Die Programmiersprache C besteht aus relativ **wenigen Schlüsselwörten**. Für häufig benötigte Aufgaben stehen in Bibliotheken leistungsfähige **Funktionen** bereit, wie z.B. **`printf()`** und **`scanf()`** (Header-Dateien). Diese Bibliothek kann vom Benutzer beliebig um eigene Funktionen erweitert werden.

Diese Technik bietet zwei **Vorteile**:

- Häufig wiederkehrende Programmteile brauchen nur einmalig programmiert zu werden und lassen sich auch in andere Programme einbinden. Solche Funktionen sollten allgemeingültig deklariert werden. Im Laufe der Zeit entsteht so u.U. eine eigene Programmbibliothek. Es ist auch möglich mehrere Funktionen in einer **eigenen Header-Datei** abzulegen und mit **`#include`** in andere Programme einzubinden.

- Größere Programme lassen sich besser strukturieren. Dabei wird versucht, das Hauptprogramm so kurz und übersichtlich wie möglich zu halten und die eigentlichen Aufgaben den Funktionen zu übertragen (**Modulare Programmierung**). Ein anschauliches Beispiel dafür bilden menügesteuerte Programme(s.u.).

Deklaration und Initialisierung:

Benutzerdefinierte Funktionen sind nach folgendem Muster zu deklarieren:

```
typ name(typ parameter1, typ parameter2,…)
      {
      anweisungen;
      …;
      return(…);
      }
```

- Funktionen müssen vor ihrer Verwendung mit **Typ**, **Name** und **Übergabeparametern** deklariert werden.

- Funktionen haben höchstens **einen Rückgabewert**. Der Rückgabewert wird mit der **`return(…)`**-Anweisung an das Hauptprogramm übergeben. Der Typ des Rückgabewertes muss mit dem deklarierten Typ der Funktion übereinstimmen. Funktionen ohne Rückgabewert sind vom Typ **`void`**.

- Den Funktionen können beliebig viele Parameter in den ()-Klammern übergeben werden. Bei der Deklaration ist der Typ des Übergabeparameters und ein Bezeichner anzugeben. Unter diesem Bezeichner ist die Variable der Funktion bekannt (**Formalparameter**). Die Typen der Formalparameter können auch unterschiedlich sein.

- Der Aufruf der Funktion erfolgt im Hauptprogramm durch Angabe des Namens mit den Parametern in ()-Klammern (**Aktualparameter**). Die Reihenfolge der Übergabeparameter muss der Reihenfolge in der Deklaration entsprechen. Die Typen der Aktualparameter müssen mit den in der Funktion deklarierten Typen übereinstimmen.

Die Technik von Übergabeparameter und Rückgabewert lässt sich am Beispiel einer einfachen mathematischen Funktion erklären: Die Header-Datei **`math.h`** stellt z.B. die Funktion **`sin()`** zur Berechnung des Sinus eines Winkels bereit. Mit den Anweisungen

```
…
float s, pi=3.1415;
s=sin(30*pi/180);
printf(„\nErgebnis : %8.3f“, s);
…
```

wird der Sinuswert von 30° errechnet und ausgegeben. Übergabeparameter an diese Funktion ist der mathematische Ausdruck **`30*pi/180`** und das Ergebnis (der Rückgabewert) wird der Variablen s zugewiesen (Die Funktion erwartet den Winkel im Radiant-Maß; deswegen erfolgt die Umrechnung von Grad nach Radiant).

Auf die Zuweisung des Funktionsergebnisses zu der Variablen s kann verzichtet werden. Als Ausgabe wird grundsätzlich das Ergebnis der Funktion geliefert. Deswegen führen die folgenden Anweisungen zum gleichen Ergebnis:

```
float pi=3.1415;
printf(„\nErgebnis : %8.3f“,sin(30*pi/180));
```

Es gibt auch Funktionen, welche keine Übergabeparameter verlangen:

```
char c;
c = getch();
```

In diesem Beispiel besitzt die Funktion **`getch()`** keine Übergabeparameter aber sehr wohl einen Rückgabewert (den ASCII-Code der gedrückten Taste). Die runden Klammern '()' müssen bei dem Funktionsaufruf trotzdem angegeben werden.

