Standardaufgaben zum C-Kurs

Franz Schenk

März/April 2007

Standardaufgaben

S1 [Fehlermeldungen des Compilers]

Speichern Sie den nachfolgenden Quelltext in einer Datei `alphabet.c`.

```
#include <stdio.h>
/*
Dieses Programm gibt die ersten 6 Groszbuchstaben des Alphabets aus,
einen Buchstaben pro Zeile.
*/

int main(void)
{
    printf("A\n");
    printf("B\n");
    printf("C\n");
    printf("D\n");
    printf("E\n");
    printf("F\n");

    return 0;
}
```

Compilieren Sie den Quelltext mit dem Aufruf
`gcc -Wall -ansi -pedantic -O alphabet.c`. Wenn das Programm compiliert wurde, so rufen Sie es auf.

Nehmen Sie nun nacheinander die untenstehenden Änderungen am Quelltext vor und versuchen Sie erneut zu compilieren. Der Compiler wird manchmal Fehler finden und auch eine Zeilnnummer dazu in der Quelltextdatei angeben (Dies ist nicht immer genau die Zeile, die den Fehler verursacht hat, sondern sehr oft eine Position weiter hinten im Quelltext!) Lesen Sie die jeweilige Fehlermeldung und versuchen Sie sie zu verstehen. Fragen Sie gegebenenfalls einen Betreuer oder einen fortgeschrittenen Teilnehmer. Für späteres Arbeiten ist es nützlich, sich eine Liste der Fehlermeldungen anzulegen mit stichpunktartigen Hinweisen auf die Fehlerursache.

Wichtig!: Nehmen Sie jede Änderung zurück, bevor Sie die nächste vornehmen!

- Löschen Sie die Zeile `#include <stdio.h>`
- Löschen Sie `*/` am Ende des ersten Kommentars.
- Löschen Sie `int` vor `main`.
- Löschen Sie irgendein Semikolon im Quelltext.
- Schreiben Sie `print` anstatt `printf`.
- Schreiben Sie `return O` (Buchstabe „Oh“) anstelle `return 0`.
- Löschen Sie die Zeile `return 0;`
- Schreiben Sie `retörn` anstelle `return`.

S2 [`printf`, Escape-Sequenzen zur Ausgabe besonderer Zeichen]

Schreiben Sie ein Programm, das die Wirkung der Escapesequenzen

`\b  \n  \r  \t  \"  \\`

als Argumente der Funktion `printf` verdeutlicht.

S3 [Formatierte Ausgabe von Zahlwerten]

Lernen Sie folgenden Quelltext auswendig. Zur Kontrolle speichern Sie ihn in einer Datei, die Sie anschließend compilieren. Falls es Fehlermeldungen gibt, so korrigieren Sie, ohne auf die Vorlage zu sehen. Anschließend führen Sie das compilierte Programm aus.

```
#include <stdio.h>
/* Ein paar hilfreiche Zahlen werden ausgegeben */
int main( void)
{
    printf( "Aller guten Dinge sind %d.\n", 3);
    printf( "Pi ist ungefaehr %f.\n", 3.141593);

    return 0;
}
```

S4 [Ausgabe von Zahlwerten mit `printf`, Standard-Konstanten]

Schreiben Sie ein Programm, das die Werte der Konstanten

`CHAR_MIN  CHAR_MAX  INT_MIN  INT_MAX`
`FLT_MIN  FLT_MAX  DBL_MIN  DBL_MAX`

ausgibt (Standard–Headerdateien `limits.h` bzw. `float.h`).

S5 [Ausgabe von Zeichen, `char`-Variablen, `while`]

Schreiben Sie ein Programm, das alle Zeichen ausgibt, die „größer“ oder gleich 'A' und „kleiner“ oder gleich 'z' sind.

S6 [`while`, Eingabe-Ende, Zahldarstellungen]

Es sollen wiederholt (bis zur Eingabe von `C-d`) nichtnegative Dezimalzahlen eingelesen und die Werte jeweils im Oktal– und Hexadezimalsystem ausgegeben werden

S7 [`while`, `scanf`, ganzzahlige Division, `%`-Operator]

Schreiben Sie ein Programm, das eine nichtnegative ganze Zahl einliest, ihre Quersumme iterativ berechnet und ausgibt.

