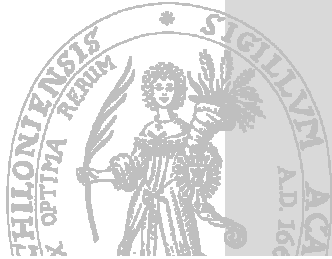


# Programmieren in C

Simone Knief – Rechenzentrum der CAU



## Programmieren in C

### Felder

- Was ist ein Feld ?
- Wie definiert man ein- und mehrdimensionale Felder ?
- Wie deklariert und initialisiert man Felder ?

## Programmieren in C

### Gliederung

- Einleitung
- Komponenten eines C-Programms
- Daten speichern: Variablen und Konstanten
- Anweisungen und Ausdrücke
- Operatoren
- Kontrollstrukturen
- Felder und Strings
- Funktionen, globale und lokale Variablen
- Zeiger
- Arbeiten mit Dateien
- Strukturen

## Programmieren in C

### Felder

- Zusammenfassung von Speicherstellen für Daten desselben Datentyps
- werden auch als Arrays oder Vektoren bezeichnet
- einzelne Speicherstellen werden als Elemente bezeichnet
- können ein- oder mehrdimensional sein
- können sowohl Integer- als auch Fließkommazahlen speichern

- lineare Anordnung von Elementen im Speicher
- Elemente werden im Arbeitsspeicher als ein ganzer Block gespeichert
- Felder deklarieren:
  - allg. `datentyp feldname [elementanzahl];`
  - Bsp.: `float ausgaben [12];`
- in C sind Felder immer nullbasiert, d.h. 1. Element ist `feldname[0]`
  - Bsp.: `ausgaben[12]: ausgaben[0], ausgaben[1], ... ausgaben[11]`

5

- **Felder initialisieren:**
  - allg. `feldname[]={Element1, Element2, ..., ElementN};`
  - Beispiel:
    - `int feldname[4]={100, 200, 300, 400};`
    - `int feldname[]={100, 200, 300, 400};`
    - `int feldname[10]={1,2,3};` --> nur die ersten 3 Elemente sind initialisiert
    - `int feldname[10]={0};` ---> alle Elemente haben den Wert 0
    - `int feldname[10]={1};` ---> nur das erste Element hat den Wert 1
    - `int feldname[3]={12,34,55,54,33};` → *Compilerfehler*

6

- einzelne Feld-Elemente werden über Positionsnummer (Index) angesprochen
- Positionsnummern werden in eckigen Klammern angegeben
- Indexnummern starten mit 0
- Beispiel:
  - `temperatur [365]`
  - 1. Element: `temperatur[0]`
  - 2. Element: `temperatur[1]`
  - 365. Element: `temperatur[364]`
- mit Feld-Elementen kann wie mit Variablen gearbeitet werden
  - z.B.: Mittelwert von 3 Tagen:
    - mit Variablen: `mittel=(temp1+temp2+temp3)/3.0`
    - mit einem Feld: `mittel=(temp[0]+temp[1]+temp[2])/3.0`
- Achtung: nie über das Ende eines Feldes hinausschreiben: z.B.: `temp[370]`

7

```
#include <stdio.h>
int main()
{
    float ausgaben[4];
    int count;
    for (count=1; count<4;count++)
    {
        printf("Ausgaben fuer Monat %d eingeben.\n",count);
        scanf("%f",&ausgaben[count]);
    }
    for (count=1;count<4;count++)
    {
        printf("Monat %d = %f Euro \n",count,ausgaben[count]);
    }
    printf("Programm beendet.\n");
    return 0;
}
```

8

```
#include <stdio.h>
int main()
{
    float ausgaben[4];
    int count;
    for (count=1; count<4;count++)
    {
        printf("Ausgaben fuer Monat %d
        eingeben.\n",count);
        scanf("%f",&ausgaben[count]);
    }
    for (count=1;count<4;count++)
    {
        printf("Monat %d = %f Euro
        \n",count,ausgaben[count]);
    }
    printf("Programm beendet.\n");
    return 0;
}
```

- Programmausgabe:
- Ausgaben für Monat 1 eingeben  
33.10  
Ausgaben für Monat 2 eingeben  
100.00  
Ausgaben für Monat 3 eingeben  
22.22
- Monat 1 = 33.10 Euro  
Monat 2 = 100.00 Euro  
Monat 3 = 22.22 Euro

