Christian-Albrechts-Universität zu Kiel

Rechenzentrum

Programmieren in C

Simone Knief – Rechenzentrum der CAU

knief@rz.uni-kiel.de

01.03.-03.03.2022

C A U
Christian-Albrechts-Universität zu Kiel
Rechenzentrum

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

C A U
Christian-Albrechts-Universität zu Kiel
Rechenzentrum

Datenfelder

- Was ist ein Datenfeld ?
- Wie definiert man ein- und mehrdimensionale Datenfelder ?
- Wie deklariert und initialisiert man Datenfelder ?

Datenfelder

- Aufgabe:
  - Programm zu schreiben, das Temperaturverlauf eines Jahres aufzeichnet
- bisherige Lösung
  - Definition von 365 Variablen (T1,T2, …, T365)
  - Mittelwert: (T1+T2+T3+…)/365
- wünschenswert
  - alle Werte in einer Variablen speichern und in Gesamtheit verarbeiten

  → realisierbar über Datenfelder

Felder

```
float w1, w2,w3, w4,w5,mittel=0.0;

w1=10.4;
w2=14.8;
w3=20.5;
w4=15.5;
w5=12.2;

mittel=(w1+w2+w3+w4+w5)/5;
```

Felder

```
float w1, w2,w3, w4;mittel=0.0;

w1=10.4;
w2=14.8;
w3=20.5;
w4=15.5;
w5=12.2;

mittel=(w1+w2+w3+w4+w5)/5;
```

```
float temp[5];mittel=0.0;

temp[0]=10.4;
temp[1]=14.8;
temp[2]=20.5;
temp[3]=15.5;
temp[4]=12.2;

mittel=(temp[0]+[temp[1]+temp[2]+temp
    [t3]+temp[4])/5;
```

Felder


```
float w1, w2,w3, w4;mittel=0.0;

w1=10.4;
w2=14.8;
w3=20.5;
w4=15.5;
m5=12.2;

mittel=(w1+w2+w3+w4+w5)/5;
```

```
float temp[5];mittel=0.0;
int i=0;
temp[0]=10.4;
temp[1]=14.8;
temp[2]=20.5;
temp[3]=15.5;
temp[4]=12.2;

mittel=(temp[0]+[temp[1]+temp[2]+temp
    [t3]+
    temp[4])/5;
for (i=0;i<5;i++)
    mittel=mittel+temp[i];
mittel=mittel/5;
```

Datenfelder

- Zusammenfassung von Speicherstellen für Daten **desselben** Datentyps unter einem Namen
- können sowohl Integer- als auch Gleitkommazahlen speichern
- werden auch als Arrays, Vektoren oder Reihungen bezeichnet
- Beispiel für eine zusammengesetzte Datenstruktur
- einzelne Speicherstellen werden als Elemente bezeichnet
- können ein- oder mehrdimensional sein

→ Erlauben systematische Verwendung einer Vielzahl von Variablen

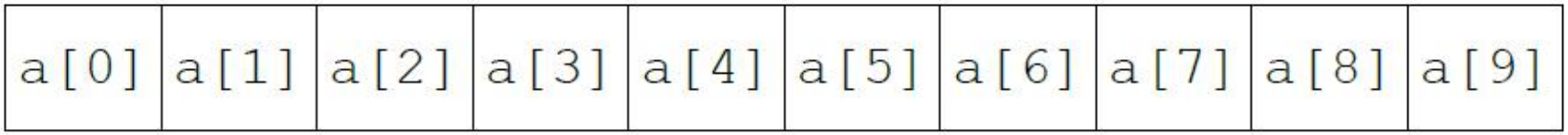
Eindimensionale Felder

- lineare Anordnung von Elementen im Speicher
- Elemente werden im Arbeitsspeicher als ein ganzer Block gespeichert

- “einfache” Variablen deklarieren
  - datentyp varname;
  - Bsp.: int xwert;

- Felder deklarieren:
  - allg. datentyp feldname [elementanzahl];
  - Bsp.: float ausgaben [12];
  - Feldgröße kann nach Deklaration nicht mehr verändert werden

- in C sind Felder immer nullbasiert, d.h. 1. Element ist feldname[0]



