Aufgabensammlung zum C-Kurs

Franz Schenk

März/April 2007

Ergänzungsaufgaben

E1 [Formatierte Ausgabe von Zahlwerten]

Speichern Sie den folgenden Quelltext in einer Datei, die Sie anschliessend compilieren und ausführen. Vergleichen Sie die Formatbeschreiber mit dem Ausgabeergebnis. Merken Sie sich die Systematik und experimentieren Sie ein wenig damit herum.

```
#include <stdio.h>
/* Zahlen werden mit vorgegebener Stellenzahl ausgegeben */
int main( void)
{
    printf( "Aller guten Dinge sind %d.\n", 3);
    printf( "Aller guten Dinge sind %3d.\n", 3); /* mind. 3 Stellen Laenge */
    printf( "Aller guten Dinge sind %6d.\n", 3); /* mind. 6 Stellen Laenge */

    printf( "Pi ist etwa %f.\n", 3.141593);
    printf( "Pi ist etwa %6.3f.\n", 3.141593); /* mind. 6 Stellen, davon ... */
                                               /* .. mind. 3 nach dem Punkt  */
    printf( "Pi ist etwa %.9f.\n", 3.141593);  /* mind. 9 Stellen nach dem . */

    return 0;
}
```

E2 [Schleifen, Formatierte Ausgabe von Zahlwerten, benannte Konstanten]

Lassen Sie eine Multiplikationstafel ausgeben, etwa so:

```
 * |   1   2   3   4   5
-------------------------
 1 |   1   2   3   4   5
 2 |   2   4   6   8  10
 3 |   3   6   9  12  15
 4 |   4   8  12  16  20
 5 |   5  10  15  20  25
```

Verwenden Sie eine benannte Konstanten für die Grösse der Multiplikationstafel, so dass das Programm einfach auf andere Grössen angepasst werden kann.

E3 [Escape-Sequenzen]

Schreiben Sie ein Programm, das folgende Texte ausgibt:

```
Er kam l\"assig heran und sagte nur "Na, wie geht's?".
Kommentare beginnen mit /* und enden mit */. Verwechseln Sie
das bitte nicht mit \* bzw. *\!
```

E4 [`printf`, `char`, `for`, `ctype.h`, Funktion `isprint`]

Rufen Sie in der Konsole die online-Hilfe zum ASCII-Zeichensatz auf: `man ascii`. Schreiben Sie ein Programm, das die dort angegebene Tabelle erzeugt.

E5 [Felder, `for`, `scanf`, benannte Konstanten, `while`, `EOF`]

Schreiben Sie ein Programm, dass die Koeffizienten eines Polynoms $p(x)$ mit vorgegebenem maximalen Grads einliest und anschliessend bis zum Eingabeende Stellen a (Gleitkommawerte) anfordert und den Wert $p(a)$ des Polynoms an dieser Stelle berechnet.

Verwenden Sie zur Berechnung des Werts das Hornerschema. Beispiel: 5

$$p(x) = 5.1{*}x^3 - 1.8{*}x^2 - 0.02{*}x + 17.3 = 17.3 + x{*}(-0.02 + x{*}(-1.8 + 5.1{*}x)).$$

E6 [`scanf`, Rekursion, `%`-Operator]

Rufen Sie in der Konsole den Befehl `factor 144` auf. Die Ausgabe `144: 2 2 2 2 3 3` gibt die Primzahlzerlegung von 144 an. Programmieren Sie dies in C nach, um beliebige ganze Zahlen bis zur Grösse `UINT_MAX` verarbeiten zu können. Die zu verarbeitende Zahl soll mit `scanf` eingelesen werden.

E7 [`getc`, `putchar`, Escapesequenzen]

Textdateien sehen unter UNIX und DOS/Windows unterschiedlich aus. Während in UNIX-Textdateien eine Zeile lediglich durch ein „Linefeed"-Zeichen `\n` (oder Newline) beendet wird, steht in DOS-Textdateien dort die Zeichenfolge `\r\n`. Schreiben Sie zwei Programme, die die Umformungen von-UNIX-nach-DOS bzw. von-DOS-zu-UNIX vornehmen. Die Programme sollen jeweils zeichenweise von der Standardeingabe lesen und auf die Standardausgabe ausgeben. Das erste Programm fügt vor jedem gelesenen `\n` einfach ein `\r` ein und reicht die anderen Zeichen unverändert weiter. Das zweite Programm „verschluckt" einfach die `\r`-Zeichen, die vor einem `\n` auftreten.

