Technische Universität München
Fakultät für Informatik
R. Kuntschke, I. Nagy,
Dr. M. Schneider

SS 2006
Lösungsvorschläge zu Blatt 3
26. Mai 2006

Übungen zu Einführung in die Informatik I

Aufgabe 9 Arbeiten mit Modulen (Lösungsvorschlag)

Zur besseren Übersicht und um gleichnamige Operationen für verschiedenen Datentypen zu unterscheiden verwendet OCaml sogenannte *Module*. Ein Beispiel ist: `Random.float`, das die Funktion *float* aus dem Modul *Random* aufruft.

In dieser Aufgabe verwenden wir Funktionen aus dem Modul *List* zur Operation auf Listen von Datentypen.

a) Definieren und belegen Sie in OCaml die beiden Zeichensequenzen `vorname` und `nachname` jeweils mit Ihrem Vor- bzw. Nachnamen. Verknüpfen Sie anschließend diese beiden Zeichensequenzen zu einer einzigen Zeichensequenz `name` unter Verwendung des Konkatenationsoperator `@` und einem Komma sowie einem Leerzeichen zwischen den beiden Namensteilen.

```
# let nachname = ['l'; 'e'; 'g'; 'r'; 'a'; 'n'; 'd']
  and vorname = ['w'; 'i'; 'l'; 'l'; 'i'; 'a'; 'm'];;
val nachname : char list = ['l'; 'e'; 'g'; 'r'; 'a'; 'n'; 'd']
val vorname : char list = ['w'; 'i'; 'l'; 'l'; 'i'; 'a'; 'm']


# let name = nachname @ [','] @ [' '] @ vorname;;
val name : char list =
  ['l'; 'e'; 'g'; 'r'; 'a'; 'n'; 'd'; ','; ' ';
   'w'; 'i'; 'l'; 'l'; 'i'; 'a'; 'm']
```

b) Verwenden Sie die Funktion *length* aus dem Modul *List* um die Länge des ganzen Namens festzustellen.

```
# List.length name;;
- : int = 16
```

c) Revertieren Sie Ihren (oben definierten) Namen mit Hilfe von OCaml durch Zerlegung und unter ausschließlicher Verwendung der Funktionen *hd* und *tl* aus dem Modul *List* und des Operators `::` ('Prepend').
Hinweis: Zur Vereinfachung der Schreibweise bietet es sich an zuerst zwei Funktionen `head` und `tail` zu definieren, welche genau die entsprechenden Funktionen aus dem Modul *List* aufrufen.

```
# let head = List.hd;;
val head : 'a list -> 'a = <fun>
# let tail = List.tl;;
val tail : 'a list -> 'a list = <fun>

# head(tail(tail(tail(tail(tail(tail(name)))))) ::
head(tail(tail(tail(tail(tail(name))))) ::
head(tail(tail(tail(tail(name)))) ::
head(tail(tail(tail(name)))) ::
head(tail(tail(name))) ::
head(tail(name)) ::
[head(name)]
;;
- : char list = ['d'; 'n'; 'a'; 'r'; 'g'; 'e'; 'l']
```

d) Definieren Sie die folgende *Cast*-Funktion `list_of_string` die mit Hilfe von Funktionen aus den Modulen *String* und *Stream* aus einem String eine Liste von Einzelzeichen macht.

```
let list_of_string s =
  Stream.npeek (String.length s) (Stream.of_string s);;
```

Testen Sie diese Funktion mit ein paar Eingabestrings.

```
# list_of_string("evitez les courrants d'air");;
- : char list =
['e'; 'v'; 'i'; 't'; 'e'; 'z'; ' '; 'l'; 'e'; 's'; ' ';
 'c'; 'o'; 'u'; 'r'; 'r'; 'a'; 'n'; 't'; 's'; ' '; 'd'; '\''; 'a'; 'i'; 'r']
```

e) Definieren Sie nun eine rekursive Funktion *revert*, die eine beliebige Liste umkehrt.
Testen Sie diese Funktion für Ihren oben definierten Namen sowie für den String `reliefpfeiler` (unter Verwendung der Cast-Funktion aus Teilaufgabe c).

```
let rec revert l = match l with
  | []    -> []
  | a::l  -> revert l @ [a];;

# revert name;;
- : char list =
['m'; 'a'; 'i'; 'l'; 'l'; 'i'; 'w'; ' '; ',';
 'd'; 'n'; 'a'; 'r'; 'g'; 'e'; 'l']

# revert (list_of_string("reliefpfeiler"));;
- : char list =
['r'; 'e'; 'l'; 'i'; 'e'; 'f'; 'p'; 'f'; 'e'; 'i'; 'l'; 'e'; 'r']
```

Aufgabe 10 Komplexe Zahlen (Lösungsvorschlag)

a)
```
# type complex = Complex of float * float;;
type complex = Complex of float * float
# Complex(3.0,5.5);;
- : complex = Complex (3., 5.)
```

b)
```
# let add (Complex(x,y)) (Complex(u,v)) = Complex(x +. u, y +. v);;
val add : complex -> complex -> complex = <fun>
# add (Complex(3.0,5.5)) (Complex(1.0,1.0));;
- : complex = Complex (4., 6.5)
```

c)
```
# let (++) (Complex(x,y)) (Complex(u,v)) = Complex(x +. u, y +. v);;
val ( ++ ) : complex -> complex -> complex = <fun>
# (Complex(3.0,5.5)) ++ (Complex(1.0,1.0));;
- : complex = Complex (4., 6.5)
```

