Die Deklaration von Objekten erfolgte bisher durch Angabe des Typs und des Namens für das Objekt. Durch die Deklaration wird ein gewisser Speicherbereich reserviert. Anschließend kann das Objekt mit Werten initialisiert werden.

Beispiel:

```
…
int zahl;       //reserviert 4 Byte für einen Integerwert
zahl = 9;       //Initialisierung
…
```

Der Zugriff auf das Objekt erfolgt durch den Bezeichner `zahl.`

Das folgende Beispiel demonstriert die Speicherbelegung für verschiedene Objekte:

```
#include <stdio.h>
#include <conio.h>
main()
{
      char z=0x41;
      int zahl=9;
      float fwert=3.1415;
      double dwert=1.2345;
      char text[12]="abcdefg";
      printf("\nAdresse: %u  Inhalt: %c  Laenge: %i Byte",&z,z,sizeof(z));
      printf("\nAdresse: %u  Inhalt: %i  Laenge: %i Byte",&zahl,zahl,sizeof(zahl));
      printf("\nAdresse: %u  Inhalt: %f  Laenge: %i Byte",&fwert,fwert,sizeof(fwert));
      printf("\nAdresse: %u  Inhalt: %lf  Laenge: %i Byte",&dwert,dwert,sizeof(dwert));
      printf("\nAdresse: %u  Inhalt: %s  Laenge: %i Byte\n",text,text,sizeof(text));
      return(0);
}
```

Das Beispiel liefert folgende Ausgabe:

Adresse: 1245052 Inhalt: A Laenge: 1 Byte
Adresse: 1245048 Inhalt: 9 Laenge: 4 Byte
Adresse: 1245044 Inhalt: 3.141500 Laenge: 4 Byte
Adresse: 1245036 Inhalt: 1.234500 Laenge: 8 Byte
Adresse: 1245024 Inhalt: abcdefg Laenge: 12 Byte

Das Beispiel zeigt für verschiedene Objekte die Startadressen, den Inhalt und den belegten Speicherbereich unter Verwendung der sizeof()-Funktion.

Man erkennt, dass die Objekte in der Reihenfolge abgelegt werden wie sie im Programm angelegt werden. Die tatsächlichen Adressen werden vom Compiler vergeben und können je nach System unterschiedlich sein.

Der Zugriff auf den Inhalt der Objekte erfolgt durch den vergebenen Bezeichner (innerhalb der printf()-Anweisung). Es würde aber ebenso ausreichen, wenn nur die Adresse des Objektes bekannt ist. Für die Verwaltung von Adressen gibt es in C++ einen eigenen Datentyp, den Pointer.

Eine Zeiger- oder Pointervariable ist eine Variable, deren Wert eine Adresse ist.

Die Deklaration einer Zeigervariablen erfolgt durch ein Sternchen ‚*' nach der Angabe des Datentyps. Der Bezeichner ist frei wählbar (hier `dptr`).

Beispiel:

```
double wert;  // reserviert Platz für eine double-Variable
double* dptr; // dptr ist ein Zeiger auf einen Wert vom Typ double
              // oder: dptr ist eine Adresse, ab welcher ein double-Wert
              // abgelegt werden kann
dptr=&wert    // Zeigervariable wird mit Adresse von wert initialisiert
```

Zeigervariable sind eigene Objekte, welche in einem 32-Bit-System 4 Byte Speicherplatz benötigen.

Zeigervariable sind vor ihrer Verwendung mit einer entsprechenden Adresse zu initialisieren!

Auf den Inhalt des Objektes kann jetzt auch über die Adresse zugegriffen werden. Der Zugriff auf den Inhalt erfolgt durch voranstellen eines Sternzeichens ‚*' vor der Zeigervariablen. Der Inhalt muss vom gleichen Typ wie die Zeigervariable sein, sonst liefert der Compiler eine Fehlermeldung.

z.B:

```
*dptr=3.1415;   //Belegt den Inhalt der Adresse dptr mit einem Wert
```

oder:

```
printf(„%lf“,*dptr);  //Gibt den Inhalt der Adresse dptr aus
```

Das folgende Beispiel zeigt die Zusammenhänge auf:

```
#include <stdio.h>
#include <conio.h>
main()
{
   double wert=3.1415; //Speicherplatz für double-Wert reservieren und initialisieren
   double* dptr;       //Speicherplatz für eine Zeigervariable auf int-Typ reservieren
   dptr=&wert;         //Zeiger mit Adresse des double-Wertes initialisieren
   printf(„\nAdresse wert: %u   Inhalt wert: %lf  Laenge wert: %i“,&wert,wert,sizeof(wert));
   printf("\nAdresse dptr: %u   Laenge dptr: %i",&dptr,sizeof(dptr));
   printf("\nInhalt *dptr: %lf\n",*dptr);
   return(0);
}
```

