Teilklausur I in Informatik

Es sind insgesamt 80 Punkte zu erreichen
(Jeder Punkt entspricht einer Bearbeitungszeit von etwa 1 Minute)

Aufgabe 1: Hotel Confusio (12 Punkte) – DF, PA
Aufgabe 2: Spiegelzahlen (43 Punkte) – EBNF, PAD/PN, PH, C/C++
Aufgabe 3: Kleinster Winkel (25 Punkte) – DH, Struktogramm, 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

(2+10 = 12 Punkte) **Hotel Confusio:** Ein Freund erzählt Ihnen, dass er ein Hotel geerbt hat und dass dieses 23 Zimmer mit 37 Betten hätte. Später überlegen Sie, wieviel Einzel- und wieviel Doppelzimmer das Hotel wohl hat und nach etwas raten kommen Sie auf die Lösung, dass es 9 Einzelzimmer und 14 Doppelzimmer sein müssten.

Um in Zukunft auf solche Situationen besser vorbereitet zu sein, beschließen Sie eine C/C++-Funktion zu entwerfen, die als Eingangsparameter die Anzahl der Zimmer und Betten bekommt und als Ausgangsparameter die Anzahl der Einzel- und Doppelzimmer berechnet. Ferner soll die Funktion über eine ebenfalls zu übergebene `bool`-Variable den Wert `true` liefern, wenn die Berechnung fehlerfrei möglich ist, und `false`, falls die Eingaben zu keiner gültigen Lösung führen können (z.B. muss die Anzahl der Betten immer mindestens so groß wie die der Zimmer sein).

Geben Sie das Datenflussdiagramm (DF) der Ebene 0 an (Funktion als ein Kästchen) (2 Punkte), sowie einen Programmablaufplan (PA) für die von Ihnen entworfene Funktion mit den nötigen Verfeinerungen (10 Punkte). Der Entwurf soll so detailliert sein, dass eine Umsetzung 1:1 in ein C/C++-Programm möglich wäre, 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. Eine Umsetzung in C/C++ ist hier *nicht* gefordert.

Aufgabe 2 — 43 Punkte

(4+4+10+10+3+6+6 = 43 Punkte) **Spiegelzahlen:** Eine Spiegelzahl ist eine Zahl, die von vorne nach hinten gelesen identisch mit der Zahl ist, die von hinten nach vorne gelesen wird. Somit sind 343, 1001 und 7 Spiegelzahlen, 62263 und 42 jedoch nicht.

(4 Punkte) Geben Sie zunächst EBNF-Regeln an, die genau solche Spiegelzahlen erzeugen. (Wer dabei beachtet, dass außer der 0 keine Spiegelzahl am Anfang und Ende Nullen hat, erhält 2 Zusatzpunkte.)

Für eine Umsetzung in eine C/C++-Funktion liegt bereits folgendes Programmfragment vor:

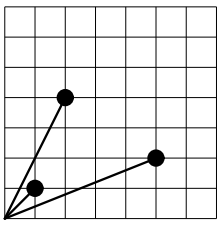
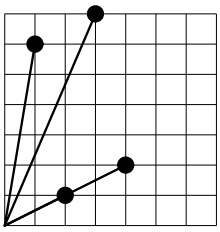
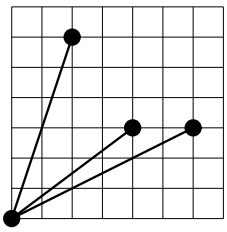
```
typedef int ziffern[10]; // ein unsigned int hat maximal 10 Ziffern
void gespiegelt(const unsigned int n, bool & erg) {
   ziffern zehnziffern;
   int2ziffern(n,zehnziffern);
   spiegeltest(zehnziffern,erg);
}
```

Die Idee ist also, zunächst die als `unsigned int` übergebene Zahl mit der Funktion `int2ziffern` in ihre einzelnen Stellen zu zerlegen (bei 343 also in 3, 4 und 3), diese Stellen in das Array `zehnziffern` zu speichern (ggf. den Rest des Arrays `zehnziffern` mit Nullen zu füllen) und danach diese einzelnen Stellen mit der Funktion `spiegeltest` auf geeignete Weise miteinander zu vergleichen und dabei die Variable `erg` auf `true` oder `false` zu setzen. Führen Sie den Entwurf zu Ende und setzen diesen dann in die entsprechenden C/C++-Funktionen um. Das Hauptprogramm `int main()` muss nicht geschrieben werden, ebenso brauchen Sie sich nicht um das Einlesen der Zahl `n` oder die Ausgabe des Ergebnisses zu kümmern.

