Grundlagen d. Informatik f. Ing. bei Prof. Paul

Eine (praktische) Zusammenfassung

Alexander Kort

6. Februar 2007

Dieses Dokument ist für die Leute unter uns bestimmt, die bisher wenig oder gar nicht in die Verlegenheit gekommen sind, etwas selbst programmieren zu müssen. Ich habe mich daher bemüht die Programme einfach zu gestalten und verständlich zu kommentieren. Diese Zusammenfassung entstand auf Eigeninitiative und erhebt keinen Anspruch auf Vollständigkeit und Richtigkeit. Letzteres wird allerdings angestrebt. Ich möchte hiermit eine Möglichkeit schaffen, sich ohne viel Fachchinesisch in die Materie einzuarbeiten. Sollte ich wesentliche Sachverhalte vergessen haben, so bitte ich um eine entsprechende Benachrichtigung. Auch für Verbesserungs- und Korrekturvorschläge wäre ich sehr dankbar.

Inhaltsverzeichnis

1	Sequenzen	3
1.1	Grundlagen	3
1.2	Aufgaben	5
2	Verzweigungen	7
2.1	Grundlagen	7
2.2	Aufgaben	7
3	Schleifen	8
3.1	Grundlagen	8
3.2	Aufgaben	9
4	Felder	10
4.1	Grundlagen	10
4.2	Aufgaben	11
5	Funktionen	11
5.1	Grundlagen	11
5.2	Aufgaben	13
6	Klausuraufgaben	14

1 Sequenzen

1.1 Grundlagen

Zum Einstieg will ich die Grundstruktur eines einfachen *C-Programms* veranschaulichen und erklären.

Grundstruktur eines C-Programms

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(void)
5 {
6
7     //ProgrammCode
8
9     system("PAUSE");
10    return 0;
11 }
```

Die einzelnen Befehle bedeuten folgendes:

- `#include <*.h>` - schließt die angegebene (*.h) Programmbibliothek ein.
- `int main(){}` - definiert eine Funktion des Typs *integer* mit dem Namen `main`. Innerhalb der “()“ stehen in der Regel die an die Funktion übergebenen Werte/Variablen. `void` bedeutet, dass nichts an die Funktion übergeben wird.
- `system("PAUSE");` - unterbricht an dieser Stelle das Programm mit der Aufforderung `Drücken Sie eine beliebige Taste...`
- `return 0;` - gibt normalerweise den Rückgabewert der Funktion aus.

Die einfachsten Befehle für ein *C-Programm* sind die Eingabe- und Ausgabebefehle `scanf("%i", &x);` und `printf("%i", x);`.

Berechnungen können intuitiv in den Quellcode geschrieben werden. Um zum Beispiel die Gleichung $d = a + \frac{b}{c}$ berechnen zu lassen, braucht man nur `d = a + b/c;` einzutippen. Damit das Programm auch ein Ergebnis errechnen kann, müssen wir die Variablen `a`, `b`, `c` und `d` definieren und mit Werten belegen. Die verschiedenen Wege um dieses zu realisieren möchte ich am Beispiel der Variablen `a` bis `c` aufzeigen. Dazu nehmen wir an, dass es sich bei den Variablen jeweils um einen ganzzahlige Werte (Typ *integer*) `int` handelt.

1. Variable können unmittelbar bei der Typdeklaration mit einem Wert belegt werden. Dazu schreiben wir am Anfang der Funktion die Zeile

```
int a=10;
```

Damit deklarieren wir die Variable `a` als Typ *integer* und setzen den Wert der Variablen auf 10.

2. Variablen können im Funktionstext nach der Typdeklaration mit einem Wert belegt werden. Dazu erweitern wir die zuvor geschriebene Zeile zu

```
int a=10, b;
```

Damit deklarieren wir die Variable ebenfalls als *integer*. Nun können wir im weiteren Verlauf des Codes die Zeile

```
b = -3;
```

einfügen. Damit überschreiben wir an dieser Stelle des Programms den Wert der Variablen zu -3 .

3. Um sich nicht beim Programmieren schon Gedanken über die Werte der Variablen machen zu müssen, empfiehlt es sich die Werte der Variablen einzulesen. Dazu müssen wir im Vorfeld wieder den Variablentyp deklarieren, und zwar indem wir die Deklarationszeile um einen weiteren Eintrag erweitern und dann nachfolgend den Wert für die Variable `c` einlesen lassen.