Noch ein Beispiel:

Die in der Header-Datei **`math.h`** deklarierte Funktion **`pow()`** berechnet einen Exponentialwert. Im Hilfe-System findet man folgende Deklaration (**Funktionsprototyp**):

```
double pow( double x, double y);
```

Die Funktion erwartet zwei Übergabeparameter vom Typ double. Der Rückgabewert ist vom Typ double das Ergebnis x^y.

```
printf(„\nErgebnis: %8.3lf“, pow(2.0,3.0);       // Ergebnis ist 8.000
```

Der Anwender der Funktion braucht sich nicht um die eigentliche Berechnung zu kümmern. Die Kenntnis von Übergabe- und Rückgabeparameter und deren Typ reicht aus, um die Funktion sinnvoll einsetzen zu können.

Nun zu den benutzerdefinierten Funktionen.

Im folgenden Demonstrationsbeispiel wird eine Funktion deklariert, welche das Quadrat eines int-Wertes berechnet. Im Hauptprogramm wird diese Zahl eingegeben und das Quadrat der Zahl ausgegeben.

```
1:    #include<conio.h>
2:    #include<stdio.h>

//Funktionsdeklaration

3:    int quadrat(int x)
4:    {
5:    return(x*x);
6:    }

//Hauptprogramm

7:    int main()
      {
            int zahl;
            do
            {
            printf("\nGanze Zahl: ");
            scanf("%i",&zahl);
8:          printf("\nQuadrat: %i",quadrat(zahl));
            printf("\nEnde = 'ESC' ...");
            }
            while (getch()!=0x1b);
9:          return(0);
      }
```

Erläuterungen:

Zeilen 1 und 2:

Einbinden der vordefinierten Funktionen.

Zeile 3:

Deklaration der Funktion. Der Typ des Rückgabewertes ist int. Der Name der Funktion ist quadrat . Der Übergabewert ist vom Typ int und hat innerhalb der Funktion die Bezeichnung x (**Formalparameter**). Achtung: Die Variable x ist nur innerhalb der Funktion gültig (sog. **Lokale Variable**). Der Name des Formalparameters ist frei wählbar, kann sogar mit dem Aktualparameter des Hauptprogramms übereinstimmen. Aber selbst bei gleicher Bezeichnung sind es unterschiedliche Variablen!

Zeile 4:

Beginn des Anweisungsblockes (Funktionsrumpf). Hier stehen die Anweisungen, welche die Funktion ausführen soll. Achtung: Der Anweisungsblock ist Teil der Funktionsinitialisierung. Deshalb darf vor dem Anweisungsblock bzw. hinter der Funktionsdeklaration kein Semikolon (;) stehen.

Zeile 5:

Die **`return()`**-Anweisung beinhalt den Wert, welche die Funktion als Ergebnis liefern soll. Nach return erfolgt ein Rücksprung zum Hauptprogramm bzw. zur aufrufenden Funktion.

Zeile 6:

Ende des Anweisungsblockes.

Zeile 7:

Beginn des Hauptprogramms, welches die Funktionen verwenden kann. Funktionen müssen vor dem Hauptprogramm deklariert werden.

Zeile 8:

Aufruf der Funktion und Ausgabe des Ergebnisses. Der Funktion wird die Variable **`zahl`** übergeben (**Aktualparameter**). Die Ausgabe ist das Ergebnis der Funktion. Die Funktion wird mit dem Namen **`quadrat(zahl)`** aufgerufen. Das Programm verzweigt zum Unterprogramm. Der Wert von Zahl wird in den **Formalparameter** x **kopiert**! Die Variable x ist nur innerhalb der Funktion bekannt. Man sagt die Variable ist **lokal** in der Funktion. Die Variable **`zahl`** des Hauptprogramms ist der Funktion unbekannt! Genauso ist dem Hauptprogramm die Variable x nicht bekannt. Die Art der Parameterübergabe nennt man auch **Übergabe per Wert**!

Zeile 9:

Auch **`int main()`** ist eine Funktion! Der Rückgabewert dieser Funktion gelangt an das aufrufende Programm. Das ist in diesem Falle die DOS-Eingabeaufforderung.

