

Einführung in die Programmiersprachen C und C++

Prof. Dr. Ulf Rehmann, Fakultät für Mathematik

Übungsblatt 5 (8 Seiten)

Zur Wiederholung wichtiger Spracheigenschaften etliche interessante Programme:

Kalenderprogramme:

1. Bestimmung eines Wochentages zu einem (gregorianischen) Datum:

Die Funktion `schalt(j)` gibt 1 zurück, wenn `j` ein Schaltjahr bezeichnet, sonst 0.

Ein Schaltjahr liegt vor, wenn die Jahreszahl durch 4, aber nicht durch 100 teilbar ist, oder aber durch 400 teilbar ist.

Das Array `int days[]` gibt bei Position `m-1` an, wieviele Tage eines Jahres in einem Nicht-Schaltjahr vor dem 1. des Monats `m` vergangen sind.

Daraus lässt sich leicht die Funktion `int tag-im-jahr(int t, int m, int j)` ableiten, die zu jedem Datum `t, m, j` angibt, um den wievielten Tag im Jahr es sich handelt: nämlich um den Tag mit der Nummer `t + days[m-2]`, wozu man 1 addieren muss, wenn – in einem Schaltjahr – ein Monat nach Februar gefragt ist.

Das wird ausgedrückt durch $(m \leq 2 ? 0 : \text{schalt}(j))$, also 0 für Januar oder Februar, und `schalt(j)` sonst.

Die Funktion `int w.diff(int j, int jj)` bestimmt die Zahl der Tage modulo 7 zwischen dem 1. Januar des Jahres `j` und dem 1. Januar des Jahres `jj`. Daher lässt sich der Wochentag des 1. Januar für jedes Jahr ermitteln, wenn man ihn für ein Jahr kennt (der 1. 1. 1995 war ein Sonntag).

Der Wert der Funktion wird so ermittelt: Gäbe es keine Schaltjahre, so lägen zwischen dem 1.1.j und dem 1.1.jj genau $(j-jj)*356$ Tage, also modulo 7 genau $j-jj$ modulo 7.

Bei gregorianischer Zeitrechnung hätte es seit dem 1.1. im (fiktiven) Jahr 0 bis zum 1.1.j genau $(j-1)/4 - (j-1)/100 + (j-1)/400$ Schalttage gegeben, analog für das Jahr `jj`. Die Differenz der (seit 1.1.0 verflossenen) Schalttage für die Jahre `j` und `jj` ist zu $j - jj$ hinzuzuzählen und das Ergebnis modulo 7 zu nehmen.

Für den Fall eines negativen Ergebnisses ($j < jj$) hat man zu beachten, dass $-a \% u = -(a \% u)$ ist, man also für den Divisor 7 einen Rest zwischen -6 und 0 erhält, also die Anzahl der Tage, die man von Sonntag zurückrechnen muss, um auf den richtigen Wochentag zu kommen, man muss also stattdessen die um 7 erhöhte Zahl nehmen, um die Position des Wochentags im Array `char *tage[]` zu finden.

Aufgabe 5.1: Schreiben Sie ein Programm, das die Anzahl der Tage zwischen zwei gregorianischen Daten ermittelt.

Hinweis: Modifizieren Sie die Betrachtungen zur Funktion `w.diff` geeignet.

Aufgabe 5.2: Der erste Teil des Programms zur Bestimmung des Osterdatums enthält eine einfache Routine zur Bestimmung des Wochentages des 21. März eines gegebenen (gregorianischen) Jahres. Verwenden Sie dies zur Erstellung eines Programms, dass den Wochentag für ein beliebiges Datum ermittelt.

```

#include <stdio.h>

/* Schaltjahr */

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

/* Anzahl der am 1.1., 1.2., 1.3. usw. im Nicht-Schaltjahr verfloßenen Tage */

int days[] = {0,31,59,90,120,151,181,212,243,273,304,334,365};

/* gibt fuer ein Datum die Nummer des Tages im Jahr zurueck */

int tag_im_jahr(int t, int m, int j) {
    return t + days[m-1] + (m <= 2 ? 0 : schalt(j) );
}

/* Berechnet die Anzahl der Tage zwischen dem 1.1 des Jahres j
   und dem 1.1. des Jahres jj modulo 7 (gregorianisch) */

int w_diff(int j, int jj) {
    int d;
    j--; jj--;
    d = (j - jj + j/4 - j/100 + j/400 - jj/4 + jj/100 - jj/400) % 7;
    return d >= 0 ? d : 7+d;
}

char *tage[] = { "Sonntag", "Montag", "Dienstag",
                 "Mittwoch", "Donnerstag", "Freitag", "Sonnabend" };

/* Ausgabe des Wochentages eines gregorianischen Datums */

main() {
    int t,m,j, n;
    while(1) {
        printf(" t m j : ");
        scanf("%d%d%d", &t, &m, &j);
        printf("Wochentag 1. 1. %4d : %s\n",j, tage[ w_diff(j, 1995) ] );
        n = (w_diff(j, 1995) + tag_im_jahr(t,m,j) - 1 ) % 7;
        printf("Wochentag %2d.%2d. %4d : %s\n",t,m,j, tage[n] );
    }
}

