Teilklausur I in Informatik – TEL09GR2 – Lewandowski

Es sind insgesamt 84 Punkte zu erreichen (80 Punkte werden als 100% gewertet)
(Jeder Punkt entspricht einer Bearbeitungszeit von etwa 1 Minute)

Aufgabe 1: Schaltjahr auf Rigel 7 (12 Punkte) – Struktogramm
Aufgabe 2: Gut gefüllt (34 Punkte) – DF, PA, C/C++
Aufgabe 3: Datenhierarchie und Modularisierung (20 Punkte) – DH, C/C++(.h)
Aufgabe 4: Einfach verkettete Ringlisten (18 Punkte) – C/C++

Bitte kennzeichnen Sie jedes Blatt lesbar mit Ihrem Namen und Ihrer Matrikelnummer

Zugelassene Hilfsmittel: Keine

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

Aufgabe 1 — 12 Punkte

(12 Punkte) **Schaltjahr auf Rigel 7:** Auch auf anderen Planeten gibt es Schaltjahre – auf Rigel 7 gelten folgende Regeln

- Regel 1: ist das Jahr ganzzahlig durch 7 teilbar, so ist es immer ein Schaltjahr
- Regel 2: von den verbleibenden Jahren sind die ganzzahlig durch 5 oder ganzzahlig durch 12 teilbaren Jahre Schaltjahre (Regel 2a), mit Ausnahme der ganzzahlig durch 60 teilbaren Jahre (Regel 2b)

Somit sind z. B. die Jahre 28 und 42 aufgrund der Regel 1 Schaltjahre, ebenso sind die Jahre 15 und 24 (Regel 2a) Schaltjahre. Das Jahr 120 ist hingegen kein Schaltjahr (Regel 2b). Die Regel 1 wird stets zuerst angewendet (so ist das Jahr 420 ein Schaltjahr trotz Regel 2b).

Vervollständigen Sie das Struktogramm auf der Folgeseite, sodass die Ausgabe „Ja“ erfolgt, wenn das eingegebene Jahr ein Schaltjahr auf Rigel 7 ist, und „Nein“, wenn es kein Schaltjahr ist. Sie dürfen die Anweisung zur entsprechenden Ausgabe mit „`"Ja"`“ bzw. „`"Nein"`“ abkürzen.

Das Struktogramm soll so detailliert sein, dass eine Umsetzung 1:1 in ein C/C++-Programm möglich wäre, d. h., dass alle Fallunterscheidungen und Schleifen (wenn benötigt) auch im Entwurf als solche vorkommen müssen. Beachten Sie: Eine Umsetzung in C/C++ ist hier *nicht* gefordert.

```
Name    : SchaltjahrRigel7
Eingabe : Jahr (die zu untersuchende Jahreszahl)
Ausgabe : "Ja"/"Nein" (nur auf der Konsole)

Jahr - ganzzahliger Wert
```

```
Lese Jahreszahl Jahr ein
```

Der Platz ist hier großzügig bemessen :-)

Aufgabe 2 — 34 Punkte

(2+18+14 = 34 Punkte) **Gut gefüllt:** In dieser Aufgabe ist eine Funktion `fuellEinsUm(.,.)` im Kontext eines Füllproblems zu entwerfen und in C- oder C++-Code umzusetzen.

Die Funktion soll dabei zwei Arrays als Parameter bekommen: Das eine Array enthält Werte, die die derzeitige Befüllung (in Liter) einer Reihe von Gefäßen angeben, ein zweites Array enthält Werte, die die Kapazitäten (in Liter) der betreffenden Gefäße angeben. Die Funktion soll zunächst das erste nicht-leere Gefäß in dem Array finden. Dieses soll – wenn möglich – nach folgenden Regeln in ein anderes Gefäß gefüllt werden, sodass

- der komplette Inhalt in das Zielgefäß passt und
- wenn es mehrere solche Zielgefäße gibt, dasjenige Gefäß gewählt wird, sodass der freie Platz in dem Zielgefäß nach dem Umfüllprozess möglichst gering ist;
- gibt es kein solches Gefäß (keines der Gefäße hat ausreichend Platz oder keines hätte nach dem Umfüllen weniger Restkapazität als das Ausgangsgefäß vor dem Umfüllen), so bleiben die Inhalte aller Gefäße unverändert.