S8 [`while`, `scanf`, einfache Berechnungen]

Schreiben Sie ein Programm, das eine beliebige Anzahl von Gleitkommazahlen einliest und ihren Mittelwert ausgibt. Die Zahlen müssen dazu nicht alle gleichzeitig gespeichert werden, sondern können nacheinander eingelesen und bearbeitet werden.

S9 [Felder, Sortieralgorithmus für Zahlen]

Es soll eine feste Anzahl ganzer Zahlen eingelesen und in einem Feld abgespeichert werden. Dann soll der Inhalt des Feldes ausgegeben, das Feld sortiert, und der Inhalt des Feldes erneut ausgegeben werden.

Realisieren Sie „Sortieren durch direkte Auswahl“: Wenn n Zahlen zu sortieren sind, bestimmt man zunächst das kleinste Element und vertauscht es mit dem, das an Position 1 steht. Dann bestimmt man von den Elementen an den Positionen $2, \ldots, n$ das kleinste Element und vertauscht es mit dem an Position 2, usw.

S10 [Arbeiten mit Strings]

Schreiben Sie ein Programm, das zwei Zeilen als Strings einliest und den zweiten String an den ersten String anhängt. Die eingelesenen Strings sollen jeweils maximal 50 Zeichen (einschließlich abschließendem Nullbyte) lang sein. Ist dieses nicht der Fall, soll das Programm mit einer entsprechenden Fehlermeldung abgebrochen werden.

S11 [`getchar`, `if`]

Schreiben Sie ein Programm, das eine Binärzahl einliest und den entsprechenden Dezimalwert ausgibt. Dabei sollen führende „white spaces“ überlesen und fehlerhafte Eingaben zuückgewiesen werden (z.B. Eingabe von Buchstaben oder Überschreitung des Wertebereichs).

S12 [Felder, ASCII-Zeichensatz]

Es soll ein Text von der Standardeingabe eingelesen und bestimmt werden, wie oft die einzelnen ASCII–Zeichen in ihm vorkommen.

S13 [Mehrdimensionale Felder, benannte Konstanten]

Schreiben Sie ein Programm, das zwei Matrizen liest, ihr Produkt berechnet und ausgibt.

Dimensionieren Sie die Matrizen mit benannten Konstanten. Das Programm kann dadurch (ohne weiteres) keine größeren Matrizen verarbeiten, aber es soll beliebige kleinere Matrizen verarbeiten können. Vom Benutzer sollte daher vorher das tatsächlich zu verarbeitende Format erfragt werden.

S14 [Felder, `while`, `if`, `%`-Operator]

Zu bestimmen sind alle Primzahlen kleiner als 1000. Zwei Verfahren:

a) Durch Division wird geprüft, ob die Zahlen n ($1 < n < 1000$) einen Teiler k ($1 < k < n$) besitzen. [Das kann man etwas abkürzen: Es genügt einen Teiler zu finden, der kleiner oder gleich $\sqrt{n}$ ist. In Aufgabe **??** finden Sie einen Hinweis zur Benutzung der entsprechenden Standardfunktion `sqrt` für die Quadratwurzel.]

b) Beim „Sieb des Eratosthenes“ (Eratosthenes von Kyrene, griechischer Mathematiker um 225 v. Chr.) werden zunächst alle zu untersuchenden Zahlen (hier 2 bis 999) aufgeschrieben. Jeder Schritt des eigentlichen Algorithmus besteht aus drei Einzelschritten:

   – Die erste nicht weggestrichene Zahl wird gesucht.
   – Diese Zahl wird als Primzahl notiert.
   – Ihre Vielfachen werden weggestrichen.

Realisieren Sie beide Algorithmen möglichst effizient.