Eindimensionale Felder: Initialisierung

- **Felder initialisieren**:
  - allg. feldname[]={Element1, Element2, …, ElementN};
  - Beispiel:
    - int feldname[4] → alle Elemente haben unbekannten Inhalt
    - int feldname[4]={100, 200, 300, 400};
    - int feldname[]={100, 200, 300, 400};
    - int feldname[10]={0}; ---> alle Elemente haben den Wert 0
    - int feldname[10]={1,2,3};
      - ersten 3 Elemente sind 1,2 bzw.3; anderen Elemente automatisch 0
    - *int feldname[3]={12,34,55,54,33}; →Compilerfehler*

Auf Feld-Elemente zugreifen

- “normale“ Variablen: Zugriff über Variablenname

- Zugriff auf Feld-Elemente:
  - Variablenname + Position innerhalb des Feldes
- Positionsnummern werden in eckigen Klammern angegeben
- Indexnummern starten mit 0

Auf Feld-Elemente zugreifen

- 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]

Felder: Beispiel

```
/* Beispiel mit Variablen /
# include <stdio.h>
int main (void)
{
    float mittel=0.0;
    float t1=10.2,t2=5.0,t3=13.4;
    mittel=(t1+t2+t3) / 3;
    printf("Mittelwert: %f\n",mittel)
    return 0;
}
```

Felder: Beispiel

```
/* Beispiel mit Variaben */
# include <stdio.h>
int main (void) {
    float mittel=0.0;
    float t1=10.2,t2=5.0,t3=13.4;
    mittel=(t1+t2+t3) / 3;
    printf("Mittelwert: %f\n,mittel)
    return 0;
}
```

```
/* Beispiel mit Feldern */
# include <stdio.h>
int main (void){
    int i=0;
    float mittel=0.0;
    float temp[3]={10.2,5.0,13.4}
    for (i=0; i<3,i++)
     {
      mittel=mittel+temp[i];
      }
      mittel=mittel/ 3;
      printf("Mittelwert: %f\n",mittel)
      return 0;
}
```

Felder (Beispiel 2)

```
#include <stdio.h>
int main(void)
{
   float ausgaben[13];
   int count;
   for (count=1; count<=12;count++)
   {
      printf("Ausgaben fuer Monat %d eingeben.\n",count);
      scanf("%f",&ausgaben[count]);
   }
   for (count=1;count<=12;count++)
   {
      printf("Monat %d = %f Euro \n",count,ausgaben[count]);
   }
return 0;
}
```

Felder (Beispiel)

```
#include <stdio.h>
int main(void)
{
   float ausgaben[13];
   int count;
   for (count=1; count<=12;count++)
   {
      printf("Ausgaben fuer Monat %d
      eingeben.\n",count);
      scanf("%f",&ausgaben[count]);
   }
   for (count=1;count<=12;count++)
   {
      printf("Monat %d = %f Euro
      \n",count,ausgaben[count]);
   }
      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
```

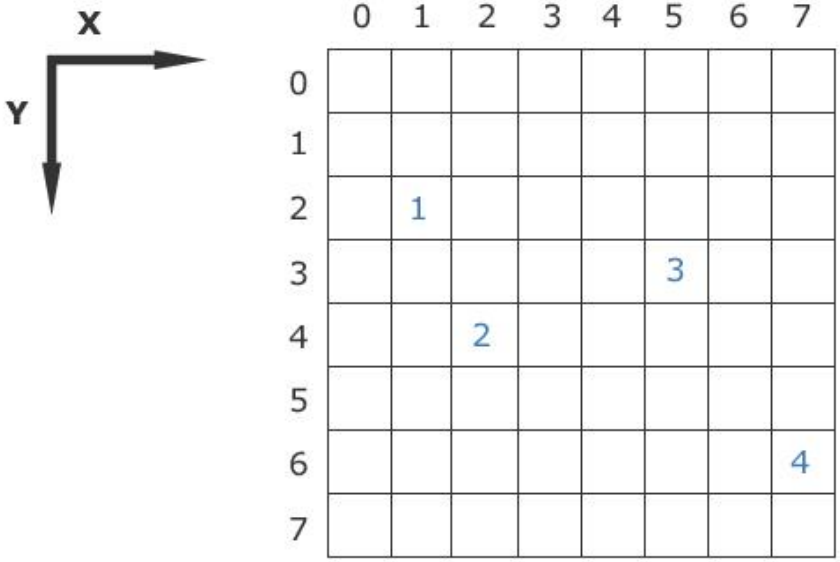
Mehrdimensionale Felder

- 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:

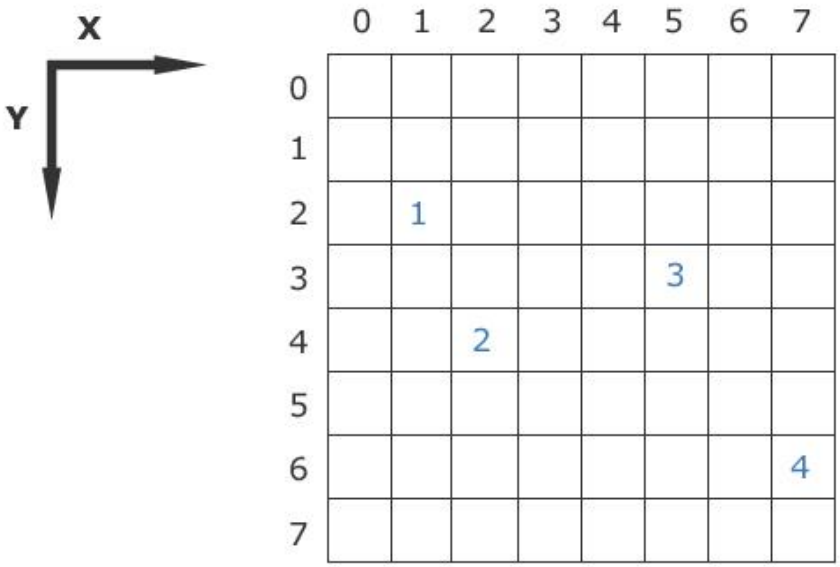
```
int matrix [4] [3] = {1,2,3,4,5,6,7,8,9,10,11,12};
int matrix [4] [3] = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};

matrix[0][0]=1
matrix[0][1]=2
matrix[0][2]=3
matrix[1][0]=4
matrix[1][1]=5
…
matrix[3][1]=11
matrix[3][2]=12
```

Felder: Beispiel Schachbrett



Felder: Beispiel Schachbrett




```
int i,j;
int brett[8][8]={0};
brett[2][1]=1;
brett[4][2]=2;
brett[3][5]=3;
brett[6][7]=4;
for (i=0;i<8;i++) // Schleife für y-Achse
{
    for (j=0;j<8;j++) // x-Achse
     printf("%d ", brett[i][j]);
     printf("\n");
}}
```

Felder: Beispiel Schachbrett

```
int i,j;
int brett[8][8]={0};
brett[2][1]=1;
brett[4][2]=2;
brett[3][5]=3;
brett[6][7]=4;
for (i=0;i<8;i++) // Schleife für y-Achse
{
    for (j=0;j<8;j++) // x-Achse
    printf("%d ", brett[i][j]);
printf("\n“);}}
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 3 0 0
0 0 2 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 4
0 0 0 0 0 0 0 0
```

Felder: Speicherbedarf ermitteln

- Speicherbedarf ermitteln mit sizeof-Operator (analog zu Variablen)

```
#include<stdio.h>
int main(void)
{int i;
int feld1[100],feld2[10000];
printf("Bytes int %d\n",sizeof(i));
printf("Bytes feld1 %d\n",sizeof(feld1));
printf("Bytes feld2 %d\n",sizeof(feld2));
return 0;
}
```

Felder: Speicherbedarf ermitteln

```
#include<stdio.h>
int main(void)
{int i;
int feld1[100],feld2[10000];
printf("Bytes int %d\n",sizeof(i));
printf("Bytes feld1 %d\n",sizeof(feld1));
printf("Bytes feld2 %d\n",sizeof(feld2));
return 0;
}
```

- Programmausgabe:

```
Bytes int 4
Bytes feld1 400
Bytes feld2 40000
```

Felder: Zusammenfassung

- fassen mehrere Datenelemente des gleichen Typs zusammen
- einzelne Feldelemente werden durch Indizes charakterisiert
- müssen wie Variabeln 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:

  int feldname[5];

  feldname [10]=3

  → Keine Fehlermeldung oder Warnung vom Compiler !

