C Programmierung

Benjamin Matthes

6. November 2012

Hello World!

C Hello World!

```c
#include <stdio.h>

int main(int argc, char **argv) {
   printf("Hello World!\n");
   return 0;
}
```

Java Hello World

```java
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Hello World!

C Hello World!

```c
#include <stdio.h>

int main(int argc, char **argv) {
   printf("Hello World!\n");
   return 0;
}
```

- Mit `#include <stdio.h>` wird das Modul stdio.h geladen welches u.a. `printf` zur Verfügung stellt.
- Jedes ausführbare C-Programm enthält eine `main` Funktion. Sie gibt einen `int` Wert zurück, welcher den exit code angibt.
- C kennt keine Klassen! Wir haben es in C mit Funktionen und nicht mit Methoden zu tun!
- Die Parameter der `main`-Funktion werden später erklärt! (Ähnlich zu `String[]` in Java)

Syntax

- Die Syntax von C hat u.a. Java beeinflusst.
- Anweisungen werden mit ; getrennt.
- Anweisungen können mit {,} zusammengefasst werden.
- Viele Syntax-Konstrukte von Java gibt es auch in C:

```
if (foo == bar) {
  printf("foo is bar\n");
} else {
  printf("bar is not foo!\n");
}
```

- Funktionsdefinitionen so wie Methoden bei Java:

```
int faculty(int n) { ... }
```

Typen und Operatoren

Ein paar Typen von C

- Verschiedene Zahlentypen, z.B. `int`, `double`, `float`, `long`
- Verschiedene Operatoren auf Zahlentypen: +,*,%
- Bitoperationen auf arithmetischen Datentypen: `x & y` entspricht bitweisem and
- Zahlentypen gibt es auch in zeichenloser Form, z.B. `uint`
- Buchstaben: `char` (arithmetischer Datentyp), z.B. `'c'`
- Keinen `boolean`-Typ! Boolsche Werte sind in C `int` Werte, 1 entspricht true und 0 entspricht false. (Modul stdbool.h führt einen Pseudotyp bool ein).
- "Zusammengesetzte Daten" (record) gibt es in C: `struct`

Typecast

```
printf("The 65 letter is: %c\n", (char) 65);
```

Ausgabe: `The 65 letter is: A`

Record Datentypen - Ein Beispiel

Beispiel zu structs

```
struct point {
    int x;
    int y;
};

struct point p = {1, 2}; // allocation on stack

if (p.x == 2)
    printf("You never reach me!\n");
else
    printf("p.x is not two, amazing.\n");

p.y = 3;
```

Record Datentypen - Definition

Einfache Definition

```
struct Foo {
    int x, y;
};
```

Achtung!

Das Semikolon gehört zur Definition dazu.

Variablendeklaration

struct Foo myFooStruct;

Record Datentypen - Definition

Alternative Methode

```
typedef struct {
    int x, y;
} Foo;
```

Alternative Methode 2

```
struct Foo {
    int x, y;
};
typedef struct Foo Foo;
```

Variablendeklaration

```
Foo myFooStruct;
```

Pointer (Zeiger)

- Ein Pointer ist ein Wert, welcher die Adresse eines Objekts repräsentiert.
- Mit dem Operator & kann man die Adresse von einem Objekt erhalten.
- Mit dem Operator * erhält man das Objekt auf das ein Zeiger zeigt. (dereferenzieren)

Beispiel zu Pointern

```
int x = 3; // x ist ein Integer
int *p; // p ist ein Zeiger auf einen Integer
p = &x; // p zeigt nun auf die Adresse von x
*p = 5; // x besitzt den Wert 5
```

Arrays (Felder)

- Syntax für Arrays:

```
double coords[3];
coords[0] = 1.5;
coords[1] = 2.0;
*(coords + 2) = 2.5;
```

- Arrays der Länge n sind von 0 bis $n - 1$ indiziert.
- Dabei ist `coords[i]` gleichbedeutend mit `*(coords + i)`.
- Anzahl der Elemente muss zur Übersetzungszeit bekannt sein.

Zusammenhang zwischen Arrays und Zeiger

In unserem Beispiel ist `coords` als Array definiert, somit wird `coords` als Wert verwendet, welcher die Adresse des ersten Elements des Arrays enthält. Wie jeden anderen Zeiger lässt sich die Adresse dereferenzieren.

Was ist "pointer arithmetic"?

Pointer arithmetic

Unter pointer arithmetic versteht man die Addition auf Zeigern. Dabei wird der "Typ des Zeigers" beachtet.

Beispiel

Worauf zeigen diese Zeiger?

