

## Einführung in die Programmiersprachen C und C++

Prof. Dr. Ulf Rehmann, Fakultät für Mathematik

Übungsblatt 5 (8 Seiten)

### Strukturen

Es können Datentypen strukturiert zusammengefaßt werden zu komplexeren Daten-”strukturen”. Zum Beispiel kann etwa in einem Grafikpaket es sinnvoll sein, Punkte auf dem Bildschirm– durch ihre Pixel-Koordinaten beschrieben– zu einem neuen Datentyp zusammenzufassen. Dies gelingt durch die folgende Deklaration.

```
struct punkt {
    int x;
    int y;
};
```

Damit wird ein neuer Datentyp geschaffen namens **struct punkt**. Es sind dann Deklarationen möglich wie

```
struct punkt a, b, z;
```

usw., syntaktisch analog zu

```
int a, b, c;
```

Die Deklaration

```
struct punkt p;
```

definiert eine Variable **p** vom Typ **struct punkt**. Durch

```
struct punkt p = { 123, 456};
```

wird eine Initialisierung vorgenommen, auf die Komponenten (= ”members”) der Struktur wird durch die syntaktische Konstruktion

```
structure-name.member
```

zugegriffen.

*Aufgabe 5.1: Welchen Wert liefert in diesem Fall `sizeof(struct punkt)` ?*

Im obigen Fall bezeichnet **p.x** die x-Komponente von **p**. Zum Beispiel kann man mit

```
printf(”%d %d”, p.x, p.y);
```

die Koordinaten von **p** ausdrucken, mit

```
double dist, sqrt(double);
...
dist = sqrt((double)p.x * p.x + (double)p.y * p.y);
```

kann man den Abstand zum Punkt (0,0) berechnen.

Strukturen können geschachtelt werden. Durch

```
struct recht {
    struct punkt lu;
    struct punkt ro;
};
```

läßt sich ein Datentyp **recht** deklarieren, der ein (horizontales) Rechteck durch Fixieren des linken unteren und rechten oberen Eckpunktes beschreibt. Nach der Deklaration

```
struct recht screen;
```

bezeichnet z. B. **screen.lu.x** die x-Koordinate der linken unteren Ecke usw.

Strukturen können als Variable an Funktionen übergeben und von ihnen als Wert zurückgegeben werden. Zum Beispiel ist folgendes eine Funktion, die aus zwei **int** ein **p** macht:

```
struct punkt mkpunkt(int x, int y) {
    struct punkt temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

Zwei Punkte können (als Vektoren) addiert werden etwa durch:

```
struct punkt add(struct punkt p1, struct punkt p2) {
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}
```

Durch **struct punkt \*p**; wird ein Zeiger **p** auf Objekte vom Typ **struct punkt** erklärt. **(\*p).x** bezeichnet dann die x-Koordinate eines **struct punkt**-Objektes, auf das **p** zeigt. Äquivalent hierzu ist die Notation **p->x**.

Strukturen können ”selbstreferentiell” sein, d. h. auf Zeigeobjekte des eigenen Typs verweisen.

In Verzeichnis der Beispielprogramme findet sich die Datei **baum.c**, die die Struktur eines Binärbaums implementiert.

```
struct node {
    char *wort;
    int zahl;
    struct node *links;
    struct node *rechts;
};
```

```
typedef struct node knoten;
```

Jedes Objekt **struct node** enthält zwei Zeiger **links**, **rechts** auf ein **struct node**.

Diese Struktur ist geeignet, ungeordnete Daten zu sortieren.

Mit dem zugehörigen Programm kann man die Häufigkeit des Auftretens von Wörtern abzählen.

*Aufgabe 5.2: Studieren Sie die Wirkungsweise des Programms **baum.c**.*

**Beispiel: Komplexe Zahlen**