Zeichen und Strings

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

Einzelne Zeichen

- 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

Einzelne Zeichen: Beispiel

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

Einzelne Zeichen: Beispiel

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

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

  Bitte geben Sie einen Buchstaben ein

  D

  Sie haben den Buchstaben D eingegeben.

Strings

- 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

Strings: Verwendung

- 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);
  - bei scanf kein Adressoperator vor Stringname

- Ausgabe von Strings:
  - Formatbezeichner: %s
  - printf("%s",vorname);

Strings: Beispiel

```
/* Einlesen von Strings */
#include <stdio.h>
int main(void)
{
    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);
    return 0;

}
```

Strings: Beispiel

```
/* Einlesen von Strings */
#include <stdio.h>
int main(void)
{
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);
return 0;
}
```

- Programm übersetzen:
  - gcc –o stringc string.c
- Programm ausführen:
  - ./string
- Programmausgabe:

  Bitte geben Sie Ihren Vornamen ein
  Max
  Bitte geben Sie Ihren Nachnamen ein
  Mustermann
  Sie heißen Max Mustermann

Strings einlesen mit fgets

- scanf-Funktion liest nur bis zum nächsten Leerzeichen ein

- Beispiel: einlesen von Vor- und Nachnamen in einen String
- möglich mit der Funktion fgets
- allgemeine Form:
  fgets (stringname, zeichenanzahl, stdin);

- Beispiel:
  printf (“Bitte Vor- und Nachnamen eingeben \n“);
  fgets (name,30,stdin);
  printf (“der eingebene Name ist %s \n“,name);

- auch zum zeilenweisen Lesen aus Dateien geeignet

Einlesen: Problem Tastaturpuffer

- Beim Einlesen von Zeichen oder Strings treten u.U. Probleme auf, wenn sich noch Zeichen im Tastaturpuffer befinden

→ scanf-Anweisungen werden scheinbar übersprungen

→ Lösung: Tastaturpuffer nach jedem scanf löschen

→ Aufruf getchar-Funktion

Einlesen: Problem Tastaturpuffer

```
#include<stdio.h>
int main(void){
char a,b,c;
printf(“Zeichen 1 eingeben\n“);
scanf(“%c“,&a);
printf(“Zeichen 2 eingeben\n“);
scanf(“%c“,&b);
printf(“Zeichen 3eingeben\n“);
scanf(“%c“,&c);
printf(“eingegeben wurden %c,%c und%c\n“,a,b,c);
return 0; }
```

Einlesen: Problem Tastaturpuffer

```
#include<stdio.h>
int main(void){
char a,b,c;
printf("Zeichen 1 eingeben\n");
scanf("%c",&a);
printf("Zeichen 2 eingeben\n");
scanf("%c",&b);
printf("Zeichen 3eingeben\n");
scanf("%c",&c);
printf("eingegeben wurden %c,%c
    und%c\n",a,b,c);
return 0; }
```

→ Programm arbeitet nicht korrekt

```
Zeichen 1:
E
Zeichen 2: Zeichen 3: eingegebene
    Zeichen: 32766,0 und 0
```

Einlesen: Problem Tastaturpuffer

fehlerhafte Version

```
char a,b,c;
printf("Zeichen 1 eingeben\n");
scanf("%c",&a);
printf("Zeichen 2 eingeben\n");
scanf("%c",&b);
printf("Zeichen 3eingeben\n");
scanf("%c",&c);
printf("eingegeben wurden %c,%c
    und%c\n",a,b,c);
return 0; }
```

Korrekte Version

```
char a,b,c;
printf("Zeichen 1 eingeben\n");
scanf("%c",&a);
getchar();
printf("Zeichen 2 eingeben\n");
scanf("%c",&b);
getchar();
printf("Zeichen 3 eingeben\n");
scanf("%c",&c);
getchar();
printf("eingegeben wurden %c,%c
    und%c\n",a,b,c);
return 0}
```

Einlesen: Problem Tastaturpuffer

Korrekte Version

```
char a,b,c;
printf("Zeichen 1 eingeben\n");
scanf("%c",&a);
getchar();
printf("Zeichen 2 eingeben\n");
scanf("%c",&b);
getchar();
printf("Zeichen 3 eingeben\n");
scanf("%c",&c);
getchar();
printf("eingegeben wurden %c,%c
    und%c\n",a,b,c);
return 0}
```

