

Einführung in die Programmiersprachen C, C++

Prof. Dr. Ulf Rehmann, Fakultät für Mathematik

Übungsblatt 1 (8 Seiten)

Diese Blätter enthalten Beispielprogramme und leichte Übungsaufgaben, anhand derer die Sprachen C und C++ kennengelernt werden sollen. Sie sind kein vollständiges Skript für den Kurs. Ausführliche Erläuterungen werden in der Vorlesung gegeben. Programmieranleitungen beziehen sich auf das Betriebssystem UNIX. Sie sind gegebenenfalls abzuändern.

Die Programme zum Kurs und diese Übungsblätter sowie weitere Informationen sind zu finden unter der WWW-Adresse

<http://www.mathematik.uni-bielefeld.de/~rehmann/CC++2000/>

Literatur:

- [K&R] Brian W. Kernighan, Dennis M. Ritchie,
The C Programming Language,
based on Draft-Proposed ANSI C
(2nd Edition, Prentice Hall, ISBN 0-13-110362-8).
[S] Bjarne Stroustrup, Die C++ Programmiersprache, 3. Auflage,
Addison-Wesley (1997)
englische Version:
Bjarne Stroustrup, The C++ Programming Language, 3. ed.,
Addison-Wesley (1997)

Das erste C-Programm mit Namen `welt.c`:

```
/* Name: Welt.c; diese Zeile ist ein Kommentar */

#include <stdio.h>
main() {
    printf("Hallo, Welt!\n");
}
```

Dies Programm wird übersetzt mit dem Befehl `gcc welt.c`. Hierbei ist `gcc` der Name des Compilers (ggf. ersetzen!). Der Compiler erstellt dann das "ausführbare" Programm mit Namen (unter UNIX) `a.out`. Dies kann man durch Eingeben von `./a.out` ausführen lassen. Das Programm bewirkt die Bildschirmausgabe des Textes `Hallo, Welt!`.

Aufgabe 1.1: Ändern Sie das Programm so ab, dass bei der Ausgabe die Wörter `Hallo`, und `Welt!` in zwei aufeinander folgenden Zeilen ausgegeben werden. Experimentieren Sie mit anderen Ausgabeformen.

Ausdrücke, Anweisungen:

Ein C- oder C++ -Programm besteht aus *Ausdrücken (Expressions)* und *Anweisungen (Statements)*.

Ein Ausdruck ist "alles, was einen Zahlenwert hat": z. B. $(3 + 4) * 5$, $7 - 2$, $a + b$, im letzteren Fall sind `a`, `b` selbst Ausdrücke, der Wert von $a + b$ ist dann die Summe der Werte von `a`, `b`. Eine Anweisung ist ein Befehl an den Rechner, etwas zu tun. Z. B. ist

```
printf("Hallo, Welt!\n");
```

der Befehl, die Zeichenfolge `Hallo, Welt!` auf den Bildschirm zu schreiben und anschließend die Schreibmarke an den Anfang der nächsten Zeile zu positionieren (ein "newline" auszugeben).

```
int a, c;
```

ist die Anweisung, Variable namens `a`, `c` vom Typ `int`, d. h. vom ganzzahligen Typ zu deklarieren.

```
c = a + 3;
```

ist die Anweisung, zum Wert von `a` den Wert 3 zu addieren und das Ergebnis der Variablen `c` zuzuweisen.

Eine Folge von Anweisungen kann durch ein Paar `{ }` geschweifter Klammern zu einer (zusammengesetzten) Anweisung (compound statement) gebündelt werden.

Hinweis: Compiliert man mit dem Befehl `gcc -v datei.c`, so erhält man wichtige Hinweise zur Compilerfunktion. Weitere Informationen bekommt man aus dem Manual mit dem Befehl `man gcc`.

