DEPENDABLE SYSTEMS AND SOFTWARE

Fachrichtung 6.2 — Informatik
Tutoren der Vorlesung

1. Probeklausur zur Programmierung I

Name: Musterlösung Matrikelnummer:

- Bitte öffnen Sie das Klausurheft erst dann, wenn Sie dazu aufgefordert werden.
- Hilfsmittel sind nicht zugelassen. Am Arbeitsplatz dürfen nur Schreibgeräte, Getränke, Speisen und Ausweise mitgeführt werden. Taschen und Jacken müssen an den Wänden des Klausursaals zurückgelassen werden.
- Das Verlassen des Saals ohne Abgabe des Klausurhefts gilt als Täuschungsversuch.
- Wenn Sie während der Bearbeitung zur Toilette müssen, geben Sie bitte Ihr Klausurheft bei der Aufsicht ab. Es kann immer nur eine Person zur Toilette.
- Alle Lösungen müssen auf den bedruckten rechten Seiten des Klausurhefts notiert werden. Die leeren linken Seiten dienen als Platz für Skizzen und werden **nicht korrigiert**. Notizpapier ist nicht zugelassen. Sie können mit Bleistift schreiben.
- Wenn bei einer Aufgabe nichts anderes angegeben ist, dürfen Sie genau die auf dem letzten Blatt des Klausurhefts aufgeführten Hilfsprozeduren benutzen, ohne sie selbst deklarieren zu müssen.
- Für die Bearbeitung der Klausur stehen 90 Minuten zur Verfügung. Sie sollten also mit durchschnittlich 15 Minuten Bearbeitungszeit pro Aufgabe rechnen.
- Bitte legen Sie zur Identifikation Ihren Personalausweis bzw. Reisepass sowie Ihren Studierendenausweis neben sich.

Viel Erfolg!

gut	ok	durchgefallen

Aufgabe 1: (Sortieren)

Schreiben Sie eine Prozedur $sort : int\ list \rightarrow int\ list$, die eine Liste von ganzen Zahlen sortiert.

Sortieren durch Einfügen:

```
fun insert (x, nil)   = [x]
  | insert (x, y::yr) = if x <= y then x::y::yr else y::insert(x, yr)

fun isort xs = foldl insert nil xs
```

Sortieren durch Mischen:

```
fun split xs = foldl (fn (x, (ys, zs)) => (zs, x::ys)) (nil, nil) xs

fun merge (nil, ys)      = ys
  | merge (xs, nil)      = xs
  | merge (x::xr, y::yr) = if x <= y then x::merge(xr, y::yr)
                                     else y::merge(x::xr,yr)

fun msort []  = []
  | msort [x] = [x]
  | msort xs  = let val (ys, zs) = split xs
                in merge(msort ys, msort zs) end
```

Aufgabe 2: (Arithmetische Ausdrücke)

Wir betrachten arithmetische Ausdrücke mit Addition, Multiplikation, Variablen und Konstanten. Variablen identifizieren wir mit ganzen Zahlen vom Typ *int*.

(a) Erweitern Sie den folgenden Datentyp für arithmetische Ausdrücke um eine Operation zur Bestimmung des minimalen Ergebnisses für eine Liste von Ausdrücken:

```
type var = int
datatype exp = Const of int
             | Var of var
             | Add of exp * exp
             | Mul of exp * exp

             | Min of exp list
```

(b) Schreiben Sie eine Prozedur $eval : exp \rightarrow env \rightarrow int$, die einen wie oben definierten Ausdruck in einer Umgebung auswertet. Wird die Minimumsoperation auf eine leere Liste von Ausdrücken angewendet, soll die Ausnahme *Subscript* geworfen werden.

Zur Erinnerung: Umgebungen sind Werte vom Typ $var \rightarrow int$.

```
exception Subscript

fun eval (Const c)     env = c
  | eval (Var v)       env = env v
  | eval (Add(e,e'))   env = eval e env + eval e' env
  | eval (Mul(e,e'))   env = eval e env * eval e' env
  | eval (Min nil)     env = raise Subscript
  | eval (Min (x::xs)) env = foldl (fn (a, b) => if eval a env < b
                                                 then eval a env else b)
                                   (eval x env)
                                   xs
```

Aufgabe 3: (Listen)

(a) Geben Sie die Baumdarstellung der durch

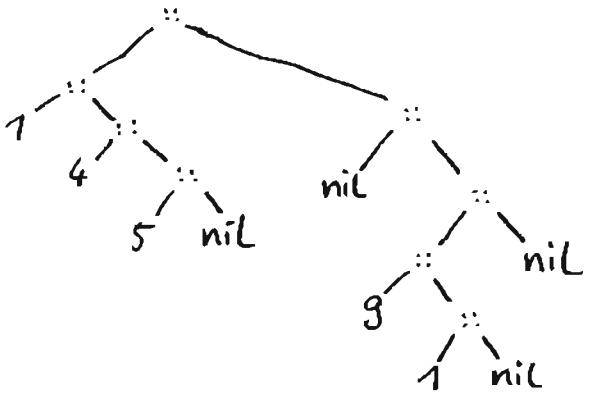
```
[[1, 4, 5], [], [9, 1]]
```

gegebenen Liste an.