E8 [Felder, `for`-Schleife, `while`-Schleife]

Ein Permutation der Länge 9 ist eine Abbildung der Menge $\{1, 2, \ldots, 9\}$ auf sich selbst. Permutationen kann man durch zweireihige Matrizen angeben. Die folgende Permutation zum Beispiel bildet 1 auf 3 ab, 2 auf 4, 3 auf 9 usw.:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 4 & 9 & 1 & 7 & 5 & 2 & 8 & 6 \end{pmatrix}$$

Da die Einträge der oberen Reihe immer die gleichen sind, würde es auch genügen, nur die untere Reihe 3, 4, 9, 1, 7, 5, 2, 8, 6 anzugeben.

Schreiben Sie ein Programm, das Folgendes leistet: Der Nutzer wird aufgefordert, eine Permutation der Zahlen 1 bis 9 einzugeben (durch Angabe der unteren Reihe, wie oben beschrieben). Es wird geprüft, ob die Eingabe korrekt war (d.h., ob jede der Zahlen 1 bis 9 genau einmal angegeben wurde). Bei Korrektheit wird die Permutation in einem Feld gespeichert (ansonsten wird eine erneute Eingabe verlangt) und zur Kontrolle wie eine zweireihige Matrix ausgegeben.

E9 [`rand`, `srand`]

Schreiben Sie eine `int`-Funktion `muenze`, die bei jedem Aufruf (pseudo)zufällig entweder 0 oder 1 als Funktionswert liefert. Testen Sie mit dieser Funktion, ob die „Münze“ tatsächlich wie erhofft etwa ebenso häufig den Wert 1 liefert, wie den Wert 0.

Testen Sie auch, wie oft Ergebniskombinationen wie `00`, `01`, etc. entstehen.

E10 [`rand`, `srand`]

Schreiben Sie eine `int`-Funktion `gezinkterwuerfel`, die bei jedem Aufruf (pseudo)zufällig einen ganzzahligen Wert zwischen 1 und 6 liefert. Dabei soll die 6 etwa doppelt häufig auftauchen, wie jeder andere Wert. Testen Sie dies durch entsprechend häufiges „würfeln“ mit dieser Funktion.

E11 [Felder, `rand`, `srand`, Schleifen]

Schreiben Sie ein Programm, das pseudozufällig eine Permutation der Zahlen 1 bis 9 erzeugt. Gehen Sie dabei wie folgt vor: Die Permutation wird wie in Aufgabe 8 durch ein `int`-Feld der Länge 9 beschrieben. Anfangs ist in jeder Position noch jeder der Werte 1 bis 9 erlaubt (denn keiner ist bereits vergeben und damit „verboten“ für weitere Positionen). Beginnend bei der Feldkomponente mit Index 0 wird nun 8 mal hintereinander ein noch nicht verbotener Wert pseudozufällig gewählt und in der nächsten freien Feldkomponente gespeichert. Die „Menge der noch erlaubten Werte“ wird bei jeder Iteration aktualisiert, so dass nach der 8ten Iteration nur noch ein einziger Wert möglich ist, der in der letzten Komponente gespeichert ist.

E12 [`math.h`, `sqrt`]

Lernen Sie den folgenden Programmtext auswendig:

```
/* Quadratwurzelberechnung */
#include <stdio.h>
#include <math.h>

int main( void)
{
    printf( "Die Wurzel aus %f ist etwa %f\n", 2.0f, sqrt( 2.0f));
    return 0;
}
```

Schreiben Sie dann ohne Vorlage das Programm in eine Datei namens `wurzel.c`, compilieren sie mit dem Aufruf

```
gcc -Wall -ansi -pedantic -O wurzel.c -lm
```

und führen das Binärprogramm aus.

Lernen Sie auch den Compile-Aufruf auswendig. Die Option `-lm` am Ende ist für den Linker bestimmt. Er wird damit angewiesen, die Bibliothek der mathematischen Funktionen zu verwenden.