Deklarationen, Definitionen:

Objekte wie Konstante, Variable und Funktionen (s.u.) müssen benannt und definiert werden. Der Name ist im wesentlichen frei wählbar; sogenannte "Schlüsselwörter" von C (wie `main`, `if`, `else`, `while`, `for`, usw.) sind zu vermeiden. Z. B. kann eine `double`-Variable namens `xyz` durch eine Anweisung

```
double xyz;
```

definiert werden: Damit wird sowohl der Name `xyz` verabredet (Deklaration), wie auch Speicherplatz bereitgestellt, in dem der jeweilige "Wert" codiert wird (Definition). Die Variable (oder Konstante) ist nach der Deklaration für den Rest der zusammengesetzten Anweisung (also bis zur nächsten Klammer `}`, oder, falls die Deklaration außerhalb jeder zusammengesetzten Anweisung geschah, für den Rest der Programmdatei bekannt. (Allerdings gelten gewisse sogenannte Sichtbarkeits- oder Scope-Regeln, die später besprochen werden.) Durch eine Zuweisung

```
xyz = 3.14159;
```

wird dann in diesen Speicherplatz die `double`-Codierung für die Zahl 3.14159 eingetragen. Groß- und Kleinschreibung wird unterschieden, die Definition `double xyz`, `Xyz`, `xYz`; definiert drei verschiedene Variablen vom Typ `double`.

Programme in C:

Ein vollständiges C-Programm ist eine Folge von Anweisungen und enthält immer einen Abschnitt von der Bauart

```
main() {
    Folge von Anweisungen
}
```

Aufgabe 1.2: Welche Ausdrücke und Anweisungen enthält das folgenden Programm ?

```
/* euro.c */
/* Umwandlung von Euro in DM und umgekehrt */
#include <stdio.h> /* Präprozessor-Anweisung */
main() {
    const float faktor = 1.95583; /* Deklaration u. Initialisierung */
    float x, Euro, DM; /* Deklaration */
    char ch; /* Deklaration */
    printf("Bitte e (EURO) oder d (DM) gefolgt von Betrag eingeben: ");
    /* Ausgabe */
    scanf("%c%f",&ch,&x); /* Eingabe nach ch und x */
    if (ch == 'e') { /* Falls ch gleich dem Buchst. e */
        Euro = x; /* kopiere x nach Euro */
        DM = x*faktor; /* mult. x mit faktor, Erg. nach DM */
    }
    else { /* andernfalls (ch ungleich e) */
        Euro = x/faktor; /* teile x durch faktor, Erg. n. Euro */
        DM = x; /* kopiere x nach DM */
    }
    printf("%4.2f EURO = %4.2f DM\n", Euro, DM); /* Ausgabe ... */
}
```

Elementare Programmfluss-Konstrukte:**if-Konstrukt, if-else-Konstrukt:**

Syntax:	Semantik:
if (Ausdruck) Anweisung	Ausdruck wird ausgewertet, falls \neq 0, wird Anweisung ausgeführt, sonst nicht
if (Ausdruck) Anweisung 1	Ausdruck wird ausgewertet, falls \neq 0, wird Anweisung 1 ausgeführt
else Anweisung 2	andernfalls wird Anweisung 2 ausgeführt

Beispiel: siehe oben. Die Anweisungen sind in der Regel “zusammengesetzte Anweisungen”.

*Aufgabe 1.3: Verändern Sie das Programm `inch.c` derart, dass für den Fall der Eingabe eines von *i* und *c* verschiedenen Buchstabens die Ausgabe `0 in = 0 cm` erfolgt.*

while-Konstrukt, while-Schleife:

Syntax:	Semantik:
while (Ausdruck) Anweisung	Ausdruck wird ausgewertet, falls \neq 0, wird Anweisung ausgeführt, und das Konstrukt wird erneut durchlaufen; falls Ausdruck = 0, wird Anweisung übersprungen

Das folgende Programm benutzt ein **while**-Konstrukt, um eine Tabelle von Geschwindigkeiten (m/sec – km/std) zu erstellen.

```
/* Name:  ms-kmh.c
   rechnet Geschwindigkeiten um   */
```

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

```
main() {
float msec, kmh;
msec = 0;
while ( msec <= 120 ) {
    kmh = msec*3.6;
    printf("%6.2f  m/sec = %6.2f km/h\n", msec, kmh);
    msec = msec + 20;
}
}
```

for-Konstrukt, for-Schleife

Syntax:	Semantik:
for (Anw 1; Ausdr; Anw 2) Anweisung 3	Schritt 1: Anw 1 wird ausgeführt. Schritt 2: Ausdr wird ausgewertet, falls Ausdr \neq 0, wird Anweisung 3 ausgeführt, danach wird Anw 2 ausgeführt und Schritt 2 wird wiederholt, falls Ausdr = 0, wird ans Ende des Konstrukts gegangen

Beispiel: Das folgende Programm tut das gleiche wie das vorige, jedoch mit einer **for**-Schleife anstelle der **while**-Schleife:

```
/* Name:  ms-kmh-for.c
   rechnet Geschwindigkeiten um   */
```

