Beachten Sie die Hinweise zum Programmierstil auf Blatt 5. Bei groben Verstößen können Ihnen ab dieser Woche Punkte abgezogen werden.

Übung 1 *Umgebungen*

Gegeben sei das folgende Programm:

```
5  int g = 0;
6
7  int a_mod_b( int a, int b)
8  {
9    int m = a%b;
10   return m;  [1]
11 }
12
13 int ggT( int a, int b)
14 {
15   g = g+1;
16   int Null=0;
17   if (b==Null)
18     return a;  [2]
19   else
20     return ggT(b, a_mod_b(a,b));
21 }
22
23 int main ()
24 {
25   int a = 2;
26   int b = 14;
27   {
28     int a = 7;
29     int g = ggT(b,a);
30     b=g;
31   }
32   a=g;
33   return 0;  [3]
34 }
```

Verwenden Sie das Umgebungsmodell (Definition 8.2 aus der Vorlesung) um zu bestimmen, welche Umgebungen existieren, wenn das Programm die mit [1], [2] und [3] markierten Zeilen erreicht. Die markierten Zeilen sollen erreicht sein, aber noch nicht ausgeführt – im Falle [3] ist Zeile 32 ausgeführt worden, Zeile 33 aber noch nicht.

Stellen Sie die Umgebungen graphisch dar (wie in der Vorlesung) um zu zeigen welche Namen existieren und welche Werte Sie haben. Namen ohne definierten Wert markieren Sie mit `?`.

(6 Punkte)

Übung 2 *Primfaktorzerlegung*

a) Schreiben Sie ein Programm, welches für eine gegebene Zahl deren Primfaktorzerlegung prozedural berechnet und diese auf der Konsole ausgibt. Schreiben Sie ihr Programm so, dass die Primfaktoren automatisch vom kleinsten zum größten sortiert ausgegeben werden. Beachten Sie außerdem die Bemerkung im Skript, dass man bei der Suche nach Teilern von n nur den Bereich $2, \ldots, \sqrt{n}$ absuchen muss.

b) Bestimmen Sie die algorithmische Komplexität Ihres Programms für den schlechtesten Fall, d.h. wenn n prim ist.

(6 Punkte)

Übung 3 *Ein einfacher Taschenrechner*

Ziel dieser Aufgabe ist es, ein Programm zu schreiben, das eine Zeichenfolge als Eingabe erhält, und versucht, diese als mathematische Formel in Umgekehrter Polnischer Notation (Postfixnotation)[†] zu

[†]Siehe `http://de.wikipedia.org/wiki/Umgekehrte_Polnische_Notation` oder Übung 2.1

interpretieren. Entweder wird im Falle eines gültigen Ausdrucks das Ergebnis der Berechnung ausgegeben oder im Falle eines ungültigen Ausdrucks eine Fehlermeldung ausgegeben. So soll zum Beispiel die Zeichenfolge

```
67 55 - 54 6 / + 2 *
```

zur Ausgabe "42" führen, denn es ist die Postfixnotation für die Formel

$$(((67 - 55) + (54/6)) \cdot 2).$$

Für die Eingabe sollen die folgenden Regeln gelten:

- Ziffernfolgen stellen positive ganze Zahlen inklusive der Null dar. Negative ganze Zahlen sollen von diesem Taschenrechner hier nicht unterstützt werden.
- Die Symbole '+','-','*' und '/' stehen für die entsprechenden Operationen, wie Sie sie schon in C++ benutzen.
- Alle weitere Zeichen werden ignoriert und haben nur den Effekt, zwei Ziffernfolgen und damit Zahlen voneinander zu trennen, falls sie zwischen diesen auftauchen.

Benutzen Sie einen Stack, auf dem Sie der Reihe nach die gelesenen Zahlen ablegen. Dieser soll ein **`struct`** sein, das ein **`int`**-Array fester, ausreichender Größe und ein **`int`** zum Zählen der hinterlegten Elemente enthält.

Sie dürfen keine Funktionen verwenden, die Zeichenketten in Zahlen umwandeln, und müssen also "zu Fuß" die Zahlen Zeichen für Zeichen konstruieren. Benutzen Sie dafür ein **`bool`**, um sich zu merken, ob das vorherige Zeichen eine Ziffer war, und ein **`int`**, mit dem Sie die Zahl zusammensetzen, bis die letzte Ziffer gelesen ist. Implementieren Sie die Funktion **`void`** `push(`**`int`** `element)`, mit der Sie die gelesene Zahl `element` auf den Stack ablegen. Beachten Sie, dass das Speichern der Zahl auf dem Stack erst bei der nächsten Nichtziffer erfolgen darf, denn erst dann können Sie sicher sein, dass die Zahl zu Ende gelesen wurde.

Sobald einer der vier Rechenoperatoren gelesen wird, müssen Sie sich zwei Zahlen vom Stack holen, auf diese den Operator anwenden und das Ergebnis zurück auf den Stack legen. Implementieren Sie dazu die Funktion **`int`** `pop()`, welche die letzte Zahl auf dem Stack zurückgibt und den Zähler um Eins zurücksetzt. Durch diese Rechenreihenfolge wird der binäre Baum der Formel (vgl. Übung 1, Blatt 2) an den Blättern "gekürzt", bis am Ende ein äquivalenter Baum mit nur einem Knoten (dem Ergebnis) übrig bleibt.

Schreiben Sie den Taschenrechner so, dass Zeichenketten zur Auswertung von der Kommandozeile gelesen werden. Verwenden Sie zum Einlesen der Zeichenketten Felder vom Typ **`char`**, mit denen Sie mit dem Operator `[]` auf die einzelnen Zeichen zugreifen können. Die Länge der Zeichenkette erhalten Sie mit der `C`-Funktion `strlen`. Auf der Homepage zur Vorlesung finden Sie dazu die Datei `taschenrechner.cc`. Benutzen Sie diese als Vorlage für Ihr Programm. Testen Sie Ihr Programm mit den folgenden Eingaben:

- `"67 55 - 54 6 / + 2 *"`
- `"123456789987654321"`
- `"2 7 * - 4 +"`
- `"16 2 2 4 2 /*Dies ist (k?)ein Kommentar*/ 4 2 / *"`

Beachten Sie, dass die Anführungszeichen rund um das Kommandozeilenargument essentiell sind. Die Shell würde die Eingabe sonst, aufgrund der enthaltenen Leerzeichen, in mehrere Argumente zerlegen. **(8 Punkte)**