Operator	Name	Stelligkeit	Assoziativität	Vorrang [1]
++, --	Postfix-Inkrement/Dekrement	unär	– [2]	1
. *Komponente*	Auswahl	unär	– [2]	1
-> *Komponente*	Inhalt mit Auswahl	unär	– [2]	1
[*Index*]	Indizierung	unär [3]	– [2]	1
(*Parameterliste*)	Methodenaufruf	unär [3]	– [2]	1
++, --	Präfix-Inkrement/Dekrement	unär	– [2]	2
+, -	Vorzeichen Plus/Minus	unär	– [2]	2
!	Logische Negation	unär	– [2]	2
~	Bitweise Invertierung	unär	– [2]	2
*	Inhalt	unär	– [2]	2
&	Adresse	unär	– [2]	2
(*Typ*)	Typanpassung	unär	– [2]	2
sizeof	Speicherplatzbedarf	unär	– [2]	2

Operator	Name	Stelligkeit	Assoziativität	Vorrang
*	Multiplikation	binär	links	3
/	Division	binär	links	3
%	Modulo	binär	links	3
+	Addition / Konkatenation	Binär	links	4
-	Subtraktion	binär	links	4
<<	Links-Shift	binär	links	5
>>	Rechts-Shift	binär	links	5
<	kleiner	binär	links	6
<=	kleiner-gleich	binär	links	6
>	größer	binär	links	6
>=	größer-gleich	binär	links	6
==	Gleichheit	binär	links	7
!=	Ungleichheit	Binär	links	7
&	bitweises Und	binär	links	8
^	bitweises XOR	binär	links	9
|	bitweises Oder	binär	links	10

Operator	Name	Stelligkeit	Assoziativität	Vorrang
&&	Logisches Und	binär	links	11
||	Logisches Oder	binär	links	12
? :	Bedingung	ternär	rechts	13
=	Zuweisung	binär	rechts	14
+=	Additions-Zuweisung	binär	rechts	14
-=	Subtraktions-Zuweisung	binär	rechts	14
*=	Multiplikations-Zuweisung	binär	rechts	14
/=	Divisions-Zuweisung	binär	rechts	14
%=	Modulo-Zuweisung	binär	rechts	14
&=	Bitweise-XOR-Zuweisung	binär	rechts	14
^=	Bitweise-Oder-Zuweisung	binär	rechts	14
|=	Bitweise-Und-Zuweisung	binär	rechts	14
<<=	Links-Shift-Zuweisung	binär	rechts	14
>>=	Rechts-Shift-Zuweisung	binär	rechts	14
,	Sequenz	binär	links	15

MAIN-Funktion:

Erläuterung zu den Parametern:

```
argv                    Feld von String-Pointern (argument vector)
argc + 1    Feldgröße (argument count)
argv[0]                 Programm-Name (Kommando)
argv[1]                 erstes Kommandozeilenargument
argv[argc -1]           letztes Kommandozeilenargument
argv[argc]  0(Null-Pointer)
```

Bsp.: `int main(int argc, char *argv[])`

MAKE:

- **explizite Regeln:**

```
hallo.o: hallo.c
    gcc -c hallo.c
```

(Ziel: hallo.o; abhängig von: hallo.c; Quelle; Kommando (muss mit Tabulator eingerückt sein): gcc -c hallo.c)

- **implizite Regeln:** beziehen sich auf Dateiendungen

```
.c.o:
    gcc -c $<
```

(Quelle etwas.c; Ziel etwas.o; $< ist die Quelle, auf die die Regel angewendet wird)

- **Musterregeln:** explizite Regeln mit Platzhalter % für beliebige Zeichenfolgen

```
%.o: %.c
    gcc -c $<
```

(Ziel etwas.o; Quelle etwas.c; $< ist die Quelle, auf die die Regel angewendet wird)

```
# Makefile
# Kommando-Variablen
CC = gcc
CFLAGS = -W -Wall -ansi -pedantic
CPPFLAGS = -I.
RM = rm -f

# Hilfsvariablen
TARGET = hallo
OBJECTS = gruss.o
SOURCES = $(TARGET).c $(OBJECTS:.o=.c)
HEADERS = $(OBJECTS:.o=.h)

# Musterregeln
%.o: %.c
   $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@

# Standardziele
.PHONY: all clean
all: $(TARGET)
clean:
   $(RM) $(TARGET) $(TARGET).o $(OBJECTS) depend
depend: $(SOURCES) $(HEADERS)
   $(CC) $(CPPFLAGS) -MM $(SOURCES) > $@

# Ziele zur Programmerstellung
$(TARGET): $(TARGET).o $(OBJECTS)
   $(CC) $(CFLAGS) $(LDFLAGS) $^ -o $@

# Abhaengigkeiten
include depend
```