Gegeben sind die C/C++-Definitionen:

```
#define N 7
typedef unsigned int Vol[N];
```

Ein Beispiel: seien zwei Arrays wie folgt definiert:

```
Vol inh = {0,0,2,7,4,5,1};
Vol kap = {6,5,7,8,9,9,6};
```

Das erste nicht-leere Gefäß ist das Gefäß mit Index 2 (das dritte im Array). Dieses kann in alle anderen Gefäße außer dem mit Index 3 (Inhalt 7, Kapazität 8) gefüllt werden. Es würde in das vorletzte Gefäß (mit Index 5) gefüllt werden, da hier nach dem Umfüllen nur noch Platz für weitere 2 Liter bleiben (bei allen anderen Gefäßen würden mindestens 3 weitere Liter Platz haben).

Beispiel 2: Hätte im obigen Beispiel das erste Gefäß die Kapazität 2 oder 3, so würde das Gefäß mit Index 2 in dieses erste Gefäß umgefüllt werden (kleinste Restkapazität von 0 bzw. 1).

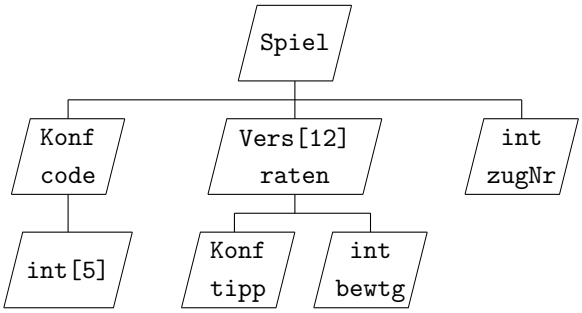
Beispiel 3: Wäre der Inhalt des Gefäßes mit Index 2 groß gewesen, z. B. 7 Liter, so hätte es kein Gefäß gegeben, in das dieses komplett umgefüllt hätte werden können. Die Inhalte der Gefäße würden unverändert bleiben. Ebenso hätte man nicht umgefüllt, wenn der Inhalt 2 Liter und die Kapazität 3 Liter betragen hätte, da dann die Restkapazität (1 Liter) geringer wäre als die eines beliebigen anderen Zielgefäßes nach dem Umfüllen.

Der Aufruf `fuellEinsUm(inh,kap)` soll das Array `inh` so verändern, dass entweder das erste nicht-leere Gefäß nach obigen Regeln umgefüllt wird oder – wenn dies nicht möglich ist (oder es kein nicht-leeres Gefäß gibt) – die Ausgabe „`Umfuellen ist nicht moeglich!`“ erfolgen.

Geben Sie das Datenflussdiagramm (DF) der Ebene 0 an (Funktion als ein Kästchen) (2 Punkte), sowie einen Programmablaufplan für die von Ihnen entworfene Funktion mit den nötigen Verfeinerungen (18 Punkte). Der Entwurf soll so detailliert sein, dass eine Umsetzung 1:1 in ein C/C++-Programm möglich ist, d. h., dass alle Fallunterscheidungen und Schleifen auch im Entwurf als solche vorkommen müssen. Beachten Sie: die Parameter sind an die Funktion zu übergeben und werden von dieser zurückgeliefert, Sie müssen sich *nicht* um die Ein- und Ausgabe der Werte kümmern. Geben Sie schließlich noch die Umsetzung der Funktion in C/C++ an (14 Punkte).

Aufgabe 3 — 20 Punkte

(20 Punkte) **Datenhierarchie und Modularisierung:** Gegeben ist folgendes Datenhierarchiediagramm und eine Liste von Funktionen, die mit diesen Datentypen arbeiten. Ihre Aufgabe ist hier, die Headerdateien (Dateiendung `.h`) für die (nach der in Vorlesung und Labor vorgestellten Standardmethode) resultierenden Module anzugeben. Die Datenstrukturen sollen dabei vollständig in den Headerdateien implementiert werden. Ordnen Sie die Funktionen den entsprechenden Modulen zu und achten Sie darauf, welche Module welche anderen Module in der Headerdatei einbinden müssen. (Die Implementierung der Funktionen selbst ist hier nicht gefordert, auch können Sie hier ausnahmsweise auf Kommentarzeilen in den Headerdateien verzichten.)




```
void neuesSpiel(void);
void putSpiel(const Spiel & spiel);
void getBewtg(Vers & rat, const Konf code);
void getKonf(Konf & code);
void putKonf(const Konf code);
```

