2.8 Funktionen/Dokumentation

2.8.1 Aufrufdiagramm

Ein Programm mit vielen Funktionen soll beschrieben werden. Im C-Quelltext liegen alle Funktionsdefinitionen nacheinander, so will es die Sprache. Im Lauf des Programms jedoch ruft oft eine Funktion eine andere auf. Vor allem in der Funktion `main()` findet man in der Regel sehr viele Aufrufe anderer Funktionen. Und diese Funktionen können wiederum andere Funktionen aufrufen. Eine Lösung bei diesem Gewirr liefert das so genannte Aufrufdiagramm der Funktionen. Jede Funktion wird durch ein Rechteck beschrieben. Ein Pfeil symbolisiert einen Aufruf. Dabei zeigt der Pfeil von der aufrufenden Funktion (z. B. `main()`) zur aufgerufenen Funktion. Abbildung 1

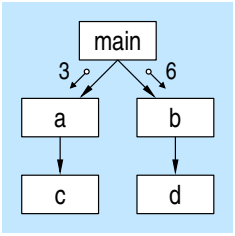

Abbildung 1: Aufrufdiagramm zu Codebeispiel 1

zeigt das Aufrufdiagramm zum folgenden kleinen Codebeispiel 1:

```c
#include <stdio.h>
void c(void) { printf(",");}
void d(void) { printf("1");}
void a(int x)
{
    printf("%i", x);
    c();
}
void b(int x)
{
    d();
    printf("%i\n", x);
}
int main(void)
{
    a(3);
    b(6);
    return 0;
}
```

Der Aufruf eingebauter Funktionen wie printf() oder scanf() braucht nicht eingezeichnet werden.

Die kleinen Pfeile mit dazugehörigen Zahlen bei den Aufrufen von a und b stellen die Parameter dar. Sie sind optional; man kann sie hinzeichnen, wenn das der Verständlichkeit der Darstellung dient. Ein kleiner Pfeil in umgekehrter Richtung stellt die Übergabe eines Rückgabewertes dar.

Ruft `main()` die Funktion `a()` mehrmals auf, so kann man man dafür einen Aufrufpfeil zeichnen oder mehrere.

Im folgenden Beispiel rufen zwei Funktionen eine dritte auf; Abbildung 2 zeigt die Möglichkeiten, es darzustellen.

```c
#include <stdio.h>
void c(void) { printf(".");}
```

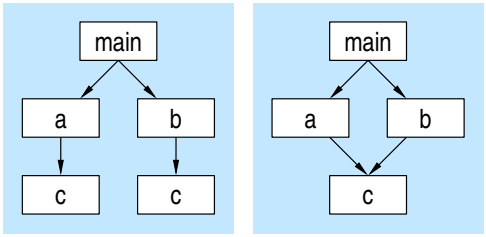

Abbildung 2: Aufrufdiagramm zu Codebeispiel 2

```
3 void a(void) { c(); }
4 void b(void) { c(); }
5 int main(void)
6 {
7     a(); b();
8     return 0;
9 }
```

2.8.2 Funktionsaufruf im Struktogramm

Ursprünglich unterschiedt man im Struktogramm nicht zwischen Blöcken und Funktionsaufrufen. Ein Rechteck mit einer Bezeichnung konnte stets entweder ein Block sein, den man in Zukunft an dieser Stelle noch näher beschreiben musste. Oder er konnte einen Funktionsaufruf darstellen, zu dem man an einer anderen Stelle eine Beschreibung anfertigte. In der entsprechenden Norm wurde dann für den Funktionsaufruf ein eigenes Symbol empfohlen (Abbildung 3).

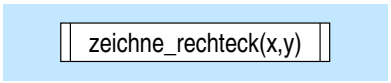

Abbildung 3: Symbol für einen Funktionsaufruf im Struktogramm

2.8.3 Beschreibung einer Funktionsdefinition im Struktogramm

In einem Struktogramm beschreibt man jede Funktion[1] auf einem eigenen Blatt oder zumindest in einer getrennten Zeichnung. Bei Codebeispiel 1 heißt das: Es sind fünf Funktionen (`main`, `a`, `b`, `c`, `d`) und damit fünf Zeichungen. Bei jeder Zeichnung muss der Name der Funktion (bei `main()` eventuell noch der Name des Programmes) als Überschrift vorhanden sein.

Eine Besonderheit bei C ist es, dass der Rückgabewert keinen festen Namen hat (z. B. einen festgelegten Namen oder den Namen der Funktion), sondern erst durch eine return-Anweisung gefüllt wird — in der Regel durch den Inhalt einer Variable des richtigen Typs, die man vorher klugerweise angelegt hat. Gleichzeitig ist die return-Anweisung eine Sprunganweisung. Dafür gibt es im Struktogramm tatsächlich ein Symbol (obwohl die Erfinder des Struktogrammes Sprunganweisungen gar nicht mochten). Abbildung 4 zeigt dieses Symbol.

