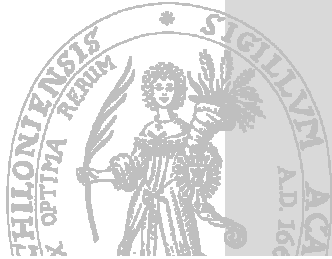


## Programmieren in C

Simone Knief – Rechenzentrum der CAU



### Gliederung

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

2

- Was ist ein Zeiger ?
- Wo setzt man Zeiger ein ?
- Wie deklariert und initialisiert man Zeiger ?
- Wie verwendet man Zeiger mit einfachen Variablen und Feldern ?
- Wie übergibt man Felder mithilfe von Zeigern an Funktionen ?

3

- wird auch als Pointer bezeichnet
- Zeigervariable enthält die Adresse einer anderen Variablen
- unabhängig vom Variablennamen kann auf den Inhalt der Variablen zugegriffen werden
- Zeiger zeigen auf einen bestimmten Speicherbereich
- mit einer Zeigervariable kann ein Objekt über die Speicheradresse angesprochen werden

4

## Zeiger (Anwendungsgebiete)

Rechenzentrum

- Speicherbereiche können dynamisch reserviert, verwaltet und gelöscht werden
- Datenobjekte können direkt an Funktionen übergeben werden (call-by-reference)
- Felder lassen sich als Argumente an andere Funktionen übergeben
- rekursive Datenstrukturen (Listen, Bäume, Stacks und Queues) können implementiert werden

5

## Zeiger (Nachteile)

Rechenzentrum

- am Anfang schwer zu verstehen
- Programmstruktur wird häufig sehr unübersichtlich
- Zeiger sind die häufigste Fehlerquelle in C
- Zeiger erfordern Disziplin vom Programmierer

6

## Zeiger (Einführung)

Rechenzentrum

- folgende Eigenschaften kennzeichnen eine Variable
  - Variablenname
  - Datentyp der Variablen
  - Wert der Variablen
  - Speicheradresse, an der der Wert gespeichert wird

7

## Zeiger: Deklaration

Rechenzentrum

- allgemeine Form  
Datentyp \*zeigername;
- Datentyp spezifiziert Typ der Variablen, auf die der Zeiger verweist
- \* Indirektionsoperator:
  - kennzeichnet eine Variable als Zeiger
- Beispiele:
  - char \*zgr1, \*zgr2; ---> Zeiger vom Typ char deklarieren
  - float \*wert, prozent; --> wert ist ein Zeiger, prozent eine Variable
- der Datentyp des Zeigers muss vom selben Datentyp wie der sein, auf den er zeigt

8

## Zeiger: Initialisierung

Rechenzentrum

- Zeiger erhält die konkrete Adresse einer Variablen
- allgemeine Form:  
zeigername = &variable;
- Zeiger wird mit der Adresse einer vorher bekannten Variablen initialisiert
- Zeiger besitzt als Wert die Adresse der Variablen, auf die er zeigt
- & = Adressoperator
- &variable gibt die Speicheradresse einer Variablen wieder
- Beispiele:
  - zrate= &zinsrate; --> zrate erhält als Wert die Adresse von zinsrate
  - zpx=&f[5] --> zpx erhält als Wert die Adresse des 5.Elementes von Feld f
- Zeiger weist immer auf die erste Speicheradresse der Variablen
- Null-Zeiger: zvariable=0;
- Praktizieren Sie sichere Programmierung: Zeiger immer initialisieren

9

## Zeiger (Beispiel)

Rechenzentrum

```
/* Inhalt und Adresse einer Variablen */
#include <stdio.h>
int main (void)
{
    int x=5;
    printf("Der Wert von x ist %d\n",x);
    printf("Die Speicheradresse von x ist %d\n",&x);

    return 0;
}
```

10

## Zeiger (Beispiel)

Rechenzentrum

```
/* Inhalt und Adresse einer Variablen */
#include <stdio.h>
int main (void)
{
    int x=5;
    printf("Der Wert von x ist %d\n",x);
    printf("Die Speicheradresse von x ist %d\n",&x);

    return 0;
}
```

- Programm übersetzen:
  - gcc -o zeiger1.exe zeiger1.c
- Programm ausführen:
  - ./zeiger1.exe
- Programmausgabe:  
Der Wert von x ist 5  
Die Speicheradresse von x ist 12211111

11

## Vergleich: Zeiger und Variable

Rechenzentrum

- Variable
  - variablenname
    - gibt den Wert einer Variablen aus
    - direkter Zugriff auf eine Variable
  - &variablenname
    - bezieht sich auf die Adresse einer Variablen