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

```
main() {
float msec, kmh;
for ( msec = 0; msec <= 120; msec = msec + 20 ) {
    kmh = msec*3.6;
    printf("%6.2f  m/sec = %6.2f km/h\n", msec, kmh);
}
}
```

for und while sind vollkommen äquivalent:

Das linke und das rechte Konstrukt tun exakt das gleiche:

while: Anw 1; while (Ausdr) { Anweisung 3 Anw 2; }	for: for (Anw 1; Ausdr; Anw 2) Anweisung 3
---	--

Weiteres Beispiel für ein **while**-Konstrukt mit äquivalenter **for**-Version:

```
/* io.c                               io-for.c
   liest die Eingabe zeichenweise und gibt sie auf den Bildschirm aus.   */
```

```
/* while-Version */                    /* for-Version   */
```

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

```
main() {                                main() {
int c;                                  int c;
while ( (c=getchar()) != EOF )          for ( ; (c = getchar()) != EOF ; )
    putchar(c);                          putchar(c);
}                                          }
/*
```

```
Die Bibliotheks-Funktion
getchar()
liest ein Zeichen von der Standard-Eingabe = Tastatur,
dies Zeichen wird nach c kopiert. Falls es nicht das
EOF (End Of File)-Zeichen ist, wird c mit der Bib-Funktion
putchar()
auf den Bildschirm geschrieben.
Das EOF-Zeichen wird nach Tastatur-Eingabe von Ctrl-D
von getchar() zurueckgegeben.
*/
```

Aufgabe 1.4: Mit der Funktion `io.c` (oder mit `io-for.c`) kann man Dateien kopieren: Kompilieren Sie z. B. mit dem Befehl: `gcc -o io io.c`. Dann heißt die ausführbare Datei `io`. Mit dem Befehl

```
./io < quelledatei > zieldatei
```

können Sie dann eine Datei namens `quelledatei` auf eine Datei namens `zieldatei` kopieren. Probieren Sie dies aus.

Die Zeichen <, > sind hierbei “Umleitungen” der Eingabe und Ausgabe. < leitet die Eingabe für das Programm `io` in folgender Weise um: Statt “von der Tastatur zu lesen”, wird nun zeichenweise aus der Datei `quelldatei` gelesen. > wirkt folgendermaßen: Statt “auf den Bildschirm zu schreiben”, wird in die Datei `zieldatei` geschrieben. Der Befehl könnte auch so gegeben werden:

```
./io > zieldatei < quelldatei
```

switch-Konstrukt

Das `switch`-Konstrukt ist eine multiple Fallunterscheidung:

Angaben in [] sind optional.

Syntax:	Semantik:
<code>switch (Ausdruck) {</code>	Ausdruck wird ausgewertet.
<code>case wert1:</code>	
<code> [Anweisung 1] [break;]</code>	Ist der Wert = <i>wert1</i> ,
<code>case wert2:</code>	werden ohne weiteren Test
<code> [Anweisung 2] [break;]</code>	alle Anweisungen nach <code>case wert1</code>
<code>case wert3:</code>	(keine frühere) bis zum nächsten <code>break</code> ausgeführt;
<code> [Anweisung 3] [break;]</code>	danach wird ans Ende des Konstrukts gegangen.
<code>...</code>	
<code>[default:</code>	Ist der Wert \neq <i>wert1</i> für alle <i>i</i> ,
<code> [Anweisung]]</code>	wird die Anweisung nach <code>default</code> ausgeführt.
<code>}</code>	

```
/* odh.c
   wandelt Zahldarstellungen um:   oktal, dezimal, hex          */
#include <stdio.h>
main() {
    int a; char c;
    while (1) {
        printf("Bitte Zahl eingeben mit vorangestelltem o,d oder h: ");
        scanf("%c",&c); /* liest eingegebenen Wert als char nach c ein;
                           wichtig: Das fuehrende Leerzeichen im
                           Control-String schluckt vorangegangenes \n */

        switch (c) {
            case 'o':
                scanf("%o",&a); break; /* liest naechsten eing. Wert oktal nach a ein */
            case 'h':
                scanf("%x",&a); break; /* liest naechsten Wert hexadezimal nach a ein */
            default:
                scanf("%d",&a); break; /* dezimal */
        }
        printf("Oktal:   %o\n",a); /*druckt oktal          */
        printf("Dezimal: %d\n",a); /*druckt dezimal       */
        printf("Hex:     %x\n",a); /*druckt hexadezimal */
        if (a == 0)
            break;
    }
}
```

Aufgabe 1.5: Ändern Sie das Programm `odh.c` so ab, dass es als Präfix nicht nur `o,d,h`, sondern auch `O,D,H` akzeptiert und sinngemäß arbeitet.