E13 [`switch`, `if`, (für Ergänzung: Stringfelder)]

Olympische Spiele der Neuzeit finden in Jahren statt, deren Jahreszahl gerade ist und zwar Sommerspiele, falls die Jahreszahl durch 4 teilbar ist und Winterspiele sonst. Schreiben Sie ein Programm, das den Nutzer nach einer Jahreszahl fragt und die Information ausgibt, ob Olympische Spiele in diesem Jahr stattfinden und ob es sich um Winterspiele oder Sommerspiele handelt.

Die oben genannte Regelung gilt erst seit einigen Jahren. Finden Sie im Internet heraus, seit wann. Finden Sie auch heraus, seit wann überhaupt Olympische Spiele der Neuzeit veranstaltet werden und in welchen Jahren es Pausen gab. Bauen Sie dann das Programm aus, in dem Sie zusätzlich die aktuelle Jahreszahl ermitteln lassen (entweder durch Anfrage an den Nutzer oder durch Benutzen einer Bibliotheksfunktion) und die Antwort zeitlich und grammatisch korrekt geben, wie etwa: „Im Jahre 2002 fanden olympische Winterspiele statt.“ oder „Im Jahre 2008 werden olympische Sommerspiele stattfinden.“

Sie können auch eine deLuxe-Variante erstellen, indem Sie die bekannten Austragungsorte in einem Feld konstanter Strings ablegen und die Ausgabe damit ergänzen.

E14 [`math.h`, `sqrt`, `scanf`]

Schreiben Sie ein C-Programm, das vom Nutzer zunächst die Eingabe einer Zahl verlangt, dann intern die Quadratwurzel berechnet und den Nutzer dann die Quadratwurzel „erraten“ lässt: Gibt der Nutzer irgendeinen Wert ein, so reagiert das Programm so lange wahrheitsgemä mit „Grösser! Nächster Versuch:“ oder „Kleiner! Nächster Versuch:“ bzw. „Richtig!!!!“ auf die Eingabe, bis genügende Übereinstimmung mit dem vom Rechner ermittelten Wert erzielt wurde. Die Umschreibung „Genügende Übereinstimmung“ soll andeuten, dass auch der Rechner die Wurzel nur näherungsweise ermitteln kann.

E15 [`while`, Newton-Iteration, benannte Konstanten, ganzzahlige Division]

Schreiben Sie ein C-Programm, das vom Nutzer die Eingabe einer Zahl a verlangt und eine Näherung b für deren Quadratwurzel wie unten beschrieben durch Newton-Iteration berechnet. Prüfen Sie die Güte des Ergebnisses, indem Sie b^2 und a bei verschiedenen Grössenordnungen von a vergleichen und auch die Anzahl der Iterationen ausgeben. Geben Sie die Genauigkeit $\varepsilon > 0$ durch eine benannte Konstante vor.

Die Idee des Newton-Verfahrens ist folgende: Die Quadratwurzel von a ist eine Nullstelle der Funktion $f(x) = a - x^2$. Ausgehend vom Startwert

$x_0 = 1$ werden iterativ $x_1, x_2, \ldots, x_n$ berechnet, bis $|f(x_n)|$ kleiner als ε ist. Dabei wird x_{i+1} aus x_i bestimmt als Schnittpunkt der x-Achse mit der Tangente an den Graphen von $f(x)$ im Punkt x_i. Der Schnittpunkt kann mit Hilfe der Ableitung von $f(x)$ bestimmt werden. Im unserem Fall ergibt sich die folgende Iterationsvorschrift:

$$x_{i+1} = \frac{1}{2} * (a/x_i + x_i).$$

Treffen Sie Vorkehrungen, damit bei der Umsetzung obiger Vorschrift nicht ganzzahlig dividiert wird! Felder sind für die Lösung dieser Aufgabe nicht erforderlich.

E16 [`sizeof`]

Benutzen Sie den `sizeof`-Operator, um zu ermitteln, wieviel Bytes für Werte der Ganzzahltypen reserviert werden. Ermitteln Sie dann rechnerisch (z.B. mittels `kcalc`) die grössten darstellbaren Werte der einzelnen Ganzzahltypen und vergleichen Sie diese mit den Werten aus der Standardaufgabe 4.

E17 [`fgetc`, `fputc`, Kommandozeilenargumente]

Unter UNIX bewirkt der Aufruf `cp` *`dateiname1 dateiname2`*, dass versucht wird, den Inhalt der ersten Datei in der zweiten abzulegen. Falls die Anzahl der Kommandozeilenargumente nicht stimmt, so wird der Nutzer darauf hingewiesen und das Programm beendet. Das Gleiche passiert, falls die erste Datei nicht zum Lesen oder die zweite nicht zum Schreiben geöffnet werden kann.

