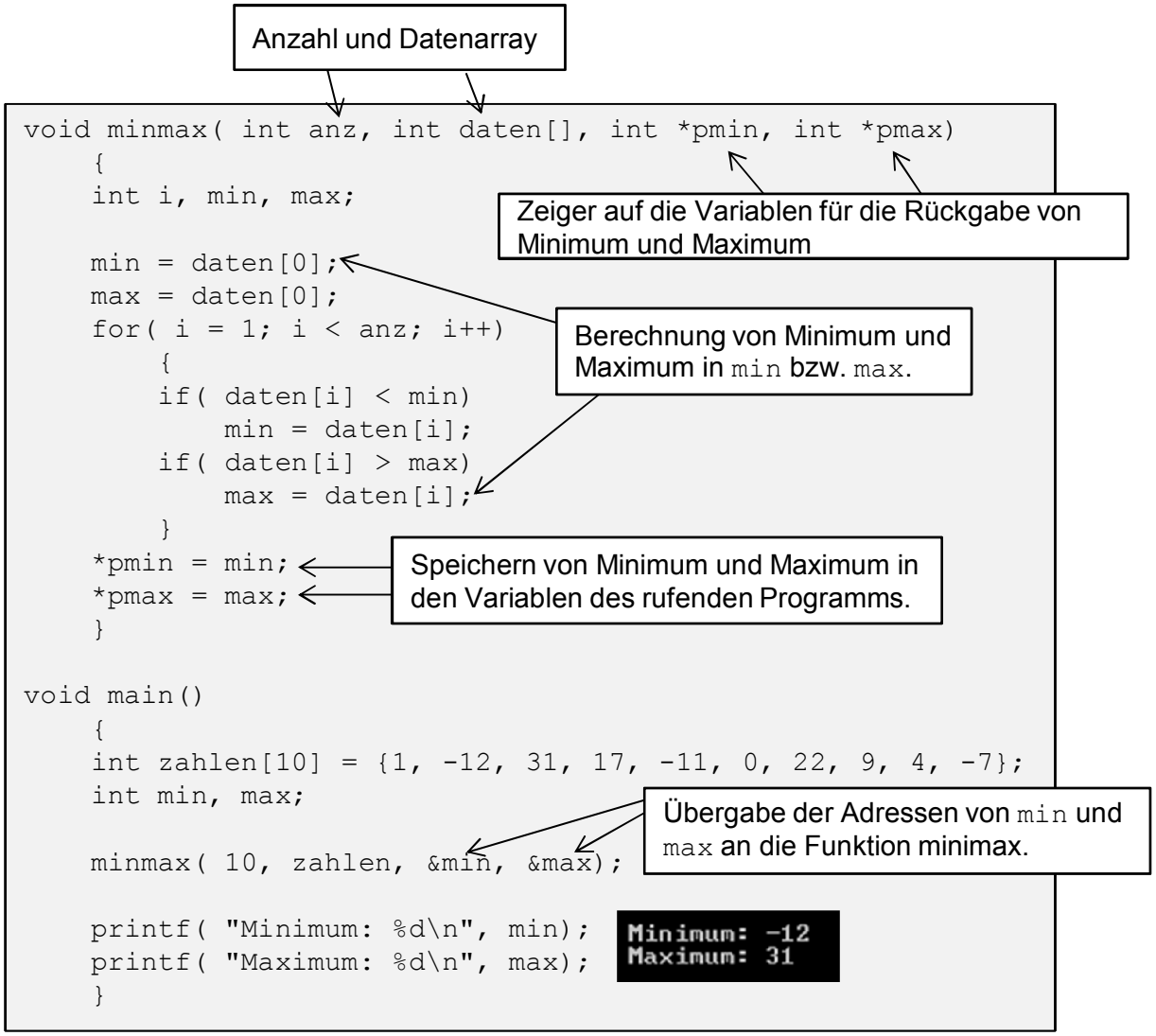
Man kann Zeiger verwenden, wenn man von einer Funktion mehr als einen Rückgabewert erwartet.

Das rufende Programm stellt Variablen bereit, in die die aufgerufene Funktion Rückgabewerte über Zeiger einträgt.

An der Schnittstelle werden die Adressen der Variablen übergeben.