9

- werden für die Darstellung von Matrizen verwendet
- Deklaration:
  - allgemein: Datentyp feldname [Dimension1] ... [DimensionN];
  - Beispiel: int matrix [4][5] oder int matrix [4][5][2];
- bei der Initialisierung wird der letzte Index zuerst durchlaufen
- Beispiel:
 

```
int array [4] [3] = {1,2,3,4,5,6,7,8,9,10,11,12};
int array [4] [3] = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
```

```
array[0][0]=1
array[0][1]=2
array[0][2]=3
array[1][0]=4
array[1][1]=5
...
array[3][1]=11
array[3][2]=12
```

10

- fassen mehrere Datenelemente des gleichen Typs zusammen
- einzelne Feldelemente werden durch Indizes charakterisiert
- müssen wie Variablen vor der Verwendung deklariert und initialisiert werden
- Deklaration
  - Datentyp feldname [Dimension1] ... [DimensionN];
- Initialisierung:
  - feldname[]={Element1, Element2, ..., ElementN};
  - feldname[]={0}
- können wie normale Variablen verwendet werden
- Felder sind nullbasiert: 1. Element ist feldname[0] und nicht feldname[1]
- Felder führen keine Bereichsprüfung durch

11

- Datentyp char dient zur Aufnahme von einzelnen Zeichen
- C kennt keinen Datentyp für Zeichenketten
- Strings sind Zeichenketten
- C kennt keinen Datentyp für Zeichenketten
- Strings müssen als Felder deklariert werden

12

- mit einzelnen Zeichen arbeiten
  - Deklaration:
    - char a,b,c;
  - Deklaration und Initialisierung
    - char code = 'x';
  - Zuweisung
    - code = 'a';
  - Formatbezeichner bei der Ein- und Ausgabe: %c

13

```
/* Ein- und Ausgabe von einzelnen Zeichen */
#include <stdio.h>
int main()
{
    char buchstabe;
    printf("Bitte einen Buchstaben eingeben \n");
    scanf("%c",&buchstabe);
    printf("Sie haben den Buchstaben %c eingegeben. \n",buchstabe);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

14

```
/* Ein- und Ausgabe von einzelnen Zeichen */
#include <stdio.h>
int main()
{
    char buchstabe;
    printf("Bitte einen Buchstaben eingeben \n");
    scanf("%c",&buchstabe);
    printf("Sie haben den Buchstaben %c eingegeben. \n",buchstabe);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

- Programm übersetzen:
  - gcc -o zeichen.exe zeichen.c
- Programm ausführen
  - ./zeichen.exe
- Programmausgabe:
 

Bitte geben Sie einen Buchstaben ein

D

Sie haben den Buchstaben D eingegeben.

Programm erfolgreich beendet.

15

- Strings sind Zeichenketten
- C kennt keinen Datentyp für Zeichenketten
- Strings müssen als Felder vom Datentyp char deklariert werden
- Strings müssen immer um ein Zeichen größer deklariert werden
- String-Terminierungszeichen \0 kennzeichnet das Ende eines Strings

16

- Arbeiten mit Strings
  - Deklaration
    - char eingabe [15]; → Platz für 14 Zeichen
  - Deklaration und Initialisierung
    - char eingabe[]={"Programmierung"};
    - char eingabe[]={'P','r','o','g','r','a','m','m','i','e','r','u','n','g','\0'};
  - Formatbezeichner bei der Ein- und Ausgabe: %s
- Einlesen von Strings:
  - Formatbezeichner: %s
  - scanf("%s %s",vorname,nachname);
- Ausgabe von Strings:
  - Formatbezeichner: %s
  - printf("%s",vorname);

17

```
/* Einlesen von Strings */
#include <stdio.h>
int main()
{
    char vorname[20];
    char nachname[20];
    printf("Bitte geben Sie Ihren Vornamen ein \n");
    scanf("%s",vorname);
    printf("Bitte geben Sie Ihren Nachnamen ein \n");
    scanf("%s",nachname);
    printf("Sie heißen %s %s. \n",vorname,nachname);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

18

```
/* Einlesen von Strings */
#include <stdio.h>
int main()
{
    char vorname[20];
    char nachname[20];
    printf("Bitte geben Sie Ihren Vornamen ein \n");
    scanf("%s",vorname);
    printf("Bitte geben Sie Ihren Nachnamen ein \n");
    scanf("%s",nachname);
    printf("Sie heißen %s %s. \n",vorname,nachname);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

- Programm übersetzen:
  - gcc -o stringc.exe string.c
- Programm ausführen:
  - ./string.exe
- Programmausgabe:
  - Bitte geben Sie Ihren Vornamen ein  
Max
  - Bitte geben Sie Ihren Nachnamen ein  
Mustermann
  - Sie heißen Max Mustermann
  - Programm erfolgreich beendet.

