Jedes Programm benötigt Objekte. Ein **Objekt** ist ein reservierter Bereich im Systemspeicher in welchem Informationen abgelegt werden. Informationen sind z.B. Zahlen, Buchstaben, Texte, Bilder, Audio-/Video-Dateien, Tabellen, Datenbanken,… . Im Beispiel Kreisberechnung existieren Objekte mit den Bezeichnern (Variablennamen) **`radius`**, **`umfang`**, **`flaeche`** und **`pi`**. Alle Objekte in diesem Beispiel beinhalten Fließkommazahlen bzw. die Objekte sind vom Typ **`double`**.

Objekte werden eindeutig beschrieben durch ihren Typ und einen beliebig wählbaren Bezeichner.

Objekte müssen vor ihrer Verwendung im Programm deklariert werden. Eine einfache **Deklaration** erfolgt in der Form

```
typ bezeichner;
```

z.B.:

```
float volumen;
```

reserviert Speicherplatz für eine Fließkommazahl mit dem Bezeichner **`volumen.`**

Objekte gleichen Typs können in einer Anweisung deklariert werden, wenn die Bezeichner durch Kommata getrennt werden.

z.B.:

```
float radius,umfang,flaeche;
```

Der Bezeichner ist frei wählbar. Zur besseren Lesbarkeit des Programms sollte er jedoch einen Bezug zum Inhalt des Objektes herstellen. Der Bezeichner darf nicht mit einer Zahl beginnen. Einleitende Buchstaben mit einer Zahlenfolge sind zulässig. Groß- und Kleinschreibung wird unterschieden. Sonderzeichen sind bis auf wenige Ausnahmen, z.B. dem Unterstrich ‚_', nicht zulässig. Die Länge des Bezeichners ist beliebig. In der Programmiersprache C reservierte Schlüsselwörter dürfen nicht verwendet werden. Jede Deklarationsanweisung ist mit einem Semikolon abzuschließen.

Durch den Typ in der Deklaration wird die Art der Information, z.B. ganze Zahl oder Fließkommazahl, festgelegt. Außerdem bestimmt der Typ den erforderlichen Speicherplatz. Die Größe des Speicherbereiches wird in Byte (= 8 Bit Binärinformation) angegeben. Je mehr Speicherbereich für eine Zahl reserviert wird, desto größer der mögliche Zahlenbereich bzw. die Genauigkeit der (Fließkomma-)Zahl.

Die folgende Tabelle zeigt die vordefinierten Typen für Zahlen, welche in der Programmiersprache C Verwendung finden.

Grundlegende Datentypen

Datentyp	*Länge in Byte*	*Wertebereich*
char	1	ganze Zahlen -128 bis + 127
unsigned char	1	ganze Zahlen 0 bis 255
short	2	ganze Zahlen -32,768 bis 32,767
unsigned short	2	ganze Zahlen 0 bis 65,535
int	4	ganze Zahlen -2,147,483,648 bis 2,147,483,647
unsigned int	4	ganze Zahlen 0 bis 4,294,967,295
long	4	ganze Zahlen -2,147,483,648 bis 2,147,483,647
unsigned long	4	ganze Zahlen 0 bis 4,294,967,295
long long	8	ganze Zahlen -9,223,372,036,854,775,808 bis 9,223,372,036,854,775,807
unsigned long long	8	ganze Zahlen 0 bis 18,446,744,073,709,551,615
float	4	Fließkommazahlen $3.4 \cdot 10^{+/-38}$ - Genauigkeit: 7 Stellen
double	8	Fließkommazahlen $1.7 \cdot 10^{+/-308}$ - Genauigkeit: 15 Stellen