```
int a=10, b, c;
scanf("%i", &c);
```

Mit dem Platzhalter `%i` sagen wir dem Programm, dass genau an dieser Stelle ein Wert vom Typ *integer* eingelesen werden soll und auf der Variablen `c` gespeichert werden soll. Die Speicheraufforderung wird durch `&c` erzeugt.

Um den berechneten Wert `a` ausgeben zu können, müssen wir die Variable ebenfalls zu Beginn der Funktion deklarieren. Da wir in der Gleichung einen Bruch haben, können wir nicht davon ausgehen, dass das Ergebnis ebenfalls einen ganzzahligen Wert annimmt. Aus diesem Grund müssen wir die Variable `a` als *Gleitkommazahl* deklarieren:

```
float d;
```

Anschließend können wir uns das Ergebnis mit dem Befehl

```
printf("%5.2f\n", d);
```

ausgeben lassen. Der Platzhalter `%5.2f` bedeutet in diesem Fall, dass an genau dieser Stelle der Wert der Variablen `a` ausgegeben werden soll. Dabei wird die Ausgabe auf 5 Zeichen mit 2 Nachkommastellen beschränkt. Die Zeichenfolge `\n` erzwingt an genau dieser Stelle einen Zeilenumbruch (Hinweis: `\t` bewirkt einen Tab an dieser Stelle - damit lassen sich einfach Matrizen oder Tabellen erzeugen!).

Das fertige Programm würde dann folgendermaßen aussehen:

1. Beispielprogramm

```
1 #include <stdio.h>
```

```
2 #include <stdlib.h>
3
4 int main(void)
5 {
6     int a=10, b, c;
7     float d;
8
9     b = -3;
10
11     printf("Geben Sie bitte den Wert für 'c' ein: ");
12     scanf("%i", &c);
13
14     d = a + b/c;
15
16     printf("Der berechnete Wert für 'd' beträgt %5.2f.\n", d);
17
18     system("PAUSE");
19     return 0;
20 }
```

1.2 Aufgaben

1. Geben Sie einen Text in zwei Zeilen auf den Bildschirm aus, geben Sie einen weiteren Text aus, nachdem eine Taste gedrückt wurde!
2. Vereinbaren Sie 3 ganzzahlige Variablen m , n und i . Lesen Sie auf m einen Wert von der Tastatur ein und geben Sie ihn wieder aus! Weisen Sie n den Wert 77 zu, berechnen Sie $m + 20$ und weisen Sie den Wert i zu! Berechnen Sie nun $n * i$ und weisen Sie den Wert m zu. Geben Sie die Werte aus! Lesen Sie dann 3 Zahlen auf einmal ein und geben Sie sie aus!
3. Lesen Sie eine reelle Zahl ein, geben Sie sie einmal im Standardformat und einmal mit 10 Zeichen und 2 Nachkommastellen aus! Subtrahieren Sie dann von der eingelesenen Zahl den Kehrwert und geben Sie das Ergebnis ebenfalls in verschiedenen Formaten aus!
(Hinweis: $x = c-d$; $c = x$; ist das gleiche wie $c = c-d$; oder $c -= d$!)
4. Schreiben Sie ein Programm, das Sie nach 3 Zahlen fragt (auch negative Werte sollen erlaubt sein) und dann die Summe der 3 Zahlen ausgibt. Danach soll nach einer weiteren Zahl gefragt werden, mit der die Summe dann multipliziert wird. Das Ergebnis ist ebenfalls auszugeben! Schreiben Sie das Programm einmal so, dass alle Zahlen und Ergebnisse weiter verwendet werden könnten und einmal so, dass möglichst wenige Variablen benötigt werden!
5. Es ist ein Programm zu entwickeln, welches den reellen Variablen A und B verschiedene Werte zuweist, dann die Werte von A und B formatiert ausgibt, anschließend eine Vertauschung der Werte von A und B vornimmt und danach nochmals die Werte dieser beiden Variablen im gleichen Format ausgibt!

6. Testen Sie die Verwendung von Zufallszahlen. Erzeugen Sie je 3 Zufallszahlen zwischen 0 und 50, zwischen -40 und 90 sowie als reelle Zahl zwischen 0 und 1!
(Hinweis: `srand(time(0))` initialisiert den Zufallsgenerator und `rand()%50` liefert eine Zufallszahl ≥ 0 und < 50 !)
7. Gegeben seien die drei Seiten eines Quaders und seine Dichte. Zu berechnen sind das Volumen und die Masse des Quaders. Bei der Ein- und Ausgabe sind die Maßeinheiten anzugeben.
8. Schreiben Sie ein Programm, das eine reelle Zahl einliest und folgende Funktionswerte berechnet und ausgibt: x^2 , x^3 , $\sin(x)$, $\cos(x)$ und $\exp(x)$!
(Hinweis: Verwenden Sie `#include (<)math.h` um die Funktionen zu definieren!)

2 Verzweigungen

2.1 Grundlagen

Um Verzweigungen in einem C-Programm zu realisieren, können sogenannte *if-Anweisungen* verwendet werden.

Dies sieht im Allgemeinen so aus:

```
if(/*Bedingung*/)
    //Anweisung
else if (/*Bedingung*/) //wenn nicht, aber
    //Anweisung
else //ansonsten
    //Anweisung
```

Mögliche Vergleichsoperatoren sind == (gleich), <= (kleiner, gleich), => (größer gleich), < (kleiner als), > (größer als), != (nicht gleich), && (logisches UND) und || (logisches ODER).

Wenn wir also sagen wollen, dass $x = 10$ sei, wenn $y = 5$ und $z \neq -3$ ist und sonst $x = -5$ gelten soll, so können wir folgende Verzweigung nutzen:

Verzweigung 1

```
1 if(y==5 && z!=-3)
2   x=10;
3 else
4   x=-5;
```

Nach diesem Vorgehen lassen sich die folgende Aufgaben lösen.

2.2 Aufgaben

1. Schreiben Sie ein Programm, das einen Wert auf die float-Variable x einliest und $sign(x)$ ausgibt. $sign(x) = 1$ für $x > 0$, $sign(x) = 0$ für $x = 0$ und $sign(x) = -1$ für $x < 0$.
2. Drei Werte sind auf die int-Variablen a , b und c einzulesen und
 - 2.1 aufsteigend der Größe nach auszugeben, ohne die Werte der Variablen zu ändern,
 - 2.2 aufsteigend der Größe nach zu sortieren (kleinste Zahl auf a , nächste auf b usw.) und in der Folge a , b , c auszugeben.
3. Das sogenannte Idealgewicht berechnet sich bei männlichen Personen aus Körpergröße (in cm) - 100 und davon 95%, bei weiblichen Personen Körpergröße (in cm)

- 100 und davon 90%. Schreiben Sie ein Programm, das nach Eingabe von Körpergröße, Gewicht und Geschlecht das Idealgewicht ausgibt und, ob Über-, Unter- oder Idealgewicht vorliegt (mit einer Toleranz für das Idealgewicht von $\pm 2\%$). Bei falschen Eingaben ist ein entsprechender Kommentar auszugeben!
4. Lesen Sie zwei natürliche Zahlen auf a und b ein und stellen Sie fest, ob a durch b und/oder b durch a teilbar ist. Hinweise: a ist durch b teilbar, wenn a Vielfaches von b ist, bzw. wenn $a \% b = 0$ ist! Falls negative Zahlen eingegeben werden, ist das Vorzeichen umzukehren, bei Eingabe einer 0 ist ein entsprechender Text auszugeben!
 5. Schreiben Sie ein Programm, das folgendes Menü anbietet: s - Summe, d - Differenz, p - Produkt, q - Quotient, zwei beliebige Zahlen als Operanden sowie den Buchstaben für das gewünschte Resultat einliest und die entsprechende Aktion realisiert.

3 Schleifen

3.1 Grundlagen

Um in einem Programm gleiche Abläufe wiederholt ablaufen zu lassen, benutzt man in der Regel Schleifen. Unter *C* stehen uns drei Schleifentypen mit den folgenden (Bsp.)Strukturen zur Verfügung:

for-Schleife

