Prüfung Informatik D-MATH/D-PHYS 10.8.2011

Dr. Bernd Gärtner, Prof. Juraj Hromkovič

Lösung.

Aufgabe 1.

```
int k = 1;
int z = 1;
double d = 1.5;
float f = 0.25f;
```

Aufgabe	Ausdruck	Typ	Wert
a)	`false && 1.77 / d == 1.18`	`bool`	`false`
b)	`127 % 7 * 67 % 17 % k`	`int`	`0`
c)	`&k == &z`	`bool`	`false`
d)	`9 / 4.0f + 1 - f * 5`	`float`	`2.0`
e)	`--z + k++ + d`	`double`	`2.5`

Punktvergabe. Bei jeder Teilaufgabe gibt es **+1 Punkt** sowohl für den richtigen Typ als auch für den richtigen Wert. Insgesamt also **+10 Punkte**.

Aufgabe 2.

a) Die beiden Deklarationen unterscheiden sich nur in den Zugriffsrechten. Bei `class` ist das Datenmitglied `i` implizit als `private` deklariert; beim `struct` jedoch als `public`.

b) Der Rumpf der Schleife wird genau 3 Mal ausgeführt. Man beachte, dass bei jeder Zuweisung zur Variablen `i` die Nachkommastellen des exakten Wertes abgeschnitten werden.

c) Die Ausgabe ist

1 2 3 5 6 7 9

mit einem Leerzeichen zwischen den einzelnen Ziffern.

d) Die Binäre Darstellung der Dezimalzahl $(1.625)_{10}$ lautet $(1.101)_2$.

d) Die Aussage ist wahr. O.B.d.A. können wir annehmen, dass p/q ein echter Bruch ist. Wählen wir für unser Fliesskommasystem den Nenner, also q, als Basis, ist die Darstellung von p/q einfach $(0.p)_q$, wobei p die entsprechende Ziffer des Systems ist.

Punktvergabe. Für jede der Teilaufgaben gibt es **+5 Punkte**.

Aufgabe 3.
Vorbedingung: Das Programm liest zuerst eine ganze positive Zahl ein. Diese Zahl gibt an wie viele weitere Zahlen in der Folge maximal eingelesen werden. So viele weitere Zahlen müssen auf dem Eingabestrom vorhanden sein.

Nachbedingung: Sei n die erste Zahl die eingelesen wurde. Das Programm entscheidet, ob die n Zahlen, die auf dem Eingabestrom folgen aufsteigend sortiert sind. Es gibt "Yes" aus, falls dies der Fall ist. Andernfalls gibt es "No" aus.

Als Verständnisstütze hier noch eine kleine Erklärung des Programmes: Die Zahl n wird ins `Register(2)` eingelesen, welches in der Folge als Schleifenzähler dient. In Zeile `10` wird von `Register(2)` jeweils `1` abgezogen, damit der Test in Zeile `6` zum richtigen Zeitpunkt angibt, dass alle n Zahlen eingelesen worden sind. Zeilen `6-12` sind die Hauptschleife des Programmes. Hier wird jeweils eine neue Zahl eingelesen, und es wird überprüft, ob sie grösser als die zuletzt eingelesene ist. Ist dies der Fall wird mit der Schleife weiter gefahren. Falls nicht, dann wird direkt zur Ausgabe "No" gesprungen. Die Zeilen `3-6` dienen dazu, zu erkennen, wenn die Folge aus keiner oder nur einer Zahl besteht. Dann kann direkt "Yes" ausgegeben werden.

Punktvergabe. Es gibt insgesamt **+10 Punkte** für die Vorbedingung. Dabei trägt am meisten Gewicht, dass n ganzzahlig und positiv sein muss, damit das Programm nicht in eine Endlosschleife geht. Das gibt **+6 Punkte**. Die weiteren **+4 Punkte** gibt es, wenn auch noch gesagt wird, dass n die Anzahl der Zahlen angibt, die noch auf dem Ausgabestrom folgen müssen.