19

- C enthält Bibliotheksfunktionen zur Bearbeitung von Strings
- zusätzliche Headerdatei <string.h> erforderlich
- Beispiele:
  - strcpy: Kopieren von Strings
  - strcmp: Vergleich von zwei Strings
  - strcat: Verbinden von Strings
  - strlen: Länge eines Strings ermitteln

20

## Stringfunktion strcpy

Rechenzentrum

- Kopiert einen String in einen anderen
- Beispiel:
 

```
#include <stdio.h>
#include <string.h>
int main()
{
    char[20] person1="Arbeiter";
    char[20] person2 = "Student";
    printf ("Status Anfang: %s und %s",person1,person2);
    strcpy(person2,person1);
    printf("Status nach Kopieren: %s und %s", person1,person2);
    return 0;
}
```

21

## Stringfunktion strcpy

Rechenzentrum

```
#include <stdio.h>
#include <string.h>
int main()
{
    char[20] person1="Arbeiter";
    char[20] person2 = "Student";
    printf ("Status Anfang: %s und %s",person1,person2);
    strcpy(person2,person1);
    printf("Status nach Kopieren: %s und %s", person1,person2);
    return 0;
}
```

- Programmausgabe  
Status am Anfang: Arbeiter und Student  
Status nach Kopieren: Arbeiter und Arbeiter

22

## Stringfunktion strcmp

Rechenzentrum

- vergleicht zwei Strings miteinander
- Strings werden Zeichen für Zeichen miteinander verglichen
- Rückgabewert:
  - 0: Strings sind identisch
  - ungleich 0: Strings sind unterschiedlich

23

## Stringfunktion strcmp (Beispiel)

Rechenzentrum

```
#include <stdio.h>
#include <string.h>
int main()
{char text1[20]="Hello", text2[20]="World",text3[20]="Hello";
if (strcmp(text1,text2) == 0)
    printf ("Text1 und Text2 sind identisch \n");
else
    printf("Text1 und Text2 sind unterschiedlich \n");
if (strcmp (text1,text3) == 0)
    printf("Text1 und Text3 sind identisch \n");
else
    printf("Text1 und Text3 sind unterschiedlich \n")
return 0;
}
```

## Stringfunktion strcmp (Beispiel)

Rechenzentrum

- Programmausgabe

Text1 und Text2 sind unterschiedlich

Text1 und Text3 sind identisch

## Stringfunktion strcat (Beispiel)

Rechenzentrum

```
#include <stdio.h>
#include <string.h>
int main(
{char text1[100]="Hallo "
  text2[50]="Programmierer";
printf ("%s \n",text1);
printf("%s \n",text2);
strcat(text1,text2);
printf("%s \n",text1);
printf("%s \n",text2);
return 0;
}
```

- Programmausgabe:

Hallo

Programmierer

Hallo Programmierer

Programmierer

## Stringfunktion strcat

Rechenzentrum

- verbindet zwei Strings miteinander
- Beispiel:

```
#include <stdio.h>
#include <string.h>
int main()
{char text1[100]="Hallo", text2[50]="Programmierer";
printf ("%s \n",text1);
printf("%s \n",text2);
strcat(text1,text2);
printf("%s \n",text1);
printf("%s \n",text2);
return 0;
}
```

## Stringfunktion strlen

Rechenzentrum

- Funktion zur Ermittlung der Länge eines Strings
- Beispiel:

```
#include <stdio.h>
#include <string.h>
int main()
{
char satz[]="Programmieren macht mir viel Spass.";
int laenge=strlen(satz);
printf("Laenge: %d Zeichen\n",laenge);
return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
    char satz[]="Programmieren macht mir viel
    Spass.";
    int laenge=strlen(satz);
    printf("Laenge: %d Zeichen\n",laenge);
    return 0;
}
```

- Programmausgabe:  
Laenge: 34 Zeichen

29

## Übungsblatt 2

### Aufgaben 1 bis 4

30

- Was ist eine Funktion ?
- Wie erzeugt man eine Funktion ?
- Wie gibt man einen Wert aus einer Funktion zurück ?
- Wie übergibt man einer Funktion Argumente ?
- Was ist der Unterschied zwischen globalen und lokalen Variablen ?
- Was sind rekursive Funktionen ?

