C: Funktionen

Philip Gawehn

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Do., 22.05.2014

Gliederung

Was ist eine Funktion?

- Unterprogramm
  - Unterbrechung des sequenziellen Ablaufs $\Rightarrow$ 'Springen'
- Bestimmte Aufgabe $\rightarrow$ 'Spezialist'
- Programmübergreifend
- Meist übersichtlicher

Grundaufbau

```
1  schluesselwoerter rueckgabetyp funktionsname (parameterliste)
2  {
3      Anweisungen;
4  }
```

- Schlüsselwörter: ‘inline’, ‘extern’, ‘static’
- Rückgabetypen: void, int, char, ...
- Funktionsname
  - (Maximal) 31 Zeichen
  - Besteht aus: A-z, 0-9, _
- Optionale Rückgabe mittels ‘return’

Schlüsselwörter (1)

- extern
  - Funktion ist global sichtbar
    - Muss im Header vorhanden sein.
  - Implizit gegeben bei Funktionen
- static
  - Funktion ist nur in eigener Datei aufrufbar
    - Vergleiche ‘private’ in Java
    - Durch Funktionspointer trotzdem in anderen Dateien aufrufbar
  - Erlaubt dem Compiler implizit zu inlinen
  - ‘static’ überschreibt ‘extern’ sinngemäß
- Es existieren keine ‘extern static’ Funktionen!

Schlüsselwörter (2)

- inline
  - Code wird vom Compiler direkt an die geforderte Stelle eingefügt
    - Kein Sprung in die Funktion und zurück
  - Optimal für kleine Funktionen
  - Programmierer kann inlinen ‘empfehlen’
  - Compiler kann ‘inline’ ignorieren
- Nicht mit Inline-Assembler verwechseln!
  - Assembler direkt im C-Code

Deklaration und Implementation

- Deklaration
  - Bekanntmachen einer Funktion / Variablen beim Compiler
- Definition / Implementation
  - Was die Funktion macht

```
1  int mul(int x, int y)
2  {
3      // ...
4      return (x + 1) * y;
5  }
```

Variadische Funktionen

- Möglichkeit nahezu unendlich viele Parameter anzugeben
- Implementation mit <stdarg.h> möglich
- Beispiele:
  - printf(“Ausgabe: %s”, ...);
  - Vektoren

Parameterübergabe

- call-by-value ⇒ Wertparameter
  - Kopiert den Parameter ⇒ ineffizient
  - Standardisiert in C
    - Structs / ... werden komplett kopiert!
    - Arrays werden nicht kopiert ⇒ Ausnahme!
- call-by-reference ⇒ Referenzparameter
  - call-by-value mit Pointern
  - Pointer wird übergeben
  - Änderungen von Variablen bleiben erhalten
  - Oft schneller und effizienter

Die Main-Funktion - Der Einstieg

- return implizit gegeben $\Rightarrow$ return 0;
- Parameter optional
- Gibt Integer zurück

- Das kleinstmöglichste Programm:

```
1  int main()
2  {
3      // Anweisungen
4  }
```

Die Main-Funktion - Der Einstieg

```
1  int main(int argc, char *argv[])
2  {
3      // Anweisungen
4  }
```

- argc ⇒ Anzahl der Parameter
- argv ⇒ Die Parameter als Array
  - argv[0] ⇒ Programmname

Prototypen

- Voranstellen der Deklaration vor die Implementation
- Parameternamen sind optional
- ‘extern’, ‘static’ und ‘inline’ entfallen
- Jede Funktion kann jede andere Funktion aufrufen.
- Teilt dem Compiler den Rückgabewert mit
  - ‘int’ wird als Standard angenommen
  - Compiler findet mehr Fehler

Prototypen

```
1   // Prototyp
2   int mul(int, int);
3
4   // ...
5
6   // Implementation bzw. Definition
7   int mul(int x, int y)
8   {
9       // ...
10      return ((x + 1) * y);
11  }
```

Header-Dateien

- Header-Dateien
  - Beinhalten Prototypen
  - Kein Ausführbarer Code - nur Deklarationen!
  - Modulkonzept
    - Viele Dateien
    - Übersichtlicher, geordneter Code
    - Geringerer Compileraufwand
- #include
  - Inhalte bestehender Dateien werden direkt eingefügt
  - In der Regel Header-Dateien
  - 2 Arten von #include
    - #include <file.h>⇒ Bibliotheksverzeichnis
    - #include “file.h” ⇒ Aktuelles Projektverzeichnis

Header-Dateien - Der Include-Guard

- Verhindert das doppelte Einbinden von Header-Dateien

```
1  #ifndef MEIN_HEADER
2  #define MEIN_HEADER
3
4  // Namespacesm, Prototypen, ...
5
6  #endif
```

Funktionspointer

- Funktionen sind Code im Speicher
- Zeiger auf den Code

- Funktionen als Parameter übergeben
- Funktionen können Funktionen zurückgeben

Funktionspointer

```
1   // Zeiger auf eine Funktion wird definiert
2   double (*function) (int);
3
4   // Prototyp
5   double fct_name(int);
6
7   // Zuweisen der Funktionsadresse in die Variable
8   function = &fct_name;
9   function = fct_name;
10
11  // Aufrufen
12  (*function)(42);
13  function(42);
```

- typedef für bessere Lesbarkeit

Was ist Rekursion?

- Eine Funktion ruft sich selber auf
- Es muss erreichbare Abbruchbedingungen geben
  - Alternative: Dauerschleife
  - Gefahr: Stack Overflow
- Teilweise inperformant und schlecht lesbar
- Iteration wird oft bevorzugt
- Jede rekursive Funktion existiert auch als iterative Funktion

Beispielform der Rekursion

```
1  int rek(int ihelp)
2  {
3      // ...
4      rek(<int>)
5      // ...
6      return <int>;
7  }
```

Zusammenfassung

- Grundlagen von Funktionen
- Parameterübergabe
- Main-Funktion
- Header-Dateien
- Prototypen
- Funktionspointer
- Rekursion

Noch Fragen?

Literatur

- Praktische Einführung in C
  Peter Baeumle-Courth, Thrsten Schmidt
- C - Programmieren von Anfang an
  Helmut Erlenkötter
- C von Kopf bis Fuß
  David Griffiths, Dawn Griffiths
- C++11 für Programmierer
  Rainer Grimm

Literatur

- http://de.wikibooks.org/wiki/C-Programmierung
- http://www.c-howto.de/tutorial-einfuehrung.html
- http://openbook.galileocomputing.de/c_von_a_bis_z/index.htm
- http://manderc.manderby.com/types/functionpointertype/
- http://stackoverflow.com/
- http://en.wikipedia.org/
- http://de.wikibooks.org/wiki/C%2B%2B-Programmierung
- http://en.wikipedia.org/wiki/Entry_point#C_and_C.2B.2B