S15 [Rekursion, `%`-Operator]

Schreiben Sie eine rekursive Funktion

```
unsigned long ggt (unsigned long a, unsigned long b);
```

die den größten gemeinsamen Teiler der nichtnegativen ganzen Zahlen a und b berechnet. Verwenden Sie die Formel

$$\mathrm{ggt}(a,b) = \begin{cases} \mathrm{ggt}(b, a \bmod b) & \text{falls} \quad b > 0 \\ a & \text{falls} \quad b = 0 \end{cases}$$

S16 [Rekursion, `%`-Operator]

Schreiben Sie eine *rekursive* Funktion zur Berechnung der Quersumme einer nichtnegativen ganzen Zahl (vgl. Aufgabe 7).

S17 [`%`-Operator, logische Operatoren]

Zu einer eingegebenen Jahreszahl soll bestimmt werden, ob es sich um ein Schaltjahr handelt oder nicht. Dies soll als `int`-Funktion realisiert werden, die den Wert `0` liefert, falls das Argument *keine* Schaltjahreszahl ist und einen Wert ungleich `0` sonst.

Der Gregorianische Kalender legt fest, dass jedes Jahr mit durch 4 teilbarer Jahreszahl ein Schaltjahr ist, auer wenn die Jahreszahl durch 100 teilbar ist. In diesem Fall handelt es sich nur dann um ein Schaltjehr, wenn die Jahreszahl auch durch 400 teilbar ist.

S18 [Backtracking, mehrdimensionale Felder]

Schreiben Sie ein Programm, das für ein (rechteckiges) Labyrinth bei vorzugebendem Startpunkt einen Weg (alle Wege, den kürzesten Weg, ...) zu einem Ausgang findet, wobei als Ausgang alle Punkte auf dem Rand des Labyrinths zählen.

Die Form des Labyrinths soll vom Benutzer erfragt werden; seine Eingabe könnte etwa in der Form

```
XXXXXXXXXX
X   X    X
X*XXX XX X
X     X  X
XXXXXXX XX
```

erfolgen, wobei `*` den gewünschten Startpunkt bezeichnet. Überlegen Sie sich selbst eine geeignete Form der Ausgabe.

S19 [mehrdimensionale Felder, Backtracking]

Gegeben sei ein $(n \times n)$–Spielfeld. Ein Springer, der nach den Schachregeln bewegt werden kann, wird auf das Feld mit den Koordinaten (x, y) gesetzt.

Schreiben Sie ein Programm, das alle Wege des Springers findet, die von (x, y) aus genau einmal über jedes der n^2 Felder führen. Überlegen Sie sich vorab: Bestimmte Konfigurationen von Feldgröße und Startpunkt **können keine** Lösung besitzen!

Bauen Sie für den Test ein, daß das Programm abgebrochen wird, sobald der erste Weg gefunden wurde. Wenn Sie diesen „Notausgang“ ausbauen, sollten Sie berücksichtigen:

- Bei einem (5×5)–Spielfeld gibt es je nach Startpunkt zwischen 56 und 304 Lösungen (sofern es überhaupt Lösungen gibt).
- Bei einem (6×6)–Spielfeld gibt es je nach Startpunkt zwischen ca. 50,000 und 500,000 Lösungen.
- Bei einem (7×7)–Spielfeld dürfte es für jeden Startpunkt weit über 100 Mio. Lösungen geben (sofern es überhaupt Lösungen gibt).

Lassen Sie sich vom Programm die benötigte Rechenzeit melden.

S20 [Felder mit `const`-Inhalt]

Schreiben Sie Funktionen, die ein Datum in den Tag im Jahr (z.B. der 11.2. ist der 42. Tag im Jahr) bzw. einen Tag im Jahr in ein Datum umrechnen und schreiben Sie auch eine entsprechende Testumgebung dazu.

S21 [Felder mit `const`-Inhalt, `%`-Operator]

Lassen Sie bei Aufgabe 20 auch „exotische“ Eingaben zu wie in diesen Beispielen: Der 366. Tag im Jahr 1991 ist der 1.1.1992; der 0. Tag im Jahr 1992 ist der 31.12.1991; usw. Verwenden Sie dazu Ihre Lösung von Aufgabe 17 wieder.