Die nebenstehende Funktion `minmax` bestimmt Minimum und Maximum in einem Array und schreibt die Werte über Zeiger in die Variablen des rufenden Programms.

Bei `scanf` hatten wir dieses Prinzip schon immer verwendet.




```
void minmax( int anz, int daten[], int *pmin, int *pmax)
    {
    int i, min, max;

    min = daten[0];
    max = daten[0];
    for( i = 1; i < anz; i++)
        {
        if( daten[i] < min)
            min = daten[i];
        if( daten[i] > max)
            max = daten[i];
        }
    *pmin = min;
    *pmax = max;
    }

void main()
    {
    int zahlen[10] = {1, -12, 31, 17, -11, 0, 22, 9, 4, -7};
    int min, max;

    minmax( 10, zahlen, &min, &max);

    printf( "Minimum: %d\n", min);
    printf( "Maximum: %d\n", max);
    }
```

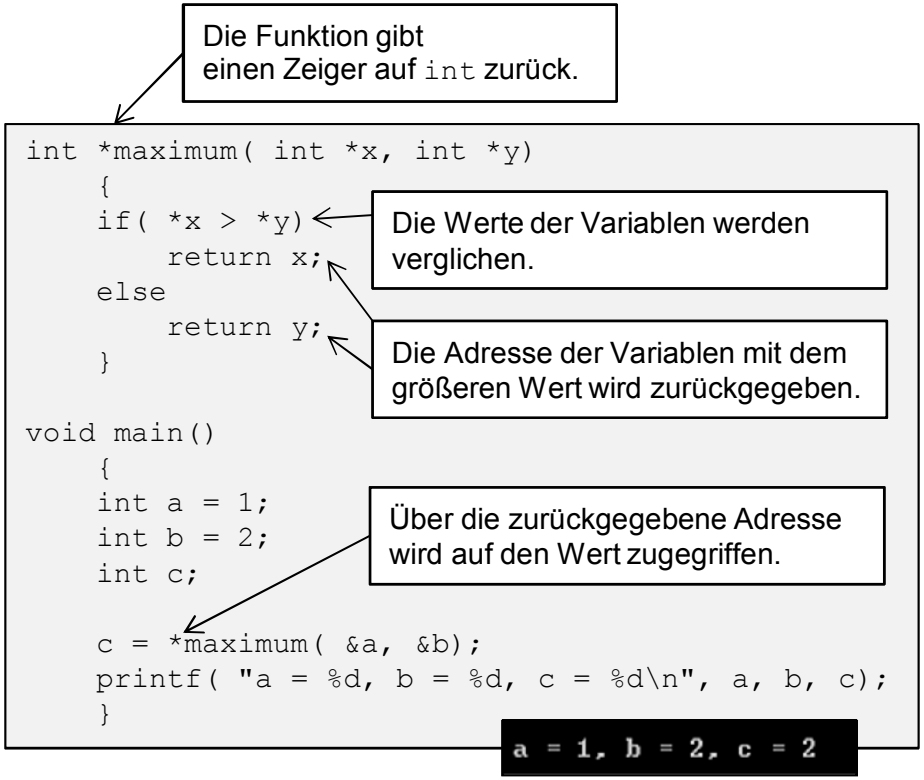
Rückgabe von Adressen

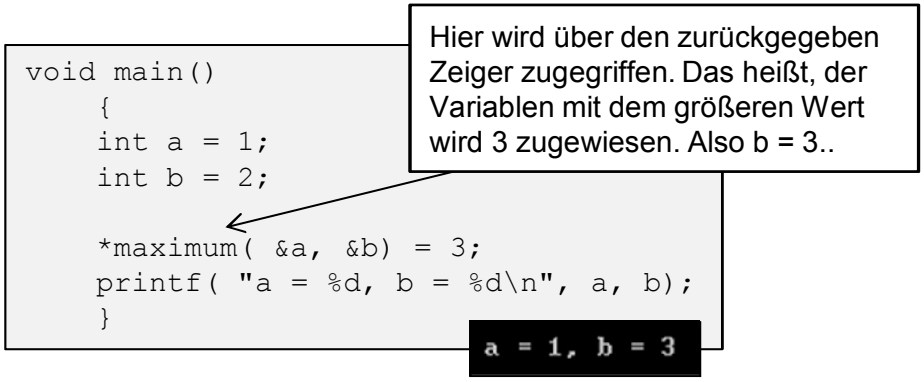
Eine Funktion kann auch eine Adresse zurückgeben.