Durch die bisherigen Deklarationen wurde lediglich Platz im Systemspeicher reserviert. Einen konkreten Wert haben die Objekte dadurch nicht erhalten. Der Inhalt der Objekte ist tatsächlich undefiniert (auch nicht 0!). Durch die **Initialisierung** erhalten die Objekte einen konkreten Inhalt. Die Initialisierung erfolgt durch den Zuweisungsoperator ‚=' (Gleichheitszeichen). Eine allgemeine Initialisierung erfolgt mit

```
bezeichner = ausdruck;
```

Links vom Gleichheitszeichen steht der Bezeichner des Objektes und rechts der Wert, welcher ihm zugewiesen wird. Die Seiten dürfen nicht vertauscht werden!

z.B.:

```
int i;
float zahl;
i=10;           //die Variable i erhält den Wert 10.
zahl=1.1415;    //die Variable zahl erhält den Wert 1.1415
```

Deklaration und Initialisierung können wahlweise auch in einer einzelnen Anweisung erfolgen, indem der Zuweisungsoperator mit in die Deklaration eingefügt wird.

z.B.:

```
int i = 10;           //Die Variable i wird deklariert und mit
                      //10 initialisiert
float zahl = 1.1415; //Die Variable zahl wird deklariert und mit
                      //1.1415 initialisiert
```

Der beim Zuweisungsoperator rechts stehende Ausdruck kann ein beliebiger mathematischer Ausdruck sein. Zugewiesen wird immer das Ergebnis der Berechnung.

z.B.:

```
double betrag=10.0;                  //betrag ist 10.0
double brutto=betrag+betrag*0.19;    //brutto ist betrag + 19% des
                                     //Betrages = 11.90
```

Will man verhindern, dass einem Objekt ein anderer als der initialisierte Wert zugewiesen wird, kann der sogenannte Modifizierer **`const`** der Typangabe vorangestellt werden.

z.B.:

```
const double pi=3.141593 ; //der Wert für pi kann nicht mehr verändert werden
```

Im Allgemeinen wird auf ein Objekt mit dem zugehörigen Bezeichner zugegriffen. Manche Funktionen in C benötigen jedoch zusätzlich die Information wo das Objekt im Speicher abgelegt ist (z.B. die **`scanf()`**-Funktion zur Eingabe von Werten). Dazu muss man sich den Aufbau des Systemspeichers etwas genauer anschauen.

Der Systemspeicher ist in Bytes (=8 Bit) organisiert. Jeder einzelne Speicherplatz hat eine Nummer, genannt **Adresse**. Beispielsweise hat ein Systemspeicher mit 512Mbyte die Speicherplatzadressen 0 bis 536870911 (=512*1024*1024). Jedes Objekt hat eine Startadresse und belegt ab dieser Adresse je nach Größe weitere Adressen in aufsteigender Reihenfolge. Auf die Adresse, welche der C-Compiler für ein Objekt belegt, hat der Programmierer keinen Einfluss. Man kann diese Adresse aber ermitteln indem man vor dem Bezeichner für das Objekt den sogenannten Adressoperator ‚&' voranstellt.

Beispiel:

Es werden folgende Objekte deklariert und initialisiert:

```
short wert=10;
float zahl=1.4142;
```

Die Speicherbelegung könnte folgendermaßen aussehen:

Adress-bezeichner	Adresse	Inhalt	Objekt-bezeichner
	2005	10	wert
&wert	2004		
	2003	1.4142	zahl
	2002		
	2001		
&zahl	2000		

Das Objekt **`wert`** belegt 2 Bytes ab der Startadresse 2004 und hat den Inhalt 10. Der Bezeichner für die Startadresse ist **`&wert`**.

Das Objekt zahl belegt 4 Bytes ab der Startadresse 2000 mit dem Bezeichner &zahl und hat den Inhalt 1.4142.

Die Adressen sind hier willkürlich gewählt und sind abhängig vom Computersystem. Zum Verständnis ist es wichtig klar zwischen dem Inhalt eines Objektes und der (Start-)Adresse auf welcher dieses abgelegt ist zu unterscheiden!