Die folgenden Hinweise sind nur relevant, wenn Sie die Aufgabe nicht komplett lösen können:

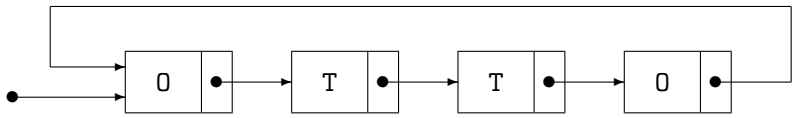
- Falls Sie die Aufteilung in einzelne Module nicht durchführen können, so geben Sie nur die Umsetzung des Datenhierarchiediagramms in C/C++-Code an (bis zu 8 Punkte).
- Falls Sie das Datenhierarchiediagramm nicht in C/C++-Code umsetzen können, so beschreiben Sie in 4–5 Sätzen, nach welchem Prinzip Sie auf Basis des Datenhierarchiediagramms eine Aufteilung in Module und die Zuordnung der Funktionen zu den Modulen vornehmen würden. (Diese Erklärung ist nicht nötig, wenn Sie die obige Aufgabenstellung komplett lösen.) (bis zu 6 Punkte)

Aufgabe 4 — 18 Punkte

(4+14 = 18 Punkte) **Einfach verkettete Ringlisten:** In dieser Aufgabe beschäftigen wir uns mit einfach verketteten Ringlisten, der Anker zeige dabei auf ein beliebiges Element der Ringliste.

1. (4 Punkte) Geben Sie eine C- oder C++-Typdefinition für eine einfach verkettete Ringliste aus Buchstaben an. Der Datentyp für den Anker soll dabei `charRgLstPtr` heißen.

2. (14 Punkte) Schreiben Sie eine C/C++-Funktion `void anz(charRgLstPtr liste)`, die als Parameter den Anker auf eine einfach verkettete Ringliste bekommt. Ist die Liste leer, so soll die Meldung „`Liste ist leer.`“ ausgegeben werden, ansonsten die Anzahl der Elemente in der Ringliste. Zu der Liste

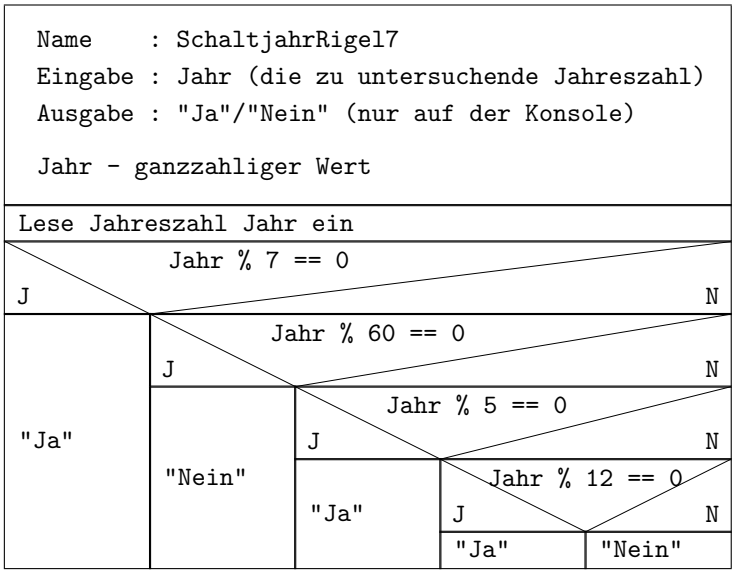


soll die Ausgabe lauten: „`Die Ringliste enthaelt 4 Element(e).`“

Beachten Sie: der Parameter ist an die Funktion zu übergeben, Sie müssen sich *nicht* um die Eingabe der Elemente kümmern. Ihre Funktion braucht nicht zu überprüfen, ob es sich wirklich um eine Ringliste handelt.