31

- Problemstellung wird in kleinere Teilprobleme unterteilt
- Unterprogramm zur Lösung von Teilproblemen
- jede Funktion hat einen eindeutigen Namen
- Funktionen sind unabhängig voneinander
- dienen der besseren Strukturierung von Programmen
- werden oft für wiederkehrende Aufgaben verwendet
- C-Programmbibliothek enthält bereits zahlreiche Funktionen (z.B. printf, scanf)

32



## Vorteile von Funktionen

Rechenzentrum

- Programm wird strukturierter, da Aufgabenstellung in kleinere Einheiten aufgeteilt wird
- Quellcode ist besser lesbar
- Funktionen können mehrmals in einem Programm aufgerufen werden
- Funktionen lassen sich in andere Programme einbauen
- Fehler sind schneller auffindbar, da der Code nur an einer Stelle bearbeitet werden muss
- Veränderungen im Code sind leichter vorzunehmen und zu testen, weil nur Codefragmente betroffen sind

33

## Funktionen Beispiel

Rechenzentrum

- Gesamtaufgabe:
  - Daten einlesen, verarbeiten und Ergebnisse wieder ausgeben
- Aufteilung:
  - Eine Funktion ist für die Eingabe des Benutzers zuständig
  - Eine andere Funktion berechnet aus den Eingaben einen bestimmten Wert
  - Eine weitere Funktion übernimmt die Ausgabe der Ergebnisse
  - Eine Funktion ist für die Fehlerbehandlung zuständig
  - alle Funktionen werden aus dem Hauptprogramm main aufgerufen

34

## Funktionsweise einer Funktion

Rechenzentrum

/\* Funktionsweise einer Funktion \*/

#include <stdio.h>

int main()

{

....

Aufruf von Funktion1  Funktion1

....

Aufruf von Funktion2  Funktion2

...

....

Aufruf von Funktion3  Funktion3

...

return 0;

}

35

## Funktion (Beispiel)

Rechenzentrum

/\* Beispiel für eine einfache Funktion \*/

#include <stdio.h>

int kubik (int x); ---> Funktionsprototyp

int eingabe, antwort;

int main()

{

printf("Bitte geben Sie eine ganze Zahl ein.\n");

scanf("%d",&eingabe);

antwort=kubik(eingabe); ----> Funktionsaufruf

printf("Die Kubikzahl von %d ist %d.\n",eingabe,antwort);

return 0;

}

int kubik(int x)

{

int x3; -----> Funktionsdefinition

x3=x\*x\*x;

return x3;

}

36

## Funktionssyntax

Rechenzentrum

- Funktionsprototyp
  - allg. : rückgabetyf funktionsname (parameterliste);
  - z.B.: int kubik (int x);
- Funktionsdefinition
  - rückgabetyf funktionsname (parameterliste)
 

```
{
/* Anweisungen */
}
```

37

## Funktionsdefinition

Rechenzentrum

- Funktion besteht aus 2 Teilen
  - Funktionsheader
  - Funktionsrumpf
- allg. Form:
 

```
rückgabetyf funktionsname (Parameterdeklaration) ----> Funktionsheader
{
    Anweisungen ----> Funktionsrumpf (Funktionsdefinition)
}
```
- Beispiel:
 

```
int kubik(int x) ---> Funktionsheader
{
    int x3; -----> Funktionsdefinition
    x3=x*x*x;
    return x3;
}
```

38

## Funktionsheader

Rechenzentrum

- Aufbau:
  - rückgabetyf Funktionsname (Parameterliste)
  - int kubik (int x)
- Rückgabetyf:
  - legt Datentyp des Rückgabewertes fest
  - void wenn Funktion keinen Rückgabewert hat
- Funktionsname
  - eindeutiger Name
  - über diesen Namen wird die Funktion aufgerufen
  - Funktionsnamen beginnen häufig mit einem Großbuchstaben
- Parameterliste
  - Werte, die der Funktion zur Verarbeitung übergeben werden
  - jedes Argument besteht aus einem Datentyp und Variablennamen
  - verschiedene Argumente werden durch Komma getrennt (z.B. void name (int x, float y, char z))
  - (void) wenn keine Argumente übergeben werden