Programmausgabe:

```
Zeichen 1 eingeben: s
Zeichen 2 eingeben: F
Zeichen 3 eingeben: h

eingegebene Zeichen: s, F und h
```

Stringfunktionen

- 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

Stringfunktion strcpy

- Kopiert einen String in einen anderen

- Beispiel:

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

Stringfunktion strcpy

```
#include <stdio.h>
#include <string.h>
    int main()
      {
      char person1 [20]="Arbeiter“;
      char person2 [20]= "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

Stringfunktion strcpy: Bsp. 2

- Stringinitialisierung erst nach Deklaration

- Beispiel:

```
#include <stdio.h>
int main()
{
    char vorname[20];
    vorname="Hans";
    printf("Vorname: %s ", vorname);
return 0;
}
```

→ Fehler bei Programmübersetzung

Stringfunktion strcpy: Bsp. 2

```
#include <stdio.h>
int main()
{
    char vorname[20];
    vorname="Hans";
    printf("Vorname: %s ",
    vorname);
return 0;
}
```

→ Fehler bei Programmübersetzung

```
#include <stdio.h>
#include <string.h>
int main()
{
    char vorname[20];
    strcpy(vorname,"Hans");
    printf("Vorname: %s ",
    vorname);
return 0;
}
```

Stringfunktion strcmp

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

Stringfunktion strcmp (Beispiel)

```
#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)

- Programmausgabe

  Text1 und Text2 sind unterschiedlich
  Text1 und Text3 sind identisch

Stringfunktion strcat

- 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 strcat (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;
}
```

- Programmausgabe:

  Hallo
  Programmierer
  Hallo Programmierer
  Programmierer

Stringfunktion strlen

- Funktion zur Ermittlung der Länge eines Strings
- Beispiel:

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

Stringfunktion strlen (Beispiel)

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

- Programmausgabe:

  Laenge: 35 Zeichen

Funktionen: Einführung (Bsp. 1)

- Aufgabe Programm zu schreiben, das Klimadaten auswertet

- Berechnungen, die ausgeführt werden sollen:
  - durchschnittliche Monatstemperatur
  - durchschnittliche Jahrestemperatur
  - durchschnittliche Niederschlagsmenge
  - durchschnittliche Anzahl von Sonnenstunden
  - ....

Funktionen: Einführung (Bsp. 1)

```
#include <stdio.h>
int main() {
...
Codezeilen Temperatur (Monat)
...
Codezeilen Temperatur (Jahr)
...
Codezeilen Niederschlag
...
Codezeilen Sonnenstd.
...
return 0; }
```

Funktionen: Einführung (Bsp. 1)

```
#include <stdio.h>
int main() {
...
Codezeilen Temperatur (Monat)
...
Codezeilen Temperatur (Jahr)
...
Codezeilen Niederschlag
...
Codezeilen Sonnenstd.
...
return 0; }
```

→ gleiche Art von Berechnung mit unterschiedlichen Daten

Funktionen: Einführung (Bsp. 1)

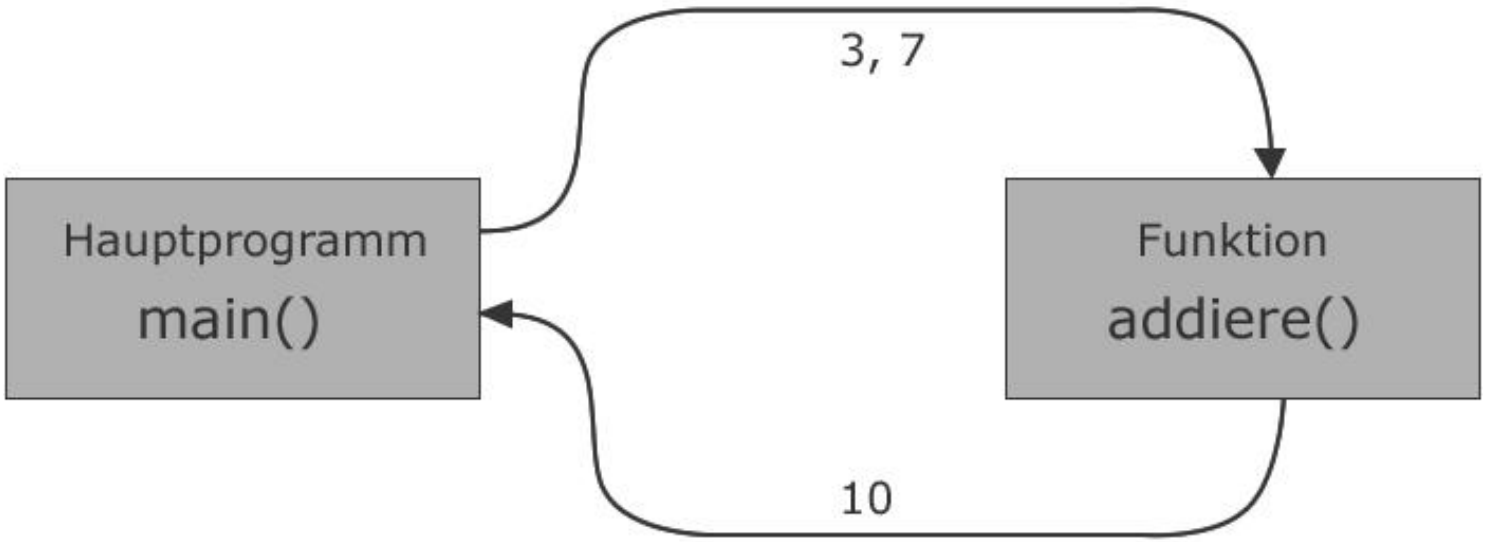
- Code ohne Funktion