(Variablen für die vordefinierten C-Übersetzungsregeln; Include-Dateien im aktuellen Verzeichnis suchen; die C-Übersetzungsregel ist vordefiniert und braucht deshalb nicht angegeben zu werden; Pseudoziele)

$^ == Alle Quellen zum aktuellen Ziel (alles nach :)
$< == Erste Quelle zum aktuellen Ziel (1. nach :)
$@ == Ziel auf das die Regel angewendet wird (vor :)

BIBLIOTHEK:

<stdlib.h>:

```
void* calloc(size_t n, size_t size);
void* malloc(size_t size);
void free(void* p);
double atof(const char* s);
int atoi(const char* s);
/* Zufallszahlengenerator */
int rand(void);
/* Anfangswert für Zufallszahlen */
void srand(unsigned int seed);
void qsort(void* p, size_t n, size_t size,
int (*cmp)(const void*, const void*));
```

<stdio.h>:

```
/* print error msg (errno) */
int printf(const char *format, ...);
void perror(const char *s);
FILE *fopen(const char *dateiname, char
*mode); /* mode = w o. r */
int fclose(FILE *fp);
int fgetc(FILE *fp); /* get next char */
/* write next char (and return it as int) */
int fputc(int c, FILE *fp);
/* get next n chars */
char *fgets(char *s, int n, FILE *fp);
/* write s to fp */
int fputs(const char *s, FILE *fp);
```

<string.h>:

```
char *strcpy(char *s1, const char *s2);
char *strcat(char *s1, const char *s2);
int strcmp(const char *s1, const char
*s2);
size_t strlen(const char *s);
int memcmp(const void cs, const void ct,
size_t n); /* Compare Memory*/
void *memcpy(void *p1, const void *p2,
size_t n); /* Copy Memory */
/* Initialize Memory with „c“ */
void *memset(void *p, int c, size_t n);
/* POSIX */
```

unistd.h:

```
int close(int fd);
ssize_t read(int fd, void *p, size_t n); /*
read n bytes from fd */
ssize_t write(int fd, const void *p, size_
n); /* write n bytes to fd */
```

<fcntl.h>:

```
int open(const char dateiname, int flags);
/* open file, return fd */
```

<errno.h>:

```
/* error number of the last POSIX function */
extern int errno;
```

<sys/stat.h>:

```
int stat(const char *dateiname, struct sta
*buf);
```

PARAMETER FÜR GCC:

- `-o` | Bestimmt den Namen der Ausgabedatei (Standard: „a.out“)
- `-c` | Der Quellcode wird nur kompiliert, aber noch nicht gelinkt (neue Endung oft „.o“)
- `-g` | Fügt Debugging-Symbole in die Binärdatei ein
- `-pedantic` | Weist auf nicht ISO konforme mit einer Warnung hin
- `-Wall` | Aktiviert sinnvolle Warnungen, die vom gcc-Compiler unterstützt werden.
- `-ansi` | Erlaubt nur ANSI C-konforme Konstrukte
- `-L` | Verzeichnisauswahl des Linkers (Suchpfad für Bibliothek)
- `-l` | Bibliothek die eingebunden werden soll
- `-I` | Verzeichnisauswahl des Preprocessor (Suchpfad für Header)

empfohlene Option: `-W -Wall -ansi -pedantic`

PRÄPROZESSORANWEISUNGEN:

```
#define KONSTANTE 25    /*Symbolische Konstante */
#define max(a, b) ((a) > (b) ? (a) : (b)) /* Makro */
int x = KONSTANTE;      /* x = 25 */
int y = max(x, 5);      /* max wird durch Makro ersetzt, y = x */
```

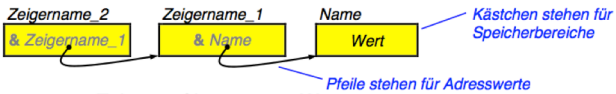
POINTER:

- **Variablen-Definition:** *Typ Name = Wert;*

  *Typ *Zeigername_1 = &Name;*

  *Typ **Zeigername_2 = &Zeigername_1;*

- Grafische Darstellung:



- konstanter Zeiger auf konstanten Wert:

  const *Typ* * **const** *Zeigername* = &*Name*;

ILP32 / LP64:

C TYP	ILP32 Bit/Byte	LP64 Bit/Byte
char	8 / 1	8 / 1
short	16 / 2	16 / 2
int	32 / 4	32 / 4
long	32 / 4	64 / 8
long long	64 / 8	64 / 8
pointer	32 / 4	64 / 8

Ausrichtung:
ILP32 – Summierte Byteanzahl durch 4 teilbar
LP64 – Summierte Byteanzahl durch 8 teilbar

FORMATANGABEN FÜR printf/scanf:

```
%s – Zeichenketten
%d – dezimal Format
%o – oktal Format
%x – hexadezimales Format
%f – Festkommaformat
%p – Pointerformat
%lu – “unsigned long“

'\n' – new Line
'\t' – Horizontaler Tab
```

WERKZEUGE:

- GCC – Compiler
- ddd – Debugging
- valgrind – Heapdebugger (Speicherfehler)
- astyle – Formatierung
- make – Automatisierung

FEHLERARTEN:

- **Absturz:** z.B. Speicherzugriffsfehler
- **Endlosschleife:** Programm scheint zu hängen, läuft aber weiter
- **Speicherüberlauf:** der ganze Rechner wird langsam, weil das Programm sämtlichen Speicher belegt.
- **Fehverhalten:** Programm tut nicht, was es soll, liefert z.B falsche Ergebnisse

ÜBERSETZUNGSEINHEIT:

```
/* hallo.c */
#include "gruss.h"
int main()
{
    gruessen();
    return 0;
}
```

```
/* gruss.h */
#ifndef GRUSS_H
#define GRUSS_H
void gruessen(void);
#endif
```

```
/* gruss.c */
#include "gruss.h"
#include <stdio.h>
void gruessen(void)
{
    printf("Hallo\n");
}
```

static Funktionen sind nur innerhalb der eigenen Implementierungs-Datei aufrufbar (private in Java)

PRÄPROZESSORANWEISUNGEN:
#ifndef == Prüft ob Header-Dateien vorhanden ist
#define == Definiert eine symbolische Konstante
#endif == Ende der if-Bedingung (Schließende Klammer zu ifndef)

- **Komponentenauswahl-Operatoren** (Punkt und Pfeil):

  Name.Komponente_1

  Zeigername->Komponente_1 — *Pfeil ist Kurzschreibweise für (*Zeigername).Komponente_2*

- Adresse einer Komponente:

  & Name . Komponente_1

  & Zeigername -> Komponente_1 — *Adresse der ersten Komponente ist Adresse der Struktur insgesamt*

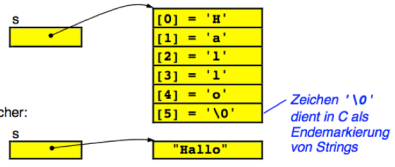
STRINGS:

Ein String ist ein Feld von Einzelzeichen mit '\0' als letztes Zeichen.
Strings werden über Zeiger-Variablen benutzt.

Variablen-Definition: const char **s = "Hallo"; /*** const da String-Literale nicht änderbar */
Wert: Anfangsadresse eines Strings (d.h. die Adresse seines ersten Zeichens)
Platzbedarf: sizeof "Hallo" = 6 /* Anzahl Zeichen inkl. '\0')
sizeof s = sizeof (char*)

strlen(s) = 5 /* ohne '\0' */

- Grafische Darstellung:



UNTERSCHIED: JAVA, C, C++

FEATURES	C	C++	Java
Paradigmen	Prozedural	Prozedural/OOP	OOP
Produkt	Ausführ. Maschinencode	Ausführ. Maschinencode	Java Bytecode
Speicherverwaltung	manuell	manuell	Garbage collector
Pointer	Ja	Ja	Nein
Stringtyp	Char Array	Char Array/ Objekt	Objekt
Datentypen	Struct, union	Struct, union, class	class
Vererbung	-	Mehrfach Vererbung	Einfach Vererbung

C++ SPEICHERVERWALTUNG:
In C++ wird Heap-Speicher mit dem Operator **new** allokiert und muss mit dem Operator **delete** wieder freigegeben werden:

```
Bsp.:       int *p = new int(1);
            std::cout << *p << '\n';
            delete p;
```