Die nebenstehende Funktion `maximum` liefert mehr als nur den größeren von zwei Werten. Sie liefert die Adresse der Variablen, die den größeren Wert enthält.

Achtung:

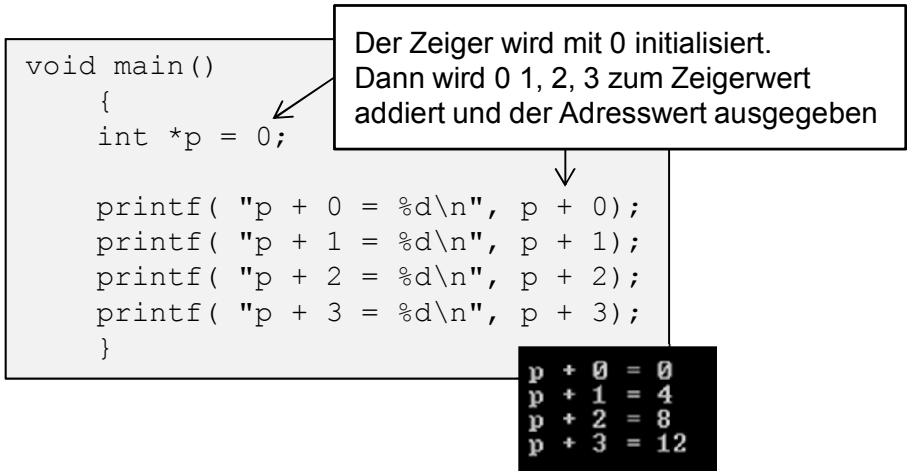
Eine Funktion kann nicht die Adresse einer internen, lokalen Variable zurückgeben, da diese Variable nach dem Rücksprung aus der Funktion nicht mehr existiert.

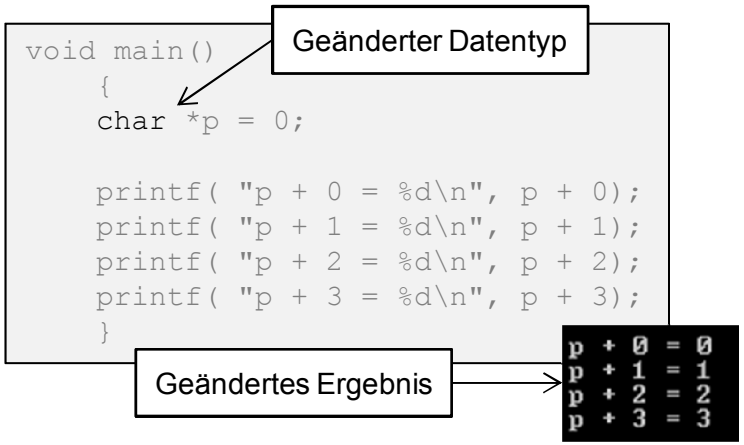




Zeigerarithmetik

Mit Zeigern kann gerechnet werden. Die Addition von zwei Zeigern ist nicht sinnvoll, aber zu einem Zeiger kann eine Zahl (offset) addiert werden. Das Ergebnis einer solchen Operation ist aber etwas anders, als auf den ersten Blick vermutet:





Wenn man zu einem Zeiger den Wert 1 addiert, erhöht sich der Adresswert des Zeigers um die Größe des Datentyps, auf den der Zeiger zeigt. Der Zeiger zeigt damit nach Addition von 1 auf das nächstmögliche Datum des gleichen Typs.

Bei Addition oder Subtraktion ganzer Zahlen ändert sich der Adresswert um entsprechende Vielfache der Größe des referenzierten Datentyps.

Man kann auch die Differenz (p-q) zweier Zeiger des gleichen Zeigertyps bilden. Als Ergebnis erhält man das Offset, das man zu q addieren müsste, um p zu erhalten. Das ist anschaulich die Anzahl der Daten des entsprechenden Typs, die zwischen p und q passen.