Hinweis: In OCaml gibt es kein Überladen von Funktionen und Operatoren (d.h.: es gibt nicht mehrere Definitionen einer Funktion oder eines Operators für verschiedene Datentypen). Der Operator ++ ist in OCaml standardmäßig nicht vorhanden. Gäbe es ihn würde er durch diese Aufgabe überschrieben werden.

d)
```
# type complex = Complex of float * float
                | Polar of float * float;;
# let pi = 2. *. asin 1.;;
# let polar c = match c with
        Complex(x,y) ->
            let r = sqrt(x *. x +. y *. y) in
            let psi = if y < 0. then acos((-. x) /. r) -. pi
                                else acos(x /. r) in
            Polar(r,psi)
      |  _ -> c;;
val polar : complex -> complex = <fun>
# polar (Complex(3.0,5.5));;
- : complex = Polar (6.2649820430708338, 1.0714496051147666)
# polar (Complex(0.0,1.0));;
- : complex = Polar (1., 1.5707963267948966)
# polar (Polar(1., 1.));;
- : complex = Polar (1., 1.)
# let normal c = match c with
          Polar(r,psi) ->
                let x = r *. cos psi in
                let y = r *. sin psi in
                Complex(x,y)
        | _ -> c;;
val normal : complex -> complex = <fun>
# normal (Polar(1., 0.));;
- : complex = Complex (1., 0.)
# normal(polar(Complex(1.0,1.0)));;
- : complex = Complex (1., 1.0000000000000002)
```

e)
```
# let rec (++) a b = match (a,b) with
        (Complex(x,y),Complex(u,v)) -> Complex(x +. u, y +. v)
        | (_,Polar(_,_)) -> a ++ (normal(b))
        | (Polar(_,_),_) -> (normal(a)) ++ b;;
```

```
val ( ++ ) : complex -> complex -> complex = <fun>
```

Alternativ:

```
# let rec add a b = match (a,b) with
        (Complex(x,y),Complex(u,v)) -> Complex(x +. u, y +. v)
        | (_,Polar(_,_)) -> add a (normal(b))
        | (Polar(_,_),_) -> add (normal(a)) b;;
val add : complex -> complex -> complex = <fun>
# let (++) = add;;
val ( ++ ) : complex -> complex -> complex = <fun>
# (Complex(3.0,5.5)) ++ (polar(Complex(1.0,1.0)));;
- : complex = Complex (4., 6.5)
```

Aufgabe 11 Kalender des Jahres 2006 (Lösungsvorschlag)