```
for (i=0; i<5; i++)
{
    //Anweisung
}
```

Solange die Zählvariable *i* kleiner als fünf ist, wird die Anweisung wiederholt. Durch den Befehl *i++* wird *i* bei jedem Durchlauf um eins erhöht. Dieser Schleifentyp empfiehlt sich bei einer definierten Anzahl von Schleifendurchläufen.

while-Schleife

```
while (i<5)
{
    //Anweisung
    i++;
}
```

Bei diesem Typ wird die Anweisung sooft wiederholt, solange die Zählvariable *i* kleiner als fünf ist. Allerdings muss bei diesem Schleifentyp innerhalb des Schleifenkörpers die Zählvariable erhöht werden, bzw. eingelesen werden. Dieser Schleifentyp empfiehlt sich bei Algorithmen, bei denen eine Anweisung solange durchgeführt werden soll, bis ein bestimmter Wert eingegeben wurde, bzw. solange ein bestimmter

Wert eingegeben wurde. Außerdem kann mit diesem Typ auf einfache Art die Anzahl der Schleifendurchläufe bestimmt werden. Diese Schleife ist *kopfkontrolliert*, das heißt, dass vor dem ersten Durchlauf die Bedingung geprüft wird und dadurch die Anweisung innerhalb der Schleife gar nicht ausgeführt wird.

```
do-while-Schleife    do
                      {
                        //Anweisung
                        i++;
                      }
                      while (i<5)
```

Dieser Schleifentyp gleicht vom Prinzip dem der *while*-Schleife, nur ist diese *fußkontrolliert*, das heißt, dass die Anweisung innerhalb der Schleife mindestens einmal ausgeführt wird, da erst nach dem Schleifendurchlauf die Bedingung geprüft wird.

3.2 Aufgaben

1. Schreiben Sie ein Programm, das eine natürliche Zahl auf die Variable n einliest und dann die Summe der Zahlen von $1 + 2 + \dots + n$ berechnet, und zwar je einmal mit einer *do*-, einer *for*- und einer *while*-Schleife. Falls für n eine negative Zahl eingelesen wird, ist mit deren Betrag zu arbeiten, falls $n=0$ eingegeben wird, ist die Summe 0 auszugeben! Für jede Schleife soll eine neue Zahl eingelesen werden!
2. Aus einer Menge von k rationalen Zahlen (k einlesen) sind die größte und die kleinste zu bestimmen und auszugeben. Das Programm soll solange wiederholt werden, bis für die Anzahl k eine 0 eingegeben wird.
3. Es sind beliebige rationale Zahlen einzulesen, bis 9999 eingegeben wird. Bestimmen Sie die Anzahl, die Summe, die größte und die kleinste der Zahlen!
4. Bei einem Leichtathletik-Wettkampf werden die Anzahl der Teilnehmer und von jedem Teilnehmer die im Weitsprung erreichte Weite eingegeben. Bestimmen Sie den Sieger und seine Weite, sowie die mittlere Weite (arithmetisches Mittel) aller Teilnehmer an diesem Wettkampf!
5. Eine Wetterstation mißt täglich die Temperatur in Grad Celsius. Bestimmen Sie für n Tage (n einlesen) die mittlere Temperatur, welcher Tag der kälteste war und wie kalt es an diesem Tag war! Die Temperaturen (nicht gerundet) sind einzulesen, falsche Eingaben sind abzuweisen. Sollte die minimale Temperatur mehrfach auftreten, ist der letzte Tag anzugeben!
6. Für die Auswertung einer Leistungskontrolle werden die Anzahl der Studenten sowie die (ganzzahligen) Noten eines Fachs eingelesen. Es sind der Notendurchschnitt und die Häufigkeitsverteilung der Noten zu berechnen und auszugeben. Falsche Eingaben sind abzuweisen.
7. Verändern Sie das Programm so, dass nicht nach der Anzahl der Studenten gefragt wird, und dass entweder die Note oder die Zeichen 'E' für entschuldigt bzw. 'U'

für unentschuldigt eingegeben werden. Abgeschlossen wird die Eingabe mit dem Zeichen '*'. Auszugeben sind dann die Anzahl der Arbeiten, der Notendurchschnitt, die Anzahl der entschuldigt und die Anzahl der unentschuldigt Fehlenden!

8. In einem Geschäft wurden an einem Tag von mehreren Artikeln 'anz' Stück bei einem Preis von 'p' EUR verkauft, beides ist jeweils einzulesen. Bei Eingabe negativer Werte ist die Eingabe zu wiederholen! Die Eingabe wird beendet, wenn als Preis 0 eingegeben wird! Es sind zu ermitteln: der Gesamtumsatz des Geschäftes an diesem Tag, die Anzahl der verschiedenen Artikel, die Anzahl der verkauften Artikel, die im Mittel verkaufte Stückzahl sowie der billigste Artikel und sein Preis.

4 Felder

4.1 Grundlagen

Nachdem wir jetzt Schleifen kennengelernt haben, werde ich nun eine Hauptanwendung von Schleifen vorstellen: *Felder*. Felder sind eine besondere Form von Variablen. Ein Feld besteht aus dem Feldnamen und aus mindestens einem Element. Felder müssen ebenso wie Variablen zu Beginn des Programms deklariert werden:

```
int feld1[m][n];
```

Mit dieser Zeile deklarieren wir ein zweidimensionales Feld mit dem Namen `feld1` und `m` Zeilen und `n` Spalten. Um die einzelnen Datenpunkte des Feldes mit Werten zu belegen, schreiben wir die folgenden Zeilen in unser Programm:

```
1 for(i=0;i<m;i++)
2 {
3     printf("Geben Sie bitte die Werte der %i. Zeile ein: ", i+1);
4     for(j=0;j<n;j++)
5     {
6         scanf("%f", &feld1[i][j]);
7     }
8 }
```

Diese Zeilen bewirken, dass innerhalb der zweiten Schleife die Plätze 0–n (Spaltenwerte) mit Werten belegt werden. Welche Zeile des Feldes belegt wird, hängt von der ersten Schleife ab. Diese wird jedoch erst gewechselt, wenn die innere Schleife durchlaufen ist.

Mit diesem Wissen können die folgenden Aufgaben gelöst werden.

4.2 Aufgaben

1. Erzeugen Sie zuerst zwei n-dimensionale Vektoren mit Zufallszahlen. Berechnen Sie dann die Summe und das Skalarprodukt der beiden Vektoren. Geben Sie danach alle 3 Vektoren und das Skalarprodukt aus!
2. Simulieren Sie eine Lottoziehung, indem Sie 6 Zufallszahlen ≥ 1 und ≤ 49 erzeugen und sie der Größe nach ausgeben, wobei keine doppelt auftreten darf! (Hinweis: Lottoschein als Feld mit 49 Elementen!)
3. Eine Einheitsmatrix mit n Zeilen und Spalten ist zu erzeugen und anzuzeigen (n einlesen, $n < 40$)!
4. Eine Matrix vom Format (m,n) ist mit Zufallszahlen zu belegen und auszugeben (m, n einlesen, ≤ 10)! Danach ist die transponierte Matrix zu bilden und auszugeben!
5. Erzeugen Sie zwei Matrizen vom Format (m, n ≤ 20). Geben Sie die beiden Matrizen, ihre Summe und ihre Differenz aus!
6. Durch Hinzufügen von Zeilen- und Spaltensummen ist aus einer Matrix vom Format (m,n) eine Matrix vom Format (m+1, n+1) zu erzeugen und auszugeben!

5 Funktionen

5.1 Grundlagen

Mit dem bis jetzt erworbenen Wissen können wir schon eine große Anzahl von Aufgabentypen mit einem C-Programm lösen.

Was uns noch fehlt, sind Algorithmen, die uns erlauben, bestimmte Aufgaben innerhalb verschiedener Schleifen durchzuführen. Hier kommen die *Funktionen* ins Spiel. Wie ich bereits im Abschnitt 1.1 erklärt habe, sind unsere bis jetzt erstellten Programme bereits eine Funktion. Damit kennen wir im Prinzip auch schon den Grundaufbau einer Funktion:

Beispielfunktion