Programmieren Sie dieses Verhalten nach.

E18 [`fgetc`, `stdlib.h`, `atoi`] uings Verändern Sie Ihr Programm aus Aufgabe 6 dahingehend, dass die zu verarbeitende Zahl jetzt per Kommandozeilenargument als String übergeben wird und mittels `atoi` als Ganzzahl interpretiert wird. Die Dokumentation zu dieser Funktion erhalten Sie durch Benutzung der online-Hilfe `man 3 atoi`. Die Funktion `atoi` überprüft zwar nicht ohne weiteres, ob ein Ganzzahlüberlauf stattfindet (und auch andere Probleme bei der Eingabe bleiben unbehandelt)— Sie brauchen sich in Ihrer Lösung aber nicht darum kümmern.

E19 [`fgetc`, `switch`, `ctype.h`, `isspace`, Kommandozeilenargumente]

Unter UNIX bewirkt der Aufruf `wc` *`dateiname`*, dass in der angegeben Datei die Anzahl der Zeichen, der Wörter und der Zeilen gezählt und ausgegeben wird. Was ein Zeichen ist, sollte klar sein. Ein „Wort“ ist eine beliebige Folge von „non whitespace-Zeichen“, deren Anfang entweder durch den Dateianfang oder eine Whitespace-Zeichen begrenzt wird und deren Ende entweder durch ein Whitespace-Zeichen oder das Dateiende begrenzt wird — die Anzahl der Wörter lässt sich so aus der Anzahl der Übergänge whitespace-non whitespace ermitteln. Eine Zeile ist schlielich eine Folge von Zeichen, deren letztes entweder das Newline-Zeichen (`char`-Wert `'\n'`) oder das Dateiende ist.

Probieren Sie dies an einigen Testdateien aus und programmieren Sie dann das Verhalten nach.

E20 [`fgetc`, `fputc`, Kommandozeilenargumente, Strings]

Schreiben Sie Ihre Programme von Aufgabe 7 um, so dass es ein Kommandozeilenargument als Dateinamen interpretiert, den Inhalt der angesprochenen Datei in die gewünschte Form umwandelt und das Ergebnis in einer neuen Datei abspeichert. Über Fehler (z.B. nicht lesbare Datei) soll der Nutzer informiert werden.

E21 [Felder als Funktionsargumente]

Programmieren und testen Sie eine Funktion:
`int permtest( const unsigned int* p, int laenge)`, die von einem übergebenen Feld der Länge `laenge` testet, ob der Inhalt eine Permutation repräsentiert (Vgl. Aufgabe 8) und in diesem Fall einen Wert ungleich 0 zurückliefert und sonst 0.

E22 [mehrdimensionale Felder als Funktionsargumente]

Eine Permutationsmatrix ist eine quadratische, ganzzahlige Matrix, die in jeder Zeile und jeder Spalte genau einen Eintrag 1 und sonst nur 0-Einträge hat. Programmieren und testen Sie eine Funktion `int permatcheck(const int** p, int laenge)`, die von einer durch ein zweidimensionales `int`-Feld der Grösse `laenge`×`laenge` testet, ob es sich um eine Permutationsmatrix handelt.

E23 [mehrdimensionale Felder als Funktionsargumente]

Eine *Sudoku-Matrix* ist eine 9×9-Matrix, deren Einträge ganze Zahlen zwischen 1 und 9 sind und die weitere Bedingungen erfüllen. Zur Formulierung der Bedingungen betrachten wir die Matrix als Blockmatrix aus 9 Blöcken von 3×3-Matrizen, die durch die Zeilen bzw. Spalten 1 bis 3, 4 bis 6 und 7 bis 9 gebildet werden. Jede dieser 3×3-Matrizen bezeichnen im weiteren als *Block*. Nun können wir die Bedingungen formulieren, die aus einer Matrix obiger Form eine Sudoku-Matrix machen:

- Jede Zeile und jede Spalte gibt eine Permutation der Länge 9 an.
- Jeder Block enthält jeden der Werte 1 bis 9 genau einmal.

Programmieren und testen Sie eine Funktion `int sudokucheck(const int** s)`, die von einer durch ein zweidimensionales `int`-Feld der Grösse 9×9 testet, ob es sich um eine Sudoku-Matrix handelt.

