Hardwarenahe Softwareentwicklung Tutorium

Funktionen, Header und Source Files

SS 16

Tobias Trabelsi

9. Mai 2016

Hochschule Bochum
Bochum University
of Applied Sciences

Inhaltsverzeichnis

1 Motivation

Eine komplexe Aufgabe erfordert meist viele Funktionen. Diese alle in eine einzige Datei zu packen, würde diese Datei sehr schnell unübersichlich und auch unhandlich machen.
Das Auslagern von Funktionen ist daher eine willkommene Möglichkeit für Übersicht zu sorgen.

1.1 Ankündigen von Funktionen

Da C eine iterative Programmiersprache ist, muss eine bestimmte Funktion vor dem Benutzen zumindest angekündigt werden.
Zum ankündigen von Funktionen, muss sich in C zumindest den Funktionsprototyp vor der Main Funktion befinden.

1.1.1 Beispiel

```
1  #include <stdio.h>
2
3  int sum(int a, int b);
4  int minus(int a, int b);
5  double devide(int a, int b);
6  int multiply(int a, int b);
7
8
9  int main()
10 {
11     int a = 5;
12     int b = 3;
13
```

```
14      printf("\n Summe \t");
15      printf("%d",sum(a,b));
16      printf("\n Teilen \t");
17      printf("%f",devide(a,b));
18      printf("\n Minus \t");
19      printf("%d",minus(a,b));
20      printf("\n Dividieren \t");
21      printf("%d",multiply(a,b));
22      printf("\n");
23
24      return 0;
25
26  }
27
28  int sum(int a, int b)
29  {
30      return a+b;
31  }
32
33  int minus(int a, int b)
34  {
35      return a-b;
36  }
37
38  double devide(int a, int b)
39  {
40      return (double)a/b;
41  }
42
43  int multiply(int a, int b)
44  {
45      return a*b;
46  }
```

2 Header und Source Dateien

Das übersetzen von C-Quelltext in ein ausführbares Programm ist normalerweise unterteilt in zwei schritte: Kompillieren und Linken.
Als erstes übersetzt der Compiler den Quelltext in so genannte „Object-Files “(*.o). Dann verknüpft der Linker diese Object-Files mit statisch verknüpften Programm-Bibliotheken und erzeugt dann das ausführbare Programm.
In dem ersten Schritt nimmt der Compiler eine zusammenführung aller C-Dateien, welche durch so genannte „Präprozessordirektiven “aufbereitet wurde und übersetzt diese datei in eine Object-File.
Bei jeder Übersetzung von Quelltext in ein ausführbares Programm muss JEDE verwendete Funktion vor dem aufrufen angegeben sein.

2.1 Aufbau einer Header Datei

In einer Header Datei stehen die zu implementierenden Funktionsprototyp so wie die nötigen Präprozessordirektiven. Hier können ebenfalls andere includes stehen

2.1.1 Beispiel

```
1 #ifndef MATH_UTIL
2 #define MATH_UTIL
3
4 int sum(int a, int b);
5 int minus(int a, int b);
6 double devide(int a, int b);
7 int multiply(int a, int b);
8
```

```
9
10 #endif
```

2.2 Aufbau einer Source Datei

Die Source-Datei muss die zugeordete Header-Datei inkludieren und die dort angekündigten Methoden implementieren.

2.2.1 Beispiel

```
1  #include "math_util.h"
2
3  int sum(int a, int b)
4  {
5      return a+b;
6  }
7
8  int minus(int a, int b)
9  {
10     return a-b;
11 }
12
13 double devide(int a, int b)
14 {
15     return (double)a/b;
16 }
17
18 int multiply(int a, int b)
19 {
20     return a*b;
21 }
```

3 Aufgaben

3.1 Ankündigen von Funktionen

Programmieren Sie ein Programm, was die ersten 15 Fibonacci Zahlen ausgibt. Ihre main Methode soll nur den Funktionsaufruf „fibonacci(15) “beinnhalten.

3.2 Auslagern von Funktionen

Lagern Sie nun all Ihre Funktionen in Source- und Header-Dateien aus.

3.3 Rekursion

Falls Sie schon sicher sind, versuchen Sie die Fibonacci-Funktion rekursiv zu programmieren.