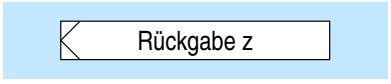

Abbildung 4: Symbol für eine Sprunganweisung im Struktogramm

[1] Bei anderen Programmiersprachen: Jede Prozedur, jedes Unterprogramm, jede Subroutine, jede Methode – also jeder Sprung, bei dem die Rücksprungadresse gespeichert wird

2.8.4 Struktogramm-Beispiel

Im folgenden Programm bekommt eine Funktion zwei Zahlen als Parameter und gibt ein Rechenergebnis zurück.

```c
#include <stdio.h>
int xhochy(int x, int y)
{
    int erg=1, lauf;
    if(x<0 || y<0)
        return -1;
    if(x==0 && y==0)
        return -2;
    for(lauf=0; lauf<y; ++lauf)
        erg *= x;
    return erg;
}
int main(void)
{
    int x, y, z;
    scanf("%i", &x);
    scanf("%i", &y);
    z=xhochy(x, y);
    printf("%i\n", z);
    return 0;
}
```

Abbildung 5 zeigt das entsprechende Struktogramm.

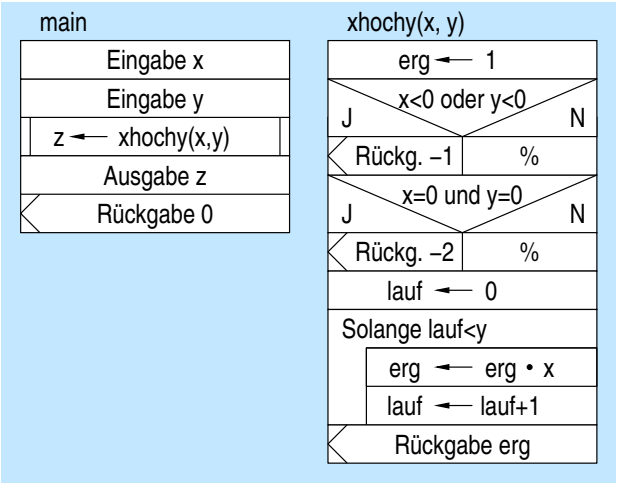

Abbildung 5: Struktogramm zu Codebeispiel 3

2.8.5 Kommentare im Quelltext

Immer wieder wird gefordert, dass der Programmierer sein Programm auch im Quelltext dokumentieren soll, und zwar durch Kommentare. Das ist oft sehr hilfreich, wenn man es richtig macht. Hier zwei Beispiele:

```c
    x=x+48; /* Hier wird zu x der Wert 48 addiert */
```

```
2     x=x+48; /* Wandelt Ziffer x (0-9) zu Zeichen x ('0'-'9') um */
```

Hilfreich ist oft nicht (nur) die Beschreibung dessen, *was* man macht (oberes Beispiel), sondern *warum* man es macht (unteres Beispiel).

Besonders sinnvoll ist eine ausführliche Beschreibung am Anfang einer Funktionsdefinition. Traditionell werden dabei folgende Aspekte als wichtig angesehen[2]:

a) Bezeichnung: Wie heißt die Funkton?

b) Übersicht: Benötigte Headerdateien, Prototyp der Funktion

c) Beschreibung: Was macht die Funktion, wozu ist sie geeignet?

d) Parameter: Für jeden Parameter sind der Typ und der Zweck anzugeben

e) Rückgabewert: Was wird zurückgegeben? Falls der Rückgabewert ein Fehlercode ist, sollte hier für jeden Wert die Bedeutung angegeben werden

f) Fehler: Welche Grenzen hat die Funktion? Was passiert bei Überschreiten dieser Grenzen?

g) Siehe auch: Optional - wo gibt es weitere Information?

h) Änderungen: Das ist am wichtigsten – bei jeder Änderung des Quelltextes (auch bei Fehlerbeseitigungen!) muss man angeben: Autor, Datum, Art der Änderung

Für die Funktion `xhochy()` aus Codebeispiel 3 könnte das so aussehen:

```
1  /*************************************
2     BEZEICHNUNG
3        xhochy() - x hoch y
4     UEBERSICHT
5        int xhochy(int x, int y);
6     BESCHREIBUNG
7        xhochy berechnet x hoch y und gibt das Ergebnis zurueck.
8        Bei x=0 und y=0 und bei y<0 wird ein Wert < 0 zurueckgegeben.
9     PARAMETER
10       int x - Basis
11       int y - Exponent
12    RUECKGABEWERT
13       >=0   Ergebnis
14       -1    Fehler: x<0 oder y<0
15       -2    Fehler: x=0 und y=0
16    FEHLER
17       Es findet keine Ueberpruefung auf Ueberlauf statt.
18    AENDERUNGEN
19       Ich, 04.10.2017, Version 1.00, Neu
20 *************************************/
```

Am Anfang des Quelltextes oder direkt vor der `main`-Funktion könnte entsprechend eine Beschreibung des Gesamtprogramms stehen.

[2] Angelehnt an das Hilfe-System von Linux