8 Unterprogramme und Funktionen

Damit für gleichartige, häufiger sich wiederholende Programmabschnitte die Schreibarbeit beim Programmieren sowie die Programmlänge reduziert werden, wurden Unterprogramme und Funktionen als Sprachkonstrukte eingeführt.

Ein bei größeren Programmen nicht zu unterschätzender Effekt von Unterprogrammen und Funktionen ist die Wiederverwendbarkeit einmal aufgeschriebener Algorithmen in anderen Programmen und besonders der Einsatz von Unterprogrammen zur Strukturierung des Programms. Diese Strukturierung kann zu einem hierarchischen Aufbau führen, so daß einzelne Unterprogramme, von einem übergeordneten Programm aufgerufen, Teilaufgaben vollständig bearbeiten und die Ergebnisse abliefern.

8.1 Vereinbarung

Ein Unterprogramm oder eine Funktion ist ein separater Programmteil mit Programmkopf, Vereinbarungsteil und Anweisungsteil, der von beliebigen Stellen im Hauptprogramm aufgerufen werden kann. Nach Abarbeitung des Unterprogramms oder der Funktion erfolgt ein Rücksprung an den nächsten Befehl nach dem Aufruf des Unterprogramms (s. Abb. 34).

Von einem Unterprogramm bzw. einer Funktion aus können weitere Unterprogramme und/oder Funktionen aufgerufen werden. Die hierbei zulässige Schachtelungstiefe ist 20. Darüber hinaus erfolgt die Fehlermeldung “ÜBERLAUF PROGRAMMSCHACHTELUNG”. Der rekursive Aufruf von Unterprogrammen oder Funktionen ist erlaubt. Das heißt, ein Unterprogramm oder eine Funktion kann sich selbst wieder aufrufen.

DEF

Alle Unterprogramme werden genau wie die Hauptprogramme mit der DEF-Vereinbarung plus Namen deklariert und mit END abgeschlossen, z.B.:

```
DEF UNTERPR()
...
END
```

