[Strukturen]

- Vereinbarung
- Verwendung
- Operationen mit *struct*
- Zeiger auf s*truct*
- Strukturen auf dem Heap
- Datenstrukturen gebildet mit *struct*
- *union*

[Strukturen] Vereinbarung

```
struct name
{
   typ komponente1;
   typ komponente2;
   ...
   typ komponenteN;
};
```

[Strukturen] Vereinbarung, Beispiel

```
/* Deklaration der Struktur student_s */

struct student_s
{
    char vorname[30];
    char nachname[30];
    int matrikelnummer;
    char wohnort[30];
    char strasse[30];
    int hausnummer;
};
```

```
/* Deklaration der Struktur student_s
    und Definition zweier Studenten*/

struct student_s
{
    char vorname[30];
    ....
} haensel, gretel;
```

```
/* oder so: */

struct student_s haensel, gretel;
```

[Strukturen] Vereinbarung, noch ein Beispiel

```
/* Deklaration der Struktur student_s
    sowie eines typs student_t
    (endlich die ersehnte sinnvolle
     Anwendung des typedef)*/

typedef struct student_s
{
    char vorname[30];
    char nachname[30];
    int matrikelnummer;
    char wohnort[30];
    char strasse[30];
    int hausnummer;
} student_t ;

int main()
{
    struct student_s haensel;
    student_t gretel;

    return 0;
}
```

[Strukturen] Vereinbarung, noch ein Beispiel

```
/* Deklaration der Struktur student_s
    sowie eines typs student_t
    (endlich die ersehnte sinnvolle
     Anwendung des typedef)*/

typedef struct student_s
{
    char vorname[30];
    char nachname[30];
    int matrikelnummer;
    char wohnort[30];
    char strasse[30];
    int hausnummer;
} student_t ;

int main()
{
    student_t gretel =
        {"Gretel","Grimm",12345,
         "Göttingen",
         "Gebrüder Grimm Allee",12 };

    return 0;
}
```

[Strukturen] Operationen

```
/*  Zugriff auf Komponenten der Struktur
    student_s */

student_t gretel, gretchen;

/* Zugriff auf einzelne Komponenten: Punkt-Operator */
scanf("%s",gretel.vorname);
scanf("%s",gretel.nachname);
scanf("%d",&gretel.matrikelnummer);

printf("Vorname: %s\n",gretel.vorname);
printf("Nachname: %s \n",gretel.nachname);
printf("Matrikelnummer: %d\n",gretel.matrikelnummer);

/* Zuweisung einer Struktur an eine andere ist auch möglich: */
gretchen = gretel;
```

[Strukturen] Schachtelung

```
typedef struct adresse_s
{
    char wohnort[30];
    int plz;
    char strasse[30];
    int hausnummer;

} adresse_t;
```

```
typedef struct student_s
{
    char vorname[30];
    char nachname[30];
    int matrikelnummer;
    adresse_t adresse;
} student_t ;
```

```
student_t beispielstudent;

...

(if beispielstudent.adresse.wohnort ==
    "Göttingen")
    ...

scanf("%d",&beispielstudent.adresse.plz);
```

[Strukturen] Zeiger auf Strukturen

```
funktion(student_t *stud_p)
{
    printf("Vorname:  %s\n",(*stud_p).vorname);
    printf("Nachname: %s\n",stud_p->nachname);
}


int main(){

    student_t gretel;
    gretel.vorname = "Gretel";
    gretel.nachname = "Grimm";
    funktion(&gretel);
    ...
}
```

[Strukturen] Modularisierung

```
#include <stdio.h>
#include "stud_struct.h"

void satz_lesen(student_t *const s)
{       printf("Nachname: ");
        scanf("%s", (*s).nachname);
        printf("Vorname: ");
        scanf("%s", s->vorname);
        adresse_lesen(&s->adresse);
}
void adresse_lesen(adresse_t *const a)
{       printf("Ort: ");
        scanf("%s", a->wohnort);
        printf("Plz: ");
        scanf("%d", &a->plz);
        printf("Strasse: ");
        scanf("%s", a->strasse);
        printf("Hausnummer: ");
        scanf("%d", &a->hausnummer);
}
void satz_schreiben(student_t *s)
{       printf("Nachname:  %s\n",s->nachname);
        printf("Vorname:   %s\n",s->vorname);
        printf("Ortname:   %s\n",s->adresse.wohnort);
        printf("PLZ:       %d\n",s->adresse.plz);
        printf("Strasse:   %s\n",s->adresse.strasse);
        printf("Hausnummer:%d\n",s->adresse.hausnummer);
}
```

```
/* stud_struct.h */

#ifndef STUD_STRUCT_H
#define STUD_STRUCT_H

typedef struct adresse_s
{
        char wohnort[30];
        int plz;
        char strasse[30];
        int hausnummer;
} adresse_t;

typedef struct student_s
{
        char vorname[30];
        char nachname[30];
        int matrikelnummer;
        adresse_t adresse;
} student_t ;


void satz_lesen(student_t *const s);
void satz_schreiben(student_t *const
s);
void adresse_lesen(adresse_t *const a);

#endif
```