S22 [Modularisierung]

Fassen Sie Funktionen aus den Aufgaben 17 und 20 zu einem Modul `datumsfunktionen` zusammen, der aus der Implementationsdatei `datumsfunktionen.c` und der Header-Datei `datumsfunktionen.h` besteht.

Nutzen Sie die Funktionen für ein Programm zur Lösung dieser oder ähnlicher Aufgaben: Welches Datum hat der 100. Tag vor oder nach einem einzugebenden Datum?

S23 [Stringfelder, %-Operator, Modularisierung]

Schreiben Sie ein Programm, das zu einem einzulesenden Datum den Wochentag berechnet. Dazu sollten Sie Ihre Lösung von Aufgabe 22 benutzen können.

Hinweise:

- Das Datum soll in der Form `tt.mm.jjjj` eingegeben werden.
- Der Wochentag soll in Klarschrift gemeldet werden (keine Kennziffer!).
- Unzulässige Daten (z.B. 30.2.1992) müssen zurückgewiesen werden.
- Bei uns gilt der „Gregorianische Kalender“, der durch Papst Gregor XIII. am 15.10.1582 unserer Zeitrechnug eingeführt wurde. In ihm ist ein Jahr ein Schaltjahr, wenn seine Jahreszahl entweder durch 4 und nicht durch 100 oder durch 400 teilbar ist.
- Die Schaltjahrsregel des Gregorianischen Kalenders ist nicht ganz exakt: In circa 3000 Jahren wird eine zusätzliche Korrektur erforderlich. Das Programm sollte deshalb neben Daten vor dem 15.10.1582 auch Daten aus dem Jahr 5000 oder später zurückweisen.
- Der 1.1.1600 war ein Sonnabend.

Verwenden Sie ggf. Modul-weit definierte Variablen (z.B. eine Tabelle der Monatslängen). Achten Sie besonders auf die Strukturierung des Programms!

Sie können sich die Programmierung etwas erleichtern, wenn Sie zunächst darüber nachdenken, wie man diese Aussage nachweisen kann: „In dem Aberglauben, daß man an einem Freitag den 13. besonders häufig Pech hat, steckt ein Körnchen Wahrheit – es gibt nämlich keinen Tag, der häufiger ist als Freitag der 13.“ (Wenn Sie wollen, können Sie die Lösung natürlich auch programmieren.)

S24 [Arbeit mit Strings]

Realisieren Sie die folgenden Funktionen:

- `int strend (const char *s, const char *t)` liefert 1, wenn der String `t` am Ende des String `s` steht, und 0 sonst.
- `char *strrchr (const char *s, int c)` liefert den Zeiger auf das letzte Vorkommen des Zeichens `c` im String `s` bzw. den `NULL`–Zeiger, wenn `c` in `s` nicht vorkommt.
- `char *strstr (const char *s, const char *t)` liefert den Zeiger auf das erste Vorkommen des String `t` im String `s` bzw. den `NULL`–Zeiger, wenn `t` in `s` nicht vorkommt.

Testen Sie die Funktionen in einem Rahmenprogramm.

S25 [Modularisierung, dynamische Speicherverwaltung]

Schreiben Sie einen Modul (Headerdatei und Implementation) zur dynamischen Bereitstellung (und Freigabe) von Speicher für Vektoren und Matrizen. Realisieren Sie in einem weiteren Modul die Matrizenmultiplikation als Funktion.

Testen Sie beide Module mit einem geeigneten Rahmenprogramm.

S26 [Strukturen, Modularisierung]

Definieren Sie den Typ `BRUCH` als Struktur mit zwei `int`–Komponenten `zaehler` und `nenner`. Realisieren Sie für diesen Typ die „Standardoperationen“

- Eingabe eines Bruches
- Ausgabe eines Bruches
- Addition, Subtraktion, ... zweier Brüche
- Berechnung des Quotienten (Funktionswert `float`!)
- Erweitern um einen (als Parameter zu übergebenden) Faktor
- Kürzen

als Funktionen in einem Modul mit Headerdatei.

Testen Sie die Funktionen mit einem geeigneten Rahmenprogramm.