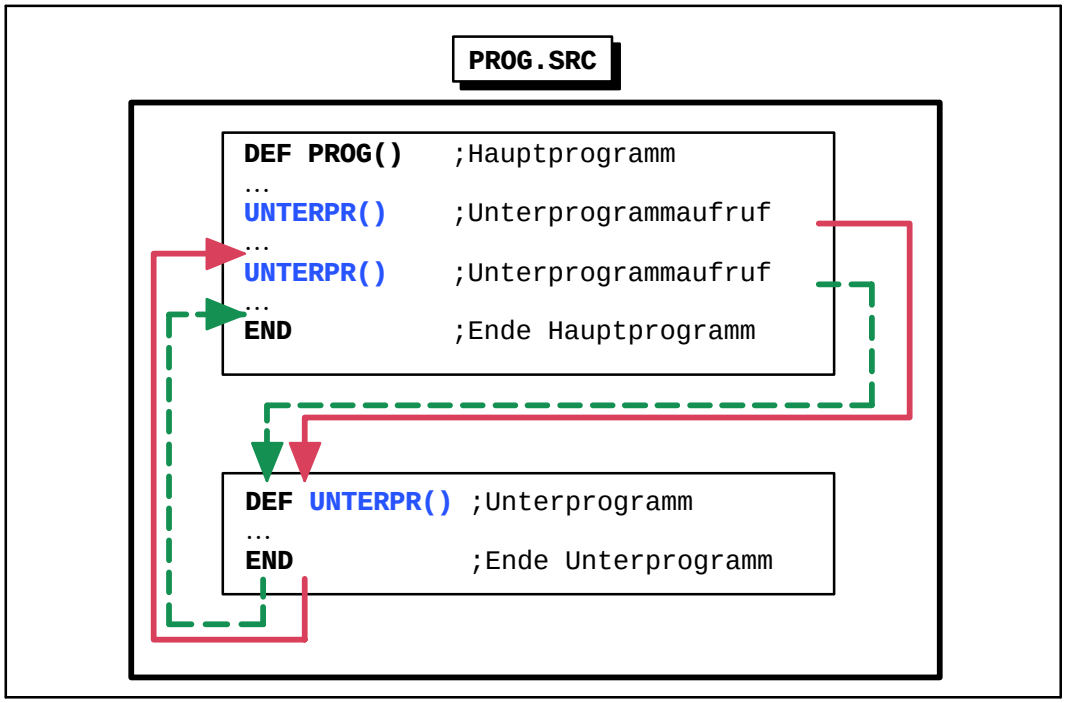

Abb. 34 Unterprogrammaufruf und Rücksprung

DEFFCT

Eine Funktion ist eine Art Unterprogramm, jedoch ist der Programmname gleichzeitig eine Variable eines bestimmten Datentyps. Damit läßt sich das Ergebnis der Funktion durch einfache Wertzuweisung an eine Variable übergeben. Bei der Vereinbarung von Funktionen mit dem speziellen Schlüsselwort DEFFCT muß daher neben dem Namen der Funktion auch der Datentyp der Funktion angegeben werden. Abgeschlossen wird eine Funktion mit ENDFCT. Da eine Funktion einen Wert übergeben soll, muß dieser Wert vor der ENDFCT-Anweisung mit der RETURN-Anweisung spezifiziert werden. Beispiel:

```
DEFFCT INT FUNKTION()
...
RETURN(X)
ENDFCT
```

lokal

Grundsätzlich unterscheidet man zwischen lokalen und globalen Unterprogrammen bzw. Funktionen. Bei lokalen Unterprogrammen oder Funktionen befinden sich das Hauptprogramm und die Unterprogramme/Funktionen in der selben SRC-Datei. Die Datei trägt den Namen des Hauptprogrammes. Dabei steht das Hauptprogramm im Quelltext immer an erster Stelle, während die Unterprogramme und Funktionen in beliebiger Reihenfolge und Anzahl nach dem Hauptprogramm folgen.

global

Lokale Unterprogramme/Funktionen können nur innerhalb des SRC-Files, indem sie programmiert wurden, aufgerufen werden. Sollen Unterprogramm-/Funktionsaufrufe auch von anderen Programmen möglich sein, so müssen sie global sein. Globale Unterprogramme oder Funktionen werden in einem eigenen SRC-File abgespeichert. Somit ist jedes Programm ein Unterprogramm, wenn es von einem anderen Programm (Hauptprogramm, Unterprogramm oder Funktion) aufgerufen wird.

- In lokalen Unterprogrammen bzw. Funktionen sind alle in der Datenliste des Hauptprogramms deklarierten Variablen bekannt. Variablen die im Hauptprogramm (SRC-File) deklariert wurden sind sogenannte “Runtime Variablen” und dürfen nur im Hauptprogramm verwendet werden. Ein Versuch diese Variablen im Unterprogramm zu verwenden führt zu einer entsprechenden Fehlermeldung.
  In globalen Unterprogrammen oder Funktionen sind die im Hauptprogramm deklarierten Variablen nicht bekannt.
- Im Hauptprogramm sind die in Unterprogrammen oder Funktionen deklarierten Variablen nicht bekannt.
- Ein Hauptprogramm kann nicht auf lokale Unterprogramme oder Funktionen eines anderen Hauptprogramms zugreifen.
- Der Name von lokalen Unterprogrammen/Funktionen kann maximal 12 Zeichen lang sein. Bei globalen Unterprogrammen/Funktionen darf er höchstens 8 Zeichen lang sein (wegen der Dateierweiterungen).

EXT
EXTFCT

Namen und Pfade aller aufzurufenden externen Unterprogramme und Funktionen sowie die verwendeten Parameter müssen dem Compiler mit der EXT- bzw. EXTFCT-Vereinbarung kenntlich gemacht werden. Mit der Angabe der Parameterliste (s. 8.2) ist auch der benötigte Speicherplatz eindeutig festgelegt. Beispiele:

```
EXT PROG_3()
EXTFCT FUNCTION(REAL:IN)
```

In Abb. 35 ist der Unterschied zwischen globalen und lokalen Unterprogrammen bzw. Funktionen dargestellt: LOCAL ist ein lokales Unterprogramm und LOCALFUN eine lokale Funktion des Programmes PROG, GLOBAL und PROG_2 sind globale Unterprogramme, GLOBFUN ist eine globale Funktion.

