

## Einführung in die Programmiersprachen C und C++

Prof. Dr. Ulf Rehmann, Fakultät für Mathematik

Übungsblatt 3 (4 Seiten)

**Zur Syntax von printf und scanf** (Vergleiche die Manual-Einträge: `man printf`, `man scanf`):

`printf("Was soll das?")` druckt: Was soll das?     `printf("Was soll das?\n")` druckt: Was soll das?

und fügt einen Newline-Character hinzu: Der nächste Ausdruck beginnt eine Zeile weiter unten.

`printf("%d", a)` druckt den Inhalt der (`int`-) Variablen `a` dezimal,

`printf("%o", b)` druckt den Inhalt der (`int`-) Variablen `b` oktal,

`printf("%x", c)` tut was?

`printf("%.6f", d)` hält `d` für eine `float`-Variable und druckt??

Ersetze `f` im letzten Beispiel durch `e`, `g`. Was geschieht?

`printf("%s", string)` hält `string` für eine Zeichenkette,

`printf("%c", charact)` deutet `charact` als Charakter.

Was bedeuten die Kontroll-Strings `"%10s"`, `"%-20s"`, `"%-20.4s"` usw. ?

Generell bedeutet der Ausdruck

```
printf(<string> [, <var_1>, ..., <var_n>]);
```

folgendes: Der Control-String `<string>` wird Zeichen für Zeichen auf den Bildschirm geschickt, mit folgenden Ausnahmen: Ein `%`-Zeichen in `<string>` leitet ein Interpretationsfeld innerhalb dieses Strings ein, das das Ausgabeformat einer der Variablen oder Ausdrücke `<var_1>`, ..., `<var_n>` festlegt, dabei bezieht sich das `i`-te `%`-Zeichen in `<string>` auf `<var_i>`.

`%` gefolgt von `d, o, x` setzt voraus, daß die zugehörige Variable (oder der entsprechende Ausdruck) `<var>` vom Typ `int` oder `char` ist, und gibt den Zahlenwert dezimal, oktal oder hexadezimal aus. Steht hinter dem `%`-Zeichen eine Zahl `n` wie in `%4x` oder `%-10d`, wird die Zahl in einem Feld der Breite `|n|` ausgegeben, und zwar rechtsbündig bei positiver Zahl, linksbündig bei negativer Zahl. Vor `d, o, x` kann ein `l` stehen, dann muß `var` vom Typ `long int` sein.

`%` gefolgt von `f, lf` erwartet `<var>` vom Typ `float` oder `double`, eine Zahl nach `%` und vor `f` oder `lf` wie in `%7.4f` legt das Ausgabeformat fest (in diesem Fall: Feldbreite 7 und 4 Nach"komma"zahlen), Vorzeichenregel wie oben. Es gibt weitere Ausgabe-Anweisungen für "wissenschaftliche" Notation etc, siehe `man printf`.

Soll ein `%`- oder `\`-Zeichen ausgegeben werden, verwendet man `\\` bzw. `\\` im Control-String.

`%c` wie in `printf("%c", x)` erwartet die Variable `x` vom Typ `char`, `%s` wie in `printf("%s", z)` erwartet, daß `z` einen "String" bezeichnet.

Rückgabewert von `printf()` ist die Anzahl der geschriebenen Zeichen, ein `int`  $\geq 0$ .

Ähnliche Regeln gelten für die Argumente der Input-Funktion `scanf`. Beispiel:

`scanf("%d", &a)` erwartet (z. B. von der Tastatur) eine dezimale `int`-Eingabe und weist diese als Wert der (`int`-) Variablen `a` zu.

(Als Argument erscheint nicht `a` selbst, sondern `&a`, die "Adresse" oder ein auf `a` weisender "Pointer".)

Der allgemeine Aufruf von `scanf` sieht so aus:

```
scanf(<string> [, <&var_1>, ..., <&var_n>]);
```

die optionalen Variablen sind **Adressen** (vorangestelltes `&`).

(Wiederholte) Aufrufe von `scanf` behandelt den Strom (die Folge) der Eingabezeichen so: Der Eingabe-Strom muß den Zeichen des Control-Strings `<string>` entsprechen, wobei Konversionsspezifikationen (beginnend mit `%` wie oben) erwarten, daß die Eingabe entsprechende Datenformate liefert, die dann in die zugehörigen Variablen eingelesen werden. White Space (beliebiger Länge, incl. 0) im Control-String entspricht White Space beliebiger Länge im Eingabe-Strom, andere Zeichen im Control-String müssen im Eingabe-Strom einzeln "gematcht" werden.

Rückgabewert von `scanf()` ist die Anzahl der erkannten und zugewiesenen Felder (ein `int`  $\geq 0$ ), man kann also über den Rückgabewert erkennen, ob die Eingabe inkorrekt war (wenn dieser verschieden ist von der Anzahl der mit Wert zu belegenden Variablen).

**Beispiele zur Syntax von while und for:**

Die folgenden Beispiele 3.1 - 3.5 sind programmiertechnisch keineswegs alle vorbildlich. Sie sollen hier nur anhand eines einfachen Beispiels die formale Äquivalenz des **for**-Konstrukts mit dem in der Vorlesung angegebenen erweiterten **while**-Konstrukt verdeutlichen. Urteilen Sie selbst über die Zweckmäßigkeit und Lesbarkeit der verschiedenen Varianten.