39

## Funktionsrumpf

Rechenzentrum

- wird von geschweiften Klammern umschlossen
- folgt unmittelbar an den Funktionsheader
- erledigt die eigentliche Arbeit, da in ihm alle Anweisungen aufgelistet sind
- Beispiel:
 

```
{
    int x3;
    x3=x*x*x;
    return x3;
}
```
- innerhalb einer Funktion darf keine neue Funktion definiert werden.
- Funktionsrumpf sollte möglichst kurz sein

40

## Wert aus einer Funktion zurückgeben

Rechenzentrum

- erfolgt mit der return-Anweisung
- allgemeine Form: return Ausdruck;
- z.B.: return ergebnis;
- gibt einen bestimmten Wert an den Aufrufer zurück
- beendet eine Funktion und es wird an die Position des Aufrufs zurückgesprungen
- stehen häufig am Ende einer Funktion
- es wird immer nur eine return-Anweisung ausgeführt
- Datentyp wird sowohl im Funktions-Header als auch im Funktionsprototyp festgelegt
- Beispiel:
 

```
int funk1 (int var)
{
    int x;
    /* Funktionscode */
    return x;
}
```

41

## Return-Anweisung (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int x,y,z;
int groesser_von(int a, int b);
int main()
{printf("Bitte 2 verschiedene Integer-Zahlen eingeben.\n");
  scanf("%d %d", &x, &y);
  z=groesser_von(x,y);
  printf("Der groessere Wert betraegt %d.\n",z);
  return 0; }

int groesser_von (int a, int b)
{
    if(a>b)
        return a;
    else
        return b;
}
```

42

## Return-Anweisung (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int x,y,z;
int groesser_von(int a, int b);
int main()
{printf("Bitte 2 verschiedene Integer-Zahlen
  eingeben.\n");
  scanf("%d %d", &x, &y);
  z=groesser_von(x,y);
  printf("Der groessere Wert betraegt
    %d.\n",z);
  return 0; }
```

```
int groesser_von (int a, int b)
{ if(a>b)
  return a;
  else
    return b;}
```

- Programmausgabe

```
Bitte 2 verschiedene Integer-Zahlen eingeben
1 3
Der groessere Wert betraegt 3
```

```
Bitte 2 verschiedene Integer-Zahlen eingeben
8 5
Der groessere Wert betraegt 8
```

43

## Funktionsprototypen

Rechenzentrum

- vergleichbar mit einer Variablendeklaration
- dienen zur Deklaration von Funktionen vor dem Aufruf
- legen Eigenschaften der Funktion fest und stellen Speicherplatz bereit
- müssen vor dem ersten Funktionsaufruf stehen
- identisch mit dem Funktionskopf mit angehängtem Semikolon:
 

```
allg.: rückgabetyf funktionsname (parameterliste);
z.B.: int kubik (int x);
```
- Datentyp muss in der Parameterliste vorhanden sein, variablenname ist optional
- sollten möglichst am Anfang eines Programms stehen
- Compiler kann überprüfen ob Funktion mit den richtigen Argumenten aufgerufen wird
- für die main-Funktion ist kein Prototyp erforderlich
- enthält folgende Informationen:
  - Namen der Funktion
  - Parameterliste, die an die Funktion übergeben wird
  - Datentyp des Rückgabewertes

44

## Aufruf einer Funktion

Rechenzentrum

- wird über den Funktionsnamen aufgerufen
- rückgabewert = funktionsname (var1, var2, ...);
- Funktion muss vor dem Aufruf definiert werden
- Anzahl und Typ der zu übergebenen Variablen muss identisch mit dem Funktionsprototyp sein
- Klammer bleibt leer, wenn keine Parameter übergeben werden
- Rückgabewert kann mit Hilfe eines Gleichheitszeichens einer Variablen zugewiesen werden
- Beispiel:
  - antwort = kubik (eingabe);