Teilklausur I in Informatik – TEL09GR2 – Lösungshinweise

Aufgabe 1 — Schaltjahr auf Rigel 7



Bei der Bearbeitung ist unerheblich, ob die Befehle C-Syntax haben oder Pseudo-Code sind.
Wichtig ist hier nur die Reihenfolge der `if`-Abfragen (erst Rest der Division durch 7, dann durch 60 – die Abfrage durch 5 und 12 hätte man auch vertauschen oder in einer Abfrage mit „oder“ („`||`“) zusammenfassen können – oder auch etwas komplizierter: `(Jahr % 5 == 0) && (Jahr % 60 != 0)`, analog mit 12).

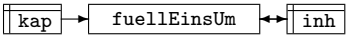
Aufgabe 4 — Einfach verkettete Ringlisten

Der entscheidende Punkt bei dieser Aufgabe, ist zu beachten, dass die Zeiger und nicht die Inhalte der Liste mit dem Anker verglichen werden. Insbesondere muss man sich zu Beginn den Anker merken. Benutzt man eine `while`-Schleife, so muss die Initialisierung mit `count=1` und `liste=anker->next` erfolgen.

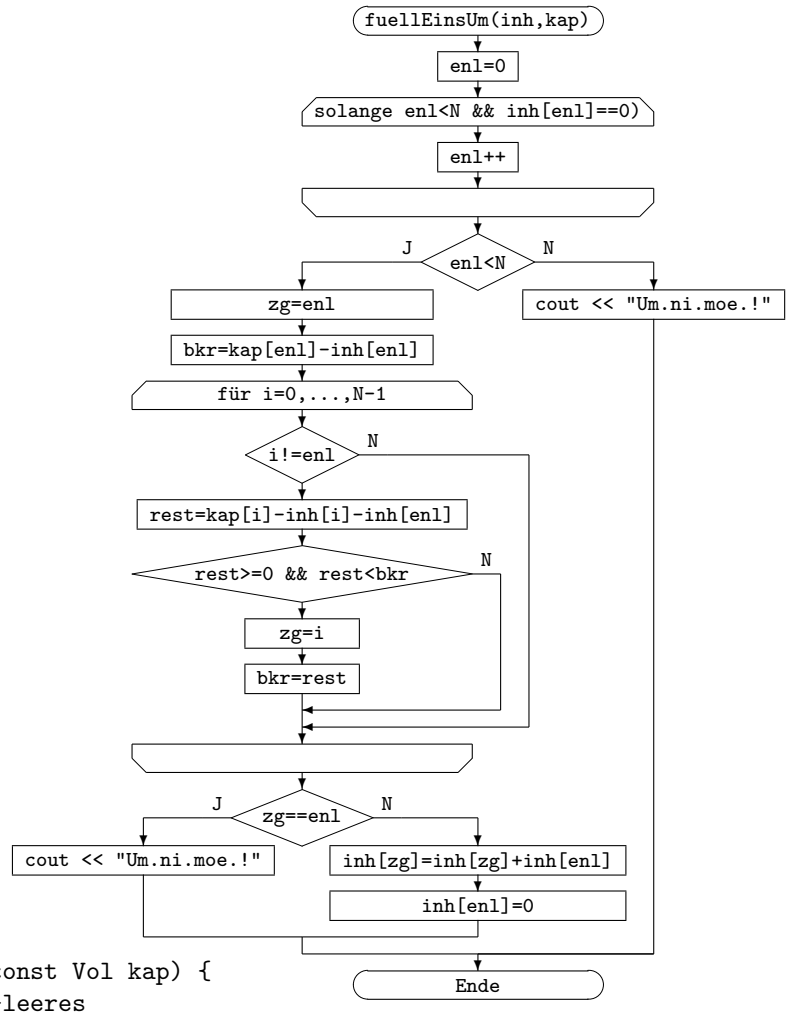
```
struct charRgLst;
typedef charRgLst* charRgLstPtr;
struct charRgLst {
  char value;
  charRgLstPtr next;
};

void anz(charRgLstPtr anker) {
  if (anker==NULL) {
    cout << "Liste ist leer." << endl;
  } else {
    int count=0;
    charRgLstPtr liste=anker;
    do {
      count++;
      liste=liste->next;
    } while (liste!=anker);
    cout << "Die Ringliste enthaelt " << count << " Element(e)." << endl;
  }
}
```

Aufgabe 2 — Gut gefüllt

Das Vorgehen kann direkt der Aufgabenstellung entnommen werden: Finde zunächst das erste nicht-leere Gefäß und – falls vorhanden – suche eines mit kleinerer Restkapazität. Wenn letzteres gefunden wurde, fülle um, sonst nicht. Das Datenflussdiagramm:



Ein Programmablaufplan: Da die Restkapazität des ersten nicht-leeren Gefäßes unterboten werden soll, können hier das Zielgefäß und das Minimum mit den entsprechenden Werten initialisiert werden.