Jedes Objekt hat einen Inhalt und eine Startadresse. Die **Größe des Objektes** bestimmt die Anzahl der benötigten Speicherplätze in Byte. Mit der **`sizeof(bezeichner)`**-Funktion lässt sich die Größe des Objektes bestimmen. Der Bezeichner für das Objekt muss der Funktion innerhalb der () übergeben werden.

Beispiel:

```
...
double zahl=1.4142;
printf(„\n%i“,sizeof(zahl));
...
```

Das Ergebnis ist der Wert 8 (Länge des Typs **`double`**).

Beispiel:

Dieses Beispiel demonstriert die oben dargelegten Zusammenhänge. Es werden 4 Objekte vom Typ char, int, float und double erzeugt. Anschließend werden die Startadresse, der Inhalt und die Größe der Objekte ausgegeben.

```
//Grundlegende Deklarationen
#include<stdio.h>
#include<conio.h>
int main()
{
char z=41;
int izahl=100;
float fzahl=1.4142;
double dzahl=3.1415927;
printf("\nAdresse: %u Inhalt: %i Groesse: %i",&z,z,sizeof(z));
printf("\nAdresse: %u Inhalt: %i Groesse: %i" ,&izahl,izahl,sizeof(izahl));
printf("\nAdresse: %u Inhalt: %f Groesse: %i" ,&fzahl,fzahl,sizeof(fzahl));
printf("\nAdresse: %u Inhalt: %lf Groesse: %i\n" ,&dzahl,dzahl,sizeof(dzahl));
return(0);
}
```

Die Ausgabe könnte folgendermaßen aussehen:

```
Adresse: 1310563 Inhalt: 41 Groesse: 1
Adresse: 1310548 Inhalt: 100 Groesse: 4
Adresse: 1310536 Inhalt: 1.414200 Groesse: 4
Adresse: 1310520 Inhalt: 3.141593 Groesse: 8
```

Die Adressen sind vom Computersystem abhängig.

Anmerkungen zum Programm:

Innerhalb der printf()-Anweisungen sind im so genannten Formatstring (innerhalb der „“) Platzhalter für die auszugebenden Werte eingefügt. Die Platzhalter beginnen mit dem Prozentzeichen ‚%’. Hinter diesem Zeichen steht ein Buchstabe, welcher das Format für die Ausgabe festlegt.

Es bedeuten:

%u - Ausgabe als große ganze Zahl
%i - Ausgabe als Integerzahl
%f - Ausgabe als Fließkommazahl (Typ float)
%lf - Ausgabe als Fließkommazahl (Typ double)

Auf die printf()-Funktion wird in einem späteren Kapitel noch genauer eingegangen.

Übung 1:

Welche Fehler enthält folgende Deklarationsanweisung?

```
float zahl; int Zahl;_zahl; int 1zahl;
```

Übung 2:

Welche Fehler enthält das folgende Programm? Berichtigen und vervollständigen Sie bitte das Programm!

```
#include<stdio.h>
#include<conio.h>
int main()
{
      float radius=2,durchmesser,umfang;
      2*radius=umfang;
      durchmesser=umfang*pi;
      printf("\nDurchmesser: %f Umfang: %i",durchmesser,umfang);
      Return(0);
}
```

Übung 3:

Das folgende Programm zur Berechnung der Fläche eines Rechteckes läßt sich trotz einiger Warnungen kompilieren und starten. Zur Laufzeit verhält sich das Programm fehlerhaft. Wo liegt die Ursache?

```
//Rechteckfläche berechnen
#include<stdio.h>
#include<conio.h>
int main()
{
      double laenge=0,breite=0,flaeche=0;
      printf("\nBitte Laenge eingeben: ");scanf_s("%lf",laenge);
      printf("\nBitte Breite eingeben: ");scanf_s("%lf",breite);
      flaeche=laenge*breite;
      printf("\nDie Flaeche betraegt: %lf\n",flaeche);
      return(0);
}
```