C++ OPERATOROVERLOADING:
C++ erlaubt das Überladen von Operatoren für benutzerdefinierte Typen
(wird unter anderem in der Ein-/Ausgabebibliothek verwendet)

Beispiel C++:

```
// Einbinden einer C++ Bibliothek
 #include <iostream>
 // Klasse
 class MeineZahl
 {
 public:
    MeineZahl(int wert);
    ~MeineZahl();
    bool operator>=(const MeineZahl m);
    int wert;
 private:
    // Private Attribute
 };
 // Konstruktor
 MeineZahl::MeineZahl(int val)
 {
    this->wert = val;
 }
 // Destruktor
 MeineZahl::~MeineZahl()
 {
    std::cout << "Objekt wird zerstört!\n";
 }
 // Operator-Overloading
 bool MeineZahl::operator>=(const MeineZahl m)
 {
    /* Vergleiche eigenen wert (im Objekt)
       mit Uebergebenem Wert */
    // m.wert, da m eine Stack-Variable
    // this->wert, da this ein Pointer
    if ((this->wert) >= (m.wert))
        return true; // ist groesser o. gleich
    else
        return false; // ist kleiner
 }
 int main(int argc, char const *argv[])
 {
    // lege x u. y als Objekte auf
    // Stack
    MeineZahl x = MeineZahl(5);
    MeineZahl y = MeineZahl(4);
    // das selbe wie if( x.operator>=(y) )
    if (x>=y)
        // Ausgabe auf Stream "out" im ns std
        std::cout << "X ist groesser gleich y\n";
    return 0;
 }
```

```
/* bin/notenspiegel.c */
#include "fachnote.h"
#include "liste.h"
#include <stdlib.h>
#include <stdio.h>
int main()
{
    fach_note *notenspiegel = NULL;
    fach_note *p;
    fach_note *q;
    /*-----------------------------------------
Notenspiegel einlesen */
    fprintf(stderr, "Faecher mit Noten eingeben (Ende mit
Strg-D):\n");
    for (;;)
    {
        p = (fach_note *) malloc(sizeof (fach_note));
        if (p == NULL)
        {
            fprintf(stderr, "Zu viele Faecher!\n");
            break;
        }

        if (! einlesen(p))
        {
            fprintf(stderr, "Eingabeende!\n");
            free(p);
            break;
        }
        notenspiegel = einfuegen(notenspiegel, p, &q);
        if (q != NULL)
        {
            fprintf(stderr, "Alte Eingabe
ueberschrieben!\n");
            free(q);
        }
    }
    /*-----------------------------------------
Notenspiegel ausgeben */
    printf("Notenspiegel:\n");

    schleife(notenspiegel, ausgeben);
    /*-----------------------------------------
Notenspiegel loeschen */
    while (notenspiegel != NULL)
    {
        notenspiegel = entfernen(notenspiegel, &p);
        free(p);
    }
    return 0;
}
```

```
/* lib/liste.c */
#include "liste.h"
#include <string.h>
fach_note* einfuegen(fach_note *head, fach_note *nfn,
fach_note **rfn)
{
    fach_note *i;
    i = head;
    *rfn = NULL;
    if (head == NULL)
    {
        return nfn;
    }
    while(i != NULL)
    {
        /* there already is a fach_note named like
this... */
        if (strcmp(i->name, nfn->name) == 0)
        {
            *rfn = i;
            if (i->prev != NULL)
            {
                i->prev->next = i->next;
            }
            if (i->next != NULL)
            {
                i->next->prev = i->prev;
            }
            if (i == head)
            {
                head = head->next;
            }
            break;
        }
        i = i->next;
    }
    nfn->prev = NULL;
    nfn->next = head;

    if (head != NULL)
    {
        head->prev = nfn;
    }

    return nfn;
}
void schleife(fach_note *sfn, void (*f)(const fach_note
*fn))
{
    /* Call function for every element of the list */
    for (; sfn != NULL; sfn = sfn->next)
    {