12

## Vergleich: Zeiger und Variable

Rechenzentrum

- Variable
  - variablenname
    - gibt den Wert einer Variablen aus
    - direkter Zugriff auf eine Variable
  - &variablenname
    - bezieht sich auf die Adresse einer Variablen
- Zeiger
  - zeigername
    - enthält die Adresse einer Variablen
  - \*zeigername
    - bezieht sich auf die Variable auf die der Zeiger verweist
    - indirekter Zugriff auf eine Variable

13

## Zeiger (Beispiel)

Rechenzentrum

```
/* Einfaches Zeiger-Beispiel */
#include<stdio.h>
int main()
{
    int var=1;
    int *zgr;
    zgr=&var;
    printf(" Variablenwert ausgeben:\n");
    printf("ueber direkten Zugriff: var = %d\n",var);
    printf("ueber indirektem Zugriff: var = %d\n", *zgr);
    printf("\n");
    printf("Adresse ausgeben:\n");
    printf("ueber Variable: %d\n",&var);
    printf("ueber Zeiger: %d\n",zgr);
    return 0;
}
```

14

## Zeiger (Beispiel)

Rechenzentrum

```
/* Einfaches Zeiger-Beispiel */
#include<stdio.h>
int main()
{
    int var=1;
    int *zgr;
    zgr=&var;
    printf(" Variablenwert ausgeben:\n");
    printf("ueber direkten Zugriff: var = %d\n",var);
    printf("ueber indirektem Zugriff: var = %d\n",
        *zgr);
    printf("\n");
    printf("Adresse ausgeben:\n");
    printf("ueber Variable: %d\n",&var);
    printf("ueber Zeiger: %d\n",zgr);
    return 0;
}
```

- Programm übersetzen
  - gcc -o zeiger2.exe zeiger2.c
- Programm ausführen
  - ./zeiger2.exe
- Programmausgabe:
  - Variablenwert ausgeben
  - ueber direkten Zugriff: var = 1
  - ueber indirekten Zugriff: var = 1
  - Adresse ausgeben:
  - ueber Variable: - 107374212
  - ueber Zeiger: - 1073744212

15

## Zeiger, Variablen und Adressen

Rechenzentrum

- 3 Dinge muss man beim Arbeiten mit Zeigern sicher auseinanderhalten können:
  - Variablenname und Variablenwert
    - int xwert = 123;
  - Zeigername
    - int \*zeigxwert;
  - die im Zeiger gespeicherte Adresse:
    - zeigwert=&xwert z.B. Speicherstelle 6627377
  - der Wert an der im Zeiger gespeicherten Adresse:
    - \*zeigwert → 123

16

- Feldname-Name als Zeiger
  - Feldname ohne eckige Klammern → Adresse des ersten Elementes
  - `int daten[10];`
  - `daten = &daten[0];`

17

- Anordnung von Feldelementen im Speicher
- werden in sequentieller Reihenfolge abgelegt
- Bereich der belegten Adressen hängt vom Datentyp ab
- Beispiele
  - `int x[3];`
    - `x[0]` Speicherplätze 1000 bis 1003
    - `x[1]` Speicherplätze 1004 bis 1007
    - `x[2]` Speicherplätze 1008 bis 1011
  - `double y[2];`
    - `y[0]` Speicherplätze 1240 bis 1247
    - `y[1]` Speicherplätze 1248 bis 1255

18

- `int x[3];`
  - `x[0]` Speicherplätze 1000 bis 1003
  - `x[1]` Speicherplätze 1004 bis 1007
  - `x[2]` Speicherplätze 1008 bis 1011
- `double y[2];`
  - `y[0]` Speicherplätze 1240 bis 1247
  - `y[1]` Speicherplätze 1248 bis 1255
- folgende Aussagen gelten:
  1. `x == 1000`
  2. `&x[0] == 1000`
  3. `&x[1] == 1004`
  4. `y == 1240`
  5. `&y[0] == 1240`
  6. `&y[1] == 1248`

19

- arithmetische Operationen und Vergleiche mit Zeigern durchführen:
  - Inkrementierung
  - Dekrementierung
  - Addition von Ganzzahlen
  - Subtraktion von Ganzzahlen
- beide Zeiger müssen auf Elemente des gleichen Typs zeigen

20

- Wert des Zeigers wird erhöht
- Wert wird immer um die Speichergröße seines Datentyps erhöht
- Zeiger zeigt automatisch auf das nächste Array-Element
- Beispiel:
  - `zgr_auf_int++`; -> Wert wird um 4 Bytes erhöht
  - `zgr_auf_double++`; -> Wert wird um 8 Bytes erhöht
- Regel gilt auch für Erhöhungen größer 1
  - z.B. `zgr_auf_int +=4`; -> Wert wird um 16 Bytes erhöht

