Einführung in die Programmiersprachen C und C++

Markus Kirschmer, Fakultät für Mathematik

Übungsblatt 1

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 Linux. Sie sind gegebenenfalls abzuändern.

Die Programme zum Kurs und diese Übungsblätter sowie weitere Informationen sind zu finden unter

`https://www.math.uni-bielefeld.de/~mkirschm/CC++/`

Dort findet man auch Literaturhinweise.

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

```
/* Name: welt.c;  diese Zeile ist ein Kommentar */
#include <stdio.h>

int main() {
  printf("Hallo, Welt!\n");
}
```

Dieses 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 Linux) `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 und 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 den Cursor an den Anfang der nächsten Zeile zu positionieren (ein *newline* auszugeben).
`int a, c;` ist die Anweisung, Variablen 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 und Definitionen:

Objekte wie Konstanten, Variablen 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 Deklaration für den Rest der zusammengesetzten Anweisung – also bis zur nächsten geschweiften 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

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

Aufgabe 1.2. *Welche Ausdrücke und Anweisungen enthält das folgende Programm?*

```
/* inch.c  */
/* Umwandlung von Inch in cm und umgekehrt */
#include <stdio.h>                          /* Praeprozessor-Anweisung */

int main() {
  const float faktor = 2.54;                /* Deklaration u. Initialisierung */
  float x, in, cm;                          /* Deklaration */
  char ch;                                  /* Deklaration */

  printf("Bitte Laenge eingeben, gefolgt von i oder c: "); /* Ausgabe    */
  scanf("%f %c", &x, &ch);                  /* Eingabe nach x und ch          */

  if (ch == 'i') {                          /* Falls ch gleich dem Buchst. i  */
    in = x;                                 /* kopiere x nach in              */
    cm = x*faktor;                          /* mult. x mit faktor, Erg. nach cm */
  }
  else {                                    /* andernfalls (ch ungleich i)    */
    in = x/faktor;                          /* teile x durch faktor,Erg. n. in  */
    cm = x;                                 /* kopiere x nach cm              */
  }

  printf("%4.2f in = %4.2f cm\n", in, cm); /* Ausgabe ... */
}
```

Elementare Programmfluss-Konstrukte:

Anweisungen werden in der Regel ausgeführt in der Reihenfolge, in der sie im Programm erscheinen. Jedoch gibt es Möglichkeiten, hiervon abzuweichen.

`if` und `if-else`: Im obigen Beispiel haben wir bereits ein `if`-Konstrukt gesehen.

Syntax:	Semantik:
`if (Ausdruck)` `  Anweisung`	`Ausdruck` wird ausgewertet, falls $\neq 0$, wird `Anweisung` ausgeführt, sonst nicht

`if (Ausdruck)`
`  Anweisung 1`
`else`
`  Anweisung 2`
	`Ausdruck` wird ausgewertet, falls $\neq 0$, wird `Anweisung 1` ausgeführt, sonst `Anweisung 2`

Die Anweisungen sind in der Regel *zusammengesetzte Anweisungen*. Sie können z.B. selbst auch wieder `if-else`-Konstrukte enthalten. Das kann man anwenden, um folgende Aufgabe zu lösen:

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.*

```
/* schaltjahr0.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>
int main() {
  int jahr;
  printf("Gib Jahreszahl >= 1582 ein : ");
  scanf("%d", &jahr);
  if (jahr % 400 != 0) {      /* jahr nicht durch 400 teilbar */
    if (jahr % 4 == 0 && jahr % 100 != 0) {
              /* jahr durch 4 und  nicht durch 100 teilbar    */
      printf("%4d ist ein Schaltjahr\n", jahr);
    }
    else {  /* jahr nicht durch 4 oder aber durch 100 teilbar */
      printf("%4d ist kein Schaltjahr\n", jahr);
    }
  }
  else {                      /* jahr durch 400 teilbar */
    printf("%4d ist ein Schaltjahr\n", jahr);
  }
}
```

Hier sind mehrere `if-else`-Konstrukte geschachtelt.
Regel: Jedes `else` bezieht sich auf das letzte vorangegangene `if` in der gleichen Klammerungsebene.

Aufgabe 1.4. *1. Können Sie das obige Progamm ohne verschachtelte* `if-else`*-Konstrukte schreiben?*

2. Erweitern Sie das Programm, so daß auch der Fall `0 < jahr < 1582` *korrekt behandelt wird: es galt damals jedes durch 4 teilbare Jahr als Schaltjahr.*

`while`-Schleife:

Syntax:	Semantik:
`while (Ausdruck)` `  Anweisung`	`Ausdruck` wird ausgewertet, falls $\neq 0$, wird `Anweisung` ausgeführt und das Konstrukt erneut durchlaufen. Ist `Ausdruck` gleich 0, so wird die Anweisung übersprungen.

