Name, Vorname, Unterschrift:	Matrikelnummer:

Klausur
Kompaktkurs Einführung in die Programmierung
(Dr. S. Zimmer)
17. März 2008

Wichtig:

- Die Angabe besteht aus 6 Seiten — prüfen Sie bitte Ihr Exemplar auf Vollständigkeit.
- Kennzeichen Sie jedes Blatt lesbar mit Ihrem Namen und dieses Deckblatt zusätzlich mit Ihrer **Matrikelnummer und Unterschrift**; trennen Sie die Heftung nicht auf.
- Bearbeiten Sie die Aufgaben auf den Angabenblättern. Sollte der Platz für einen Aufgabenteil nicht reichen, benutzen Sie bitte die Rückseite der Angabenblätter und machen Sie einen entsprechenden Vermerk bei der Aufgabe.
- Insgesamt können Sie 40 Punkte erreichen, bei jeder Aufgabe sind die für die jeweiligen Teilaufgaben vorgesehenen Punkte angegeben. Zum Bestehen sind 17 Punkte erforderlich.
- Als Hilfsmittel ist nur ein Ausdruck der Vorlesungsfolien erlaubt (handschriftliche Anmerkungen sind OK).
- Die Bearbeitungszeit beträgt 75 Minuten.

Aufgabe	1a)	1b)	2a)	2b)	2c)	2d)	3a)	3b)	Summe	Note
Punkte	/8	/7	/3	/3	/4	/5	/3	/7	/40	

1 Felder (8+7=15 Punkte)

Zum Speichern von n nichtnegativen ganzen Zahlen wird in dieser Aufgabe ein Feld von $n + 1$ `int` verwendet, wobei die letzte Komponente auf -1 gesetzt wird, um das Ende des Feldes zu markieren. Wenn man die drei Werte 1, 2 und 3 speichern möchte, hätte man also das Feld `{1,2,3,-1}`.

a) Schreiben Sie zwei Funktionen mit jeweils einem solchen Feld als Parameter und Ergebnistyp `bool`; bei der ersten Funktion soll das Ergebnis genau dann `true` sein, wenn *wenigstens eine* Komponente eine gerade Zahl ist, bei der zweiten soll das Ergebnis genau dann `true` sein, wenn *alle* Komponenten gerade sind — die Endemarkierung -1 zählt dabei natürlich nicht mit.

b) Nun soll eine Funktion `aussuchen` mit zwei Feldern v und w als Parameter geschrieben werden, die ein Feld als Ergebnis hat, das diejenigen Komponenten von v enthält, deren Indizes w angibt. Alle drei Felder (Parameter und Ergebnis) sollen die oben beschriebene Struktur (-1 als Endemarkierung) haben.

Wäre v etwa das Feld `{10,11,12,13,-1}` und w das Feld `{3,0,-1}`, sollte das Ergebnis das Feld `{13,10, -1}` sein (genau genommen, wie immer, ein Zeiger auf die erste Komponente).

Wenn einer der in w angegebenen Indizes einen unzulässigen Wert hat (im Beispiel etwa 4, was zu groß für dieses v ist), soll kein Feld, sondern der Nullzeiger zurückgeliefert werden.

Dazu dürfte es hilfreich sein, die Komponenten von v und von w zu zählen; man sollte prüfen, ob alle Komponenten von w im zulässigen Bereich sind; in diesem Fall Speicher für ein neues Feld passender Länge reservieren und schließlich die Werte kopieren.

2 Strukturtypen (3+3+4+5=15 Punkte)

Gegeben sei der Strukturtyp

```
struct zahl {
   int mantisse;
   int exponent;
};
```

zur Speicherung von rationalen Zahlen als *mantisse* $\cdot\, 2^{exponent}$. Schreiben Sie folgende Funktionen (soweit nichts anderes gesagt, brauchen Sie sich nicht um Fälle zu kümmern, in denen durch eine zu große oder zu kleine Zahl der Bereich der Datentypen `int` bzw. `double` verlassen würde):

a) `durchZwei` soll einen Parameter `struct zahl x` haben, das Funktionsergebnis – ebenfalls `struct zahl` – soll eine Darstellung von $x/2$ enthalten.

b) `alsZahl` soll einen `int`-Parameter x haben und eine Darstellung von x im Datentyp `struct zahl` als Funktionsergebnis liefern.

c) `alsDouble` soll einen Parameter `struct zahl x` haben und dessen Wert als Funktionsergebnis liefern (Typ `double`). Hierbei soll *keine* Funktion zum Potenzieren verwendet werden.

Ein Beispiel mit den bisher vorkommenden Funktionen: `alsDouble(durchZwei(alsZahl(5)))` würde die `double`-Zahl 2.5 liefern.

d) Schreiben Sie nun noch eine Funktion `istGleich` mit zwei Parametern vom Typ `struct zahl` und einem Wahrheitswert als Funktionsergebnis, der genau dann `true` sein soll, wenn beide Parameter dieselbe rationale Zahl darstellen. Hier dürfen Sie *nicht* davon ausgehen, dass sie die Parameter auch als `double` darstellen können, Verwendung der Funktion `alsDouble` scheidet also aus. Beispiel: `istGleich(alsZahl(5), durchZwei(alsZahl(10)))` sollte `true` ergeben.

3 Listen (3+7=10 Punkte)

Um in einem Text zu zählen, welche Zeichen wie oft vorkommen (z.B., dass `"aababcabcd"` aus 4 `'a'`, 3 `'b'`, 2 `'c'` und 1 `'d'` besteht), soll für jedes vorkommende Zeichen in einem Strukturtyp gespeichert werden: das Zeichen selbst (`char`), seine Häufigkeit (`int`) und die notwendige Information zum Aufbau einer einfach verketteten Liste mit Elementen dieses Strukturtyps. Im Beispiel hätte die Liste also 4 Elemente (je eins für `'a'`, ..., `'d'`), die Häufigkeiten wären 4, 3, 2 bzw. 1.
Eine Liste wird wie üblich durch einen Zeiger auf das erste Element repräsentiert (`NULL` für die leere Liste).

a) Geben Sie einen passenden Strukturtyp `struct element` an:

b) Schreiben Sie eine Funktion `einfuegen` mit einem Zeichen c und einer Liste als Parameter; falls c in der Liste bereits vorkommt, soll im entsprechenden Listenelement die Anzahl um eins erhöht werden, andernfalls soll ein neues Element hinten an die Liste angehängt werden (mit c als Zeichen und Anzahl 1). Das Funktionsergebnis soll die veränderte Liste sein.

   Tipp: rekursiv wird diese Funktion m.E. übersichtlicher als mit einer Schleife!