21

```
/* Zeiger inkrementieren */
#include <stdio.h>
int main (void)
{
    int array[5]={100,222,333,4,5};
    int *zeiger, *zeiger2;
    zeiger=array;    → zeiger=&array[0];
    zeiger2=zeiger+1; → zeiger2=&array[1];
    printf("Speicheradressen ausgeben \n");
    printf("Inhalt von zeiger: %d \n",zeiger);
    printf("Inhalt von zeiger2: %d \n",zeiger2);
    printf("\n");
    printf("Werte ausgeben \n");
    printf("Wert von zeiger: %d \n",*zeiger);
    printf("Wert von zeiger2: %d \n",*zeiger2);
    return 0;
}
```

22

```
/* Zeiger inkrementieren */
#include <stdio.h>
int main (void)
{
    int array[5]={100,222,333,4,5};
    int *zeiger, *zeiger2;
    zeiger=array;
    zeiger2=zeiger+1;
    printf("Speicheradressen ausgeben \n");
    printf("Inhalt von zeiger: %d \n",zeiger);
    printf("Inhalt von zeiger2: %d \n",zeiger2);
    printf("\n");
    printf("Werte ausgeben \n");
    printf("Wert von zeiger: %d \n",*zeiger);
    printf("Wert von zeiger2: %d \n",*zeiger2);
    return 0;
}
```

- Programm übersetzen
  - `gcc -o ink.exe zinkrement.c`
- Programm ausführen
  - `./ink.exe`
- Programmausgabe
 

```
Speicheradressen ausgeben
Inhalt von zeiger: -1074573848
Inhalt von zeiger2: -1074573844

Werte ausgeben
Wert von zeiger: 100
Wert von zeiger2: 222
```

23

- Wert des Zeigers wird erniedrigt
- Wert wird immer um die Speichergröße seines Datentyps erniedrigt
- Zeiger zeigt automatisch auf das vorherige Array-Element
- Beispiel:
  - `zgr_auf_int--`; -> Wert wird um 4 Bytes verringert
  - `zgr_auf_double--`; -> Wert wird um 8 Bytes verringert

24

Zeiger dekrementieren (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main ()
{
    int array[5]={100,222,333,4,5};
    int *zeiger, *zeiger2;
    zeiger=&array[4];
    zeiger2=zeiger-1;
    printf("Speicheradressen ausgeben \n");
    printf("Inhalt von zeiger: %d \n",zeiger);
    printf("Inhalt von zeiger2: %d \n",zeiger2);
    printf("\n");
    printf("Werte ausgeben \n");
    printf("Wert von zeiger: %d \n",*zeiger);
    printf("Wert von zeiger2: %d \n",*zeiger2);
    return 0;
}
```

25

Zeiger dekrementieren (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main (void)
{
    int array[5]={100,222,333,4,5};
    int *zeiger, *zeiger2;
    zeiger=&array[4];
    zeiger2=zeiger-1;
    printf("Speicheradressen ausgeben \n");
    printf("Inhalt von zeiger: %d \n",zeiger);
    printf("Inhalt von zeiger2: %d \n",zeiger2);
    printf("\n");
    printf("Werte ausgeben \n");
    printf("Wert von zeiger: %d \n",*zeiger);
    printf("Wert von zeiger2: %d \n",*zeiger2);
    return 0;
}
```

- Programm übersetzen  
– gcc -o dek.exe zdekrement.c
- Programm ausführen  
– ./dek.exe
- Programmausgabe

**Speicheradressen ausgeben**  
Inhalt von zeiger: -1076212632  
Inhalt von zeiger2: -1076212636

**Werte ausgeben**  
Wert von zeiger: 5  
Wert von zeiger2: 4

26

Zeigerarithmetik (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main()
{
    int *intzgr,count;
    int intarray[10]={0,1,2,3,4,5,6,7,8,9};
    float *floatzgr;
    float floatarray[10]={.0,.1,.2,.3,.4,.5,.6,.7,.8,.9};
    intzgr=intarray; → intzgr=&intarray[0];
    floatzgr=floatarray; → floatzgr=&floatarray[0];
    for(count=0;count<10;count++)
    {
        printf("%d %f\n",*intzgr++,*floatzgr++); → *intzgr=*intzgr+1
    }
    return 0;
}
```

27

Zeigerarithmetik (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main()
{
    int *intzgr,count;
    int intarray[10]={0,1,2,3,4,5,6,7,8,9};
    float *floatzgr;
    float floatarray[10]={.0,.1,.2,.3,.4,.5,.6,.7,.8,.9};
    intzgr=intarray;
    floatzgr=floatarray;
    for(count=0;count<10;count++)
    {
        printf("%d %f\n",*intzgr++,*floatzgr++);
    }
    return 0;
}
```

- Programmausgabe

0 0.00000  
3 0.10000  
4 0.20000  
5 0.30000  
6 0.40000  
7 0.50000  
8 0.60000  
9 0.70000  
10 0.80000  
11 0.90000

28

## Zeigersubtraktion

Rechenzentrum

- Differenzbildung
  - Subtraktion liefert Abstand von Feldelementen
  - Beispiel: `zgr1 - zgr2` .