45

## Funktion aufrufen (Beispiel)

Rechenzentrum

```
* Beispiel für eine einfache Funktion */
#include <stdio.h>
int kubik (int x);
long eingabe, antwort;
int main()
{
    printf("Bitte geben Sie eine ganze Zahl ein.\n");
    scanf("%d",&eingabe);
    antwort=kubik(eingabe); ----> Funktionsaufruf
    printf("Die Kubikzahl von %d ist %d.\n",eingabe,antwort);
    return 0;
}
int kubik(int x)
{
    int x3; -----> Funktionsdefinition
    x3=x*x*x;
    return x3;
}
```

46

## Funktion (Beispiel)

Rechenzentrum

- Aufgabe:
  - Funktion schreiben, die zwei Integer-Zahlen übernimmt und das Produkt dieser beiden Zahlen zurück gibt.
- Vorgehensweise:
  1. Funktionskopf schreiben
  2. Funktionsrumpf mit return-Anweisung schreiben
  3. Funktionsaufruf
  4. Funktionsprototypen erstellen

47

## Funktion (Beispiel)

Rechenzentrum

- Aufgabe:
  - Funktion schreiben, die zwei Integer-Zahlen übernimmt und das Produkt dieser beiden Zahlen zurück gibt.
- 1. Funktionskopf schreiben
 

```
int produkt (int wert1, int wert2)
```

48

- Aufgabe:
  - Funktion schreiben, die zwei Integer-Zahlen übernimmt und das Produkt dieser beiden Zahlen zurück gibt.
- 2. Funktionsrumpf schreiben
 

```
int produkt (int wert1, int wert2)
{
    int produkt=0;
    produkt=wert1*wert2;
    return produkt;
}
```

49

- Aufgabe:
  - Funktion schreiben, die zwei Integer-Zahlen übernimmt und das Produkt dieser beiden Zahlen zurück gibt.
- 3. Funktionsaufruf schreiben
 

```
/* Beispielprogramm */
int main()
{
    ergebnis=produkt(wert1,wert2);
    return 0;
}
int produkt (int wert1, int wert2)
{
    int produkt=0;
    produkt=wert1*wert2;
    return produkt;
}
```

50

- Aufgabe:
  - Funktion schreiben, die zwei Integer-Zahlen übernimmt und das Produkt dieser beiden Zahlen zurück gibt.
- 4. Funktionsprototypen schreiben
 

```
int produkt (int wert1, int wert2);
int main()
{
    ergebnis=produkt(wert1,wert2);
    return 0;
}
int produkt (int wert1, int wert2)
{
    int produkt=0;
    produkt=wert1*wert2;
    return produkt;
}
```

51

- Aufgabenstellung wird in mehrere unabhängige Teilprobleme aufgeteilt
- Bestandteile
  - Funktionsprototyp:
    - rückgabetyt name (datentyp wert1, ...);
  - Funktionsaufruf
    - ergebnis = name(wert1,...);
  - Funktionsdefinition:
    - rückgabetyt name (datentyp wert1, ...)
 

```
{
                                Anweisungen
                                return wert2;
                            }
```

52

- 2 Arten von Variablen
  - globale Variablen
  - lokale Variablen

53

- werden außerhalb aller Funktionen (auch außerhalb von main) deklariert
- sind im ganzen Programm verfügbar
- können von allen Funktionen verwendet werden
- Anzahl der globalen Variablen sollte möglichst gering sein
- werden automatisch mit dem Wert 0 initialisiert

54

- werden innerhalb einer Funktion definiert
- können nur innerhalb der jeweiligen Funktion verwendet werden
- lokale Variablen in verschiedenen Funktionen sind unabhängig voneinander
- werden nicht automatisch vom Compiler mit 0 initialisiert
- ohne Initialisierung haben Sie einen unbestimmten Wert
- werden zu Beginn einer Funktion neu angelegt und beim Verlassen der Funktion wieder aus dem Speicher gelöscht
- Beispiel:
 

```
int kubik(int x)
{
    int x3;    -----> x3 ist lokal
    x3=x*x*x;
    return x3;
}
```

55

```
#include <stdio.h>
int kubik(int x);
long eingabe, antwort;
int main()
{
    int x3=99;
    printf("Wert von x3 vor Funktionsaufruf: %d\n",x3);
    printf("Bitte geben Sie eine ganze Zahl ein.\n");
    scanf("%d",&eingabe);
    antwort=kubik(eingabe);
    printf("Die Kubikzahl von %d ist %d.\n",eingabe,antwort);
    printf("Wert von x3 nach Funktionsaufruf: %d\n",x3);
    return 0;
}
int kubik(int x)
{
    int x3;
    x3=x*x*x;
    printf("Wert von x3 in Funktion: %d\n",x3);
    return x3;
}
```