Damit verbleiben **+15 Punkte** für die Nachbedinung.

Aufgabe 4.

a) Der Algorithmus funktioniert wie folgt: Solange der Zeiger `b` nicht gleich dem Zeiger `e` ist, gibt es noch (Gruppen von) Ziffern, die nicht abgearbeitet wurden.

In einer äusseren Schleife lassen wir `b` also in Richtung `e` wandern. In einer inneren Schleife schauen wir immer auf die Ziffer, die auf `*b` folgt und lassen `b` gleich um eines weiter schreiten, wenn die Ziffern gleich sind. Dabei zählen wir mit, wie viel gleiche Ziffern wir antreffen. Erst wenn wir das Ende `e` oder eine fremde Ziffer erkennen, brechen wir den aktuellen Lauf ab und schreiben die Anzahl und die Ziffer ins Zielfeld `res`. Dabei können wir auch gleich die Anzahl `n` aktualisieren, indem wir 2 dazu zählen.

Zum Ende der äusseren Schleife erhöhen wir `b` noch einmal um eins, damit wir uns am Beginn der nächsten Zifferngruppe befinden (oder am Ende). Damit wird auf die selbe Weise die nächste Zifferngruppe abgearbeitet.

b)
```
void morris(const int* b, const int* e, int* res, unsigned int& n) {
  n = 0;
  while (b != e) {
    int count = 1;
    while (b+1 < e && *(b+1) == *b) {
      ++count;
      ++b;
    }
    *(res++) = count;
    *(res++) = *b;
    n += 2;
    ++b;
  }
}
```

Punktvergabe. Für eine korrekte Beschreibung gibt es **+10 Punkte**. Für die Implementierung gibt es **+15** Punkte.

Für den Programmierteil soll gelten, das 1-2 kleinere Syntaxfehler noch ohne Punkteabzug toleriert werden. Für jeden weiteren Fehler gibt es dann aber **-1 Punkt**. Effizienz ist nicht erforderlich für die volle Punktezahl.

Bei der Korrektur von Teilaufgabe b) ist ein Fehler unterlaufen. Anstatt der in der Prüfung angegebenen **+20 Punkte** wurden maximal **+15 Punkte** vergeben. Dieser Punktestand ist auf den Prüfungen eingetragen und die folgenden Regeln beziehen sich auf diesen Maximalpunktestand. Bei der Berechnung der Noten wurde der Punktestand aus b) jedoch mit einem Faktor von $4/3$ belegt. Somit war es möglich maximal **+20 Punkte** zu erhalten.

a)
- **+3 Punkte** für die Erklärung der äusseren Schleife (e-b).
- **+3 Punkte** für das Zählen aufeinanderfolgender Ziffern.
- **+2 Punkte** für die Ausgabe in `res`.
- **+2 Punkte** für das korrekte Inkrementieren von `n`.

Es sind jeweils auch Teilpunkte möglich.

b)
- **-1 Punkt** falls eine nicht `const` Kopie der `const` Eingabezeiger gemacht wird.
- **-1 Punkt** falls (möglicherweise) im nicht gültigen Bereich dereferenziert wird.
- **-3 Punkte** falls die Zählvariable oder der Zeiger der Hauptschleife falsch inkrementiert wird.
- **-2 Punkte** falls das Resultat nicht in `res` geschrieben wird.
- **-2 Punkte** falls die Ausgabe der letzen Zifferngruppe vergessen wird resp. wenn es nicht funktioniert falls die Eingabezahl einstellig ist.
- **-2 Punkte** falls die Ziffern öfter als ein mal ausgegeben wird.
- **-2 Punkte** falls nur zwei aufeinanderfolgende Zahlen geprüft werden.
- **-1 Punkt** falls fälschlicherweise etwas zurück gegeben wird (z.B. `return 0;`), anstatt nur `return`.
- **Maximal 6 Punkte** falls Lösung wirr und nicht ausgeführt, die Abzüge oben können weiterhin gelten.
- **Maximal 8 Punkte**, falls Morris falsch verstanden wurde, die Abzüge oben können weiterhin gelten.