29

## Zeiger: Subtraktion

Rechenzentrum

```
#include <stdio.h>
int main()
{
    int *intzgr,count;
    int *zeiger2;
    int intarray[10]={0,1,2,3,4,5,6,7,8,9};
    intzgr=intarray;
    zeiger2=intzgr+2;
    printf("Adresse Element1: s %d \n",intzgr);
    printf("Adresse Element3: %d \n",zeiger2);
    printf("Differenz ist %d \n",zeiger2 - intzgr);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

30

## Zeiger: Subtraktion

Rechenzentrum

```
#include <stdio.h>
int main()
{
    int *intzgr,count;
    int *zeiger2;
    int intarray[10]={0,1,2,3,4,5,6,7,8,9};
    intzgr=intarray;
    zeiger2=intzgr+2;
    printf("Adresse Element1: %d \n",intzgr);
    printf("Adresse Element3: %d \n",zeiger2);
    printf("Differenz ist %d \n",zeiger2 -intzgr);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

- Programm übersetzen
  - `gcc -o sub.exe zsubtraktion.c`
- Programm ausführen
  - `./sub.exe`
- Programmausgabe
  - Adresse Element1: -1077452016
  - Adresse Element3: -1077452008
  - Differenz ist: 2

31

## Weitere Zeigermanipulationen

Rechenzentrum

- Zeigervergleiche
  - anwendbar auf: `=`, `!=`, `>`, `<`, `<=` und `>=`
  - Zeiger müssen auf dasselbe Feld zeigen
- Multiplikation und Division sind auf Zeiger nicht anwendbar

32



- Zuweisung
  - Einem Zeiger können Sie einen Wert (Adresse) zuweisen.
- Indirektion
  - Indirektionsoperator liefert Wert der Speicherstelle, auf die Zeiger verweist
- Adresse ermitteln
  - Mit dem Adressoperator lässt sich die Adresse des Zeigers ermitteln.
- Inkrementieren
  - Zahlen können addiert werden; Zeiger verweist auf andere Speicherposition
- Dekrementieren
  - Subtraktion von Zahlen; Zeiger verweist auf eine andere Speicherposition
- Differenzbildung
  - zwei Zeiger können Sie voneinander subtrahieren; feststellen, wie weit die Elemente voneinander entfernt sind
- Vergleich
  - nur möglich bei Zeigern, die auf das gleiche Feld zeigen

- Verwenden Sie in Ihrem Programm nie nicht initialisierte Zeiger
  - Beispiel:
    - `int *zgr;`
    - `*zgr=12;`

34

- Feldname ohne eckige Klammern ist Zeiger auf das 1. Feldelement
- über Indirektionsoperator kann auf die einzelnen Feldelemente zugegriffen werden
- Beispiel:
 

```
int feldname[]={12,345,99,...}
*(array) == array[0] → 12
*(array +1) == array[1] → 345
*(array + 2) == array[2] → 99
...
*(array +n) == array[n]
```

35

- Feld kann nur über einen Zeiger an eine Funktion übergeben werden
- um ein Feld an eine Funktion zu übergeben sind zwei Angaben nötig
  - Zeiger, der auf das erste Element verweist
  - Größe des Feldes

36

```
#include <stdio.h>
int groesster(int x[], int y);
int main()
{
    int count=10;
    int
        intarray[10]={30,40,100,50,60,70,80,
            90,20,10};
    int gross;
    gross=groesster(intarray,10);
    printf("groesste wert ist %d \n",gross);
    return 0;
}
```

```
int groesster(int x[], int y)
{int count, max=x[0];
for (count=0;count<y;count++)
{
    if(x[count]>max)
        max=x[count];
}
return max;
}
```

*int x[] identisch mit int \*x*

37

- **call-by-value Verfahren**
  - beim Funktionsaufruf wird Kopie des Originalwertes angelegt
  - Funktion arbeitet nur mit Kopien
  - Wert wird in Funktion verändert → kein Einfluss auf Variable im Original
- **call-by-reference Verfahren**
  - kann nur mit Zeigern realisiert werden
  - beim Funktionsaufruf werden nur Adressen übergeben
  - Wert wird in der Funktion verändert → Wert verändert sich auch in main

38

```
#include <stdio.h>
int flaeche (int eingabe);
int main()
{
    int eingabe, ergebnis;
    eingabe=4;
    printf("Eingabe zu Beginn: %d \n",eingabe);
    ergebnis=flaeche(eingabe);
    printf("Quadratflaeche: %d\n",ergebnis);
    printf("Eingabe am Ende: %d\n",eingabe);
    return 0;
}
```

```
int flaeche (int input)
{
    int output;
    output=input*input;
    input=10;
    return output;
}
```