```

/* complex.c : Komplexe Zahlen, zu uebersetzen mit :
gcc complex.c -lm */

#define PI 3.14159265358979323846L

#include <stdio.h>
#include <math.h>

struct complex {
    double r; double i;
};

typedef struct complex comp; /* 'comp' wird Synonym fuer 'struct complex' */

comp makecomp(double, double);
comp sum(comp, comp);
comp product(comp*, comp*); /* Zeigervariable zur Demonstration */
comp power(comp, int);
void compprint(comp);
double ab(comp); /* Absolutbetrag */
double arg(comp); /* Argument im Bogenmass */
double warg(comp); /* Argument im Gradmass (als "Winkel") */

#define N 16

main() {
    /* Einige Rechnungen mit komplexen Zahlen */
    int k;
    comp arr[N]; /* Array komplexer Zahlen */
    comp basis, result;
    double w = 1/sqrt(2);
    /* basis = makecomp(w,w); 8-te Einheitswurzel */
    basis = makecomp(1,1);
    for (k = 0; k < N; k++) {
        arr[k] = power(basis, k+1);
        compprint(arr[k]);
        printf(" .. Betrag: %6.2f : Argument: %6.2f : Winkel: %6.2f\n",
            ab(arr[k]), arg(arr[k]), warg(arr[k]));
    }
    result = makecomp(0,0);
    for (k = 0; k < N; k++)
        result = sum(result, arr[k]);
    compprint(result);
    printf("\n");
}

comp makecomp(double r, double i) { /* baut komplexe Zahl mit */
    comp u = {r,i}; /* Realteil r, Imaginaerteil i */
    return u;
}

```

```

comp sum(comp a, comp b) { /* Summe zweier komplexer Zahlen */
    comp u;
    u.r = a.r + b.r;
    u.i = a.i + b.i;
    return u;
}

double ab(comp x) {
    return sqrt(x.r*x.r + x.i*x.i);
}

double arg(comp x) {
    if (x.r == 0) return (x.i == 0) ? 0 : ( x.i > 0 ? 0.5*PI : 1.5*PI);
    if (x.r < 0) return atan(x.i/x.r) + PI;
    return (x.i >= 0) ? atan(x.i/x.r) : atan(x.i/x.r) + 2*PI ;
}

double warg(comp x) {
    return arg(x)/PI*180;
}

/* Hier werden zur Demonstration Zeigervariablen fuer die Argumente verwendet */
comp product(comp *x, comp *y) { /* Produkt zweier komplexer Zahlen */
    comp u;
    u.r = x->r * y->r - x->i * y->i; /* -> als Zugriff auf Komponente */
    u.i = x->r * y->i + x->i * y->r;
    return u;
}

comp power( comp b, int n) { /* b hoch n, b komplex */
    comp u = {1, 0}; /* 1 komplex */
    while (n > 0) {
        if (n % 2) {
            n--; u = product(&b, &u); /* Adressen werden uebergeben */
        }
        else {
            n /= 2; b = product(&b, &b);
        }
    } /* beachte: b^{2 k + 1} = (b^2)^k * b */
    return u;
}

void compprint(comp z) { /* Druckt eine komplexe Zahl */
    if (z.r != 0)
        printf("%6.2f",z.r);
    else
        printf("%6c",' ');
    if (z.i > 0)
        printf(" + %6.2f i",z.i);
    if (z.i < 0)
        printf(" - %6.2f i",-z.i);
    if (z.i == 0)
        printf(" %6c ",' ');
}

```

**Zur Entspannung einige Spielprogramme:****Turm von Hanoi, Acht-Damen-Problem (sogar für n Damen), Rösselsprung****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>
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 5.3: 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.

```
/* 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("\nFalsche 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 5.4: Modifizieren Sie das Programm, so dass alle möglichen Lösungen ausgegeben werden.*

*Aufgabe 5.5: 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.

```
/* 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 5.6: 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 5.7: Schreiben Sie eine Version des Programms, die alle zyklischen Lösungen findet.*