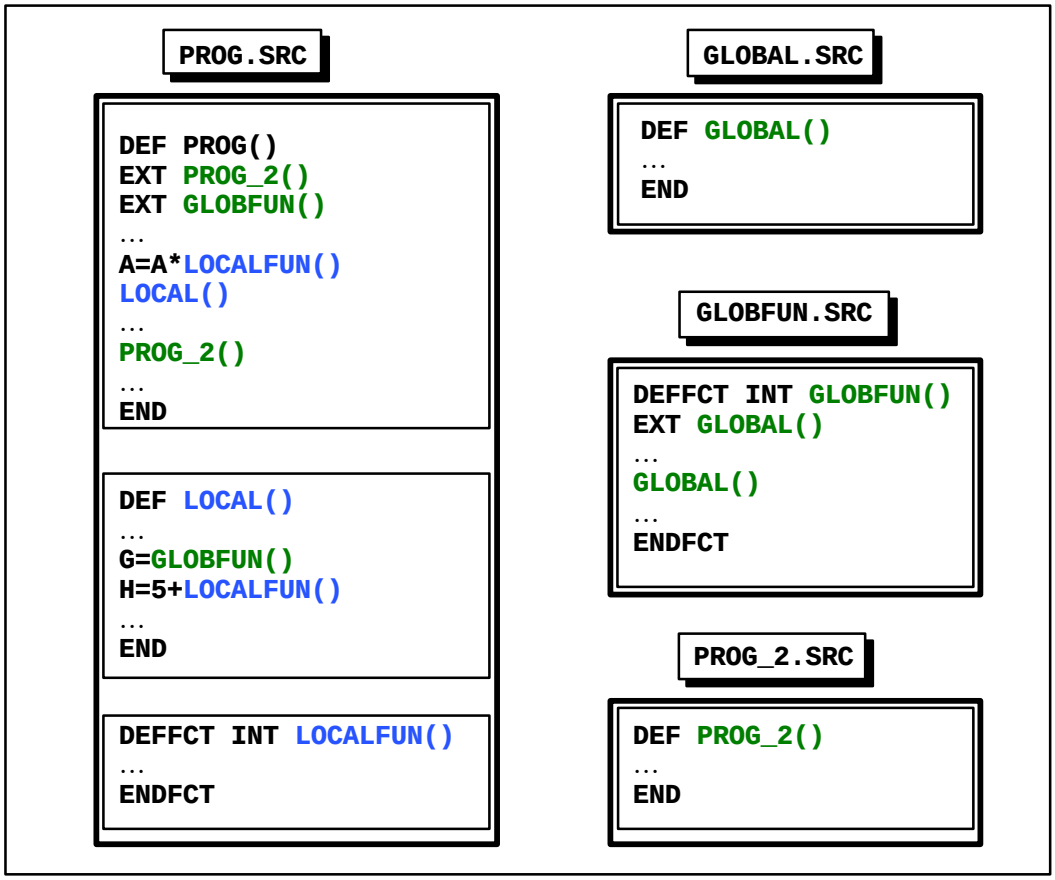

Abb. 35 Unterschied zwischen lokalen und globalen Unterprogrammen

8.2 Aufruf und Parameterübergabe

Der Aufruf eines Unterprogrammes erfolgt einfach durch die Angabe des Unterprogramm-Namens und runden Klammern. Er sieht somit aus wie eine Anweisung (s. Kap. 2.1), z.B.:

```
UNTERPROG1()
```

Ein Funktionsaufruf ist eine besondere Form einer Wertzuweisung. Eine Funktion kann daher nie alleine stehen, sondern der Funktionswert muß stets im Rahmen eines Ausdrucks einer Variablen vom gleichen Datentyp zugewiesen werden, z.B.:

```
INTVAR = 5 * INTFUNKTION() + 1
REALVAR = REALFUNKTION()
```

Parameterliste

In lokalen Unterprogrammen und Funktionen sind alle in der Datenliste des Hauptprogramms deklarierten Variablen bekannt. In globalen Unterprogrammen dagegen, sind diese Variablen nicht bekannt. Über eine Parameterliste können aber auch Werte an globale Unterprogramme und Funktionen übergeben werden.

Die Übergabe mit Parameterlisten ist oft auch in lokalen Unterprogrammen und Funktionen sinnvoll, da so eine klare Trennung zwischen Hauptprogramm und Unterprogramm/Funktion vorgenommen werden kann: Im Hauptprogramm (SRC-File) deklarierte Variablen werden nur dort verwendet, alle Übergaben in Unterprogramme und Funktionen (lokal und global) erfolgen mittels Parameterlisten. Durch diese strukturierte Programmierung reduzieren sich Programmierfehler deutlich.