Dies ist eine Liste mit häufigen Fehlern. Andere Fehler geben im Normalfall **-1 Punkt**.

Aufgabe 5.

a) Die Funktion berechnet die Summe $\sum_{i=0}^{n} i$.

b) Aus der Vorlesung kennen Sie die Formel des kleinen Gauss: $\frac{n(n+1)}{2}$. Dies kann natürlich direkt implementiert werden, und die Rekursion fällt somit weg.

Punktvergabe. Für die beiden Aufgabenteile gibt es je **+5 Punkte**.

Falls jemand die Sache iterativ und nicht rekursiv macht, so ist das nicht wesentlich effizienter. Das gibt maximal `+2 Punkte`.

Aufgabe 6.

a)
```
struct Bewohner {
  char v,n;     // Steht fuer Vornamen und Nachnamen
  int alter;    // Alter des Bewohners in Jahren
  int meister;  // Index des Meisters
};
```

b)
```
// In diesem globalen Array sind alle Bewohner gespeichert und
// korrekt initialisiert, wobei anz_bew eine zuvor initialisierte
// Konstante ist.
Bewohner bewohner[anz_bew];

// PRE: Die Meister-Struktur von Plato enthaelt keine Zyklen.
//      index ist ein gueltiger Index, 0 <= index < anz_bew.
// POST: Der Rueckgabewert ist der Index des Koenigs von
//       bewohner[index], oder -1 falls bewohner[index] selbst
//       ein Koenig ist.
int finde_koenig(int index) {
  if (bewohner[index].meister == -1) {
    return -1;
  } else {
    int res = finde_koenig(bewohner[index].meister);
    if (res == -1) {
      return bewohner[index].meister;
    } else {
      return res;
    }
  }
}
```

Man beachte, dass es im `else`-Fall nicht einfach reicht, einen rekursiven Aufruf zu mache, weil der Rückgabewert im Falle eines Koenigs nicht sein Index sondern -1 ist. Es gibt noch eine andere Variante, das ganze iterativ zu machen, was eigentlich fast noch eleganter ist.

```
int finde_koenig(int index) {
  if (bewohner[index].meister == -1) {
    return -1;
  } else {
    while (bewohner[index].meister != -1) {
      index = bewohner[index].meister;
    }
    return index;
  }
}
```

Punktvergabe. Für die beiden Aufgabenteile gibt es je **+10 Punkte**.

Bei der Teilaufgabe a) kann man dabei **+2 Punkte** jedes der 4 notwendigen Datenmitglieder vergeben, und noch zusätzlich **+2 Punkte** für die korrekte Syntax des `structs` (inklusive Strichpunkt am Ende).

Bei der Korrektur von Teilaufgabe b) gelten die allgemeinen Richtlinien für Programmieraufgaben. D.h. ein Syntaxfehler wird ohne Punktabzug toleriert, danach **-1 Punkt** für jede neue Art von Syntaxfehlern. Effizienz ist nicht erforderlich für die volle Punktezahl. Es ist jedoch darauf zu achten, dass die Nachbedingung korrekt implementiert ist, insbesondere die Rückgabe von -1 im Falle eines Königs. Insbesondere gibt es **+3 Punkte**, wenn bei einem König korrekterweise -1 zurück gegeben wird, wobei die triviale Lösung (**return -1;**) diese Punkte nicht erhält. Falls die anderen Fälle auch richtig abgedeckt sind, gibt es **+7 Punkte**.

Ein Spezialfall: Eine Lösung, die im Falle eines Königs nicht -1 zurück gibt, die anderen Fälle aber korrekt abhandelt, bekommt "nur" **+3 Punkte**, da es so viel einfacher ist. Es galt aber, die Vorbedingung genau zu befolgen.

Ende der Musterlösung zur Prüfung vom 10.8.2011.