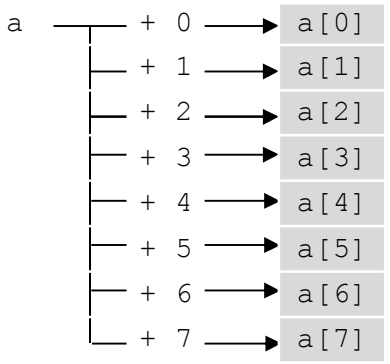
Zeiger und Arrays

Zeiger und Arrays sind eng miteinander verwandt.

Ist `a` ein Array, so ist `a` zugleich ein Zeiger auf das erste Element des Arrays.

Es gilt also `a = &a[0]`

bzw. `*a = a[0]`

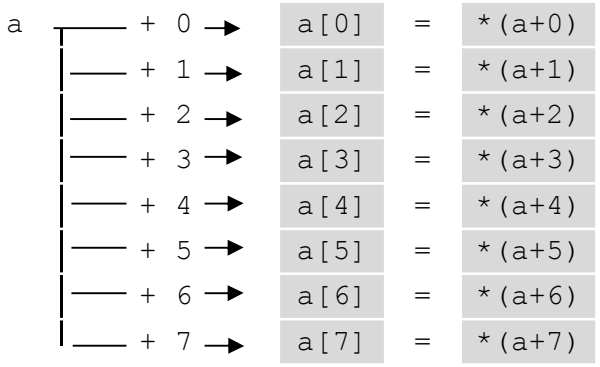


Wegen der Gesetze der Zeigerarithmetik folgt dann für einen Index i:

Ist `a` ein Array, so ist `a+i` ein Zeiger auf das i-te Element des Arrays.

