Programmieren in C

Schleifen und so...

thoto

/dev/tal e.V.

23. März 2013 (Version vom 10. April 2013)

Agenda für Heute

1. und weiter mit Funktionen
   - Wiederholung
   - Prototypen
2. Schleifen und Rekursion
   - Allgemeines
   - Schleifen
   - Rekursion

Funktion vs. Methode

- Funktion hat Rückgabewert, Methode nicht
- beide haben Parameter
- eigener Variablenraum

Beispiel: Funktion

Funktion

```
int add(int a, int b){
    int c; c=a+b;
    return c;
}
int main(){
    int c=21;
    printf("%d\n", add(c,2));
    return 0;
}
```

Prototypen: Problem

Problem a() braucht b(), was a() aufruft.

Prototypen: Problem

Problem a() braucht b(), was a() aufruft.

Ansatz Wir brauchen a() in b() ohne a() zu implementieren.

Prototypen: Problem

Problem a() braucht b(), was a() aufruft.

Ansatz Wir brauchen a() in b() ohne a() zu implementieren.

Lösung Prototypen

Prototypen: Problem

Problem a() braucht b(), was a() aufruft.

Ansatz Wir brauchen a() in b() ohne a() zu implementieren.

Lösung Prototypen

Anm: Insbesondere bei sog. Modulen nötig (später mehr!)

Prototypen: Beispiel

```
int pong(int i);

int ping(int i){ // braucht pong();
    printf("Ping...");
    return pong(i-1);
}
int pong(int i){ // braucht ping();
    printf("pong.\n");
    if(k>0) return ping(i);
    else  return 0;
}
```

Notwendigkeit von Schleifen

- „Iteration“ genannt
- Sachen mehrfach ausführen
- auf Ergebnis warten
- ...

Zählschleife vs. Solange-Schleife

- Zählschleife zählt über Reihe von Zahlen
- `for(i=0;i<=50;i++)`

Zählschleife vs. Solange-Schleife

- Zählschleife zählt über Reihe von Zahlen
- `for(i=0;i<=50;i++)`
- Solange-Schleife: wiederholt solange Bedingung erfüllt
- `while(i<=50)`

Beispiel: Fakultät mit for

```
int fak(int z){
  int r,i;
  for(i=0;i<=z;i++){
    r*=i;
  }
  return r;
}
```

Collatz-Problem

- Wenn Zahl gerade: Dividiere Zahl durch 2
- Wenn Zahl ungerade: Multipliziere Zahl mit 3 und addiere 1

Collatz-Problem

- Wenn Zahl gerade: Dividiere Zahl durch 2
- Wenn Zahl ungerade: Multipliziere Zahl mit 3 und addiere 1

$\Rightarrow$ Es kommt immer 1 raus.

$\Rightarrow$ (btw.: unbewiesen bis 2009)

Beispiel: Collatz mit While-Schleife

```c
int collatz(int p){
  int z=p; //setze z auf p
  while(z!=1){ //Solange z nicht 1 ist...
    if((z%2)==1) { // z ist ungerade
      printf("p=%d -> ungerade\n",z);
      z=(3*z)+1; // z mit 3 multipl. und 1 add.
    } else{ // z ist gerade
      printf("p=%d -> gerade\n",z);
      z=z/2; // z halbieren
    }
  }
  return z;
}
```

Rekursion

Rekursion

- Funktion ruft sich selber auf

Rekursion

- Funktion ruft sich selber auf
- Vorteil: Fix

Rekursion

- Funktion ruft sich selber auf
- Vorteil: Fix
- Vorteil: Reduzierung des Problems

Rekursion

- Funktion ruft sich selber auf
- Vorteil: Fix
- Vorteil: Reduzierung des Problems
- Nachteil: Bei C bitte nicht übertreiben

Rekursion

- Funktion ruft sich selber auf
- Vorteil: Fix
- Vorteil: Reduzierung des Problems
- Nachteil: Bei C bitte nicht übertreiben

$\Rightarrow$ Stack-Überlauf

Beispiel: Collatz mit Rekursion

```
int collatz_rek(int p){
  if(p==1)    return 1; // p ist eins => Ende!
  if((p%2)==1) { // p ist ungerade
    printf("p=%d -> ungerade\n",p);
    //Collaz-Problem fuer 3*p+1 loesen!
    return collatz_rek((3*p)+1);
  } else{ // p ist gerade
    printf("p=%d -> gerade\n",p);
    //Collaz-Problem fuer p/2 loesen!
    return collatz_rek(p/2);
  }
}
```

Beispiel: Fakultät mit Rekursion

```
int fak(int z){
  if(z==1){ return 1;} //Fakultaet von 1 ist 1.
  else return z*fak(z-1);
}
```

Schema: Fakultät mit Rekursion

$$
\begin{aligned}
& \mathit{fakultaet}(5) & \\
= \quad & \mathit{fakultaet}(4) & *5 \\
= \quad & \mathit{fakultaet}(3) & *4*5 \\
= \quad & \mathit{fakultaet}(2) & *3*4*5 \\
= \quad & \mathit{fakultaet}(1) & *2*3*4*5 \\
= \quad & 1 & *2*3*4*5
\end{aligned}
$$

Schema: Collatz mit Rekursion

$$Collatz(24) =$$

$$\begin{aligned} \text{gerade} \quad & Collatz(24/2) \\ \text{ungerade} \quad & Collatz((3*24)+1) \end{aligned}$$