```
1 int fkt1(int x)
2 {
3     int y;
4     y = x*x;
5     return y;
6 }
```

Diese Funktion brechnet aus dem übergebenen ganzzahligen Wert x das Quadrat der Zahl und speichert den Wert auf y . Der ganzzahlige Wert von y wird von der Funktion durch den Befehl `return y;` wieder zurückgegeben. (Anstatt von einzelnen Werten können auch

Felder *an* eine Funktion übergeben, jedoch nicht *von* einer Funktion zurückgegeben werden!) Um diese Funktion im Hauptprogramm (-funktion) verwenden zu können, müssen wir sie nur aufrufen. Dies kann durch die Befehlszeile

```
y = fkt1(x);
```

erfolgen. Anhand dieses Beispiels sehen wir auch sofort, dass die Variablenbezeichnung unabhängig ist. Es ist also egal, wie die Variablen innerhalb der Funktionen heißen, da jeweils nur die Werte übergeben werden. Anhand des nächsten Beispiels möchte ich die Verwendung von Funktionen aufzeigen. Ich möchte mit diesem Programm mehrere ganze Zahlen einlesen und ihnen ihr Quadrat zuordnen. Anschließend möchte ich die eingelesenen Werte zusammen mit ihren Quadraten ausgeben.

Beispielprogramm

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int zahl()
5 {
6     int y;
7
8     printf("Bitte geben Sie die Anzahl der Zahlen ein: ");
9     scanf("%i", &y);
10
11    printf("\n\n\n");
12
13    return y;
14 }
15
16 int quad(int x)
17 {
18     x*=x; //ist gleich x = x*x; bzw. y = x*x; x = y;
19
20     return x;
21 }
22
23 int main(void)
24 {
25     int anzahl=zahl(), i, z;
26     int quads[anzahl][anzahl];
27
28     for(i=0;i<anzahl;i++)
29     {
30         printf("Geben Sie die %2i. von %2i Zahlen ein: ", i+1,anzahl);
31         scanf("%2i", &quads[i][i]);
32
33         quads[i][i+1]=quad(quads[i][i]);
34     }
35
36     for(i=0;i<anzahl;i++)
37     {
```

```

38     printf("\nDie %2i. eingegebene Zahl war %2i und die Quadratzahl dazu ist %3i.",
39         i+1, quads[i][i], quads[i][i+1]);
40 }
41
42 printf("\n\n");
43
44 system("PAUSE");
45 return 0;
46 }

```

In diesem etwas komplexeren Beispiel sind alle Elemente vorhanden, die wir im vorhergehenden besprochen haben. Ich möchte nun erläutern, was die einzelnen Elemente bedeuten. Die erste Funktion (`zahl`) liest eine ganze Zahl `y` ein, welche dann von der Funktion übergeben wird. An die zweite Funktion (`quad`) wird eine ganze Zahl `x` übergeben und mit sich selbst multipliziert (quadratiert) und das Ergebnis wieder auf `x` abgespeichert. Anschließend wird der neue ganzzahlige Wert von `x` von der Funktion übergeben. Die dritte Funktion (`main`) ruft während der Variablendeklaration die Funktion `zahl` auf, um damit den Wert für `anzahl` festzulegen. Durch diese Festlegung wird der Speicherplatz für das *integer*-Feld `quads` definiert. Die Feldeinträge werden in ersten `for`-Schleife gesetzt. Die Werte der ersten Spalte werden eingelesen. Die zugehörigen Quadratzahlen werden durch den Aufruf der Funktion `quad` berechnet und in der jeweiligen Zeile, aber der zweiten Spalte abgelegt. Nach dem Berechnen und Speichern der Quadratzahlen werden die Werte durch die zweite `for`-Schleife gesammelt ausgegeben.