Gefordert sind die Programmablaufpläne mit Daten (PAD) *oder* Programmnetze (PN) der Funktionen `gespiegelt` (4 Punkte), `int2ziffern` (10 Punkte) und `spiegeltest` (10 Punkte), ein Programmhierarchiediagramm (PH) (3 Punkte), sowie die Umsetzung von `int2ziffern` und `spiegeltest` in C/C++ (je 6 Punkte).

Aufgabe 3 — 25 Punkte

(3+14+8 = 25 Punkte) **Kleinster Winkel:** Gegeben ist eine Menge von Punkten $(x, y) \in \mathbb{N}_0 \times \mathbb{N}_0$ in der Ebene. Für jeden dieser Punkte (x, y) bildet die Gerade vom Ursprung $(0, 0)$ zu diesem Punkt (x, y) einen Winkel mit der x-Achse. Ihre Aufgabe ist, eine Funktion zu entwerfen, die aus einer Menge von gegebenen Punkten den Punkt bestimmt, für den dieser Winkel am kleinsten ist, und diesen als Ergebnis der Funktion zurückgibt (keine Ausgabe auf dem Bildschirm). Wenn bei mehreren Punkten dieser Winkel identisch ist, so soll derjenige zurückgegeben werden, der dem Ursprung am nächsten ist. Ist ein Punkt der Ursprung, so soll stets dieser zurückgegeben werden. (Hinweis: Sie brauchen keine trigonometrischen Funktionen, überlegen Sie sich, wie die Steigungen der Geraden mit dem Winkel zusammenhängen.)

Im ersten Beispiel ist das Ergebnis $(5, 2)$, im zweiten $(2, 1)$ (gleicher Winkel wie $(4, 2)$, aber näher am Ursprung), im dritten ist das Ergebnis $(0, 0)$ (da der Ursprung als Punkt dabei ist).

(3 Punkte) Geben Sie die verwendeten Datenstrukturen als Datenhierarchiediagramm (DH) an. Entwerfen Sie eine Funktion, die ein Array mit Punkten übergeben bekommt und die Koordinaten des gesuchten Punktes zurückgibt. (14 Punkte) Geben Sie ein Struktogramm für die von Ihnen entworfene Funktion an. 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.

(8 Punkte) Setzen Sie danach Ihren Entwurf in C/C++ um.

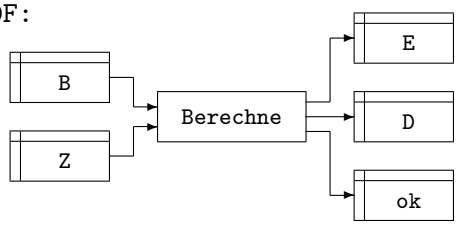
Teilklausur I in Informatik – Lösungshinweise

Aufgabe 1 — Hotel Confusio

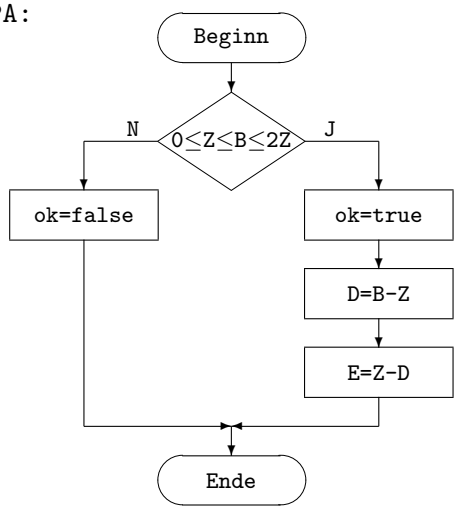
Diese Aufgabe war 1:1 zu lösen wie das Beispiel des Parkplatzproblems in der Vorlesung.