```
#include <stdio.h>
int main(void) {
...
Codezeilen Temperatur (Monat)
...
Codezeilen Temperatur (Jahr)
...
Codezeilen Niederschlag
...
Codezeilen Sonnenstd.
...
return 0; }
```

- Code mit Funktion

```
#include <stdio.h>
int main(void) {
...
Mittelwertberechnung (Input Monat)
...
Mittelwertberechnung (Input Jahr)
...
Mittelwertbrechnung (Input Regen)
...
Mittelwertberechnung (Input Sonnenstd.)
...
return 0; }
```

Funktionen: Beispiel 1



Funktionen: Beispiel 2

- 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

Funktionen

- werden oft für wiederkehrende Aufgaben verwendet
- Problemstellung wird in kleinere Teilprobleme unterteilt
  - Unterprogramm zur Lösung von Teilproblemen

- dienen der besseren Strukturierung von Programmen
- jede Funktion hat einen eindeutigen Namen
- Funktionen sind unabhängig voneinander
- C-Programmbibliothek enthält bereits zahlreiche Funktionen (z.B. printf, scanf)
  → Bibliotheksfunkionen

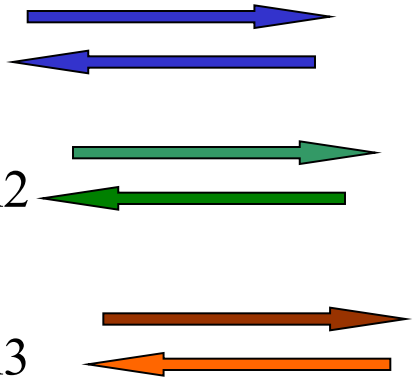
Vorteile von Funktionen

- 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

Funktionsweise einer Funktion

```
/* Funktionsweise einer Funktion */
#include <stdio.h>
int main(void)
{
….
Aufruf von Funktion1
....
Aufruf von Funktion2
…
…
Aufruf von Funktion3
…
return 0;

}
```



Funktion1

Funktion2

Funktion3

Funktion (Beispiel)

```
/* 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;
}
```

Funktionsdefinition

- Funktiondefinition besteht aus 2 Teilen
  - Funktionsheader
  - Funktionsrumpf

- allg. Form:

```
rückgabetyp funktionsname (Parameterdeklaration) ---> Funktionsheader
{
   Anweisungen ----> Funktionsrumpf
}
```

- Beispiel:

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

Funktionsheader

- Aufbau:
  - rückgabetyp Funktionsname (Parameterliste)
  - int kubik (int x)

- Rückgabetyp:
  - 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

Funktionsrumpf

- 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

Wert aus einer Funktion zurückgeben

- erfolgt mit der return-Anweisung
- allgemeine Form: return Ausdruck;
- z.B.: return ergebnis;
- gibt einen Wert an den Aufrufer zurück (beliebiger Datentyp)
- 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

- Beispiel:

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

return-Anweisung (Beispiel)