Das folgende Programm zählt verschiedene Arten von Zeichen in einer Datei. Der “White Space”-Zähler `wc` wird erhöht, wenn ein Newline-, ein Tabulator- oder ein Blank-Zeichen vorliegt.

```
/* cc.c zaehlt Charaktere (Zeichen), "White Space" und Zeilen
   einer Datei.   */

#include <stdio.h>

main() {
    int  c, cc, bc, nc, wc;
           /* Zaehler fuer char's, white space, newlines, Woerter   */
    int in_wort = 0;           /* Anzeige, ob gerade ein Wort gelesen wird */
    cc = bc = nc = wc = 0;
    while ( (c=getchar()) != EOF) {
        cc++;
        switch(c) {
            case '\n':
                nc++, bc++, in_wort = 0; break;
            case '\t': case ' ':
                bc++, in_wort = 0; break;
            default:           /* kein newline, tab oder blank   */
                if (in_wort == 0) {
                    in_wort = 1; wc++;
                }
            }
        }
    }
    printf("%d Char's, %d Whites, %d Woerter, %d Newlines\n", cc, bc,wc, nc );
}
```

Aufruf mit `./a.out < cc.c` etwa liefert die Ausgabe:

```
677 Char's, 211 Whites, 122 Woerter, 25 Newlines
```

Auch der Aufruf `./a.out < a.out` funktioniert:

```
13473 Char's, 79 Whites, 80 Woerter, 18 Newlines
```

Aufgabe 1.6: Ändern Sie das Programm so ab, dass es z. B. auch die Häufigkeit des Buchstabens `a` in `a.out` in der Datei zählt.

Aufgabe 1.7: Schreiben Sie ein Programm, das vor jede Zeile einer Datei eine Zeilennummer einfügt. (Hinweis: Zu Anfang und jeweils nach einem gelesenen Newline einen Zeilenzähler ausgeben.) Alternativ lassen Sie die Zeilennummer am Zeilenende ausgeben, mit vorangestellten `/` und nachgestellten `*/`, dass die Nummer als Kommentar erscheint.*

Weitere Anweisungen zur Programmfluss-Steuerung:

- `do`
 Anweisung
 `while (Ausdruck);`
führt Anweisung aus und wertet danach Ausdruck aus. Ist Ausdruck $\neq 0$, wird das Konstrukt erneut durchlaufen, andernfalls wird ans Ende des Konstruktes gegangen. Der Unterschied zum `while`-Konstrukt ist: Die Anweisung des `do-while`-Konstruktes wird immer wenigstens einmal durchlaufen, die Anweisung des `while`-Konstruktes wird u. U. (wenn der zu testende Ausdruck bereits zu Beginn 0 ist) gar nicht durchlaufen.
- `break`; erlaubt, aus einer Schleife vorzeitig auszubrechen (siehe `odh.c`).
- `continue`; in einer `while`- oder `for`-Schleife erlaubt, sofort den nächsten Schleifendurchgang zu beginnen.
- `goto ziel`; erlaubt einen Sprung zu einem Programmpunkt, der mit einer Marke namens `ziel`: gekennzeichnet ist (der Name, hier `ziel`) ist beliebig, der Doppelpunkt bei der Marke erforderlich.

Aufgabe 1.8: Testen Sie die Arbeitsweise dieser Anweisungen mit selbstgeschriebenen Programmbeispielen.