39

```
#include <stdio.h>
int flaeche (int eingabe);
int main()
{
    int eingabe=4, ergebnis;
    printf("Eingabe Beginn: %d \n",eingabe);
    ergebnis=flaeche(eingabe);
    printf("Quadratflaeche: %d\n",ergebnis);
    printf("Eingabe Ende: %d\n",eingabe);
    return 0;
}
```

```
int flaeche (int input)
{
    int output;
    output=input*input;
    input=10;
    return output;
}
```

→ Programmausgabe:  
Eingabe Beginn: 4  
Quadratfläche: 16  
Eingabe Ende: 4

40

```
#include <stdio.h>
int flaeche (int eingabe);
int main()
{ int eingabe=4, ergebnis;
printf("Eingabe Beginn: %d \n",eingabe);

ergebnis=flaeche(eingabe),
printf("Quadratflaeche: %d\n",ergebnis);
printf("Eingabe Ende: %d\n",eingabe);
return 0; }
int flaeche (int input)
{
output=input*input;
input=10;
return(output)}
```

```
#include <stdio.h>
int flaeche (int *zeingabe);
int main()
{int eingabe=4, ergebnis, *zeingabe;
printf("Eingabe Beginn: %d \n",eingabe);
zeingabe=&eingabe;
ergebnis=flaeche(zeingabe),
printf("Quadratflaeche: %d\n",ergebnis);
printf("Eingabe Ende: %d\n",eingabe);
return 0; }
int flaeche (int *zeingabe)
{ int output;
output=*zeingabe*(*zeingabe);
*zeingabe =10;
return(output);}
```

41

```
#include <stdio.h>
int flaeche (int *zeingabe);
int main()
{
int eingabe,=4 ergebnis, *zeingabe;
printf("Eingabe Beginn: %d \n",eingabe);
zeingabe=&eingabe;
ergebnis=flaeche(zeingabe),
printf("Quadratflaeche: %d\n",ergebnis);
printf("Eingabe Ende: %d\n",eingabe);
return 0;
}
int flaeche (int *zeingabe)
{ int output;
output=*zeingabe*(*zeingabe);
*zeingabe=10;
return(output);}
```

Programm übersetzen  
gcc -o zfmt.exe zfunktion3.c

Programm ausführen:  
./zfmt.exe

Programmausgabe:  
4  
Eingabe zu Beginn: 4  
Quadratfläche: 16  
Eingabe am Ende: 10

42

- Zeigervariable enthält die Adresse einer Variablen
- 2 Operatoren von Bedeutung
  - Adressoperator &
  - Indirektionsoperator \*
- Werte von Variablen ausgeben:
  - varname oder \*zvarname
- Speicheradresse einer Variablen ausgeben
  - &varname oder zvarname
- Arrayname ohne [] -> Zeiger auf das erste Element eines Feldes
- Zeiger ermöglichen die Übergabe von Feldern an Funktionen

43

## Übungsblatt 3

### Aufgaben 1 bis 4

44

## Arbeiten mit Dateien

Rechenzentrum

- Was ist ein Stream ?
- Welche Arten von Dateien gibt es in C ?
- Wie öffnet und schließt man eine Datei ?
- Wie liest und schreibt man in eine Datei ?

45

## Dateiarten

Rechenzentrum

- Datei
  - Datenobjekt,
    - Daten werden auf externen Datenträger (z.B. Festplatte) abgelegt
- Streams
  - einfache Datenströme mit denen Daten von der Quelle zum Ziel bewegt werden
  - Standardstreams in C:
    - Standardeingabe (stdin): Tastatur
    - Standardausgabe (stdout): Bildschirm
    - Standardfehler (stderr): Bildschirm

46

## Dateiarten

Rechenzentrum

- Text-Streams
  - reine Textdateien
  - bestehen aus einer Folge von Zeichen
  - Zeichen sind in Zeilen angeordnet
  - Zeile endet mit einem Zeilen-Ende-Zeichen (Newline)
  - sind mit einem Editor lesbar
- Binäre-Streams
  - sind mit Binärdateien verbunden
  - bestehen aus einer Folge von Bytes
  - Daten werden unverändert gelesen und geschrieben
  - haben eine eindeutige Beziehung zu einem externen Gerät

47

## Arbeiten mit Textdateien

Rechenzentrum

- Schreiben bzw. Lesen von Textdateien erfolgt in 4 Schritten:
  - Dateinamen deklarieren
  - Datei öffnen
  - Dateiinhalt lesen bzw. schreiben
  - Datei schließen

48

## Datei öffnen und schließen

Rechenzentrum

- Zugriff auf Dateien erfolgt über einen Zeiger
- Datei deklarieren und öffnen
 

```
#include <stdio.h>
FILE *datei1; ---> Dateizeiger
datei1 = fopen("dateiname", "modus");
```

  - Modi für Textdateien:
    - "r": Lesezugriff
    - "w": Schreibzugriff
    - "a": anfügender Schreibzugriff
  - für jede Datei einen Dateizeiger deklarieren
- Datei schließen:
 

```
fclose(dateiname);
```