Seien B die Anzahl der Betten, Z die Anzahl der Zimmer, E die Anzahl der Einzelzimmer, D die Anzahl der Doppelzimmer und ok die boolsche Variable. Es ergeben sich zwei Gleichungen: (1) E+D=Z und (2) E+2D=B, durch Umformung ((2)-(1)) erhält man (3) D=B-Z (und durch Einsetzen von (3) in (1) die Gleichung (4) E=2Z-B, wobei man hier auch E=Z-D verwenden kann). Randbedingungen sind wie in der Aufgabenstellung angegeben B≥Z sowie zusätzlich B≤2Z. Ein- und Ausgabe waren nicht gefordert.

DF:



PA:



Aufgabe 2 — Spiegelzahlen

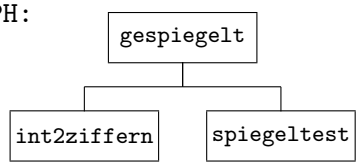
Spiegelzahlen haben links und rechts dieselbe Ziffer, dazwischen befindet sich wiederum eine Spiegelzahl (mit mehr als 2 Ziffern) oder zwei identische Ziffern oder eine Ziffer. Also

```
<S> ::= 0<S>0 | 1<S>1 | ... | 9<S>9 | 00 | 11 | ... | 99 | 0 | 1 | ... | 9
```

Um zu berücksichtigten, dass Spiegelzahlen nicht mit 0 anfangen und enden können, muss man diese Regeln duplizieren und bei `<S>` die „verbotenen Regeln“ aussparen:

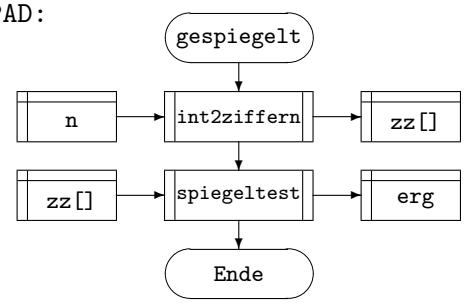
```
<S> ::= 1<T>1 | 2<T>2 | ... | 9<T>9 | 11 | 22 | ... | 99 | 0 | 1 | ... | 9
<T> ::= 0<T>0 | 1<T>1 | ... | 9<T>9 | 00 | 11 | ... | 99 | 0 | 1 | ... | 9
```

PH:

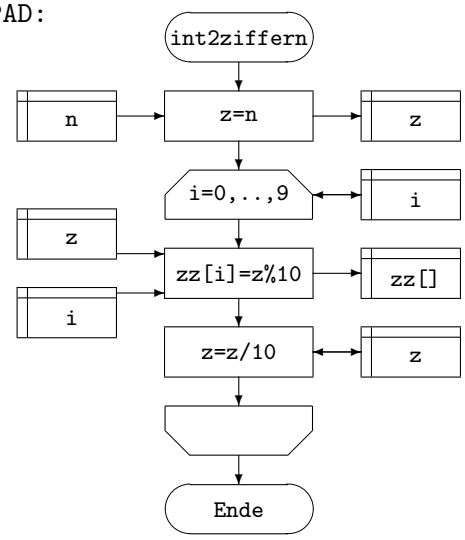


Der Zerlegen der Zahl in die Einzelstellen kann durch wiederholtes Modulo-10-Rechnen und Durch-10-Dividieren erledigt werden. Beim Überprüfen überspringt man die Nullen am Rand und überprüft dann den Rest von außen nach innen auf Gleichheit.

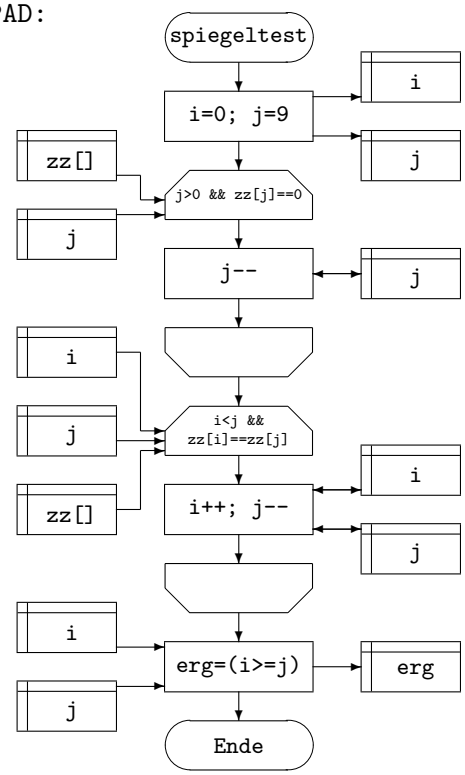
PAD:



PAD:



PAD:



Die Programme wurden nicht in derart kompakter Form erwartet, die Grundidee hätte aber schon so sein sollen. Das Zerlegen der Zahlen in Ziffern wurde so in der Vorlesung behandelt. Das Überprüfen der Spiegeleigenschaft erfolgt direkt auf den Ziffern (wozu sollte man sie sonst vorher zerlegt haben). Insbesondere in der Funktion `spiegeltest` haben die meisten den Test auf Gleichheit in die Schleife gelegt und dort den Wert `erg` gesetzt (was auch vollkommen in Ordnung ist). Die hier sehr kompakte Form ist gleichbedeutend mit `if (i>=j) erg=true; else erg=false;` – sie nutzt aus, dass `j` beim Nullen-Überspringen maximal auf `0` runtergezählt wird: War `n=0` (oder einstellig), so ist dann `i=j=0` und die Test-Schleife wird nicht betreten; sonst wird die Schleife verlassen wenn $\texttt{i} \geq \texttt{j}$ wird (dann waren die Ziffern-Gleichheitstests alle erfolgreich) oder wenn ein Test eine Ungleichheit gefunden hat (dann ist `i` noch kleiner `j`). Es können nicht beide Bedingungen der Test-Schleife gleichzeitig falsch sein (selbst überlegen).

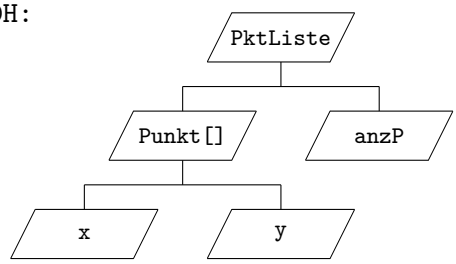
```
void int2ziffern(const unsigned int n, ziffern & zz) {
  // Eingabe-Parameter haben nur Lesezugriff
  unsigned int z=n;
  for (int i=0;i<=9;i++) {
    zz[i] = z%10;
    z = z/10;
  }
}
```

```
void spiegeltest(const ziffern zz, bool & erg) {
  int i=0, j=9;
  while (j>0 && zz[j]==0) {j--;}
  while (i<j && zz[i]==zz[j]) {i++; j--;}
  erg = (i>=j);
}
```

Aufgabe 3 — Kleinster Winkel

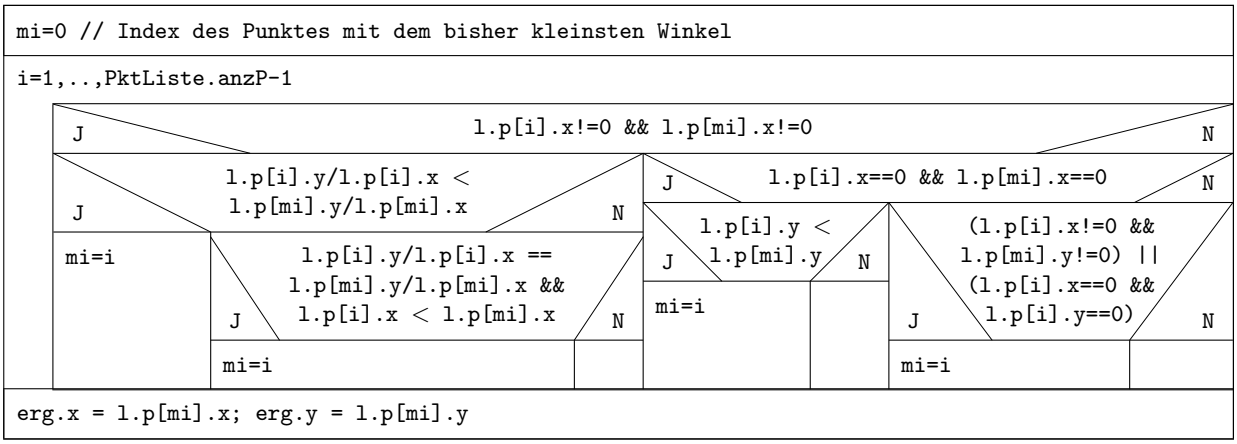
Um in einer Menge ein kleinstes (oder größtes) Element zu finden, merkt man sich das erste Element als bisher kleinstes (diese oder eine geeignete andere Initialisierung wurde oft vergessen) und vergleicht dann alle weiteren Elemente mit dem jeweils bisher kleinsten. Es war hier also im Wesentlichen eine Vergleichsroutine zu schreiben, die für zwei Punkte p_1 und p_2 entscheidet, ob p_1 im Sinne der Aufgabenstellung kleiner als p_2 ist, also mit der x-Achse den kleineren Winkel bildet bzw. bei gleichem Winkel den kleineren Abstand zum Ursprung hat. Als Parameter erhält die Funktion ein Array aus Punkten, dementsprechend sieht das Datenhierarchiediagramm aus:

DH:




```
const unsigned int MAX_ANZ = 40;
struct Punkt {
  unsigned int x,y;
};
struct PktListe {
  Punkt p[MAX_ANZ];
  unsigned int anzP;
};
```

Die Funktion bekommt als Parameter eine Liste `l` vom Typ `PktListe` und liefert über einen Referenzparameter `erg` vom Typ `Punkt` den gesuchten Punkt zurück. Eine kleinere Steigung (`y/x`) bedeutet einen kleineren Winkel; zu beachten ist, dass die Steigung bei `x=0` nicht berechnet werden kann – man dies also durch geeignete Fallunterscheidungen abfangen muss: Sind beide `x`-Werte ungleich 0, so entscheidet die kleinere Steigung und bei gleicher Steigung der kleinere `x`-Wert; sind beide `x`-Werte gleich 0, so entscheidet der kleinere `y`-Wert; ist genau ein `x`-Wert ungleich 0, so hat dieser den kleineren Winkel, außer der `y`-Wert des anderen Punktes ist auch gleich 0 (d.h., der andere Punkt ist der Ursprung). Ggf. kann man die Schleife abbrechen, wenn ein Punkt mit Koordinaten `(0,0)` gefunden wurde.



Bei der Umsetzung sollte man noch beachten, dass die Koordinaten vom Datentyp `unsigned int` sind und somit bei der Berechnung der Steigungen die Ganzzahl-Division benutzt wird. Um dieses Problem zu umgehen, berechnet man statt `l.p[i].y/l.p[i].x<l.p[mi].y/l.p[i].x` einfach `l.p[i].y*l.p[mi].x<l.p[mi].y*l.p[i].x` (selber Beispiele überlegen, bei denen es sonst zu falschen Ergebnissen kommt).

```
void kW(const PktListe & l, Punkt & erg) {
  unsigned int mi = 0;
  for (unsigned int i=1;i<l.anzP;i++) {
    if (l.p[i].x!=0 && l.p[mi].x!=0) { // beide ungleich 0
      if (l.p[i].y*l.p[mi].x<l.p[mi].y*l.p[i].x) { // sonst Ganzzahl-Division!!!
        mi = i; // kleinere Steigung
      } else { // gleiche Steigung und kleinerer Abstand ?
        if (l.p[i].y*l.p[mi].x==l.p[mi].y*l.p[i].x && l.p[i].x<l.p[mi].x)
          mi = i;
      }
    } else {
      if (l.p[i].x==0 && l.p[mi].x==0) { // beide gleich 0
        if (l.p[i].y<l.p[mi].y)
          mi = i; // kleinerer Abstand
      } else { // nur einer ungleich 0
        if ((l.p[i].x!=0 && l.p[mi].y!=0) || (l.p[i].x==0 && l.p[i].y==0))
          mi = i; // kleinere Steigung oder = Ursprung
  } } }
  erg.x = l.p[mi].x; erg.y = l.p[mi].y;
}
```