```
#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;
}
```

return-Anweisung (Beispiel)

```
#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
```

Funktionsprototypen

- vergleichbar mit einer Variablendeklaration
- dienen zur Deklaration von Funktionen vor dem Aufruf
- legen Eigenschaften der Funktion fest (Eingabeparameter und Rügabewert)
  - Bereitstellung von Speicherplatz
- müssen vor dem ersten Funktionsaufruf stehen (möglichst am Programmanfang)
- identisch mit dem Funktionskopf mit angehängtem Semikolon:
  allg.: rückgabetyp funktionsname (parameterliste);
  z.B.: int kubik (int x);
- Datentyp muss in der Parameterliste vorhanden sein, variablenname ist optional
- Compiler kann überprüfen ob Fkt. 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

Aufruf einer Funktion

- wird über den Funktionsnamen aufgerufen
- rückgabewert = funktionsname (var1, var2, ...);
- Anzahl und Typ der zu übergebenen Variablen muss identisch mit dem Funktionsprototypen sein
- Werden keine Parameter übergeben: (void) oder ()
- Rückgabewert kann mit Hilfe eines Gleichheitszeichens einer Variablen zugewiesen werden → Weiterverarbeitung
- Beispiel:
  - antwort = kubik (eingabe);

Funktion aufrufen (Beispiel)


```
#include <stdio.h>
int kubik (int x); → Funktionsprototyp
long eingabe, antwort;
int main(void)
{
    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;
}
```

Funktionssyntax: Zusammenfassung

- Funktionsprototyp
  - allg. : rückgabetyp funktionsname (parameterliste mit Datentypen);
  - z.B.: int kubik (int x);

- Funktionsdefinition
  - rückgabetyp funktionsname (parameterliste)

```
{
  /* Anweisungen */
  return wert;
}
```

Funktion (Beispiel)

- 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

Funktion (Beispiel)

- 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)

Funktion (Beispiel)

- 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;
}
```

Funktion (Beispiel)

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

- 3. Funktionsaufruf schreiben

```
int main(void)
{int ergebnis,wert1=5,wert2=10;
ergebnis=produkt(wert1,wert2);
    return 0;}
int produkt (int wert1, int wert2)
{
    int produkt=0;
    produkt=wert1*wert2;
    return produkt;}
```

Funktion (Beispiel)

- 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(void)
{ int ergebnis,wert1=5,wert2=10;
   ergebnis=produkt(wert1,wert2);
    return 0;
}
int produkt (int wert1, int wert2)
{
    int produkt=0;
    produkt=wert1*wert2;
    return produkt;
}
```

Funktion (Bsp. Produktbildung)

```
int produkt (int wert1, int wert2);
int main(void)
{ int ergebnis,wert1=5,wert2=10;
   ergebnis=produkt(wert1,wert2);
printf("ergebnis ist %d\n",ergebnis);
    return 0;
}
int produkt (int wert1, int wert2)
{
    int produkt=0;
    produkt=wert1*wert2;
    return produkt;
}
```

Funktion (Bsp. Produktbildung)

```
int produkt (int wert1, int wert2);
int main(void)
{ int ergebnis,wert1=5,wert2=10);
   ergebnis=produkt(wert1,wert2);
printf("ergebnis ist %d\n",ergebnis);
    return 0;
}
int produkt (int wert1, int wert2)
{
    int produkt=0;
    produkt=wert1*wert2;
    return produkt;
}
```

- Programmausgabe:

  ergebnis ist 10

Funktion (Bsp. Produktbildung)

```
int produkt (int wert1, int wert2);
int main(void)
{ int ergebnis; wert1=5,wert2=10;
int erg2,w3=4,w4=11;
   ergebnis=produkt(wert1,wert2);
printf("ergebnis ist %d\n",ergebnis);
erg2=produkt(w3,w4);
printf("erg2 ist %d\n",erg2);
    return 0;
}
int produkt (int wert1, int wert2)
{
    int produkt=0;
    produkt=wert1*wert2;
    return produkt;
}
```

- Programmausgabe:

  ergebnis ist 10

  erg2 ist 44

Funktionen mit oder ohne Prototypen schreiben

- **Funktion mit Prototypen** :

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

Funktionen mit und ohne Prototypen schreiben

- **Funktion mit Prototypen**