Zur Parameterübergabe gibt es zwei unterschiedliche Mechanismen:

- **Call by value (`IN`)**
  Bei dieser Übergabeart wird ein **Wert** aus dem Hauptprogramm an eine Variable des Unterprogramms oder der Funktion übergeben. Der übergebene Wert kann eine Konstante, eine Variable, ein Funktionsaufruf oder ein Ausdruck sein. Bei unterschiedlichen Datentypen wird, wenn möglich, eine Typanpassung durchgeführt.

- **Call by reference (`OUT`)**
  Durch “Call by reference” wird nur die **Adresse** einer Variablen des Hauptprogramms an das Unterprogramm bzw. die Funktion übergeben. Das aufgerufene Unterprogramm bzw. die Funktion kann nun über einen eigenen Variablennamen den Speicherbereich überschreiben und somit auch den Wert der Variablen im Hauptprogramm verändern. Die Datentypen müssen daher identisch sein, eine Typanpassung ist in diesem Fall nicht möglich.

In Abb. 36 ist der Unterschied zwischen den beiden Methoden aufgezeigt. Während die Variable `X` bei “Call by value” im Hauptprogramm aufgrund der getrennten Speicherbereiche unverändert bleibt, wird sie bei “Call by reference” durch die Variable `ZAHL` in der Funktion überschrieben.