Das Programm liefert folgende Ausgabe:

```
Adresse wert: 1245048   Inhalt wert: 3.141500 Laenge wert: 8
Adresse dptr: 1245044   Laenge dptr: 4
Inhalt *dptr: 3.141500
```

Anmerkungen:

- Der Typ der Zeigervariablen ist double* (nicht double) und der Name ist dptr (nicht *dptr).
- Zeiger können auf jeden beliebigen Datentyp eingerichtet werden (z.B: int*, float*, double*,…).
- In obigem Beispiel wird dem Zeiger dptr der Wert einer Adresse zugewiesen, welche auf eine Zahl vom Typ double zeigt. Der Typ der Zeigervariablen muss mit dem Typ des reservierten Speicherbereiches übereinstimmen. Zeiger müssen vor ihrer Verwendung mit einer Adresse initialisiert werden!
- Zeigervariable haben grundsätzlich eine Länge von 4 Byte (in 32-Bit-Betriebssystemen).

Der *-Operator findet verschiedene Verwendungen. Die Bedeutung ergibt sich aus dem Zusammenhang in welchem er verwendet wird. Bekannt ist er schon als Multiplikationsoperator, wenn links und rechts von ihm Zahlenwerte stehen. Nach einer Typangabe bewirkt der Operator die Deklaration eines Zeigers (z.B: `double*, int*, char*,…`). Als Operator vor einer Zeigervariablen gibt er den Inhalt der Adresse wieder. (z.B.: `*dptr` = Inhalt der Adresse dptr) .

An dieser Stelle tut sich die Frage auf, welchen Vorteil Zeiger eigentlich liefern, wenn sie doch auch nur Werte für bekannte Datentypen beinhalten. Die Frage wird in späteren Kapiteln noch ausführlicher behandelt.

Vorteile sind:

Zeigervariable belegen u.U. weniger Speicherplatz als die Objekte selber. Zum Beispiel: Ein double-Wert belegt 8 Byte, der Zeiger nur 4 Byte. Benutzerdefinierte Objekte können beliebig groß sein (Siehe später bei Arrays, Strukturen oder Klassen). Der Zeiger (auf ein Objekt) belegt immer nur 4 Byte.

Die Übergabe von großen Objekten an Funktionen wird erleichtert. Wie alle anderen Werte auch können Zeiger auch Übergabeparameter in Funktionen sein. Das ist besonders wichtig bei der Übergabe von Strings oder sonstigen Datenfeldern. Außerdem können Objekte dann durch Funktionen verändert werden.

Mit Zeigern kann man rechnen, zum Beispiel bei Datenfeldern (Arrays), wo der fortlaufende Zugriff auf Werte des Datenfeldes durch inkrementieren oder dekrementieren eines Indizes erfolgt. Auf die Bedeutung von Zeigern bei Datenfeldern wird später noch besonders eingegangen.

Viele vordefinierte Funktionen erwarten als Übergabeparameter eine Adresse oder eben einen Zeiger (z.B.: Die scanf()-Funktion zur Dateneingabe).

Übungen:

1) Welche Ausgabe liefert das folgende Programm?

```
#include <stdio.h>
#include <conio.h>
main()
{
	int zahl1=5, zahl2=7;
	int* iptr1;
	int* iptr2;
	iptr1=&zahl1;iptr2=&zahl2;
	printf("\nErgebnis: %i\n",*iptr1**iptr2);
	return(0);
}
```

2) Welcher Wert wird für Ergebnis ausgegeben?

```
#include <stdio.h>
#include <conio.h>
main()
{
	int zahl1=5, zahl2=7;
	int* iptr1;int* iptr2;
	iptr1=&zahl1;iptr2=&zahl2;
	*iptr1=zahl2;
	*iptr2=*iptr1+zahl1;
	printf("\nErgebnis: %i\n",*iptr1+*iptr2);
	return(0);
}
```

3) Welchen Fehler enthalten folgende Programme?

```
#include <stdio.h>
main()
{
	int zahl=5;
	double wert=2.5;
	int* dptr=&zahl;
	double* iptr;
	printf("\nErgebnis: %i\n",*dptr+*iptr);
	return(0);
}
```

```
#include <stdio.h>
main()
{
	int zahl=5;
	int* iptr=zahl;
	printf("\nErgebnis: %i\n",zahl**iptr);
	return(0);
}
```

```
#include <stdio.h>
main()
{
	double zahl=5;
	int* iptr;
	iptr=&zahl;
	printf("\nErgebnis: %i\n",zahl**iptr);
	return(0);
}
```