Beispiel:

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

#include <stdio.h>

int main() {
  float msec, kmh;
  msec = 0;
  while ( msec <= 120 ) { /* Der Ausdruck msec <= 120
       liefert Wert ungleich 0, falls der Vergleich zutrifft, andernfalls 0 */
    kmh = msec*3.6;
    printf("%6.2f  m/sec = %6.2f km/h\n", msec, kmh);
    msec = msec + 20;
  }
}
```

Weiteres Beispiel für ein `while`-Konstrukt: Die Bibliotheks-Funktion `getchar()` liest ein Zeichen von der Standard-Eingabe (normalerweise der Tastatur), dies Zeichen wird nach c kopiert. Falls es nicht das EOF (End Of File)-Zeichen ist, wird c mit der Funktion `putchar()` auf den Bildschirm ausgegeben. Das EOF-Zeichen wird nach Tastatur-Eingabe von Ctrl-D von `getchar()` zurückgegeben.

```
/* io.c liest die Eingabe zeichenweise und gibt sie auf den Bildschirm aus. */
#include <stdio.h>
int main() {
  int c;
  while ( ( c = getchar() ) != EOF )
    putchar(c);
}
```

Aufgabe 1.5. *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 < quelldatei > zieldatei`

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

Die Zeichen `<` und `>` 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.

`for`-Schleife:

Syntax:	Semantik:
`for (Anw1; Ausdr; Anw2)` `  Anw3;`	1. `Anw1` wird ausgeführt. 2. `Ausdr` wird ausgewertet. 3. Ist `Ausdr` ungleich 0, wird `Anw3` ausgeführt und dann `Anw2`. Ist `Ausdr` gleich 0, so wird ans Ende des Konstrukts gesprungen.

Die folgenden beiden Ausdrücke sind äquivalent:

```
for (Anw1; Ausdr; Anw2)
    Anw3;
```

```
Anw1;
while (Ausdr) {
    Anw3;
    Anw2;
}
```

Das heißt jede `while`-Schleife kann in eine `for`-Schleife konvertiert werden und umgekehrt.

Eines der vorangegangenen Beispiele umformuliert mit einer `for`-Schleife:

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

#include <stdio.h>

int 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);
  }
}
```

`do-while`-Schleife:

Syntax:	Semantik:
`do`	`Anweisung` wird ausgeführt, danach wird `Ausdruck` ausgewertet.
`  Anweisung`	Ist dieser $\neq 0$, wird das Konstrukt erneut durchlaufen.
`while (Ausdruck);`	

Der Unterschied der beiden `while`-Konstrukte besteht darin, daß im zweiten Fall (`do-while`) die Anweisung mindestens einmal ausgefürt wird, im ersten Fall u. U. überhaupt nicht.

Sprungkommandos:

- `break;` erlaubt, aus einer `while`- oder `for`-Schleife 'vorzeitig' auszubrechen (siehe `odh.c`).
- `continue;` in einer `while`- oder `for`-Schleife beginnt sofort den nächsten Schleifendurchlauf.

Aufgabe 1.6. *Testen Sie die Arbeitsweise dieser Anweisungen mit selbstgeschriebenen Programmbeispielen.*

`switch`-Konstrukt Das `switch`-Konstrukt ist eine multiple Fallunterscheidung. Die Angaben in [ ] sind optional.

Syntax:	Semantik:
`switch (Ausdruck) {`	Zunächst wird `Ausdruck` ausgewertet.
`  case wert1:`	Ist sein Wert gleich `werti`, so werden **ohne weitere Tests**
`    [ Anweisung 1 ]  [ break; ]`	**alle** Anweisungen nach `case werti` bis zum nächsten `break`
`  case wert2:`	ausgeführt; danach wird ans Ende des Konstrukts gesprun-
`    [ Anweisung 2 ]  [ break; ]`	gen.
`  ...`	Ist der Wert von `Ausdruck` ungleich `werti` für alle i, wird
`  [default:`	(nur) die Anweisung nach `default` ausgeführt. Danach wird
`    [ Anweisung ] ]`	ans Ende des Konstrukts gesprungen.
`}`	

Es folgen zwei Anwendungen des `switch`-Konstruktes:


```
/* odh.c, wandelt Zahldarstellungen um: oktal, dezimal, hex  */
#include <stdio.h>
int 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 eingeg. Wert oktal nach a ein */
    case 'h':
      scanf("%x",&a); break; /* liest naechsten Wert hexadezimal nach a ein */
    case 'd':
      scanf("%d",&a); break; /* dezimal */
    default:
      printf("Unzulaessige Eingabe\n");
      continue;              /* geht zum naechsten Schleifendurchlauf */
    }
    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;
  }
}
```