Es gilt also: `a+i = &a[i]`

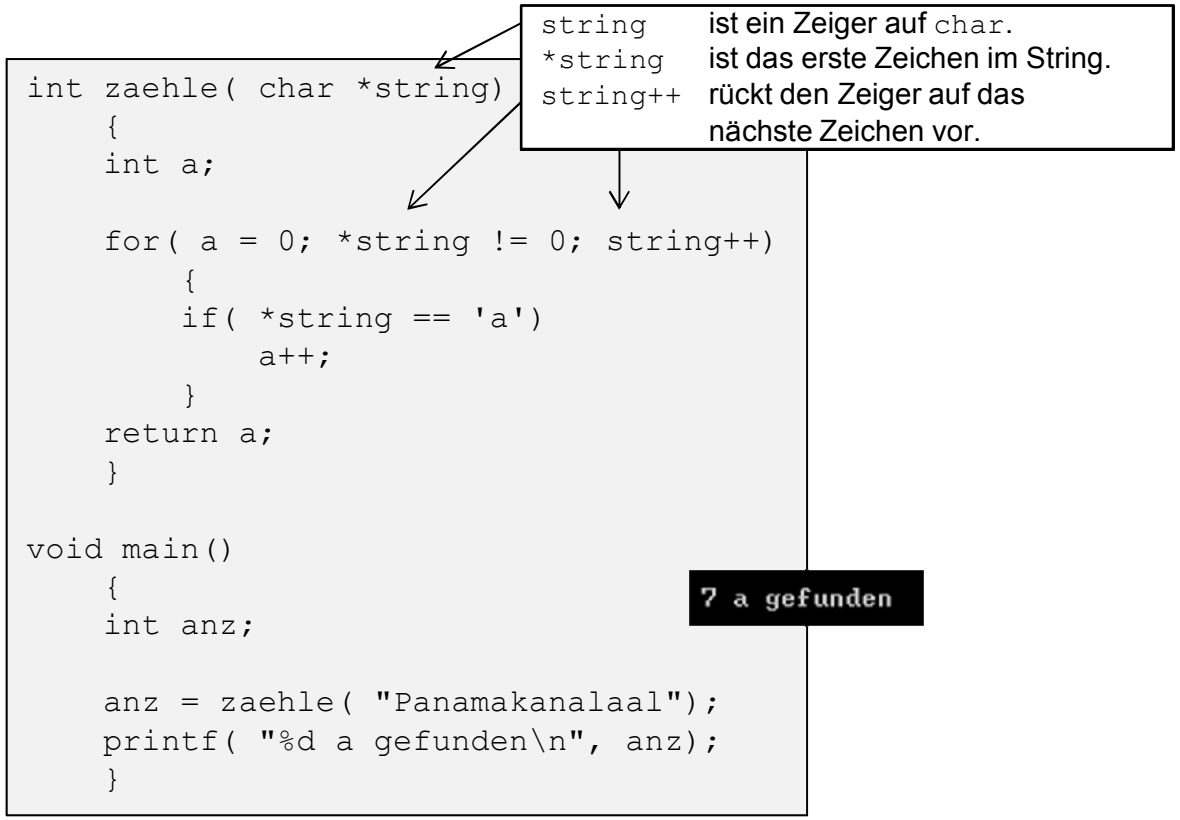
bzw. `*(a+i) = a[i]`



Arrays und Strings als Funktionsparameter

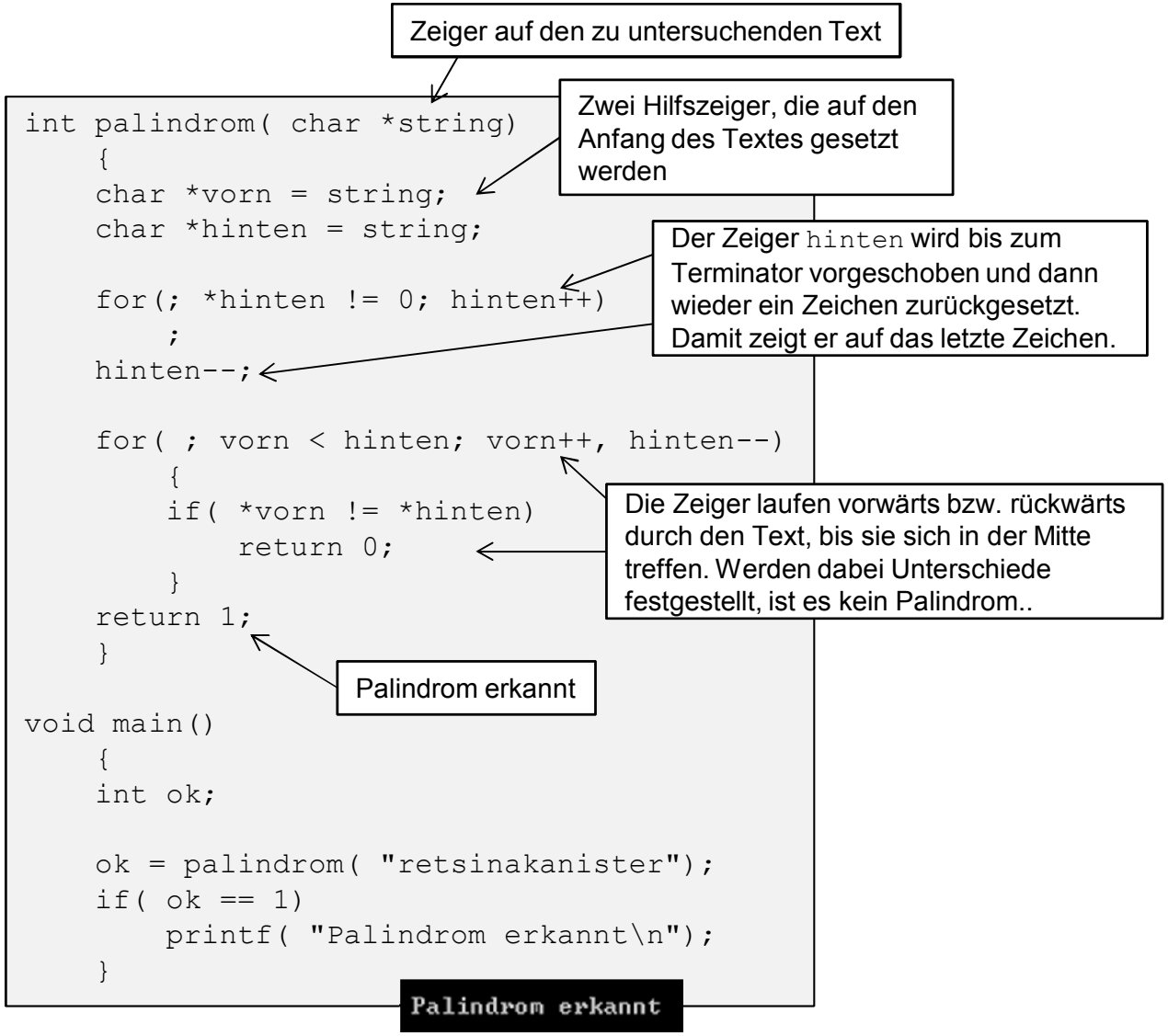
Wird ein Array (oder ein String) an einer Schnittstelle übergeben, so wird ein Zeiger übergeben und die Funktion erhält über diesen Zeiger direkten Zugriff auf die Daten im Array. Der Array wird an der Schnittstelle nicht kopiert, es entsteht nur eine Kopie des Zeigers.