49

## Ein- und Ausgabe mit Dateien

Rechenzentrum

- Arbeiten mit Standard-Streams
  - Einlesen mit der Funktion scanf
  - Ausgeben mit der Funktion printf
- Arbeiten mit Dateien
  - Ausgabefunktion:
    - allg. fprintf (datei1, "text");
    - z.B.: fprintf(datei1, "C-Programmierung macht Spass \n");
    - fprintf(datei1,"%d",xwert);
  - Einlesefunktion:
    - allg. fscanf(datei1,"%d", &variable);
    - z.B. fscanf(datei1, "%d",&xwert);

50

## Ausgabe in Dateien (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main()
{
    FILE *datei;
    datei=fopen("ausgabe.txt","w");

    fprintf(datei, "Hallo Programmierer !\n");

    fclose(datei);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

51

## Lesen aus Dateien (Beispiel)

Rechenzentrum

```
/*include <stdio.h>
int main()
{FILE *datei;
    int input[10],output[10];
    int icount;
    datei=fopen("eingabe.txt","r");
    for (icount=0;icount<10;icount++)
    {
        fscanf(datei,"%d", &input[icount]);
        printf("eingelesener Wert: %d\n",input[icount]);
        output[icount]=input[icount]+1000;
    }
    for(icount=0;icount<10;icount++)
        printf("output ist: %d\n",output[icount]);
    return 0;}
```

52

```
#include <stdio.h>
int main()
{
    int i=0;
    printf("unformatierte Ausgabe \n");
    for (i=0;i<=1000;i=i+100)

        printf("%d \n",i);

    printf("\n");
    printf("formatierte Ausgabe \n");
    for (i=0;i<=1000;i=i+100)

        printf("%10d \n",i);
}
```

53

```
#include <stdio.h>
int main()
{
    int i=0;
    printf("unformatierte Ausgabe \n");
    for (i=0;i<=1000;i=i+100)
        printf("%d \n",i);
    printf("\n");
    printf("formatierte Ausgabe \n");
    for (i=0;i<=1000;i=i+100)
        printf("%10d \n",i);
}
```

Programmausgabe:  
unformatierte Ausgabe  
0  
100  
...  
900  
1000  
formatierte Ausgabe  
0  
100  
...  
900  
1000

54

## Übungsblatt 3

### Aufgaben 5 und 6

55

- ist ähnlich einem Feld
- mehrere Variablen zu einem Thema werden unter einem neuen Namen zusammen gefasst
- Variablen können unterschiedliche Datentypen haben <-> Feld
- Variablen einer Struktur werden als Elemente bezeichnet
- jedes Element kann jederzeit angesprochen werden
- können wie normale Variablen behandelt werden
- sinnvoll, wenn Informationen der Gruppe bearbeitet werden sollen
- Beispiele:
  - Personendaten (Name, Vorname, Straße, Ort, Telefonnummer)
  - Koordinaten eines Punktes (x-, y- und z-Koordinaten)
  - Datum (Tag, Monat, Jahr)

56

## Definition einer Struktur

Rechenzentrum

- erfolgt mit dem Schlüsselwort struct
- allgemein:
 

```
struct structname {
    Datentyp strukvar1;
    Datentyp strukvar2;
    ...
    Datentyp strukvarn;
};
```
- Beispiel:
 

```
struct koord {
    int xwert;
    int ywert;
};
```

57

## Deklaration einer Struktur

Rechenzentrum

- Bereitstellung von Speicherplatz
- 2 Deklarationsarten
  - bei der Definition
 

```
struct koord {
    int xwert;
    int ywert;
} punkt1, punkt2;
```
  - unabhängig von der Definition
 

```
struct koord {
    int xwert;
    int ywert;
};
/* C-Code */
struct koord punkt1, punkt2;
```

58

## Zugriff auf Strukturelemente

Rechenzentrum

- Strukturelemente lassen sich wie normale Variablen verwenden
- auf jedes Element kann separat zugegriffen werden
- Zugriff erfolgt über den Strukturelement-Operator (.)
- wird auch als Punktoperator bezeichnet
- Punktoperator verbindet Strukturvariable mit einer Komponente
- Element ansprechen mit:
 

```
varname.strukvar
```

 z.B. punkt1.xwert; → xwert von Punkt1 wird verwendet

59

## Strukturen initialisieren

Rechenzentrum

- 2 Arten der Wertzuweisung:
  - jeder Variablen wird einzeln ein Wert zugewiesen
 

```
punkt1.xwert=2;
punkt1.ywert=4;
```
  - alle Werte in einem Schritt zuweisen
 

```
struct koord punkt1={2,4};
```