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

```
/*  cc.c zaehlt Charaktere (Zeichen), "White Space", Zeilen und Woerter
    einer Datei. Aus: Kernighan-Ritchie */
#include <stdio.h>
int main() {
  int  c, cc, bc, nc, wc;  /* Zaehler fuer char's, blanks, newlines, Woerter */
  int in_wort = 0;         /* Anzeige, ob gerade ein Wort gelesen wird       */
  cc = bc = nc = wc = 0;   /* Initialisieren der Zaehler                     */
  while ( ( c=getchar() ) != EOF) {
    cc++;                  /* Abkuerzung fuer: cc = cc+1; */
    switch(c) {
      case '\n':
        nc++, bc++; in_wort = 0; break; /* Newline- und Blank-Zaehler erhoehen
                                           sicher liegt kein Wort vor */
      case '\t':                        /* Tabulator, Leerzeichen       */
      case ' ':
        bc++; in_wort = 0; break;         /* blank-Zaehler erhoehen     */
      default:                          /* kein newline, tab oder blank   */
        if (in_wort == 0) {    /* bisher kein Wort? Also beginnt eins:  */
          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:

```
1051 Char's, 361 Whites, 153 Woerter, 22 Newlines
```

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

```
8429 Char's, 59 Whites, 57 Woerter, 6 Newlines
```

Aufgabe 1.7. *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.*

Aufgabe 1.8. *Ändern Sie das Programm* `cc.c` *so ab, daß es auch die Häufigkeit des Buchstabens* `a` *zählt.*

Aufgabe 1.9. *Schreiben Sie ein Programm, daß vor jede Zeile einer Datei eine Zeilennummer einfügt. (Hinweis: Zu Anfang und jeweils nach einem gelesenen Newline einen Zeilenzähler ausgeben.) Alternative: Lassen Sie die Zeilennummer am Zeilenende ausgeben, mit vorangestellten* `/*` *und nachgestellten* `*/`*, so daß die Nummer als Kommentar erscheint.*

Funktionen in C:

Die Deklaration einer Funktion geschieht durch Angabe ihres Names sowie der Typen Ihres Rückgabewerts und der Argumene, wie z.B.

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

Die Funktion `ggt` nimmt zwei Argumente vom Typ `int` und gibt einen Wert vom Typ `int` zurück. 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 x oder y nicht 0 */
  int c;                /* und gibt 0 zurueck fuer x=y=0.   */
  if ( x < 0 ) x = -x;
  if ( y < 0 ) y = -y;
  while ( y != 0 ) {          /* solange y != 0 */
    c = x % y; x = y; y = c;  /* ersetze x durch y und
     y durch den Rest von x modulo y */
  }
  return x;
}

int main() {
  int a,b;
  while (1) {
    printf("Gib zwei ganze Zahlen ein:  ");
    if ( scanf("%d %d", &a, &b) != 2 ) {
      printf("Eingabefehler\n ");
      break;
    }
    printf("ggt(%d,%d) = %d\n", a, b, ggt(a,b));
  }
}
```

Erläuterungen:

1. Die Funktionsdefinition von `ggt` enthält zwei *formale Variablen* `int x, int y`, deren Wert erst beim 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)
   ```

   einen Wert ungleich 2 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.

`scanf(...)` ist – wie `printf()`, `getchar()`, `putchar()` ebenfalls eine Funktion (aus der *Standardbibliothek* von C), deren Deklaration man dem Unix-Manual entnehmen kann, z.B. mit `man scanf`. Man findet für dort die Deklaration

```
int scanf( const char *format, ...);
```

Der Rückgabewert von `scanf` ist ein `int` und immer die Anzahl der korrekt eingelesenen Werte. Argumente können in unbestimmter Zahl vorkommen, daher die Punkte `...`. Der Typ der ersten Variablen `const char *format` wird später besprochen.

Aufgabe 1.10. *Schreiben Sie eine Funktion* `int kgv(int,int)`*, die das kleinste gemeinsame Vielfache zweier ganzer Zahlen berechnet. Hinweis:* `a*b = ggt(a,b) * kgv(a,b)`*.*

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. */
}
int main() {
  int a,b;
  while (1) {
    printf("Gib zwei ganze Zahlen ein:  ");
    if ( scanf("%d %d", &a, &b) != 2 ) {
      printf("Eingabefehler\n");
      break;
    }
    printf("ggt(%d,%d) = %d\n", a, b, ggt(a,b));
  }
}
```

Aufgabe 1.11. *Schreiben Sie eine nicht-rekursive und eine rekursive Variante einer Funktion*

```
int fakultaet(int),
```

so daß `fakultaet(n)` *für eine positive ganze Zahl* `n` *das Produkt aller Zahlen von 1 bis* `n` *liefert.*

Beispiel: Hier ist eine Funktion, die (für Jahreszahlen $j \geq 1582$) den Wert 1 oder 0 zurückgibt, je nachdem, ob das Jahr j ein Schaltjahr ist oder nicht.

```
int schalt(int j) {
  if ( (j % 4 == 0 && j % 100 != 0) || j % 400 == 0 )
    return 1;
  else
    return 0;
}
```

Aufgabe 1.12. *Testen Sie die Funktion, und ändern Sie sie 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 im Vergleich zum gregorianischen Kalender? Sie könnten ein Programm schreiben, um dies festzustellen.*