[Strukturen] Modularisierung

```
#include <stdio.h>
#include "stud_struct.h"


int main()
{
        student_t s;
        printf("Datensatz eingeben:\n");
        satz_lesen(&s);
        printf("Datensatz eingegeben:\n");
        satz_schreiben(&s);


        return 0;
}
```

```
/* Makefile */

GCCFLAGS = -ansi -pedantic -Wall

struct3: struct3.o stud_struct.o
        gcc -o struct3 struct3.o stud_struct.o

struct3.o : struct3.c stud_struct.h
        gcc $(GCCFLAGS) -c struct3.c


stud_struct.o : stud_struct.c stud_struct.h
        gcc $(GCCFLAGS) -c stud_struct.c

stud_mem.o : stud_mem.c stud_mem.h
        gcc $(GCCFLAGS) -c stud_mem.c

clean:
        rm -f *~ struct*o struct3
```

[Strukturen] Strukturen auf dem Heap

```
#include <stdio.h>
#include <stdlib.h>
#include "stud_mem.h"

student_t* getstudmem(int number)
{
        student_t * s;
        s = (student_t*) malloc(number * sizeof(student_t));
        if (s == NULL)
                printf("Speicherallokation fehlgeschlagen.");
        return s;
}

void freestudmem(student_t *stud)
{
        free(stud);
}
```

```
/* stud_mem.h */

#include "stud_struct.h"


#ifndef STUD_STRUCT_H
#define STUD_STRUCT_H

student_t* getstudmem(int number);
void freestudmem(student_t *stud);

#endif
```

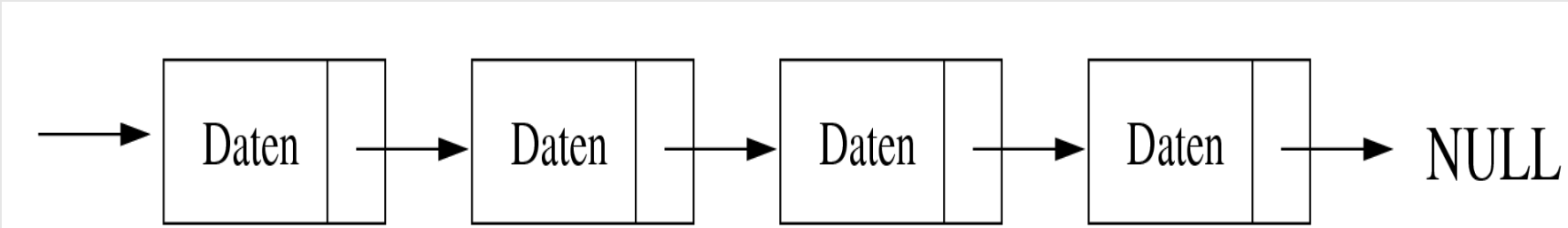
[Strukturen] Strukturen auf dem Heap

```
#include <stdio.h>
#include "stud_mem.h"
int main()
{
        student_t *stud_p;
        int anzahl_stud,i=0,j=0;
        printf("Weviele Saetze werden eingegeben: ");
        scanf("%d",&anzahl_stud);
        stud_p = getstudmem(anzahl_stud);
        if (stud_p == NULL)
                return 0;
        while (getchar() != '\n');        /*Zeilenpuffer leeren */
        while (        i < anzahl_stud )
        {
                char test;
                printf("Noch einen Datensatz eingeben (J/N)\n");
                test=getchar();
                if (test == 'J' || test == 'j')
                        satz_lesen(&stud_p[i++]);
                if (test == 'N' || test == 'n')
                        break;
                while (getchar() != '\n');   /*Zeilenpuffer leeren */
        }
        printf("Datensaetze eingegeben: %d\n",i);
        while( j < i)
                satz_schreiben(&stud_p[j++]);
        freestudmem(stud_p);
        return 0;
}
```

```
GCCFLAGS = -ansi -pedantic -Wall

struct3: struct3.o stud_struct.o
        gcc -o struct3 struct3.o stud_struct.o

struct3.o : struct3.c stud_struct.h
        gcc $(GCCFLAGS) -c struct3.c

struct4: struct4.o stud_struct.o stud_mem.o
        gcc -o struct4 struct4.o stud_struct.o \
        stud_mem.o

struct4.o : struct4.c stud_struct.h stud_mem.h
        gcc $(GCCFLAGS) -c struct4.c

stud_struct.o : stud_struct.c stud_struct.h
        gcc $(GCCFLAGS) -c stud_struct.c

stud_mem.o : stud_mem.c stud_mem.h
        gcc $(GCCFLAGS) -c stud_mem.c

clean:
        rm -f *~ struct*o struct3 struct4
```