Im folgenden Beispiel wird im Hauptprogramm eine ganze Zahl eingegeben. Für den Fall, dass die Zahl größer 100 ist erfolgt ein Funktionsaufruf mit einer Fehlermeldung.

```
#include<conio.h>
#include<stdio.h>
void fehler()         // Funktionen ohne Rückgabewert sind vom Typ void
{
     printf("\nHier ist ein Fehler!");
     printf("\nTaste betaetigen... ");
     _getch();        //die return-Anweisung entfällt,Funktion endet hier
}
int main()
{
     int zahl;
     do
     {
     printf("\nGanze Zahl: ");
     scanf_s("%i",&zahl);
     if (zahl > 100) fehler();  //Fehlermeldung, wenn zahl > 100
     printf("\nEnde = 'ESC' ...");
     }
     while (_getch()!=0x1b);
     return(0);
}
```

Die hier verwendete Funktion benötigt weder Übergabeparameter noch einen Rückgabewert. Funktionen ohne Rückgabewert sind vom Typ **void**. Die return-Anweisung im Funktionsrumpf entfällt. Die Typangabe **void** darf nicht entfallen. Wenn man beispielsweise das Hauptprogramm mit

```
void main()
{
…;
}
```

deklariert, kann die return-Anweisung im Hauptprogramm entfallen.

Das folgende Beispiel gestattet im Hauptprogramm die Eingabe von Durchmesser und Höhe eines Zylinders und gibt das Volumen des Zylinders aus. Die eigentliche Berechnung des Volumens erfolgt in einer Funktion der Form

```
float zylvolumen(float d, float h); //Prototypangabe
```

Der Rückgabewert der Funktion ist das Volumen. Die beiden Übergabewerte sind Durchmesser und Höhe des Zylinders. Achtung: Die Reihenfolge der Übergabewerte ist zu beachten!).

```
#include<conio.h>
#include<stdio.h>
float zylvolumen(float d, float h)
{
      const float pi=3.141593f;  //pi ist vom Typ float!
      float flaeche,volumen;          //Variable sind lokal
      flaeche=d*d*pi/4;
      volumen=flaeche*h;
      return(volumen);
}
int main()
{
      float durchmesser,hoehe;
      do
      {
      printf("\nDurchmesser: ");
      scanf_s("%f",&durchmesser);
      printf("\nHoehe: ");
      scanf_s("%f",&hoehe);
      printf("\nVolumen: %10.3f",zylvolumen(durchmesser,hoehe));
      printf("\nEnde = 'ESC' ...");
      }
      while (_getch()!=0x1b);
      return(0);
}
```

Erläuterungen:

In der Funktionsdeklaration sind d und h Formalparameter. Sie erhalten die Werte von **`durchmesser`** und **`hoehe`** des Hauptprogramms (Übergabe per Wert!).

Alle von der Funktion benötigten Variablen sind innerhalb des Funktionsrumpfes zu deklarieren. So ist z.B. die Zahl `pi` nur der Funktion bekannt.

Im Hauptprogramm erfolgt der Funktionsaufruf mit

```
zylvolumen(durchmesser, hoehe)
```

Die Reihenfolge der Aktualparameter **`durchmesser`** und **`hoehe`** müssen eingehalten werden.

Übung:

Ergänzen Sie dieses Programm um eine Funktion, welche die Oberfläche des Zylinders berechnet!

Funktionsprototyp:

```
float zyloberflaeche(float d, float h);
```

Das Hauptprogramm ist ebenfalls um diese Ausgabe zu ergänzen.

Fragen und Übungen:

1. Erläutern Sie die Begriffe Formal- und Aktualparameter!

2. Was bedeutet die Angabe Übergabe per Wert bei Funktionen?

3. Wie viele Übergabeparameter und Rückgabewerte hat eine Funktion maximal?

4. Welche Aufgabe hat der Typ void?

5. Wie unterscheiden sich folgende Deklarationen und welche Konsequenzen haben die unterschiedliche Deklarationen?

```
int main()
void main()
```