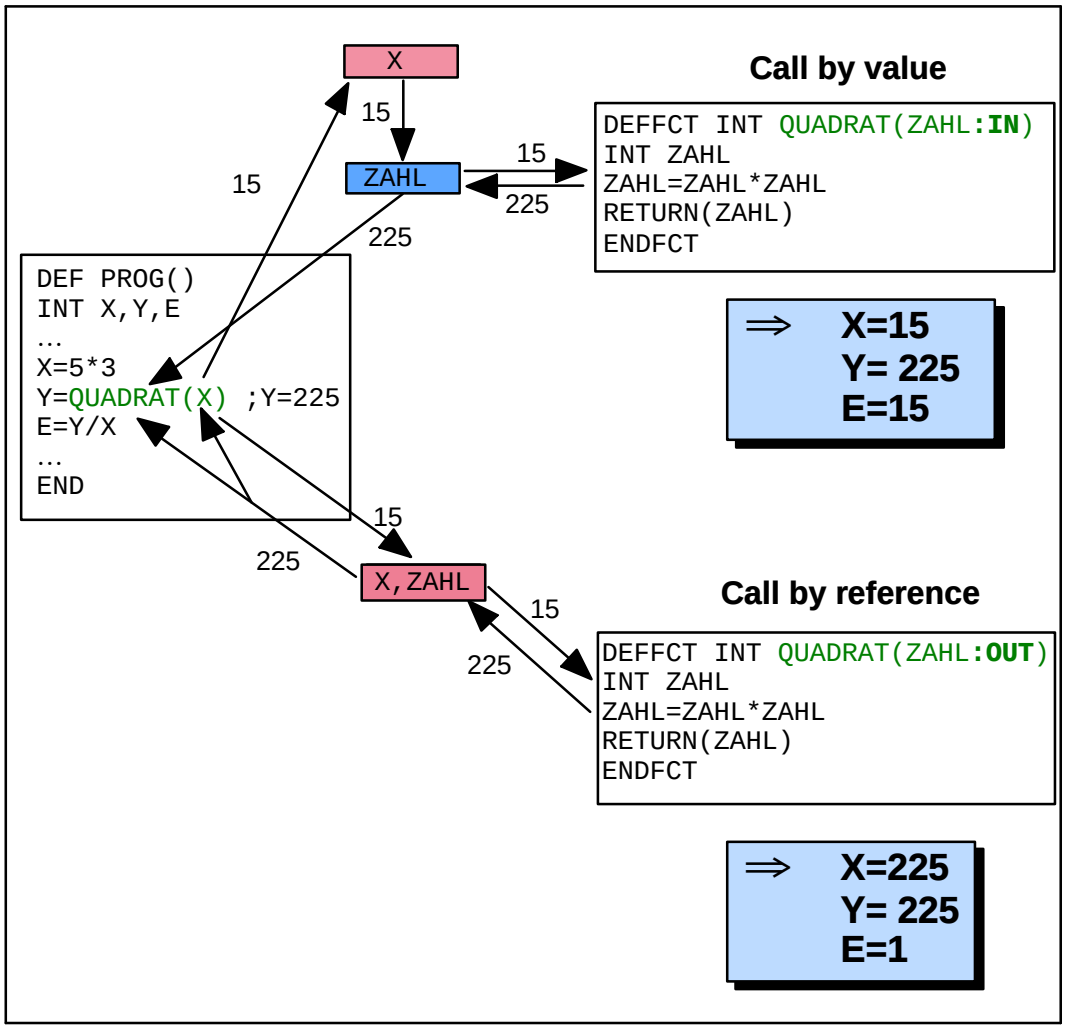

Abb. 36 Unterschied zwischen “Call by value” und “Call by reference”

“Call by value” wird im Unterprogramm- oder Funktionskopf durch das Schlüsselwort **IN** hinter jeder Variablen in der Parameterliste angegeben. “Call by reference” erhält man durch Angabe von **OUT**. OUT ist auch die Default-Einstellung. Beispiel:

```
DEF RECHNE(X:OUT,Y:IN,Z:IN,B)
```

Ist das aufzurufende Unterprogramm oder die Funktion global, so muß bei der Extern-Vereinbarung im Hauptprogramm angegeben werden, welchen Datentyp die jeweiligen Variablen haben und welcher Übergabemechanismus verwendet werden soll. OUT ist wieder die Default-Einstellung. Beispiel:

```
EXTFCT REAL FUNKT1(REAL:IN,BOOL:OUT,REAL,CHAR:IN)
```

Die Verwendung von IN und OUT soll mit folgendem Beispiel verdeutlicht werden. Das Unterprogramm und die Funktion seien global.


```
DEF PROG()
EXT RECHNE (INT:OUT,INT:IN,INT:IN)
EXTFCT REAL FUNKT1(REAL:IN,REAL:OUT,REAL:OUT,REAL:IN,REAL:OUT)
INT A,B,C
REAL D,E,F,G,H,X

A = 1
B = 2
C = 3
D = 1
E = 2
F = 3
G = 4
H = 5

RECHNE (A,B,C)
     ;A ist nun 11
     ;B ist nun 2
     ;C ist nun 3

X = FUNKT1(H,D,E,F,G)
     ;D ist nun 3
     ;E ist nun 8
     ;F ist nun 3
     ;G ist nun 24
     ;H ist nun 5
     ;X ist nun 15
END
DEF RECHNE(X1:OUT,X2:IN,X3:IN)                          ;globales UP
INT X1,X2,X3
X1=X1+10
X2=X2+10
X3=X3+10
END
DEFFCT REAL FUNKT1(X1:IN,X2:OUT,X3:OUT,X4:IN,X5:OUT);globale Fkt.
REAL X1,X2,X3,X4,X5
X1 = X1*2
X2 = X2*3
X3 = X3*4
X4 = X4*5
X5 = X5*6
RETURN(X4)
ENDFCT
```

Bei der Übergabe eines Feldes muß das Feld im Unterprogramm oder der Funktion ebenfalls neu deklariert werden, jedoch ohne Index. Dazu folgendes Beispiel, indem die Werte eines Feldes `X[]` verdoppelt werden (die Funktion ist global):

```
DEF  ARRAY ( )
EXT  BAS (BAS_COMMAND:IN,REAL:IN)
INT X[5]     ;feldvereinbarung
INT I
EXT  DOPPEL (INT[]:OUT)

BAS (#INITMOV,0)

FOR I=1 TO  5
   X[I]=I    ;array X[] initialisieren
ENDFOR       ;X[1]=1,X[2]=2,X[3]=3,X[4]=4,x[5]=5

DOPPEL (X[]) ;unterprogramm mit feldparameter aufrufen
             ;X[1]=2,X[2]=4,X[3]=6,X[4]=8,X[5]=10
END

DEF  DOPPEL (A[]:OUT)
INT A[]          ;erneute vereinbarung des feldes
INT I
FOR I=1 TO  5
   A[I]=2*A[I]  ;verdoppelung der feldwerte
ENDFOR
END
```

Bei der Übergabe von mehrdimensionalen Felder werden ebenfalls keine Indizes angegeben, allerdings muß die Dimension des Feldes durch Angabe von Kommas spezifiziert werden, Beispiele:

`A[,]` für zweidimensionale Felder

`A[,,]` für dreidimensionale Felder

NOTIZEN: