1. Übungsblatt zu Programmierung 1, WS 2012/13

Prof. Dr. Gert Smolka, Sigurd Schneider, B.Sc.
www.ps.uni-sb.de/courses/prog-ws12/

Lesen Sie im Buch: Kapitel 1

Aufgabe 1.1 Betrachten Sie das folgende Programm:

```
val x = 7+4
val y = x*(x-1)
val z = ~x*(y-2)
```

Welche Bezeichner, Konstanten, Operatoren und Schlüsselwörter kommen in dem Programm vor? An welche Werte bindet das Programm die vorkommenden Bezeichner?

Aufgabe 1.2 Deklarieren Sie eine Prozedur $p : int \to int$, die für x das Ergebnis $2x^2 - x$ liefert. Identifizieren Sie das Argumentmuster, die Argumentvariable und den Rumpf Ihrer Prozedurdeklaration.

Aufgabe 1.3 (Signum) Schreiben Sie eine Prozedur $signum : int \to int$, die für negative Argumente -1, für positive Argumente 1, und für 0 das Ergebnis 0 liefert.

Aufgabe 1.4 Schreiben Sie eine Prozedur $hoch17 : int \to int$, die zu einer Zahl x die Potenz x^{17} berechnet. Dabei sollen möglichst wenig Multiplikationen verwendet werden. Schreiben Sie die Prozedur auf zwei Arten: Mit einer Hilfsprozedur und mit lokalen Deklarationen.

Aufgabe 1.5

a) Geben Sie ein Tupel mit 3 Positionen und nur einer Komponente an.
b) Geben Sie einen Tupelausdruck an, der den Typ $int * (bool * (int * unit))$ hat.
c) Geben Sie ein Paar an, dessen erste Komponente ein Paar und dessen zweite Komponente ein Tripel ist.

Aufgabe 1.7 Schreiben Sie eine Prozedur $max : int * int * int \to int$, die zu drei Zahlen die größte liefert, auf zwei Arten:

a) Benutzen Sie keine Hilfsprozedur und drei Konditionale.
b) Benutzen Sie eine Hilfsprozedur und insgesamt nur ein Konditional.

Aufgabe 1.10 Schreiben Sie eine Prozedur $teilbar : int * int \to bool$, die für (x, y) testet, ob x durch y ohne Rest teilbar ist.

Aufgabe 1.11 (Zeitangaben) Oft gibt man eine Zeitdauer im *HMS-Format* mit Stunden, Minuten und Sekunden an. Beispielsweise ist 2h5m26s eine hervorragende Zeit für einen Marathonlauf.

a) Schreiben Sie eine Prozedur $sec : int * int * int \to int$, die vom HMS-Format in Sekunden umrechnet. Beispielsweise soll $sec(1,1,6)$ die Zahl 3666 liefern.
b) Schreiben Sie eine Prozedur $hms : int \to int * int * int$, die eine in Sekunden angegebene Zeit in das HMS-Format umrechnet. Beispielsweise soll *hms* 3666 das Tupel $(1,1,6)$ liefern. Berechnen Sie die Komponenten des Tupels mithilfe lokaler Deklarationen.

Aufgabe 1.12 Sei die folgende rekursive Prozedurdeklaration gegeben:

```
fun f(n:int, a:int) : int = if n=0 then a else f(n-1, a*n)
```

a) Geben Sie die Rekursionsfolge für den Aufruf $f(3,1)$ an.
b) Geben Sie ein verkürztes Ausführungsprotokoll für den Ausdruck $f(3,1)$ an.
c) Geben Sie ein detailliertes Ausführungsprotokoll für den Ausdruck $f(3,1)$ an. Halten Sie sich dabei an das Beispiel in Abbildung 1.1 (Buch S. 14). Wenn es mehrere direkt ausführbare Teilausdrücke gibt, soll immer der am weitesten links stehende zuerst ausgeführt werden. Sie sollten insgesamt 18 Ausführungsschritte bekommen.

Aufgabe 1.13 Schreiben Sie eine rekursive Prozedur $mul : int * int \to int$, die das Produkt einer natürlichen und einer ganzen Zahl durch wiederholte Addition berechnet. Beschreiben Sie den zugrunde liegenden Algorithmus zunächst mit Rekursionsgleichungen.

Aufgabe 1.14 Der ganzzahlige Quotient $x \; div \; y$ lässt sich aus x durch wiederholtes Subtrahieren von y bestimmen. Schreiben Sie eine rekursive Prozedur $mydiv : int * int \to int$, die für $x \geq 0$ und $y \geq 1$ das Ergebnis $x \; div \; y$ liefert. Geben Sie zunächst Rekursionsgleichungen für $x \; div \; y$ an.