60

```
#include <stdio.h>
int main()
{
    struct koord{
        int xwert;
        int ywert;
    };
    struct koord punkt1={2,6};
    struct koord punkt2={11,15};
    printf("Punkt 1 hat die Koordinaten %d und %d \n",punkt1.xwert,punkt1.ywert);
    printf("Punkt 2. hat die Koordinaten %d und %d \n",punkt2.xwert,punkt2.ywert);
    printf("Programm erfolgreich beendet. \n");
    return 0;
}
```

61

```
#include <stdio.h>
int main()
{
    struct koord{
        int xwert;
        int ywert;
    };
    struct koord punkt1={2,6};
    struct koord punkt2={11,15};
    printf("Punkt 1 hat die Koordinaten %d und %d \n",punkt1.xwert,punkt1.ywert);
    printf("Punkt 2. hat die Koordinaten %d und %d \n",punkt2.xwert,punkt2.ywert);
    printf("Programm erfolgreich beendet. \n");
    return 0;
}
```

- Programmausgabe:

Punkt 1 hat die Koordinaten 2 und 6  
Punkt 2 hat die Koordinaten 11 und 15

62

- Informationen in einer Struktur können in ihrer Gesamtheit verändert werden
- Beispiel:
  - Koordinaten von Punkt 1 sollen nach Punkt2 kopiert werden
  - bei reiner Variablenbverwendung:
 

```
xwert2=xwert1;
ywert2=ywert1;
```
  - mit Strukturen:
 

```
punkt1=punkt2;
```

63

```
#include <stdio.h>
int main() {
    struct koord
    {
        int xwert;
        int ywert;
    };
    struct koord punkt1={2,6};
    struct koord punkt2={0,0};
    struct koord punkt3={0,0};
    punkt2=punkt1;
    punkt3.xwert=punkt1.xwert+100;
    printf("Pkt. 1: %d und %d \n",punkt1.xwert,punkt1.ywert);
    printf("Pkt. 2: %d und %d \n",punkt2.xwert,punkt2.ywert);
    printf("Pkt. 3: %d und %d \n",punkt3.xwert,punkt3.ywert);
    return 0; }
```

64



## Strukturvariablen kopieren (Beispiel)

Rechenzentrum

```

#include <stdio.h>
int main() {
    struct koord{
        int xwert;
        int ywert;
    };
    struct koord punkt1={2,6};
    struct koord punkt2={0,0};
    struct koord punkt3={0,0};

    punkt2=punkt1;
    punkt3.xwert=punkt1.xwert+100;

    printf("Pkt. 1: %d und %d\n",punkt1.xwert,punkt1.ywert);
    printf("Pkt. 2: %d und %d\n",punkt2.xwert,punkt2.ywert);
    printf("Pkt. 3: %d und %d\n",punkt3.xwert,punkt3.ywert);

    return 0; }
    
```

- Programm übersetzen:  
– gcc -o struc2.exe struc2.c
- Programm ausführen:  
– ./struc2.exe
- Programmausgabe  
Pkt. 1: 2 und 6  
Pkt. 2: 2 und 6  
Pkt 3: 102 und 0

65

## Strukturen in Strukturen

Rechenzentrum

- Struktur definieren
 

```

struct koord{
    int xwert;
    int ywert;
} obenlinks, untenrechts;

struct rechteck {
    struct koord obenlinks;
    struct koord untenrechts;
};
            
```

66

## Strukturen in Strukturen

Rechenzentrum

- Struktur definieren
 

```

struct koord{
    int xwert;
    int ywert;
} obenlinks, untenrechts;

struct rechteck {
    struct koord obenlinks;
    struct koord untenrechts;
};
            
```
- Instanzen deklarieren
 

```

struct rechteck objekt1;
            
```

67

## Strukturen in Strukturen

Rechenzentrum

- Struktur definieren
 

```

struct rechteck {
    struct koord obenlinks;
    struct koord untenrechts;
};
            
```
- Instanzen deklarieren
 

```

struct rechteck objekt1;
            
```
- Struktur definieren
 

```

struct rechteck {
    struct koord obenlinks;
    struct koord untenrechts;
} objekt1;
            
```

68

## Strukturen in Strukturen

Rechenzentrum

- Struktur definieren
 

```
struct rechteck {
    struct koord obenlinks;
    struct koord untenrechts;
};
```
- Instanzen deklarieren
 

```
struct rechteck objekt1;
```
- Koordinaten definieren: Pkt1(0,10) und Pkt2(100,200)
 

```
objekt1.obenlinks.xwert=0;
objekt1.obenlinks.ywert=10;
objekt1.untenrechts.xwert=100;
objekt1.untenrechts.ywert=200;
```