Funktionen in C:

Die Deklaration einer Funktion geschieht durch einen sogenannten Funktions-*Prototyp*. Z. B. ist

```
int ggt(int, int)
```

der Prototyp einer Funktion namens `ggt`, die zwei Argumente vom Typ `int` aufnimmt und einen Wert vom Typ `int` zurückgibt. Im folgenden Beispiel berechnet eine solche Funktion den größten gemeinsamen Teiler zweier ganzer Zahlen.

```
/* ggt.c
   Groesster gemeinsamer Teiler ggt zweier Zahlen */
#include <stdio.h>

int ggt(int x, int y) {          /* gibt ggt(x,y) zurueck, falls */
    int c;                      /* x oder y ungleich 0 */
    if ( x < 0 ) x = -x;         /* und gibt 0 zurueck fuer x=y=0. */
    if ( y < 0 ) y = -y;
    while ( y != 0 ) {
        c = x % y; x = y; y = c;
    }
    return x;
}
```

```
main() {
    int a,b;
    while (1) {
        printf("Gib zwei ganze Zahlen ein: ");
        if ( scanf("%d %d", &a, &b) == 0 ) {
            printf("Eingabefehler\n");
            break;
        }
        printf("ggt(%d,%d) = %d\n", a, b, ggt(a,b));
    }
}
```

Erläuterungen: 1. Die Funktionsdefinition von `ggt` enthält zwei “formale Variable” `int x`, `int y`, deren Wert erst bei “Aufruf” der Funktion (in `main` durch den Ausdruck `ggt(a,b)`) feststeht. Die aktuellen Werte von `a`, `b` werden an die Funktion `ggt` übergeben, die Funktion berechnet dann den Rückgabewert, indem sie den Funktionscode so ausführt, als hätten `x,y` den Wert von `a,b`. Der Wert des Ausdrucks `ggt(a,b)` im Funktionsaufruf ist der Wert, den die Funktion mit `return` zurückgibt.

2. In `main` wird geprüft, ob der Ausdruck

```
scanf("%d %d", &a, &b)
```

den Wert 0 hat. Dies ist der Fall, wenn das Einlesen der beiden Werte für `a,b` fehlerhaft war, z. B. weil versehentlich ein Buchstabe statt einer Zahl eingegeben wurde. Die `break`-Anweisung sieht für diesen Fall den Abbruch der `while`-Schleife und damit des Programms vor.

*Aufgabe 1.9: Schreiben Sie eine Funktion `int kgv(int,int)`, die das “kleinste gemeinsame Vielfache” zweier ganzer Zahlen berechnet. Hinweis: Ist `d = ggt(a,b)`, so ist das kleinste gemeinsame Vielfache gegeben durch `a*b/d`.*

Hier ist eine rekursive Variante der `ggt`-Funktion: Sie liefert das gleiche Ergebnis wie die Funktion oben.

```
/* ggt-rec.c
   Groesster gemeinsamer Teiler ggt zweier Zahlen */
#include <stdio.h>
int ggt(int x, int y) {
    if ( x < 0 ) x = -x;
    if ( y < 0 ) y = -y;
    if ( y == 0 ) return x;
    return ggt(y, x%y);          /* Hier ruft die Funktion ggt 'sich selbst' auf. */
}
main() {
    int a,b;
    while (1) {
        printf("Gib zwei ganze Zahlen ein: ");
        if ( scanf("%d %d", &a, &b) == 0 ) {
            printf("Eingabefehler\n");
            break;
        }
        printf("ggt(%d,%d) = %d\n", a, b, ggt(a,b));
    }
}
```

Aufgabe 1.10: Schreiben Sie eine nicht-rekursive und eine rekursive Variante einer Funktion `int fakultaet(int)` mit der Eigenschaft, dass `fakultaet(n)` für eine positive ganze Zahl das Produkt aller Zahlen von 1 bis `n` liefert.

```
/* schaltjahr.c
   berechnet, ob ein Jahr nach den Gregorianischen Kalender ein Schaltjahr ist.
   Achtung: Vor 1582 war jedes Jahr mit durch 4 teilbarer Zahl ein Schaltjahr
   (Julianischer Kalender). */
#include <stdio.h>
main() {
    int jahr;
    while(1) {
        printf("Gib Jahreszahl ein: ");
        scanf("%d", &jahr);
        if ( schalt(jahr) )
            printf("%d ist ein Schaltjahr\n", jahr);
        else
            printf("%d ist kein Schaltjahr\n", jahr);
    }
}
int schalt(int j) {
    if ( j % 4 == 0 && j % 100 != 0 || j % 400 == 0 )
        return 1;
    else
        return 0;
}
```

Aufgabe 1.11: Ändern Sie das Programm so ab, dass für Jahre vor 1582 die Ausgabe gemäß dem julianischen Kalender erfolgt. Wieviele Schaltjahre hat es seit dem Jahre 1 zu viel gegeben? Sie könnten ein Programm schreiben, um dies festzustellen.