Damit haben wir auch das letzte Klausurrelevante Thema abgeschlossen und können nun die folgenden als auch die im Anschluss gestellten Klausurfragen lösen.

5.2 Aufgaben

1. Definieren Sie eine Funktion, welche die Summe der Zahlen von 1 bis zu einer vorgegebenen Zahl `k` liefert. Im Hauptprogramm sind innerhalb einer Schleife jeweils eine Zahl einzulesen, die Funktion aufzurufen und der Funktionswert auszugeben bis eine 0 eingegeben wird!
2. Ein Anfangskapital soll für `n` Jahre auf einer Bank fest angelegt werden. Formulieren Sie eine Funktion `endkap(anfkap, zins, n)`, die das Endkapital nach `n` Jahren berechnet und zurückgibt. Die Zinsen werden jährlich gutgeschrieben und $endkap(n) = endkap(n-1) * (1 + zins)$ wenn $endkap(0) = anfkap$. Stellen Sie mit Hilfe der Funktion `endkap` tabellarisch dar, wie sich das Guthaben (Endbeträge) bei Zinssätzen von 3,00; 3,25; 3,50; 3,75 und 4,00 Prozent im Verlauf von 10 Jahren entwickelt! Startkapital seien 1000,-EUR.
3. Erstellen Sie folgende Funktionen 'quer', 'mantel' und 'volumen', die aus Durchmesser und Länge eines Zylinders den Querschnitt bzw. die Mantelfläche und aus Querschnitt und Länge das Volumen ermitteln. Für eine Welle aus mehreren Zy-

lindern sind der Durchmesser und die Länge der verschiedenen Abschnitte (max. 10, Abbruch bei 0) einzulesen. Lassen Sie jeweils den Querschnitt, die Mantelfläche und das Volumen mit den o.g. Funktionen berechnen. Geben Sie alle Daten als Tabelle aus!

(Hinweis: $\pi=3.141593$; $m= \pi*d*l$; $q= \pi/4*d*d$; $v=q*l$)

4. Bei einem Motorentest wurden für jeden Motor die Drehzahl, der Benzinverbrauch und die Temperatur gemessen und in die Felder d, b und t gespeichert. Verwenden Sie die Funktion mini, welche aus einem Feld von n Elementen den Wert des kleinsten bestimmt, um für jedes dieser Felder den kleinsten Wert zu bestimmen! Die Anzahl der getesteten Motoren und die Meßwerte sind mit Hilfe der Funktion daten einzulesen!

(Hinweis: Es können auch Felder an Funktionen übergeben, jedoch nicht zurückgegeben werden!)

6 Klausuraufgaben

Eine willkürliche Auswahl von Klausuraufgaben ab 2001:

1. Die gemessene Niederschlagsmenge (in mm) ist für einen Monat (30 Tage) einzulesen. Falsche Eingaben (negative Werte) sind zu wiederholen! Wie war die höchste Niederschlagsmenge? Wie oft und an welchen Tagen trat sie auf? Wie groß war die durchschnittliche Niederschlagsmenge?
2. Definieren Sie eine Funktion 'faku(k)= k!', welche zu einer vorgegebenen natürlichen Zahl k den Wert k! berechnet, ohne Ausgabe auf den Bildschirm und ohne globale Variablen zu verwenden. Schreiben Sie ein Programm, das eine natürliche Zahl n einliest und mit Hilfe der o.g. Funktion (n-1)!, n! und (n+1)! berechnet und dieses solange wiederholt, bis für n eine 0 eingelesen wird! Bei Null ist das Programm zu beenden, negative Zahlen sind nicht zu verwenden!
(Hinweise: $0! = 1$ und $k! = 1 * 2 * \dots * k$ für $k > 0$)
3. In ein eindimensionales Feld sind n Messwerte z_i einzulesen (n variabel, $n \leq 100$). Die Werte z_i treten nicht mehrfach auf! Von den Messwerten z_i sind der Mittelwert m zu berechnen

$$m = \frac{1}{n} \sum_{i=1}^n z_i$$

sowie die größte Zahl und ihre Position innerhalb des Feldes zu bestimmen. Geben Sie die Resultate übersichtlich auf dem Bildschirm aus!