[Strukturen] Datenstruktur: Einfach verkettete Liste

```
struct knoten
{
   DATENTYP daten;
   struct knoten *nachfolger;
};
```



- suchen und einfügen ist relativ aufwendig, da die Liste nur in eine Richtung durchlaufen werden kann
- Einfügen am Anfang der Liste ist schnell
- Operationen am Anfang und Ende der Liste sind Sonderfälle

[Strukturen] Datenstruktur: Einfach verkettete Liste

```
/* Funktionsprototypen */

/* z.B. eine sortierte, einfach verkettete Liste von Integern */

struct knoten* erzeugeKnoten(int wert);
int listeLeer(struct knoten *anfang);
struct knoten* elementEinfuegen(struct knoten *anfang, int wert);
void elementLoeschen(struct knoten *anfang, int wert);
void elementSuchen(struct knoten *anfang, int wert);

/* (und andere mehr) */
```

[Strukturen] Datenstruktur: Einfach verkettete Liste

```
#include <stdio.h>
#include <stdlib.h>

struct knoten
{       int daten;
        struct knoten *nachfolger;
};

struct knoten*      erzeugeKnoten           (int zahl);
void                loescheListe            (struct knoten *anfang);
int                 listeLeer               (struct knoten *anfang);
struct knoten*      elementEinfuegen        (struct knoten *anfang, int wert);
void                listeZeigen             (struct knoten *anfang);

int main()
{       int zahl;
        struct knoten *anfang = erzeugeKnoten(5);
        while (scanf("%d",&zahl) != EOF)
        {
                printf("Fuege ein: %d\n",zahl);
                anfang = elementEinfuegen(anfang,zahl);
                listeZeigen(anfang);
        }
        loescheListe(anfang);
        return 0;
}
```

[Strukturen] Datenstruktur: Einfach verkettete Liste

```
struct knoten*  elementEinfuegen(struct knoten *anfang, int zahl)
{       if (zahl < anfang->daten)
        {       struct knoten *knotenneu = erzeugeKnoten(zahl);
                knotenneu->nachfolger = anfang;
                return knotenneu;
        }
        else if (anfang->nachfolger != NULL)
                anfang->nachfolger=elementEinfuegen(anfang->nachfolger,zahl);
        else
        {       struct knoten *knotenneu = erzeugeKnoten(zahl);
                anfang->nachfolger = knotenneu;
                knotenneu->nachfolger = NULL;
        }
        return anfang;
}

void listeZeigen(struct knoten *anfang)
{       printf("Wert: %d\n",anfang->daten);
        if (anfang->nachfolger !=NULL)
                listeZeigen(anfang->nachfolger);
}

struct knoten* erzeugeKnoten(int zahl)
{       struct knoten *a;
        a = malloc(sizeof(struct knoten));
        if (a == NULL)
                printf("Speicherallokation fehlgeschlagen.");
        a->daten=zahl;
        return a;
}

void loescheListe(struct knoten *anfang)
{       if (anfang->nachfolger != NULL)
                loescheListe(anfang->nachfolger);
        printf("Loesche nun: %d\n",anfang->daten);
        free(anfang);
}
```

[Strukturen] UNION (Verbunde)

- Ein Verbund kann verschiedene Komponenten speichern, je nach Deklaration
- Tatsächlich speichert er aber nur einen Wert zur Zeit
- Die Größe eines Verbundes richtet sich dann nach der größten Komponente

```
union
{   int i;
    float f;
} testunion;

testunion.i = 3.0;
printf("%d\n",testunion.i);
testunion.f = 6.5f;
printf("%f\n",testunion.f);

/* Die Gefahr besteht allerdings, auf den Verbund in einer nicht
vorgesehenen Weise zuzugreifen: */
printf("%d\n",testunion.i);
```