Aufgabe 1.15 Auch der ganzzahlige Rest $x \; mod \; y$ lässt sich aus x durch wiederholtes Subtrahieren von y bestimmen. Schreiben Sie eine rekursive Prozedur
$mymod : int * int \to int$, die für $x \geq 0$ und $y \geq 1$ das Ergebnis $x \; mod \; y$ liefert. Geben Sie dazu zunächst Rekursionsgleichungen für $x \; mod \; y$ an.

Aufgabe 1.16 (Stelligkeit) Schreiben Sie eine rekursive Prozedur $stell : int \to int$, die zu einer Zahl die Stelligkeit ihrer Dezimaldarstellung liefert. Beispielsweise soll $stell\ 117 = 3$ gelten. Geben Sie zunächst die Rekursionsgleichungen für *stell* an. Verwenden Sie ganzzahlige Division durch 10, um die Zahl zu erhalten, die durch Streichen der letzten Ziffer entsteht.

Aufgabe 1.17 (Quersumme) Schreiben Sie eine rekursive Prozedur $quer : int \to int$, die die Quersumme einer ganzen Zahl berechnet. Die Quersumme einer Zahl ist die Summe ihrer Dezimalziffern. Beispielsweise hat die Zahl -3754 die Quersumme 19. Geben Sie zunächst die Rekursionsgleichungen für *quer* an. Verwenden Sie Restbestimmung modulo 10, um die letzte Ziffer einer Zahl zu bestimmen.

Aufgabe 1.21 Schreiben Sie eine Prozedur $mul : int * int \to int$, die das Produkt aus einer natürlichen und einer ganzen Zahl mit einer endrekursiven Hilfsprozedur durch wiederholte Addition berechnet.

Aufgabe 1.22 Schreiben Sie eine Prozedur $quer : int \to int$, die die Quersumme einer ganzen Zahl mithilfe einer endrekursiven Hilfsprozedur berechnet.

Aufgabe 1.23 Unter der Reversion $rev\,n$ einer natürlichen Zahl n wollen wir die natürliche Zahl verstehen, die man durch Spiegeln der Dezimaldarstellung von n erhält. Beispielsweise soll $rev\,1234 = 4321$, $rev\,76 = 67$ und $rev\,1200 = 21$ gelten.

a) Schreiben Sie zunächst eine endrekursive Prozedur $rev' : int * int \to int$, die zu zwei natürlichen Zahlen m und n die Zahl liefert, die sich ergibt, wenn man die reversierte Dezimaldarstellung von n rechts an die Dezimaldarstellung von m anfügt. Beispielsweise soll $rev'(65,73) = 6537$ und $rev'(0,12300) = 321$ gelten. Die Arbeitsweise von rev' ergibt sich aus dem verkürzten Ausführungsprotokoll $rev'(65,73) = rev'(653,7) = rev'(6537,0) = 6537$.
b) Schreiben Sie mithilfe der Prozedur rev' eine Prozedur rev, die natürliche Zahlen reversiert.
c) Machen Sie sich klar, dass die entscheidende Idee bei der Konstruktion des Reversionsalgorithmus die Einführung einer Hilfsfunktion mit einem Akku ist. Überzeugen Sie sich davon, dass Sie rev nicht ohne Weiteres durch Rekursionsgleichungen bestimmen können.

Aufgabe 1.24 Schreiben Sie eine Prozedur $int \to int$, die für negative Argumente divergiert und für nicht-negative Argumente x das Ergebnis x liefert.

Aufgabe 1.25 Dieter Schlau liebt Prozeduren. Er deklariert die Prozedur

```
fun ifi (b:bool, x:int, y:int) = if b then x else y
```

und behauptet, dass er sie anstelle des Konditionals verwenden kann, wenn Konsequenz und Alternative den Typ *int* haben. Anna ist skeptisch und zeigt ihm schließlich die folgenden Deklarationen:

```
fun p (n:int) : int = if n=0 then p(n-1) else n
fun q (n:int) : int = ifi(n=0, q(n-1), n)
```

Dieter kann erst keinen Unterschied im Verhalten der Prozeduren p und q erkennen, aber ein Experiment mit dem Interpreter belehrt ihn eines Besseren.

Für welche Argumente verhalten sich die Prozeduren p und q unterschiedlich? Warum? Welche Eigenschaft des Konditionals geht bei der Verwendung der Prozedur *ifi* verloren?