Funktionen

Funktionen erlauben,

- dem Programmcode hierarchisch zu strukturieren –
  ein Hauptprogramm steuert dabei die Abfolge von Schritten, die einzelnen Schritte können durch Funktionen realisiert werden
- mehrfach benötigte Programmteile nur einmal zu schreiben und mehrfach aufzurufen
- Programmteile mehrmals leicht variiert auszuführen, gesteuert durch Parameter
- Rekursive Algorithmen, indem eine Funktion sich selbst mit abgeänderten Parametern aufruft
- Die Benutzung vorgefertigter Funktionen einer Standardbibliothek, z.B. für Ein- und Ausgabe, für Zeichenkettenverarbeitung usw.
- Zusammenstellung und Sammeln von Hilfs-Funktionen, die ein Programmierer persönlich oft benutzt

Funktionsdefinition (1)

Allgemeine Form:

Rückgabetyp Funktionsname (Parameterliste)

```
{
    Block mit
    Deklarationen und
    Anweisungen

    return  ausdruck ;
}
```

Funktionsdefinition (2)

Eine Funktion wird wie folgt definiert:

```
Rückgabetyp  Funktionsname ( Parameterliste )
{
  …
  return …;
}
```

Durch den *Funktionsnamen* wird die Funktion
an anderer Stelle aufgerufen.

Die *Parameterliste* enthält eine durch Komma getrennte Folge von Parametern, jeweils mit Typangabe und Variablenname.

Eine Funktion kann einen Wert mit dem angegebenen Rückgabetyp zurückgeben. Dazu muss in der Funktion *return … ;* ausgeführt werden.

C-Programme als Folgen von Funktionen

Ein C-Programm stellt eine Folge von Funktionsdefinitionen dar, wobei eine Funktion als Hauptfunktion (main) gekennzeichnet sein muss.

In den Funktionen können sich Aufrufe anderer Funktionen befinden.

Funktionsdefinitionen können in C nur global sein und dürfen nicht im Block einer anderen Funktion bzw. der Hauptfunktion definiert werden.

Funktionen in C (1)

Beispiel 1: Funktion mit float-Rückkehrwert (return!)

```
float max(float a, float b)
{   if (a>b) return a;
    else return b;
}
```

Beispiel 2: Funktion ohne Rückkehrwert

```
void druck(float x)
{  printf("\n x=%f",x);
}
```

Beispiel 3: Hauptprogramm mit Funktions-Aufruf

```
void main( )
{  float c=4.0f, d=5.99f;
   druck(max(c,d));
}
```

Funktionen in C (2)

Beispiel mit Ausgabeparameter in der Parameterliste:

```
void fakultaet(int x, double *w)
{
   int i;
  *w=1.0;
   for (i=2; i<=x; i++)
     *w = *w * (double)i;
}
```

Aufruf der Funktion:

```
int i; double y;
for (i=0;i<40;i++)
{   fakultaet(i,&y);
     printf("i=%03d, fak(i) =%.0lf \n",i,y);
}
```

Funktionen in C (3)

Benennung einer Funktion:

Funktionsnamen unterliegen den gleichen Einschränkungen wie Variablennamen. Beispielsweise darf keine Ziffer am Beginn eines Funktionsnamen stehen. Groß- und Kleinschreibung wird unterschieden, d.h. *Myfun()* und *myfun()* sind zwei unterschiedliche Funktionen.

Funktionen in C (4)

Parameter:

Eine durch Komma getrennte Liste einzelner Typbezeichner und Variablenbezeichner beschreiben die Parameter. Die als Parameter angegeben Variablen sind innerhalb des Prozedurkörpers gültig und überlagern gleichnamige globale Variablen.

Die Parameter werden bei Aufruf der Funktion auf diese Variablen kopiert (Call by Value).

Parameter, die als Ausgabe einer Funktion agieren, sind als Zeiger zu übergeben (Call by Reference). Dann wird der Zeiger kopiert, die Änderung des Werts erfolgt auf dem Originalspeicherplatz.

Beim Funktionsaufruf werden für die Parameter s.g. **Argumente** eingetragen.

Funktionen in C (4)

Ein- und Ausgabe:

- Funktionen nehmen Ihre Eingabe über die Parameter entgegen.
- Die Rückgabe erfolgt über return-Werte, oder über spezielle Parameter.
- deshalb wird normalerweise kein *scanf()* und kein *printf()* benötigt.
- Ausnahmen:
  - Funktionen, die speziell der Benutzerinteraktion dienen
  - Ausgabe von Fehlernachrichten

Gute Programmstruktur:
Trennung zwischen Funktionalität und Nutzerinteraktion

Funktionen in C (5)

Bei einer C-Funktionsdefinition stellen die Parameter quasi Platzhalter dar. Für diese Parameter werden dann beim Funktionsaufruf Argumente eingesetzt. Die Anzahl, die Reihenfolge und der Typ von Argumenten beim Funktionsaufruf muss immer mit Anzahl, Reihenfolge und Typ der Parameter bei der Funktionsdefinition übereinstimmen.