6. Was versteht man unter lokalen Variablen?

7. Was Versteht man unter einem Funktionsprototyp?

8. Welches Ergebnis wird nach folgendem Programm ausgegeben? Prüfen Sie nach!

```
#include<conio.h>
#include<stdio.h>
#include<math.h>
float rechne(float x , float y)
{
   float a=2;
   a=(pow(x,y)*a);
   printf("\n%f",a);
   return(pow(x,y));
}
int main()
{
   float a=3,x=4,y=2;
   printf("\n%f",(rechne(a,y)));
   _getch();
   return(0);
}
```

9. Entwickeln Sie eine Funktion, welche den Widerstand eines runden Kupferdrahtes errechnet. Übergabeparameter sind Durchmesser in mm und Länge des Drahtes in m. Der Rückgabewert der Funktion ist der Widerstand in Ohm. Erstellen Sie außerdem ein passendes Hauptprogramm zum Testen dieser Funktion!

10. Erstellen Sie eine Funktion zur Berechnung der Fakultät einer ganzen Zahl im Bereich von 0 bis 10. Ubergabeparameter und Rückgabewert sind ganze Zahlen (Typ **long**). Rückgabewert ist die Fakultät der Zahl. Wenn der Übergabeparameter kleiner 0 oder größer 10 ist, soll die Funktion eine Fehlermeldung ausgeben und den Wert 0 zurückliefern.

    Funktionsprototyp:

    ```
    long fakultaet(long x);
    ```

    (Beachten Sie: 0!=1)

Parameterübergabe per Referenz

Bei den bisher behandelten Beispielen mit Funktionen erfolgte die Parameterübergabe per Wert. Das heißt: Die Funktion arbeitet mit einer **Kopie des Aktualparameters** aus dem Hauptprogramm. Alle deklarierten Objekte sind **lokal**.

Das folgende Beispiel demonstriert dies noch einmal:

```
#include<conio.h>
#include<stdio.h>
void quadriere(int zahl)              //Übergabeparameter ist eine ganze Zahl
{
     zahl=zahl*zahl;
     printf("\nQuadrat: %i",zahl);   //Wert von zahl innerhalb der Funktion
}
int main()
{
     int zahl=0;
     do
     {
     printf("\nGanze Zahl: ");
     scanf_s("%i",&zahl);
     printf("\nEingegebene Zahl: %i",zahl);
     quadriere(zahl);           //Aktualparameter ist der Integerwert zahl
     printf("\nWert von Zahl: %i",zahl);   //Wert von Zahl im Hauptprogramm
     printf("\nEnde = 'ESC' ...");
     }
     while (_getch()!=0x1b);
     return(0);
}
```

Im Hauptprogramm wird der Wert für **`zahl`** eingegeben. Bei Aufruf der Funktion **`quadriere(zahl)`** wird dieser in den Formalparameter **`zahl`** (der Funktion) kopiert. Innerhalb der Funktion wird der Formalparameter quadriert und ausgegeben. Nach Rückkehr zum Hauptprogramm ist der Aktualparameter **`zahl`** (im Hauptprogramm) unverändert.

Die Ausgabe sieht folgendermaßen aus:

```
Ganze Zahl: 5

Eingegebene Zahl: 5
Quadrat: 25
Wert von Zahl: 5
Ende = 'ESC' ...
```

Das Objekt **`zahl`** ist im Hauptprogramm ein anderes als in der Funktion. Es steht in unterschiedlichen Speicherbereichen.

An der Funktion des Programms ändert sich nichts, wenn der Formalparameter umbenannt wird.

```
void quadriere(int a)           //Formalparameter ist der Integerwert a
{
     a=a*a;
     printf("\nQuadrat: %i",a); //Wert von a innerhalb der Funktion
}
```

Merke:

Bei Funktionen mit Übergabe per Wert kann das Objekt im Hauptprogramm nicht beeinflusst werden!

Gelegentlich ist es erwünscht, dass die Funktion die Objekte des Hauptprogramms verändert. Grundsätzlich ist dazu erforderlich, dass der Funktion die **Speicherplatzadresse des Objektes** im Hauptprogramm bekannt ist.

Die **Adresse eines Objektes** nennt man auch die **Referenz** des Objektes. Referenzen werden durch ein vorgestelltes kaufmännisches ‚&' dargestellt. Bei der Funktionsdeklaration muss lediglich vor dem Bezeichner des Übergabeparameters ‚&' vorangestellt werden. Diesen Mechanismus bezeichnet man als **Übergabe per Referenz**.