**Beispiel 3.1:**

```
#include <stdio.h>
main() {
    int a, b, c;
    printf("Eingabe a, b : ");

    scanf("%d %d", &a, &b);
    while (b != 0) {
        c = a % b; a = b; b = c;
    }
    printf("%s %d\n", "ggT =", a);
}
```

Bemerkung: Das **while**-Konstrukt in diesem Programm kann mit Hilfe des Komma-Operators auch so geschrieben werden:

```
while (b != 0)
    c = a % b, a = b, b = c;
```

*Aufgabe 3.1: Was geschieht, wenn die **scanf**-Zeile ersetzt wird durch folgendes?*

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

*Versuchen Sie, anhand des Manual-Eintrags von **scanf** eine Erklärung zu geben.*

**Beispiel 3.2:**

```
#include <stdio.h>

main() {
    int a, b, c;
    printf("Eingabe a, b : ");

    for( scanf ("%d %d", &a, &b); b != 0; ) {
        c = a % b; a = b; b = c;
    }
    printf("%s %d\n", "ggT =", a);
}
```

**Beispiel 3.3:**

```
#include <stdio.h>
main() {
    int a, b, c;
    printf("Eingabe a, b : ");
    for(scanf ("%d %d", &a, &b); b!=0; c = a % b, a = b, b = c)
        ; /* leeres for-Statement */
    printf("%s %d\n", "ggT =", a);
}
```

Die folgenden Beispiele sollen die bisher diskutierten Programmier Techniken weiter demonstrieren und Gelegenheit zur Wiederholung bieten. 3.1, 3.2 sind Beispiele zum Funktionsbegriff, 3.3 – 3.5 sind Abwandlungen des Idioms

```
while( (c=getchar()) != EOF)
    putchar(c)
```

3.6 ist eine Kombination dieser Dinge.

#### Beispiel 3.4

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

```
main() {
    int i;
    for ( i = 0; i < 10; i++)
        printf("%d %d %6d\n", i, power(2,i), fact(i));
}

int power(int base, int n) {      /* berechnet base hoch n    */
    int i, p = 1;                /* Beim Deklarieren wird p initialisiert */
    for ( i = n; i > 0; i--)
        p = p * base;           /* Kuerzer: p *= base;      */
    return p;
}

int fact(int y) {                /* berechnet 1*2* ... *(y-1)*y    */
    int ergebnis = 1;
    while (y != 0)               /* Achtung! Falls y < 0 ....      */
        ergebnis *= y--;       /* kurz fuer: ergebnis = ergebnis * y; y--; */
    return ergebnis;
}
```

*Aufgabe 3.2: In beiden vorangegangenen Funktionen sind keine Überprüfungen der Argumente vorgesehen. Korrigieren Sie das.*

#### Beispiel 3.5:

Beachten Sie die folgende Regel:

Eine Funktion, die einen Rückgabewert vom Typ Nicht-int hat, muß wie im nächsten Beispiel vor dem Erstaufwurf *deklariert* sein durch Angabe ihres "Prototyps" (siehe Beispiel 3.5). Der Compiler nimmt sonst beim Erstaufwurf an, daß die Funktion `int` zurückliefert, und bemängelt dann später bei der Definition der Funktion ein "type mismatch".

```
/* Quadratwurzel ohne Errorausgabe fuer negative Radikanden    */

#define abs(A) ( (A) < 0 ? -(A) : (A) )    /* "Funktion" per define */
#define DELTA 1.0e-16

double wurzel(double); /* Funktionsprototyp */

main() {
    int i;
    for (i=0; i < 10; i++)
        printf( "%d %20.16f\n", i, wurzel( (double)i ) );
}
```

```
double wurzel(double n) {
    double x = n, alt_x = 0;
    if (n <= 0)
        return 0;

    while ( abs(x - alt_x) > DELTA ) {
        alt_x = x;
        x = (n/x + x)/2;
    }
    return x;
}
```

*Aufgabe 3.3: Verbessern Sie die Funktion `wurzel()`, so daß im Fall eines negativen Radikanden eine entsprechende Fehlermeldung erfolgt.*

### Beispiel 3.6:

```
#include <stdio.h>
main() {
    /* Geheimschrift */
    int c;
    while ( (c = getchar()) != EOF)
        putchar(c^(-1));
}
```

Dies Programm verändert `c` mit dem Operator `^` (Bitweis-Oder). Dadurch werden Zeichen erzeugt, die u. U. nicht druckbar sind. Daher ist es besser, die Ausgabe in eine Datei zu schreiben mit der Unix-Umleitung

`./a.out < quelldatei > zieldatei`

Ein zweiter Aufruf

`./a.out < zieldatei > zieldatei2`

“dekodiert” die “kodierte” Datei: `zieldatei2` ist nun identisch mit `quelldatei` – warum?

*Aufgabe 3.4: Modifizieren Sie dies Programm, so daß ein beliebiger, jeweils auf eine Abfrage hin eingebbarer Schlüssel verwandt werden kann.*

### Beispiel 3.7:

```
#include <stdio.h>

main() {
    /* Fuegt Zeilennummern ein */
    int c, d, nl;
    nl = 1;
    printf("%4d %4c", nl, ' ');
    while ((c = getchar()) != EOF) {
        if (c == '\n') {
            ++nl; putchar(c);
            printf("%4d %4c", nl, ' ');
        }
        else putchar(c);
    }
}
```

*Aufgabe 3.5: Ändern Sie das Programm so ab, daß die Zeilennummern als Kommentare in einem C-Programm gedruckt erscheinen, also von `/* ... */` eingerahmt sind.*