56

## Lokale Variablen (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int kubik (int x);
long eingabe, antwort;
int main()
{
    int x3=99;
    printf("Wert von x3 vor Funktionsaufruf:
    %d\n",x3);
    printf("Bitte geben Sie eine ganze Zahl
    ein.\n");
    scanf("%d",&eingabe);
    antwort=kubik(eingabe);
    printf("Die Kubikzahl von %d ist
    %d.\n",eingabe,antwort);
    printf("Wert von x3 nach Funktionsaufruf:
    %d\n",x3);
    return 0;
}
int kubik(int x)
{
    int x3;
    x3=x*x*x;
    printf("Wert von x3 in Funktion: %d\n",x3);
    return x3;
}
```

- Programmausgabe  
Wert von x3 vor Funktionsaufruf: 99  
Bitte geben Sie eine ganze Zahl ein.  
4  
Wert von x3 in Funktion: 64  
Die Kubikzahl von 4 ist 64.  
Wert von x3 nach Funktionsaufruf: 99

57

## Lokale Variablen 2 (Beispiel)

Rechenzentrum

```
/* Initialisierung von Variablen */
#include <stdio.h>
int global;
int main()
{
    int lokal;
    printf("globale Variable hat den Anfangswert: %d\n",global);
    printf("lokale Variable hat den Anfangswert: %d\n",lokal);
    return 0;
}
```

58

## Lokale Variablen 2 (Beispiel)

Rechenzentrum

```
/* Initialisierung von Variablen */
#include <stdio.h>
int global;
int main()
{
    int lokal;
    printf("Anfangswert von global: %d\n",global);
    printf("Anfangswert von lokal: %d\n",lokal);
    return 0;
}
```

- Programmausgabe:  
Anfangswert von global: 0  
Anfangswert von lokal: 12081500

59

## Rekursive Funktionen

Rechenzentrum

- rekursive Funktionen rufen sich selbst auf
- Rekursion bezeichnet den eigentlichen Funktionsaufruf
- sinnvoll, wenn Sie zuerst Eingabedaten und dann in gleicher Weise das Ergebnis bearbeiten möchten
- Vorteil:
  - Code ist in der Regel leichter lesbar
- Nachteil:
  - Code arbeitet nicht mehr so effektiv
- direkte Rekursion:
  - Funktion ruft sich selber auf
- indirekte Rekursion:
  - Funktion 1 ruft Funktion 2 auf, die wieder Funktion 1 aufruft

60

## Rekursive Funktion: Beispiel

Rechenzentrum

- Summe der n-ten Zahl berechnen
- $\text{sum}(n) = 1 + 2 + \dots + n$
- Rekursionsformel:
  - $\text{sum}(n) = \text{sum}(n-1) + n$
- Summenfunktion
  - $\text{sum}(n) = 0$  für  $n=0$
  - $\text{sum}(n) = \text{sum}(n-1) + n$  für  $n > 0$

61

## Rekursive Funktion: Summenbildung

Rechenzentrum

- Beispiel: Summenbildung für  $n=3$
- $\text{sum}(3) = \text{sum}(2) + 3$ 

$$= \text{sum}(1) + 2 + 3$$

$$= \text{sum}(0) + 1 + 2 + 3$$

$$= 0 + 1 + 2 + 3$$

$$= 6$$

62

## Rekursion: Beispiel

Rechenzentrum

```
/* Bsp. fuer Summenbildung */
#include<stdio.h>
int sum,x;
int summe(int a);
int main()
{
    printf("Bitte geben Sie eine int-Zahl ein\n");
    scanf("%d",&x);
    sum=summe(x);
    printf("Die Summe von %d ist %d\n",x,sum);
    return 0;
}

int summe (int a)
{
    if (a == 0)
        return 0;
    else
    {
        a=a+summe(a-1);
        return a;
    }
}
```

63

## Rekursion (Beispiel)

Rechenzentrum

- Programmausgabe
  - Bitte geben Sie eine int-Zahl ein:
 

3

Die Summe von 3 ist 6
  - Bitte geben Sie eine int-Zahl ein:
 

5

Die Summe von 5 ist 15



## Übungsblatt 2

Aufgaben 5 bis 11