Beispiel: Zähle die 'a' in einem String:



Beispiel - Palindromerkennung

Ein Palindrom ist ein Wort, das vorwärts und rückwärts gelesen gleich ist.



Aufgabe: **Suche wahlweise das Minimum oder das Maximum in einem Array.**

Lösungsansatz: Über einen Parameter (`modus`) wird gesteuert, welche Funktion (`minimum` oder `maximum`) verwendet werden soll.

```
int suche( int anz, int *daten, int modus)
    {
    int i, m;

    m = daten[0];
    for( i = 1; i < anz; i++)
        {
        if( modus == 1)
            m = minimum( m, daten[i]);
        else
            m = maximum( m, daten[i]);
        }
    return m;
    }

void main()
    {
    int zahlen[10] = {1, -12, 31, 17, -11, 0, 22, 9, 4, -7};
    int min, max;

    min = suche( 10, zahlen, 1);
    printf( "Minimum: %d\n", min);

    max = suche( 10, zahlen, 2);
    printf( "Maximum: %d\n", max);
    }
```

Über den Parameter `modus` wird gesteuert, ob das Minimum oder das Maximum berechnet werden soll.

Die Funktion `suche` wird mit unterschiedlichem `modus` aufgerufen.

```
Minimum: -12
Maximum: 31
```

```
int minimum( int a, int b)
    {
    if( a < b)
        return a;
    return b;
    }
```

```
int maximum( int a, int b)
    {
    if( a > b)
        return a;
    return b;
    }
```

Elegantere Lösung über →**Funktionszeiger**

Funktionszeiger

Eine Funktion hat, wie eine Variable, auch eine Adresse. Die Adresse einer Funktion kann man als Parameter an eine andere Funktion übergeben.

```
int minimum( int a, int b)
    {
    if( a < b)
        return a;
    return b;
    }
```

```
int maximum( int a, int b)
    {
    if( a > b)
        return a;
    return b;
    }
```

Minimum	`int minimum( int, int)`
Maximum	`int maximum( int, int)`
Allgemein	`int fkt( int, int)`

```
int suche( int anz, int *daten, int fkt( int, int))
    {
    int i, m;

    m = daten[0];
    for( i = 1; i < anz; i++)
        m = fkt( m, daten[i]);
    return m;
    }

void main()
    {
    int zahlen[10] = {1, -12, 31, 17, -11, 0, 22, 9, 4, -7};
    int min, max;

    min = suche( 10, zahlen, minimum);
    printf( "Minimum: %d\n", min);

    max = suche( 10, zahlen, maximum);
    printf( "Maximum: %d\n", max);
    }
```

Im Parameter `fkt` wird eine Funktion übergeben, die zwei int-Werte übergeben bekommt und einen int-Wert zurückgibt.

Die im Parameter `fkt` übergebene Funktion wird aufgerufen

Im dritten Parameter wird die bei der Suche zu verwendende Hilfsfunktion übergeben.

```
Minimum: -12
Maximum: 31
```

Callback-Funktionen

Im letzten Beispiel haben wir einen wichtigen Programmierstil kennengelernt:

Funktionen, die einer anderen Funktion als Parameter übergeben werden, um von dieser zurückgerufen zu werden, werden auch als Callback-Funktionen bezeichnet.

Das Programmieren mit Callback-Funktionen ist eine weit verbreitete Technik, die zum Beispiel bei der systemnahen Programmierung oder auch bei der Programmierung grafischer Benutzeroberflächen intensiv verwendet wird.