```

Aufgabe 5.3: Schreiben Sie ein Programm, dass die Feiertagsdaten eines Jahres bestimmt. Hinweise: Für ein Datum d sei $n(d)$ die Nummer des Tages im betreffenden Jahr, und o das Osterdatum. Dann sind alle Tage mit $(n(d) - n(o)) \% 7 == 0$ Sonntage. Karfreitag, Ostermontag, Himmelfahrt, Pfingstmontag, Fronleichnam ergeben sich als $n(o) - 2$, $n(o) + 1$, $n(o) + 39$, $n(o) + 50$, $n(o) + 60$.

2. Bestimmung des Ostertermins eines Kalenderjahres:

Der Algorithmus ist nach D. Knuth: The Art of Computer Programming, Vol 1, Fundamental Algorithms, 2nd Ed., p. 155-156 formuliert.

Ostern ist (nach einem Beschluss des Konzils von Nicäa, anno 325) der erste Sonntag nach dem ersten Vollmond im Frühling, also nach dem 21. März.

Es sind also Wochentage und Mondphasen zu berücksichtigen.

Die "Epakte" eines Kalenderjahres gibt das "Mondalter" am 1.1. des Jahres an, also die Anzahl der Tage, die am 1.1. seit dem letzten Neumond verflossen sind. Daraus kann man für das Jahr die Mondphasen berechnen.

Nach jeweils 19 Jahren fallen die Vollmondtermine (und Mondphasen) wieder auf das gleiche Datum, daher ist 19 der "Mondzyklus". (Eine "säkulare" Korrektur ist nötig, da diese Regel nicht ganz exakt stimmt.)

Aufgabe 5.4: Verwenden Sie die Epaktenbestimmung aus dem Programm zur Bestimmung des Osterfestes, um mit einem Programm die Mondphasen für irgend ein Kalenderjahr zu ermitteln.

```

/* Ostern ist der erste Sonntag nach dem ersten Vollmond, der am
   21. Maerz oder spaeter erscheint (nicht voellig korrekt, da der
   wahre Stichtermin der Fruehlingsanfang ist) epact < 0 fuer year =
   9006. Siehe z. B.: D. Knuth: The Art of Computer Programming, Vol
   1, Fundamental Algorithms, 2nd Ed., p. 155-156 */

#include <stdio.h>

void ostern_knuth(char *, int);

int main() {
    char s[80];
    int year;
    while(1) {
        printf(" : ");
        scanf("%d", &year);
        ostern_knuth(s,year);
        printf("%s\n",s);
    }
}

void ostern_knuth(char *date, int year) {
    int golden, century;
    int x, z;
    int day, epacte, vollmond, ostern;

    century = year/100 + 1; /* z.B. 1932 --> 20. Jahrhundert) */

    x = 3 * century/4 - 12;
        /* Anzahl der Jahrhundertzahlen ohne 29. Feb. seit 1582
           1600-->0,1700-->1,1800-->2,1900-->3,2000-->3,2100-->4 */

    day = (5 * year/4 - x + 4) % 7; /* Wochentag des 21. Maerz: 0 = Sonntag */
        /* 5*year/4 = year + year/4 */

    golden = year % 19 + 1;