```
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;
}
```

- **Funktion ohne Prototypen**

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

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

Funktionen und Prototypen

- Funktionen mit Prototypen
  - Trennung von Deklaration und Definition
  - Deklaration vor main(), damit vor Funktionsaufruf
  - Defnition am Ende (nach main())

- Funktionen ohne Prototypen
  - Deklaration und Definition in einem Schritt *(analog zu Variablen)*
  - kompletter Funktionscode vor main()

Funktionen (Zusammenfassung)

- Aufgabenstellung wird in mehrere unabhängige Teilprobleme aufgeteilt

- Bestandteile
  - Funktionsprototyp:
    - rückgabetyp name (datentyp wert1, …);
  - Funktionsaufruf
    - ergebnis = name(wert1,…);
  - Funktionsdefinition:
    - rückgabetyp name (datentyp wert1, …)

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

- 2 Arten von Variablen
  - globale Variablen
  - lokale Variablen

Globale Variablen

- 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

Lokale Variablen

- werden innerhalb einer Funktion definiert
- können nur innerhalb der jeweiligen Funktion verwendet werden
- lokale Variablen in verschiedenen Funktionen sind unabhängig voneinander
- werden nicht immer automatisch vom Compiler mit 0 initialisiert
- haben unter Umständen 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;
}
```

Lokale Variablen (Beispiel)

```
#include <stdio.h>
int kubik (int x);
long eingabe, antwort;
int main(void)
{
    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;
}
```

Lokale Variablen (Beispiel)

```
#include <stdio.h>
int kubik (int x);
long eingabe, antwort;
int main(void)
{
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
```

Lokale Variablen 2 (Beispiel)

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

Lokale Variablen 2 (Beispiel)

```
/* Initialisierung von Variablen */
#include <stdio.h>
int global;
int main(void)
{
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

Statische Variablen

- Mischung aus lokalen und globalen Variablen
- werden auf einer festen Speicherstelle abgelegt
- sinnvoll innerhalb von Funktionen
- Definition:
  - static datentyp varname;
  - Beispiel: static int x = 0;

Statische Variablen (Beispiel)

```
#include <stdio.h>
int f1 ( );
int main (){
    int y;
    y=f1();
    printf("%i\n",y);
    y=f1();
    printf("%i\n",y);
    return 0; }
int f1( ) {
    int y=0;
    y=y+1
    return y;
```

Statische Variablen (Beispiel)

```
#include <stdio.h>
int f1 ( );
int main (){
    int y;
    y=f1();
    printf("%i\n",y);
    y=f1();
    printf("%i\n",y);
    return 0; }
int f1( ) {
    int y=0;
    y=y+1
    return y;
```

Programmausgabe:

1

1

Statische Variablen (Beispiel)

```
#include <stdio.h>
int f1 ( );
int main (){
    int y;
    y=f1();
    printf("%i\n",y);
    y=f1();
    printf("%i\n",y);
    return 0; }
int f1( ) {
    static int y=0;
    y=y+1
    return y;
```

Programmausgabe:

1
2

Rekursive Funktionen

- rekursive Funktionen rufen sich selbst auf
- Rekursion bezeichnet den eigentlichen Funktionsaufruf
- sinnvoll, wenn zuerst Eingabedaten verwendet werden und dann im nächsten Schritt die berechneten Daten in gleicher Weise weiter verarbeitet werden
- 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

Rekursive Funktion: Beispiel

- Summe der n-ten Zahl berechnen
- sum(n)=1+2+..+n

- Rekursionsformel:
  - sum(n)=sum(n-1)+n

- Summenfunkion
  - sum(n) = 0 für n=0
  - sum(n)=sum(n-1)+n für n >0

Rekursive Funktion: Summenbildung

- Beispiel: Summenbildung für n=3
- sum(3) = sum(2) +3
  = sum(1) + 2 + 3
  = sum(0) + 1 +2 +3
  = 0 + 1 + 2 + 3
  = 6

Rekursion: Beispiel Summenbildung

```
#include<stdio.h>
int sum,x;
int summe(int a);
int main()
{
printf("Bitte geben Sie ein 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;
    }
}
```

Rekursion (Beispiel Summenbildung)

- 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
```