Die Umsetzung in C/C++:

```
void fuellEinsUm(Vol & inh, const Vol kap) {
  int enl=0; // erstes nicht-leeres
  while (enl<N && inh[enl]==0) { enl++; }
  if (enl<N) {
    int zg=enl; // Ziel-Gefaess, Aufgabe: finde ein besseres
    int bkr=kap[enl]-inh[enl]; // bisher kleinster Rest (muss unterboten werden)
    int i, rest;
    for(i=0;i<N;i++) {
      if (i!=enl) { // was passiert sonst? (z.B. mit inh[enl]=4; kap[enl]=8;)
        rest=kap[i]-inh[i]-inh[enl]; // das wäre hier noch übrig
        if (rest>=0 && rest<bkr) { // Inhalt passt rein und zwar besser als bisher
          zg=i; bkr=rest;
        }
      }
    }
    if (zg==enl) { // nichts besseres gefunden
      cout << "Umfuellen nicht moeglich!" << endl;
    } else {
      inh[zg]=inh[zg]+inh[enl];
      inh[enl]=0;
    }
  } else { // alle Gefäße leer
    cout << "Umfuellen nicht moeglich!" << endl;
} }
```

Da ein Array intern im Wesentlichen nur ein Pointer ist, wird in der obigen Signatur das DF nicht 100%-ig richtig umgesetzt – Lob, wer dies erkannt hat, so wäre es ganz korrekt: `void fuellEinsUm(unsigned int * const inh, const unsigned int * const kap) {...}` – sonst könnte man `kap=new unsigned int[7];` lokal ein neues Array zuweisen, dies aber Dank `const` nicht ändern. Ohne die zusätzliche Variable `rest` wäre die entsprechende `if`-Abfrage optisch etwas komplizierter, z. B. `if ((inh[enl]+inh[i]<=kap[i]) && (kap[i]-inh[i]-inh[enl]<bkr))`.

Aufgabe 3 — Datenhierarchie und Modularisierung

Folgende Regeln sind bei der Umsetzung zu beachten:

- Jede neue Datenstruktur wird zu einem eigenen Modul (der Name entspricht der Datenstruktur mit angehängtem `.h`).
- Jedes Modul beginnt mit `#ifndef` *NAME*`_H` und `#define` *NAME*`_H` (wobei *NAME* durch den Bezeichner der Datenstruktur ersetzt wird) und endet mit `#endif`.
- Das DH-Diagramm wird von unten nach oben in C/C++-Code bzw. in die Module umgesetzt.
- Es müssen jeweils diejenigen Headerdateien eingebunden werden, deren Datenstruktur in den Signaturen verwendet wird.
- Jede Funktion wird dem am weitesten oben liegenden Modul zugeordnet, dessen Datenstruktur in der Signatur verwendet wird.
- Die Funktion `neuesSpiel()` gehört aufgrund des Namens offensichtlich in das Modul `spiel.h`.

Somit entstehen die 3 Headerdateien `konf.h`, `vers.h` und `spiel.h` (die Bezeichner der formalen Parameter können – wie in Headerdateien üblich – auch weggelassen werden):

konf.h:

```
#ifndef KONF_H
#define KONF_H

typedef int Konf[5];

void getKonf(Konf & code);
void putKonf(const Konf code);

#endif
```

vers.h:

```
#ifndef VERS_H
#define VERS_H

#include "konf.h"

struct Vers {
  Konf tipp;
  int bewtg;
} ;

void getBewtg(Vers & rat, const Konf code);

#endif
```

spiel.h:

```
#ifndef SPIEL_H
#define SPIEL_H

#include "konf.h"
#include "vers.h"

struct Spiel {
  Konf code;
  Vers raten[12];
  int zugNr;
} ;

void neuesSpiel(void);
void putSpiel(const Spiel & spiel);

#endif
```