Das vorige Beispiel sähe folgendermaßen aus:

```
#include<conio.h>
#include<stdio.h>
void quadriere(int &zahl)       // Übergabeparameter ist die Referenz einer
ganzen Zahl
                           // Es ist das gleiche Objekt wie im Hauptprogramm
{
     zahl=zahl*zahl;
     printf("\nQuadrat: %i",zahl);   //Wert von zahl
}
int main()
{
     int zahl;
     do
     {
     printf("\nGanze Zahl: ");
     scanf_s("%i",&zahl);
     printf("\nEingegebene Zahl: %i",zahl);
     quadriere(zahl);               //Aktualparameter ist der Integerwert
zahl
     printf("\nWert von Zahl: %i",zahl);   //Wert von zahl
     printf("\nEnde = 'ESC' ...");
     }
     while (_getch()!=0x1b);
     return(0);
}
```

Das Programm liefert folgende Ausgabe:

```
Ganze Zahl: 5

Eingegebene Zahl: 5
Quadrat: 25
Wert von Zahl: 25
Ende = 'ESC' ...
```

Der Bezeichner für den Formalparameter ist frei wählbar. So liefert die folgende Funktionsdeklaration das gleiche Ergebnis:

```
void quadriere(int &a)  // Übergabeparameter ist die Referenz einer ganzen Zahl
                        // Es ist das gleiche Objekt wie im Hauptprogramm
{
a=a*a;
printf("\nQuadrat: %i",a); //a Ist das gleiche Objekt wie zahl im Hauptprogramm
}
```

Im folgenden Beispiel soll eine Funktion **`ieingabe()`** sicherstellen, dass nur ganze Zahlen zwischen einer unteren und oberen Grenze eingegeben werden können. Bei einer anderen Eingabe erfolgt eine Fehlermeldung. Die Funktion überprüft zuerst, ob nur Zahlenwerte von 0 bis 9 eingegeben wurden. Anschließend wird der eingegebene String in Integer gewandelt (**`atoi()`**-Funktion). Als letztes werden dann die zulässigen Grenzen überprüft.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>      //Für Funktion atoi()
/****************** Funktionsdeklaration ************
Übergabeparamter: Referenz der einzugebenden Zahl
                  zulässiger unterer Wert und oberer Wert der Zahl
*/
void ieingabe(int &a,int ug,int og)
{
      char z[10];
      unsigned int m,i;
      do
      {
            m=0;
            printf("\nGanze Zahl: ");gets_s(z);//String eingeben
            for (i=0;i<strlen(z);i++)          //Prüfen ob nur Zahlen von 0...9
            {
                  if( (z[i]<0x30) || (z[i]>0x39) ) m=1;
            }
            a=atoi(z);                         //String in Int-Wert wandeln
            if (m==1 || a<ug || a>og)          //Bereichsprüfung
            {
                  printf("\nFalsche Eingabe...");_getch();
                  m=1;
            }
```

```
      }
      while(m==1);
}
int main()
{
      int zahl;
      do
      {
      ieingabe(zahl,0,999);               //nur ganze Zahlen von 0 bis 999 zulassen
      printf("\nWert von Zahl: %i",zahl); //Wert von zahl
      printf("\nEnde = 'ESC' ...");
      }
      while (_getch()!=0x1b);
      return(0);
}
```

Übung:

Entwerfen Sie eine ähnliche Funktion **`feingabe(…)`**, welche die sichere Eingabe eines positiven double-Wertes zwischen unterer und oberer Grenze gestattet. Die eingegebene Zahl darf nur Zahlen von 0 bis 9 und **einmal** den Dezimalpunkt enthalten.

Siehe auch: **`atof()`**-Funktion

Parameterübergabe per Zeiger

Zeigertypen können wie alle anderen Typen auch Übergabeparameter und Rückgabetyp einer Funktion sein.

Da Zeiger im Grunde Adressen sind, kann das Objekt, welches per Zeiger übergeben wird, von der Funktion verändert werden.

Beispiel:

```
#include <stdio.h>
#include <conio.h>

/********** Funktionsdeklaration ***************/

void quadriere(int* x)  //Der Übergabeparamter x ist ein Zeiger auf einen int-Wert
{
      *x = *x * *x;     //Quadriere den Inhalt des Wertes an der Adresse x(*x)
}

int main()
{
      int zahl;
      int* pzahl;            //pzahl ist ein Zeiger auf einen int-Wert
      pzahl=&zahl;           //Zeiger initialisieren
      do
      {
            printf("\nZahl: ");
            scanf_s("%i",pzahl);    //Eingabe einer int-Zahl
            quadriere(pzahl);       //Übergabeparameter ist ein Zeiger
            printf("\nQuadrat: %i",*pzahl); //int-Wert des Hauptprogramms ist verändert
            printf("\n'ESC' = Ende... ");
      }
      while(_getch()!=0x1b);
      return(0);
}
```

Der Aktualparamter **`pzahl`** und der Formalparameter **`x`** verweisen auf dasselbe Objekt **`zahl`** des Hauptprogramms!

Übung:

Welches Ergebnis liefert das folgende Programm, wenn für zahl1 der Wert 3 und für zahl2 der Wert 5 eingegeben wird?

```
#include <stdio.h>
#include <conio.h>

int rechne(int* x, int y) //Der Übergabeparamter x ist ein Zeiger auf einen int-Wert
{
      *x = *x * y;
      return (2**x);
}
int main()
{
      int zahl1;
      int zahl2;
      int erg;
      int* pzahl1;
      pzahl1=&zahl1;
      printf("\nZahl1: ");
      scanf_s("%i",&zahl1);
      printf("\nZahl2: ");
      scanf_s("%i",&zahl2);
      erg=rechne(pzahl1,zahl2);
      printf("\nZahl1: %i",zahl1);
      printf("\nErgebnis: %i",erg);
      _getch();
      return(0);
}
```

Ein wesentlicher Vorteil der Parameterübergabe per Zeiger ist, dass beliebig große Objekte (zum Beispiel Datenfelder) an eine Funktion übergeben werden können. Dazu ist lediglich erforderlich, die Startadresse des Objektes zu übergeben; das ist bei Datenfeldern der Bezeichner des Datenfeldes.

Im folgenden Beispiel wird im Hauptprogramm ein Datenfeld angelegt und initialisiert. Die Anzeige der Werte erfolgt in einer Funktion.

```
#include<conio.h>
#include<stdio.h>
void zeige(int* wert,int anzahl)   //Übergabeparameter sind die Adresse des
                                   //Datenfeldes(wert) und die Anzahl der Werte
{
      int j;
      for(j=0;j<anzahl;j++)
      {
            printf("\n%2i.Zahl: %i",j+1,wert[j]);
      }
}
```

```
int main()
{
      int zahl[5]={3,7,5,2,10};    //Datenfeld deklarieren und initialisieren
      zeige(zahl,5);
      _getch();
      return (0);
}
```

Als nächstes soll ein etwas umfangreicheres Beispiel betrachtet werden. Es ist ein Programm zur statistischen Auswertung eines Datenfeldes zu entwickeln. Das Hauptprogramm stellt ein Auswahlmenü zur Verfügung, welches folgende Funktionen gestattet:

```
        ***** Statistik *****

Daten eingeben  = '1'
Daten anzeigen  = '2'
Minimalwert     = '3'
Maximalwert     = '4'
Mittelwert      = '5'
Daten sortieren = '6'
Ende = '0'