69

## Strukturen in Strukturen (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main()
{
    struct koord{
        int xwert;
        int ywert; };
    struct rechteck{
        struct koord obenlinks;
        struct koord untenrechts;
    } objekt1;
    printf ("Koordinaten für den Pkt. obenlinks eingeben\n");
    scanf ("%d %d",&objekt1.obenlinks.xwert,&objekt1.obenlinks.ywert);
    printf ("Koordinaten für den Pkt. untenrechts eingeben\n");
    scanf ("%d %d",&objekt1.untenrechts.xwert,&objekt1.untenrechts.ywert);
    printf ("Pkt 1: %d %d\n",objekt1.obenlinks.xwert, objekt1.obenlinks.ywert);
    printf ("Pkt 2: %d %d\n",objekt1.untenrechts.xwert, objekt1.untenrechts.ywert);
    return 0;
}
```

70

## Strukturen in Strukturen (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main()
{
    struct koord{
        int xwert;
        int ywert; };
    struct rechteck{
        struct koord obenlinks;
        struct koord untenrechts;
    } objekt1;
    printf ("Koordinaten für den Pkt. obenlinks eingeben\n");
    scanf ("%d %d",&objekt1.obenlinks.xwert,&objekt1.obenlinks.ywert);
    printf ("Koordinaten für den Pkt. untenrechts eingeben\n");
    scanf ("%d %d",&objekt1.untenrechts.xwert,&objekt1.untenrechts.ywert);
    printf ("Pkt 1: %d %d\n",objekt1.obenlinks.xwert, objekt1.obenlinks.ywert);
    printf ("Pkt 2: %d %d\n",objekt1.untenrechts.xwert, objekt1.untenrechts.ywert);
    return 0;
}
```

Programmausgabe

```
Koordinaten für den Pkt. obenlinks eingeben
2 10
Koordinaten für den Pkt. untenrechts eingeben
7 1
```

```
Pkt 1: 2 10
Pkt 2: 8 1
```

71

## Strukturen, die Felder enthalten

Rechenzentrum

- C-Strukturen können auch Felder enthalten
- Beispiel:
 

```
struct daten {
    char vname[30];
    char nname[30];
    int telefon;
};
```

72

## Strukturen, die Felder enthalten (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main()
{
    struct daten{
        char vname[30];
        char nname[30];
        int telefon;
    };
    struct daten p1={"Peter","Mueller", 12341};
    struct daten p2={"Max","Muster", 76312};
    printf("Teilnehmer:\n");
    printf("%s %s %d\n",p1.vname,p1.nname,p1.telefon);
    printf("%s %s %d\n",p2.vname,p2.nname,p2.telefon);

    return 0;
}
```

73

## Felder von Strukturen

Rechenzentrum

- Feldelemente sind keine Variablen, sondern Strukturen
- Struktur definieren
 

```
struct daten {
    char vname[30];
    char nname[30];
    int alter;
};
```
- Array deklarieren
 

```
struct daten liste [100]
```

74

## Felder von Strukturen

Rechenzentrum

```
#include <stdio.h>
int main()
{
    struct daten {
        char vname[30];
        char nname[30];
        char ort[30];
    };
    struct daten liste [100];
    for (i=0;i<2;i++)
        {printf("Vorname, Nachname und Telefonnummer eingeben\n");
        scanf("%s %s %d",liste[i].vname,liste[i].nname,&liste[i].telefon);}
    printf("Teilnehmer:\n");
    for (i=0;i<2;i++)
        printf("%s %s %s\n",liste[i].vname,liste[i].nname,liste[i].telefon);
    return 0;
}
```

75

## Felder von Strukturen

Rechenzentrum

```
#include <stdio.h>
int main()
{
    struct daten {
        char vname[30];
        char nname[30];
        char ort[30];
    };
    struct daten liste [100];
    for (i=0;i<2;i++)
        {printf("Vorname, Nachname und
        Telefonnummer eingeben\n");
        scanf("%s %s %d",&liste[i].vname,&liste[i].nname,&liste[i].telefon);}
    printf("Teilnehmer:\n");
    for (i=0;i<2;i++)
        printf("%s %s %s\n",liste[i].nname,liste[i].vname,liste[i].telefon);
    return 0;
}
```

- Programmausgabe  
Vorname, Nachname und Telefonnummer eingeben  
Peter Müller 7623  
...  
Müller Peter 7653  
Schmidt Klaus 44211  
Meier Konrad 4321

76

```
#include <stdio.h>
struct daten {
    float spende;
    char name[30];
} beitrage;
void ausgabe (struct daten x) ;
int main()
{
    printf("Bitte Namen eingeben \n");
    scanf("%s", beitrage.name);
    printf("Spendenhöhe eingeben \n");
    scanf ("%f",&beitrage.spende);
    ausgabe (beitrage);
    return 0;
}
```

```
void ausgabe( struct daten x)
{
    printf("Spender %s gab %f Euro\n",x.name,x.spende);
}
```

Programmausgabe:

```
Bitte Namen eingeben
Schmidt
Spendenhöhe eingeben
100
Spender Schmidt gab 100 Euro
```

## Übungsblatt 3