```

```
        /* Goldene Zahl: Jahr in Mondzyklus = 19 Jahre */

z = (8 * century + 5)/25 - 5;
    /* Korrektur ?? Ab 1600:// 0,0,0,1,1,1,2,2,2,3,3,3 */

epacte  = (11 * golden + 20 + z - x) % 30;
    /* Epakte = Alter des Mondes bei Jahresbeginn
       = Anzahl der Tage seit letztem Neumond */

if ( epacte < 0 ) {
    epacte += 30; printf("Funny epacte - year: %d\n",year);
}
/*  >= 0 mod 30 */

if ( (epacte == 25 && golden > 11) || epacte == 24 )
    epacte++;

vollmond = 44 - epacte;      /* 44 Tage = 1 1/2 Monate */
if ( vollmond < 21 )
    vollmond += 30;

ostern = vollmond + 7 - ((day + vollmond) % 7);

if ( ostern > 31 )
    sprintf(date,"%2d.April.%4d", ostern - 31,year);
else
    sprintf(date,"%2d.Maerz.%4d", ostern,   year);
}
```

Zur Entspannung einige Spielprogramme:**Der Turm von Hanoi:**

Ein Turm aus einer Anzahl von der Größe nach geordneten übereinandergelegten Scheiben (größte zuunterst) ist von einer Position ("links") in eine andere Position ("rechts") zu versetzen. Dabei darf nur eine Hilfsposition ("mitte") zur Zwischenlagerung von Scheiben verwendet werden. Bedingung: Nie darf in einer der Positionen eine Scheibe auf eine kleinere gelegt werden. Für das klassische Standardproblem ist $n = 10$.

```
/* titel: turm.c, turm von hanoi */

#include <stdio.h>
#include <ctype.h>

typedef enum { links, mitte, rechts } turm;

main() {
    int scheibenzahl;
    void bewege();
    while (1) {
        printf("\nScheibenzahl (Abbruch mit 0): ");
        while ( scanf("%d", &scheibenzahl) == 0 ) {
            getchar(); printf("\nFalsche Eingabe, bitte Zahl >= 0 eingeben: ");
        }
        if (scheibenzahl <= 0)
            break;
        printf ("%d %s\n",scheibenzahl,"Scheiben erfordern folgende Bewegungen:");
        bewege (scheibenzahl, links, mitte, rechts);
    }
}

void bewege (int anzahl, turm von, turm mittels, turm nach) {
    void bewege_scheibe();
    if ( anzahl == 1 )
        bewege_scheibe (von, nach);
    else {
        bewege ( anzahl - 1, von, nach, mittels);
        bewege_scheibe ( von, nach );
        bewege ( anzahl - 1, mittels, von, nach);
    }
}

void bewege_scheibe ( turm von, turm nach ) {
    void print_turm();
    printf ("Scheibe von "); print_turm (von);
    printf (" nach ");      print_turm (nach);
    printf ("\n");
}

void print_turm ( turm welcher ) {
    switch (welcher) {
        case links : printf ("LINKS "); break;
        case mitte : printf ("MITTE "); break;
        case rechts : printf ("RECHTS"); break;
    }
}
```

Aufgabe 5.5: Schreiben Sie das Programm so um, dass die Anzahl der erforderlichen Bewegungen ausgegeben wird.

5.2 Das n-Damen-Problem: n 'Damen' sind auf einem Schachbrett der Größe n mal n so aufzustellen, dass keine von einer anderen angegriffen ist im Sinne der Regeln der Damenzüge im Schachspiel.

Typische Ein- und Ausgabe:

Welche Zahl (< 20) ? 8

1 5 8 6 3 7 2 4

Dies bedeutet: Die Dame in der ersten Spalte steht in Zeile 1, die in der zweiten Spalte steht in Zeile 5 usw. Die Funktion `int versuche()` implementiert einen "Backtracking"-Algorithmus.

Dabei wird folgendermaßen vorgegangen: Die Damen werden spaltenweise gesetzt. Ist Spalte `i` konfliktfrei besetzt (auf Position `j`), so wird die erste konfliktfreie Position in Spalte `i+1` gesucht, falls keine solche existiert, wird die Position in Spalte `i` durch die nächstmögliche Position ersetzt und wie oben fortgefahren.