        f(sfn);
    }
}
fach_note* entfernen(fach_note *fn, fach_note **addr)
{
    fach_note *next;
    next = NULL;
    if (fn != NULL)
    {
        next = fn->next;
        if (next != NULL)
        {
            next->prev = NULL;
        }
        fn->next = NULL;
        fn->prev = NULL;
        *addr = fn;
    }
    return next;
}
```

```
# Makefile lib
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic -fpic
SOFLAGS=-shared -o $(LIBNAME) $(OBJECTS)
AR=ar rs
CPPFLAGS=-I.
LDFLAGS=
SOURCES=fachnote.c liste.c
HEADERS=$(SOURCES:.c=.h)
RM=rm -f

OBJECTS=$(SOURCES:.c=.o)

LIBFNAME=libaufgabe5
LIBTYPE=a
LIBNAME=$(LIBFNAME).$(LIBTYPE)

ALLOBJECTS=$(OBJECTS) $(LIBFNAME).a $(LIBFNAME).so
depend
.SUFFIXES: # disable imlicite rules
.PHONY: all clean

all: $(LIBNAME)

clean:
        $(RM) $(ALLOBJECTS)

depend: $(SOURCES) $(HEADERS)
        $(CC) $(CPPFLAGS) -MM $(SOURCES) > $@

$(LIBFNAME).a: $(OBJECTS)
        $(RM) $(LIBFNAME).so
        $(AR) $(LIBNAME) $^

$(LIBFNAME).so: $(OBJECTS)
        $(RM) $(LIBFNAME).a
        $(CC) $(SOFLAGS)

%.o: %.c
        $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) -c $<

include depend
```

```
/* lib/fachnote.c */
#include "fachnote.h"
#include <stdio.h>
#include <string.h>

static void ausgeben_unbenotet(const fach_note *fn)
{
    if (fn->grade.cgrade == 'B')
    {
        printf("bestanden\n");
    }
    else if (fn->grade.cgrade == 'N')
    {
        printf("nicht bestanden\n");
    }
    else {
        printf("Fehler: %c\n", fn->grade.cgrade);
    }
}
static void ausgeben_benotet(const fach_note *fn)
{
    if ((fn->grade.ngrade >= 10 && fn->grade.ngrade <= 40)
|| fn->grade.ngrade == 50)
    {
        printf("%d,%d\n", fn->grade.ngrade / 10, fn-
>grade.ngrade % 10);
    }
    else
    {
        printf("Fehler: %d\n", fn->grade.ngrade);
    }
}
/* reads in a fach_note and saves it to *fn */
int einlesen(fach_note *fn)
{
    char *undrln;
    /* read name */
    if(scanf("%20s", fn->name) < 1)
    {
        return 0;
    }
    /* Remove underlines */
    while((undrln = strchr(fn->name, (int) '_')) != NULL)
    {
        *undrln = ' ';
    }
    /* initialize prev and next */
    fn->next = NULL;
    fn->prev = NULL;

    /* read grade */
    if(scanf("%d", &fn->grade.ngrade) > 0)
    {
        fn->gr_o_ungr = GRADED;
        fn->ausgeben = ausgeben_benotet;
        return 1;
    }
    else if (scanf("%c", &fn->grade.cgrade) > 0)
    {
        fn->gr_o_ungr = UNGRADED;
        fn->ausgeben = ausgeben_unbenotet;
        return 2;
    }
    else
    {
        return 0;
    }
}
void ausgeben(const fach_note *fn)
{
    printf("%-*s\t", MAXFNLEN, fn->name);
    fn->ausgeben(fn);
}
```

```
/* lib/fachnote.h */
/* strcuture for the linked list */
#ifndef FACHNOTE_H
#define FACHNOTE_H
#define MAXFNLEN 20
enum graded_type { GRADED, UNGRADED };
struct fach_note
{
    char name[MAXFNLEN+1];
    void (*ausgeben)(const struct fach_note *fn);
    struct fach_note *next;
    struct fach_note *prev;

    enum graded_type gr_o_ungr;
    union
    {
        int ngrade;
        char cgrade;
    } grade;
};
/* alias name */
typedef struct fach_note fach_note;

void ausgeben(const fach_note *fn);

int einlesen(fach_note *fn);
#endif
```

```
/* lib/liste.h */
#ifndef LISTE_H
#define LISTE_H
#include "fachnote.h"

fach_note* einfuegen(fach_note *head, fach_note *nfn,
fach_note **rfn);

void schleife(fach_note *head, void (*f)(const fach_note
*fn));

fach_note* entfernen(fach_note *fn, fach_note **addr);
#endif
```