4. Die Koordinaten von n Punkten $P_1(x_1, y_1), \dots, P_n(x_n, y_n)$ sind über Tastatur einzulesen und in zwei Feldern x und y abzuspeichern (n variabel, $n \leq 100$). Schreiben Sie eine Funktion abstand, die die Länge der Geraden zwischen zwei

Punkten nach der angegebenen Formel berechnet. Parameter sind x_1, x_2, y_1, y_2 .

$$A = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Die Abstände sind in einem Feld A abzuspeichern. Die Länge des durch die n Punkte gelegten n-Ecks (Polygons) ist mit Hilfe dieser Funktion zu berechnen (wiederholter Aufruf von `abstand!`). Schreiben Sie eine weitere Funktion `mittel`, die ein Feld und die Anzahl der Elemente übergibt und den Mittelwert der Elemente sowie das kleinste Element zurückgibt. Wenden Sie diese Funktion auf das Feld A an und geben Sie die zurückgegebenen Werte Mittelwert der Abstände bzw. kleinster Abstand auf dem Bildschirm aus!

5. Erstellen Sie eine Funktion 'volumen', die aus Durchmesser und Länge das Volumen eines(!) Zylinders ermittelt (Hinweis: $\pi = 3,1415926$, $V = \frac{\pi}{4}D^2L$), sowie eine Funktion 'lesen', die einen 'float'-Wert einliest, die Eingabe wiederholt solange er ≤ 0 ist, und ihn dann zurückliefert. Für eine Welle sind der Durchmesser und die Länge der verschiedenen Abschnitte jeweils mit Hilfe der Funktion 'lesen' einzulesen und (getrennt) zu speichern, Das Volumen der Abschnitte ist mit Hilfe der Funktion 'volumen' zu berechnen! Bestimmen Sie das Gesamtvolumen und den größten Durchmesser! Die Anzahl der Abschnitte ist einzulesen, soll aber nicht > 100 sein (absichern)!
6. Schreiben Sie ein Programm, das für einen Autohändler folgendes Menü wiederholt anbietet und mit Hilfe der Funktionen 'neu', 'tabelle' und 'senken' realisiert bis eine 0 eingegeben wird:
 - 0** Ende
 - 1** ein neues Auto mit Typ, Baujahr und Preis erfassen,
 - 2** alle Daten als Tabelle auflisten und
 - 3** den Preis aller Autos um p% senken (p einlesen).
7. Schreiben Sie ein Programm, das zu der Datenstruktur 'wein' folgende Funktionen für n Weine ($n \leq 200$) testet:
 - * einen neuen Wein mit Name, Jahrgang, Bestand und Preis erfassen,
 - * alle Daten eines Weins ausgeben,
 - * den Jahrgang des jüngsten und des ältesten Weines liefern,
 - * zuerst alle Daten der Weine des jüngsten und dann die des ältesten Jahrgangs in einer Tabelle auflisten.
8. Schreiben Sie ein C-Programm mit folgenden Funktionen:
 - 'volumen' berechnet aus dem Durchmesser das Volumen einer Kugel,
 - 'lies' liest einen float-Wert ein und wiederholt die Eingabe, wenn bzw. solange der Wert < 0 ist,
 - 'mittel' berechnet für die n Elemente eines Feldes den Mittelwert,
 - 'mini' berechnet für die n Elemente eines Feldes das Minimum.

Die Funktionen sollen die berechneten bzw. eingelesenen Werte als Wert zurückliefern. Im Hauptprogramm sind für 10 Kugeln mit Hilfe der o.g. Funktionen der Durchmesser einzulesen, deren Volumen zu berechnen und der mittlere Durchmesser sowie das mittlere Volumen zu berechnen und anzuzeigen, sowie deren minimale Werte.

(Hinweis: Es genügt eine der Funktionen 'mittel' und 'mini'. Werden beide richtig angegeben und verwendet, so gibt es 4 Zusatzpunkte. $\pi = 3.1415926$, $V = \frac{\pi}{6}d^3$)