E24 [`rand`, `srand`, mehrdimensionale Felder, Felder als Funktionsargumente]

Schreiben Sie ein Programm, dass pseudozufällig eine Sudoku-Matrix generiert. Dazu gibt es z.B. diese Möglichkeiten:

- Sie speichern ein festes Sudoku (z.B. aus der Zeitung) in einem zweidimensionalen Feld und wenden eine zufällige Permutation der Länge 9 darauf an.
- Lassen Sie zeilenweise den Inhalt des Sudokus zufällig wählen. Bei der ersten Zeile besteht noch völlig freie Wahl, bei jeder weiteren bis zur achten Zeile sind die Ausschlussbedingungen zu berücksichtigen, die sich aus den bereits gewählten Zeilen ergeben. Die letzte Zeile ist dann durch die noch fehlenden Spalteneinträge gegeben.

Zwecks möglicher Wiederverwendung sollte die Aufgabe am besten durch eine Funktion `sudokugenerate` realisiert werden, die einen Zeiger auf den Anfang eines `int`-Feldes der Grösse 9×9 zurückliefert, das die generierte Sudoku-Matrix speichert.

E25 [`rand`, `srand`, mehrdimensionale Felder, `printf`]

Ein Sudoku ist ein Rätsel, das darin besteht, in einer unvollständig gegebenen Sudoku-Matrix die fehlenden Einträge zu ermitteln.

Schreiben Sie ein Programm, das Sudoku-Rätsel generiert. Benutzen Sie zunächst die Funktion `sudokugenerate` aus Aufgabe 24 um eine Sudoku-Matrix zufällig generieren zu lassen. Dann lassen Sie die Matrix in geeigneter Form (mit Rahmen, damit man die Blöcke erkennt etc.) ausgeben, wobei aber bestimmte Einträge durch Leerzeichen ersetzt werden. Welche Einträge ausgegeben werden, können Sie durch ein „Besetzungsmuster“ steuern. Dies soll hier eine beliebige 9×9-Matrix sein. Ein Eintrag der Sudoku-Matrix wird nur dann ausgegeben, wenn der entsprechende Eintrag des Besetzungsmusters verschieden von 0 ist. Passende Besetzungsmuster können Sie z.B. aus in Zeitungen abgedruckten Sudokus ermitteln. Sie können aber auch versuchen, Besetzungsmuster generieren zu lassen, indem mehrere Permutationsmatrizen (siehe Aufgabe 22) generiert werden und deren Summe als Besetzungsmuster verwendet wird.

E26 [mehrdimensionale Felder, Rekursion, Backtracking]

Schreiben Sie eine Funktion `sudokusolve`, die eine unvollständige Sudoku-Matrix (fehlende Einträge durch Wert 0 angegeben) als Argument erhält und versucht, diese zu vervollständigen. Der Funktionswert sollte anzeigen, ob der Versuch erfolgreich war oder nicht.

E27 [`malloc`, `free`, Strings]

Schreiben Sie ein Programm, dass eine natürliche Zahl z und eine weitere natürliche Zahl b zwischen 2 und 36 dezimal einliest und die b-adische Darstellung von z ausgibt. Für die Ziffern der b-adischen Darstellung sollen die Zeichen `0`, `1`, ..., `9`, `a`, `b`, ..., `z` in dieser Reihenfolge benutzt werden. Ermitteln Sie zunächst, welche Länge die b-adische Darstellung von z hat. Dann allokieren Sie Speicher für einen String entsprechender Länge, das die Ziffern aufnimmt, der String soll dann ausgegeben werden. Anschliessend geben Sie den Speicherplatz für den String wieder frei.

E28 [`malloc`, Strings, Datei-Ein- und Ausgabe]

Schreiben Sie ein Programm, das den Inhalt einer Datei in einen String einliest. Dazu wird die Datei zunächst zum Lesen geöffnet und die Zeichen werden gezählt. Obwohl es auch eleganter geht, wird nun wie folgt verfahren: Die Datei wird wieder geschlossen. Dann wird Speicher für den String allokiert, die Datei erneut zum Lesen geöffnet und die einzelnen Zeichen in den String übertragen. Anschliessend wird die Datei geschlossen, der String zur Kontrolle ausgegeben und der Speicher darauf wieder frei gegeben.