Es wird entweder eine Lösung gefunden oder festgestellt, dass keine solche existiert.

```
/* titel: 8.c 8 Damen (und mehr) */

#include <stdio.h>

#define ZZ 20 /* Zahl der Zeilen und Spalten */
int max, DAME[ ZZ ], ZEILE[ ZZ ], DIA0[ 2*ZZ-1 ], DIA1[ 2*ZZ-1 ];

main() {
    int i;
    printf("Welche Zahl ( < %d, Abbruch mit 0) ? ", ZZ);
    while( scanf("%d", &max)==0 ) {
        getchar();
        printf("Falsche Eingabe, bitte Zahl eingeben: ", ZZ);
    }
    if (max <= 0) exit(0);
    if (max > ZZ)
        max = ZZ-1;
    else if (max < 1)
        max = 1;

    if ( versuche(0) )
        for (i=0; i < max; i++)
            printf("%d ", DAME[i]);
        printf("\n");
}

int versuche(int i){
    int j, q;
    for (j = q = 0; q == 0 && j < max; j++) {
        if ( ZEILE[j] == 0 && DIA0[i+j] == 0 && DIA1[i-j+max-1] == 0 ) {
            DAME[i] = j+1;
            ZEILE[j] = DIA0[i+j] = DIA1[i-j+max-1] = 1;
            if (i == max - 1)
                q = 1;
            else
                if ( (q = versuche(i+1)) == 0 )
                    ZEILE[j] = DIA0[i+j] = DIA1[i-j+max-1] = 0;
        }
    }
    return(q);
}
```

Aufgabe 5.6: Modifizieren Sie das Programm, so dass alle möglichen Lösungen ausgegeben werden.

Aufgabe 5.7: Ändern Sie die Ausgabe so ab, dass ein Feld von Charakteren der Größe n dargestellt wird, wobei die Positionen der Damen durch ein '' gekennzeichnet wird.*

5.3 Rösselsprung:

Das folgende Programm berechnet eine Folge von Springer-Zügen im Sinne des Schachspiels, die ein ganzes Schachfeld der Größe $ZZ \times ZZ$ bedeckt derart, dass jedes Feld genau einmal erreicht wird.

Auch hier wird in der Funktion `versuche()` ein Backtracking-Algorithmus implementiert.

```
/* titel: springer.c, Roesselsprung-Programm */

#include <stdio.h>

#define ZZ 8                                /* Feldgroesse */
int z;

int FELD[ ZZ ][ ZZ ];                      /* Feld */
int a[] = { 2, 1, -1, -2, -2, -1, 1, 2 }; /* Koordinaten der */
int b[] = { 1, 2, 2, 1, -1, -2, -2, -1 }; /* Springerzuege */

int xx, yy;

main() {
    z = 4;
    while ( 1 ) {
        printf("Zeilen: ");
        while ( scanf("%d",&z) != 1 ) {
            getchar();
            printf("\nFehlerhafte Eingabe, bitte Zahl < %d eingeben: ", ZZ);
        }
        if ( z <= 1 || z >= ZZ ) exit(0);
        { int i,j;
          for (i=0;i<z;i++)
            for (j=0;j<z;j++)
              FELD[i][j] = 0;
        }
        printf("Anfangswerte xx yy <= %d: ", z);
        while ( scanf("%d %d", &xx, &yy) != 2 ) {
            getchar();
            printf("\nFehlerhafte Eingabe, bitte zwei Zahlen <= %d eingeben: ", z);
        }
        FELD[ xx-1 ][ yy-1 ] = 1;
        if ( versuche( xx-1, yy-1 ) )
            drucke();
        else printf("\nKeine Loesung\n");
    }
}

versuche( int x, int y ) {
    int k, q, u, v;
    for ( k = q = 0; q == 0 && k < 8; k++ ) {
        u = x+a[k]; v = y+b[k];
        if ( 0 <= u && u < z && 0 <= v && v < z && FELD[u][v] == 0 ){
            FELD[u][v] = FELD[x][y] + 1;
```

```
        if ( FELD[u][v] == z*z )
            q = 1;
        else
            if ( (q = versuche( u, v ) ) == 0 )
                FELD[u][v] = 0;
    }
}
return(q);
}

drucke() {
    int i, j;
    printf("\n");
    for (i = 0; i < z; i++) {
        for (j = 0; j < z; j++)
            printf("%3d", FELD[i][j]);
        printf("\n");
    }
}
```

Typische Ein- und Ausgabe:

Zeilen: 6
Anfangswerte xx yy = 1 1

```
  1 16  7 26 11 14
34 25 12 15  6 27
17  2 33  8 13 10
32 35 24 21 28  5
23 18  3 30  9 20
36 31 22 19  4 29
```

Aufgabe 5.8: Modifizieren Sie das Programm derart, dass ein "zyklischer" Rösselsprung gefunden wird, so dass also das erste Feld durch einen Springerzug vom letzten Feld aus erreicht wird.

Aufgabe 5.9: Schreiben Sie eine Version des Programms, die alle zyklischen Lösungen findet.