$= [1,4,5] :: [] :: [9,1] :: nil$

$= (1 :: 4 :: 5 :: nil) :: nil :: (9 :: 1 :: nil) :: nil$

Baumdarstellung:



(b) Schreiben Sie eine Prozedur *dropex* : *int list* → *int list*, die Minimum und Maximum aus einer Liste ganzer Zahlen entfernt. Beispielsweise soll *dropex* $[1, 1, 4, 5] = [4]$ gelten.

Benutzen Sie keine selbst definierten Hilfsprozeduren und keine `let`-Ausdrücke.

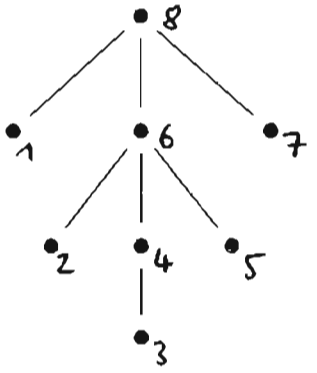
```
fun dropex nil     = nil
  | dropex (x::xr) = #1(foldl (fn (n, (xs, max, min)) => if n = max orelse n = min
                                                         then (xs, max, min)
                                                         else (xs @ [n], max, min))
                              (nil,
                               foldl (fn (a, b) => if a > b then a else b) x xr,
                               foldl (fn (a, b) => if a < b then a else b) x xr)
                              (x::xr))
```

Aufgabe 4: (Bäume)

In dieser Aufgabe betrachten wir reine Bäume, realisiert durch die Typdeklaration

```
datatype tree = T of tree list
```

(a) Geben Sie die Konstruktordarstellung des folgenden Tannenbaums an und nummerieren Sie seine Knoten nach der Postordnung.



T [T nil,
T [T nil, T [T nil], T nil],
T nil]

(b) Schreiben Sie eine Prozedur *mast'* : *int list* → *tree*, die den kleinsten Baum liefert, in dem durch das Argument ein Knoten adressiert wird. Ist die übergebene Adresse ungültig, soll eine passende Ausnahme geworfen werden.

```
exception Argument

fun listtree n ts = if n < 0 then raise Argument
                    else if n = 0 then ts
                    else listtree (n - 1) ((T nil)::ts)

fun mast' nil     = T nil
  | mast' (x::xr) = T (listtree (x - 1) [mast' xr])
```

(c) Benutzen Sie die Prozedur *mast'* von oben, um eine Prozedur *mast* : *int list* → *tree option* zu schreiben, die für ungültige Adressen eine uneingelöste Option zurückgibt.

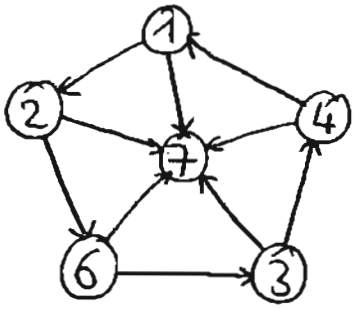
```
fun mast xs = SOME (mast' xs) handle Argument => NONE
```

Aufgabe 5: (Graphen)

Zeichnen Sie den durch

$$V = \{1, 2, 3, 4, 6, 7\},\ E = \{(1, 2), (2, 6), (3, 4), (6, 3), (4, 1), (6, 7), (2, 7), (3, 7), (4, 7), (1, 7)\}$$

gegebenen Graphen, ohne daß sich Kanten überkreuzen.
Geben Sie seine Größe und seine Tiefe an. Geben Sie, wenn vorhanden, Quellen, Senken und Wurzeln an. Ist der Graph zyklisch? Wenn ja, geben Sie einen Zyklus an. Ist er zusammenhängend, stark zusammenhängend oder baumartig?



Größe 6
Tiefe 5
Quellen: keine
Senken: 7
Wurzeln: 1, 2, 3, 4, 6
Zyklisch: Ja, Zyklus <1,2,6,3,4,1>
Zusammenhängend: Ja
Stark zusammenhängend: Nein
Baumartig: Nein

Aufgabe 6: (Typen und Bezeichnerbindung)

(a) Geben Sie das Typschema an, mit dem der Interpreter die Prozedur

```
fun f a b c d = a = b c = d
```

typisieren wird. Achten Sie darauf, daß Ihr Typschema minimal geklammert ist.

Hinweis: Der Gleichheitsoperator = klammert nach links.

$$'\alpha \to (\beta \to '\alpha) \to \beta \to bool \to bool$$

(b) Bereinigen Sie das folgende Programm durch Indizieren der benutzenden Bezeichnerauftreten und Überstreichen der definierenden Bezeichnerauftreten:

```
exception e1 of int

val (x1, y1, z1) = ( () , 2 , Empty )

fun f1 x2 = ( x2 ; raise e1 y1)

val x3 = f1 x1 handle y2 => y2

val q1 = x3
```

(c) Geben Sie zu jedem definierten Bezeichner des obigen Programms den Typ an. Woran wird q nach Ausführen des Programms gebunden?

e_1: int → exn
x_1: unit
y_1: int
z_1: exn
f_1: $\alpha \to \beta$
x_2: α
x_3: exn
y_2: exn
q_1: exn

Am Ende: q wird an (e2) gebunden.