Bitte waehlen
```

Das Programm ist modular zu gestalten; d.h.: Nach Eingabe eines Menüpunktes soll eine entsprechende Funktion aufgerufen werden und anschließend wieder zum Hauptmenü zurückgekehrt werden. Die Daten befinden sich als Integer-Werte in einem Datenfeld, welches im Hauptprogramm deklariert wird (max. 100 Werte).

Hauptprogramm und die Funktionen **`eingeben()`** sowie **`anzeigen()`** sind vorgegeben (siehe nächste Seite!).

Aufgabe:

Analysieren Sie das Programm und ergänzen Sie die fehlenden Funktionen!

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
/********* Funktionsprototypen ****************/
int eingeben(int*,int);         //Übergabe:Zeiger auf Datenfeld,Anzahl der Werte;
                                //Rückgabe:Anzahl der Werte
void anzeigen(int*,int);        //Übergabe:Zeiger auf Datenfeld,Anzahl der Werte
int minimalwert(int*,int);      //Übergabe:Zeiger auf Datenfeld,Anzahl der
Werte;Rückgabe:Minimalwert
int maximalwert(int*,int);      //Übergabe:Zeiger auf Datenfeld,Anzahl der
Werte;Rückgabe:Maximalwert
float mittelwert(int*,int);     //Übergabe:Zeiger auf Datenfeld,Anzahl der
Werte;Rückgabe:Mittelwert
void sortieren(int*,int);       //Übergabe:Zeiger auf Datenfeld,Anzahl der Werte
```

```
/******** Hauptprogramm *************/
int main()
{
int wert[100], i=0, wahl;
      do
      {
      system("cls");
      printf("\t***** Statistik *****");
      printf("\n\nDaten eingeben  = '1'");
      printf("\nDaten anzeigen  = '2'");
      printf("\nMinimalwert     = '3'");
      printf("\nMaximalwert     = '4'");
      printf("\nMittelwert      = '5'");
      printf("\nDaten sortieren = '6'");
      printf("\n\nEnde = '0'");
      printf("\n\nBitte waehlen... ");
      wahl = _getch();
            switch(wahl)
            {
                  case '1': i=eingeben(wert,i);break;
                  case '2': anzeigen(wert,i);break;
                  //case '3': printf("\nMinimalwert: %i\n",minimalwert(wert,i));getch();break;
                  //case '5': printf("\nMaximalwert: %i\n",maximalwert(wert,i));getch();break;
                  //case '3': printf("\nMittelwert: %i\n",mittelwert(wert,i));getch();break;
                  //case '6': sortieren(wert,i);break;
                  case '0': break;
                  default: printf("\nFalsche Eingabe... ");_getch();
            }
      }
while(wahl != '0');
return(0);
}
/****** Die Funktion gestattet die Eingabe von int-Werten in ein Datenfeld***
Uebergabeparamter: Zeiger auf Datenfeld ; Anzahl der vorhandenen Werte
Rueckgabewert: Anzahl aller Werte *************/
int eingeben(int *z, int i)
{
      do
      {
            printf("\n%i.Wert: ",i+1);
            scanf_s("%i",z+i); i++;
            printf("\n'ESC' = Beenden der Eingabe...");
      }
      while (_getch()!=0x1b);
return(i);
}
/********* Funktion fuer die formatierte Anzeige eines Datenfeldes ******
Uebergabeparameter: Zeiger auf Datenfeld ; Anzahl der Werte *********/
void anzeigen(int *z, int i)
{
      int j=0;
      while (j < i)
      {
            printf("\n%i. Wert: %i",j+1,z[j]);
            j++;
      }
      printf("\n\nBitte Taste betaetigen... ");_getch();
}
```

Strings als Parameter

Zeichenketten (Strings) sind in C Datenfelder vom Typ **`char`**. Strings sind eine Folge von ASCII-Zeichen, welche mit dem Zeichen 0x00 begrenzt sind. Die Übergabe von Strings an Funktionen erfolgt deshalb immer per Zeiger.

Das folgende Beispiel übergibt eine String und ein einzelnes Zeichen an die Funktion **`cinstr()`**. Der Rückgabewert der Funktion gibt an, wie oft das Zeichen im String vorkommt.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int cinstr(char* t, char z)
{
     unsigned int i=0,j=0;
     for (i=0;i<strlen(t);i++)
     {
            if (t[i]==z) j++;
     }
     return(j);
}
int main()
{
     char text[100];
     char zeichen;
     do
     {
     printf("\nText: ");
     gets_s(text);
     printf("\nZeichen: ");
     zeichen=_getche(); //Zeichen wird auf Konsole angezeigt
     printf("\nDas Zeichen %c kommt im Text %i malvor!",zeichen,cinstr(text,zeichen));
     printf("\n'ESC' = Ende ...");
     }
     while(_getch()!=0x1b);
     return(0);
}
```

So sieht's aus:

```
Text: Dies ist ein Text

Zeichen: e
Das Zeichen e kommt im Text 3 mal vor!
'ESC' = Ende ...
```

Der Rückgabewert einer Funktion kann auch ein Zeiger sein. Im folgenden Beispiel wird der Funktion leftstr() ein String und eine ganze Zahl übergeben. Die Funktion verkürzt den String von links beginnend auf die Anzahl der übergebenen Zahl. Der String wird verkürzt. In dieser Funktion ist der Rückgabewert (Zeiger auf String) identisch mit dem übergebenen Zeiger.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
char* leftstr(char* t, int i)
{
      t[i]=0;
      return(t);
}
int main()
{
      char text[100];
      int anzahl;
      do
      {
      printf("\nText: ");
      _flushall();      //löscht zur Sicherheit den Eingabebuffer
      gets_s(text);
      printf("\nAnzahl der Zeichen: ");
      scanf_s("%i",&anzahl);
      printf("\nVerkuerzter Text: %s",leftstr(text,anzahl));
      printf("\n'ESC' = Ende ...");
      }
      while(_getch()!=0x1b);
      return(0);
}
```

So sieht's aus:

```
Text: Donauschifffahrt

Anzahl der Zeichen: 5

Verkuerzter Text: Donau
'ESC' = Ende ...
```

1. Übung:

Es ist eine Funktion zu erstellen, welche einen Teilstring aus einem übergebenen String ausschneidet. Übergabeparameter sind ein String und zwei ganze Zahlen. Die erste Zahl gibt an, ab welcher Stelle der String ausgeschnitten wird und die zweite Zahl bestimmt die Anzahl der Stellen.

Vorgegeben sind Funktionsprototyp und Hauptprogramm:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
char* midstr(char* t, int i,int j);
int main()
{
      char text[100];
      int position,anzahl;
      do
      {
      printf("\nText: ");
      gets_s(text);
      printf("\nAusschneiden ab Position: ");
      scanf_s("%i",&position);
      printf("\nAnzahl der Zeichen: ");
      scanf_s("%i",&anzahl);
      printf("\nAusgeschnittener String: %s",midstr(text,position,anzahl));
      printf("\n'ESC' = Ende ...");
      }
      while(_getch()!=0x1b);
      return(0);
}
```

So sollte es aussehen:

```
Text: Donaudampfschifffahrt
Ausschneiden ab Position: 5
Anzahl der Zeichen: 5
Ausgeschnittener String: dampf
'ESC' = Ende ...
```

2. Übung:

Gegeben ist der Prototyp einer Funktion:

```
int strsuch(char* t1, char* t2);
```

Der Funktion werden zwei Strings übergeben. Es soll festgestellt werden, ob der zweite String im ersten String enthalten ist. Der Rückgabewert der Funktion ist die Position, ab welcher der zweite String im ersten String vorhanden ist. Ist der String nicht enthalten soll der Rückgabewert –1 sein. Das Hauptprogramm soll eine entsprechende Meldung ausgeben.

So sollte es aussehen:

```
Text1: Donaudampfschifffahrt
Text2: dampf
Text2 ist in Text1 ab Position 5 enthalten
'ESC' = Ende ...
```

Überladen von Funktionen

Es besteht die Möglichkeit, mehrere Funktionen mit gleichem Namen anzulegen. Die einzelnen Funktionen unterscheiden sich durch die Anzahl oder den Typ der Übergabe- und/oder Rückgabewerte.

Im folgenden Beispiel werden zwei Funktionen mit dem Namen rechne(...) angelegt. Der ersten Funktion werden zwei int-Werte und der zweiten Funktion zwei double-Werte übergeben. Die erste Funktion addiert und die zweite multipliziert zwei Zahlen. Das Hauptprogramm erkennt an Hand der Typen für die Übergabeparameter, welche Funktion verwendet werden soll.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int rechne(int a, int b)
{
     return (a+b);
}

double rechne(double a, double b)
{
     return (a*b);
}
int main()
{
     int a=2,b=4;
     double c=2.5,d=3.0;
     printf("Summe: %i\n",rechne(a,b));
     printf("Produkt: %lf\n",rechne(c,d));
     _getch();
     return(0);
}
```

Ausgabe:

```
Summe: 6
Produkt: 7.500000
```