```
*(coords+2) = 3;
*(coords+3) = 2;
```

`(coords+2)` zeigt auf das 3. Element des Arrays. `(coords+3)` zeigt auf das 4. Element des Arrays `coords`.

Achtung

Zeiger können auf ungültige Stellen im Speicher zeigen (segmentation fault)!

Pointer - Strings

Strings

Ein String ist ein `char*`. Um das Ende eines Strings erkennen zu können, ist das letzte Zeichen des Strings `'\0'`.

Länge eines Strings

```
int strlen(char *s) {
  int i = 0;
  while ( *(s+i) != '\0' ) i++;
  return i;
}
```

Pointer - Argumente an die `main`-Funktion

Die `main`-Funktion

```
int main(int argc, char **argv) { ... }
```

- `argc` gibt die Anzahl der übergebenen Argumente an (argument count)
- `argv` ist ein Array von Strings (argument vector)

Beispiel

```
$ myprog -a -b
argc = 3
argv[0] = "myprog"
argv[1] = "-a"
argv[2] = "-b"
```

Erzeugen von Datenstrukturen auf dem Heap

`malloc` und `free`

- Mit der Funktion `void* malloc(size_t size)` lässt sich ein Speicherblock allokieren.
- Mit der Funktion `void free(void *ptr)` lässt sich ein allokierter Speicherblock freigeben.
- Die Funktionen sind in `stdlib.h` zu finden.

Beispiel zu `malloc` und `free`

```
struct point *p =
    (struct point*) malloc(sizeof(struct point));
free(p);
```

Record Datentypen - Funktionsaufruf

Was passiert beim Funktionsaufruf, falls ein struct an eine Funktion übergeben wird?

```
struct foobar {
    int x,y;
    char *s;
};
void modify(struct foobar f) {
    f.x = 3;
};
struct foobar y = {0,0,""};
modify(y);
```

Call by value

Das komplette struct wird kopiert und an die aufgerufe Funktion übergeben! `y.x` ist immer noch 0!

Record Datentypen - Funktionsaufruf

Lösung des Problems beim Funktionsaufruf

```
struct foobar {
    int x,y;
    char *s;
};
void modify(struct foobar *f) {
    (*f).x = 3;
}
struct foobar y = {0,0,""};
modify(&y);
```

Syntaktischer Zucker

```
(*f).x = 3;
f->x = 3;
```

Module

Laden von Modulen

```
#include <stdlib.h>     // looks in the include path (-I)
#include "my_module.h" // looks in the same directory
```

Schreiben von Modulen

```
#ifndef VERY_UNIQUE_STRING_NOBODY_ELSE_USES
#define VERY_UNIQUE_STRING_NOBODY_ELSE_USES

// insert your declarations here

#endif
```

Zeiger und Referenzen

Wie ist das in Java?

In Java gibt es keine Zeiger. In Java hat man Referenzen auf Objekte. Referenzen zeigen immer auf ein Objekt (des richtigen Typs), egal wo es sich im Speicher befindet!

Was tut ein Zeiger?

Ein Zeiger repräsentiert eine Adresse des Speichers. Dabei ist es unabhängig davon, was sich in dieser Speicherstelle befindet. Es ist möglich ...

- eine beliebige Stelle im Speicher als Zeiger auf ein Objekt anzunehmen.
- den Zeiger auf ein Objekt vom Typ x zu einem Zeiger auf ein Objekt vom Typ y zu verändern. Der Zeiger zeigt dennoch auf das Objekt vom Typ x!

Umgang mit Zeigern

Gehen Sie sorgfältig mit Zeigern um.