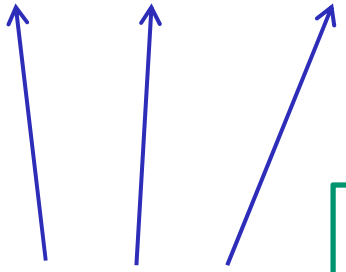
Definition:

```
double  berechneEtwas( int a, double x, double y)
{ … }
```

drei Call-by-Value Parameter

Aufruf:

```
ergebnis =  berechneEtwas( 5, wx, wy);
```



drei Argumente:
beim Aufruf wird der konstante Wert 5 an a,
und die Werte von wx und wy an x bzw. y übergeben

Funktionen in C (4)

Parameter:

Eine durch Komma getrennte Liste einzelner Typbezeichner und Variablenbezeichner beschreiben die Parameter. Die als Parameter angegeben Variablen sind innerhalb des Prozedurkörpers gültig und überlagern gleichnamige globale Variablen.

Die Parameter werden bei Aufruf der Funktion auf diese Variablen kopiert (Call by Value).

Parameter, die als Ausgabe einer Funktion agieren, sind als Zeiger zu übergeben (Call by Reference). Dann wird der Zeiger kopiert, die Änderung des Werts erfolgt auf dem Originalspeicherplatz.

Beim Funktionsaufruf werden für die Parameter s.g. **Argumente** eingetragen.

Funktionen in C (6)

Call by Reference

Definition:

*void tauscheWerte(double *x, double *y)*
{ … }

zwei Call-by-Reference Parameter

Aufruf:

TauscheWerte(&a, &b);

Zwei Argumente:
beim Aufruf wird die Adresse von a an x,
und die Adresse von b an y übergeben

Funktionen in C (7)

Vorgriff auf Felder und Zeichenketten:

Felder und Zeichenketten werden als Zeiger übergeben.
Dabei wird ausgenutzt, dass der Feldbezeichner ein Zeiger auf das erste Element ist.

```
void ausgabe_zeile(char *postitionsstring, int x, int y, float p)
{
   printf("Luftdruck an Position %s (%d,%d): %f \n",
            postionsstring, x, y, p);
}

...
ausgabe_zeile("Zugspitze", 51, 13, 998.17);
```

Funktionen in C (7)

Rückgabewert:
Funktionen können einen Wert zurückliefern.
Das erlaubt

Benutzung in Zuweisungen ...
wert = myfun(a,b); // myfun liefert einen Wert zurück

Benutzung einer Funktion als Parameter
einer anderen Funktion …
```
printf("Der Funktionswert an der Stelle %f ist %f \n",
        x, fun(x) );
```

Wird die Angabe des Rückgabetyps weggelassen, nimmt der Compiler standardmäßig integer an.

Funktionen, die keinen Wert zurückgeben sollen, werden mit dem Rückgabetyp *void* gekennzeichnet.

Funktion – Typ des Rückkehrwertes

Bei einer C-Funktionsdefinition muss vor dem Funktionsnamen ein Typ (des Rückkehrwertes) angegeben werden. Dieser Typ kann ein in C bekannter Standardtyp, wie z.B. int, unsigned long, char , double, ... sein.

Es ist darauf zu achten, dass der Typ des Ausdrucks in der return-Anweisung mit dem Typ des Rückkehrwertes kompatibel ist.

Rekursive Funktionen (1)

In C/C++ können Funktionen rekursiv definiert werden. Das kann in direkter oder indirekter Form geschehen. Bei einer direkten Rekursion enthält die Funktionsdefinition einen Aufruf von sich selbst, während im indirekten Fall eine andere Funktion aufgerufen wird, die wiederum die zu definierende Funktion ruft.

Da die Verwaltung aller lokalen Größen ohnehin durch den Compiler in einem Runtime-Stack vorgenommen wird, muss der Programmierer bei der Definition rekursiver Funktionen nichts besonderes beachten.

Allerdings müssen stets Anweisungen vorhanden sein, die den rekursiven Aufruf begrenzen, um ein „Endlos-Aufrufen“ zu verhindern.

Rekursive Funktionen

Beispiel FIBONACCI-Zahlen:

fibo(n)= fibo(n-1) + fibo(n-2) Rekursionsabbruch fibo(1)=1 fibo(0)=1

```
int fibo(int n)  //rekursive Definition
{ if (n<2) return 1;else return (fibo(n-1)+fibo(n-2)); }

void main()
{ int i=1,f=1;
  printf("\nBerechnung der FIBONACCI-Zahlen im Intervall [0,100]");
  printf("\n\n x   fibo(x)\n-----------");
  while ((f=fibo(i))<=100)
  { printf("\n%2d     %3d",i,f); i++; }
}
```

Funktionen – vorläufige Zusammenfassung

- Eine Funktion besteht aus Rückgabetyp, Funktionsname, Parameterliste und einem Anweisungsblock, der in geschweifte Klammern “{“, “}“ eingefasst wird.
- Rückgabetyp kann jeder gültige Typ sein, auch selbstdefinierte Typen, Strukturen oder Zeiger.
- Die Anweisung *return* dient zur Rückgabe eines Wertes aus der Funktion. Die Ausführung wird dann mit der Anweisung fortgesetzt, die dem Funktionsaufruf folgt.
- Funktionen, die keinen Wert zurückgeben, erhalten den Rückgabetyp *void*.
- Funktionen können sich selbst aufrufen: Rekursion