Eine mögliche Lösung könnte folgendermaßen aussehen:

```
type monat = Januar | Februar | Maerz | April | Mai | Juni | Juli | August | September |
             Oktober | November | Dezember;;

let zahl_zu_monat (z) = match z with
| 1 -> Januar | 2 -> Februar | 3 -> Maerz | 4 -> April | 5 -> Mai | 6 -> Juni | 7 -> Juli
| 8 -> August | 9 -> September | 10 -> Oktober | 11 -> November | 12 -> Dezember
| _ -> failwith "Falscher Monat";;

let monat_zu_zahl (m) = match m with
| Januar -> 1 | Februar -> 2 | Maerz -> 3 | April -> 4 | Mai -> 5 | Juni -> 6 | Juli -> 7
| August -> 8 | September -> 9 | Oktober -> 10 | November -> 11 | Dezember -> 12;;

type wochentag = Montag | Dienstag | Mittwoch | Donnerstag | Freitag | Samstag | Sonntag;;

let zahl_zu_wochentag (z) = match z with
| 0 -> Montag | 1 -> Dienstag | 2 -> Mittwoch | 3 -> Donnerstag
| 4 -> Freitag | 5 -> Samstag | 6 -> Sonntag
| _ -> failwith "Falscher Wochentag";;

type datum = Datum of (int * monat * int);;

let schaltjahr (jahreszahl) = (jahreszahl mod 4 = 0) &&
                               not (jahreszahl mod 100 = 0) ||
                              (jahreszahl mod 400 = 0);;

let anzahlTageImJahr (jahreszahl) =
  if schaltjahr (jahreszahl) then 366 else 365;;

let monatslaenge (monat, jahr) = match monat with
  | Januar | Maerz | Mai | Juli | August | Oktober | Dezember -> 31
  | Februar when (schaltjahr jahr) -> 29
  | Februar -> 28
  | _ -> 30;;

let ersterErsterWelcherWochentag (j) =
```

```
  if (j < 1900) || (j > 2099) then
    failwith "Jahr ungueltig"
  else
    let jahr = j - 1900 in
      let schalt = jahr / 4 in
        let versatz = jahr + schalt in
          versatz mod 7;;

let welcherWochentag (t, m, j) =
  let rec monatstage (monat, tage) = match monat with
  | ende when (ende = m) -> tage + t - 1 + ersterErsterWelcherWochentag (j)
  | _ -> tage + monatstage(monat + 1, monatslaenge(zahl_zu_monat(monat), j))
  in
    if schaltjahr(j) then zahl_zu_wochentag((monatstage(1, 0) - 1) mod 7)
    else zahl_zu_wochentag(monatstage(1, 0) mod 7)

let anzahlTageSeit1_1_1900(t, m, j) =
  let rec zaehleMonate(m, anz, j) = match m with
  | 0 -> anz + t - 1
  | _ -> anz + zaehleMonate(m - 1, monatslaenge(zahl_zu_monat(m), j), j) in
    let rec zaehleJahre(j, ende, anz) = match j with
    | jahr when (jahr = ende) -> zaehleMonate(monat_zu_zahl(m) - 1, anz, j)
    | _ -> anz + zaehleJahre(j + 1, ende, anzahlTageImJahr(j))
    in zaehleJahre(1900, j, 0);;

let anzahlTageSeit1_1_1900zuDatum(z) =
  let rec zaehleMonate(j, monatstage, monate) = match monatstage with
  | schranke1 when (schranke1 < monatslaenge(zahl_zu_monat(monate), j))
    -> [welcherWochentag (monatstage + 1, monate, j),
        Datum(monatstage + 1, zahl_zu_monat(monate), j)]
  | _ -> zaehleMonate(j, monatstage - monatslaenge(zahl_zu_monat(monate), j), monate + 1) in
    let rec zaehleJahre(j, anz) = match anz with
    | schranke2 when (schranke2 < anzahlTageImJahr(j)) -> zaehleMonate(j, schranke2, 1)
    | _ -> zaehleJahre(j + 1, anz - anzahlTageImJahr(j))
    in zaehleJahre(1900, z);;

let kalender(t1, m1, j1, t2, m2, j2) =
  let anz1 = anzahlTageSeit1_1_1900(t1, m1, j1)
  and anz2 = anzahlTageSeit1_1_1900(t2, m2, j2)
  in
  let rec iteriereTage(tag, datumliste) = match tag with
  | ende when (ende = anz2) -> datumliste
  | _ -> iteriereTage(tag + 1, datumliste @ anzahlTageSeit1_1_1900zuDatum(tag))
  in iteriereTage(anz1, []);;

kalender(30, Dezember, 1912, 2, Maerz, 1913);;
```

Aufgabe 12 Kalender für beliebige Zeiträume (Lösungsvorschlag)

Eine mögliche Lösung könnte folgendermaßen aussehen:

```
type monat = Januar | Februar | Maerz | April | Mai | Juni | Juli | August | September |
             Oktober | November | Dezember;;
```

```
let zahl_zu_monat (z) = match z with
| 1 -> Januar | 2 -> Februar | 3 -> Maerz | 4 -> April | 5 -> Mai | 6 -> Juni | 7 -> Juli
| 8 -> August | 9 -> September | 10 -> Oktober | 11 -> November | 12 -> Dezember
| _ -> failwith "Falscher Monat";;

let monat_zu_zahl (m) = match m with
| Januar -> 1 | Februar -> 2 | Maerz -> 3 | April -> 4 | Mai -> 5 | Juni -> 6 | Juli -> 7
| August -> 8 | September -> 9 | Oktober -> 10 | November -> 11 | Dezember -> 12;;

type wochentag = Montag | Dienstag | Mittwoch | Donnerstag | Freitag | Samstag | Sonntag;;

let zahl_zu_wochentag (z) = match z with
| 0 -> Montag | 1 -> Dienstag | 2 -> Mittwoch | 3 -> Donnerstag
| 4 -> Freitag | 5 -> Samstag | 6 -> Sonntag
| _ -> failwith "Falscher Wochentag";;

type datum = Datum of (int * monat * int);;

let schaltjahr (jahreszahl) = (jahreszahl mod 4 = 0) &&
                               not (jahreszahl mod 100 = 0) ||
                              (jahreszahl mod 400 = 0);;

let anzahlTageImJahr (jahreszahl) =
  if schaltjahr (jahreszahl) then 366 else 365;;

let monatslaenge (monat, jahr) = match monat with
  | Januar | Maerz | Mai | Juli | August | Oktober | Dezember -> 31
  | Februar when (schaltjahr jahr) -> 29
  | Februar -> 28
  | _ -> 30;;

let ersterErsterWelcherWochentag (j) =
  if (j < 1900) || (j > 2099) then
    failwith "Jahr ungueltig"
  else
    let jahr = j - 1900 in
      let schalt = jahr / 4 in
        let versatz = jahr + schalt in
          versatz mod 7;;

let welcherWochentag (t, m, j) =
  let rec monatstage (monat, tage) = match monat with
  | ende when (ende = m) -> tage + t - 1 + ersterErsterWelcherWochentag (j)
  | _ -> tage + monatstage(monat + 1, monatslaenge(zahl_zu_monat(monat), j))
  in
    if schaltjahr(j) then zahl_zu_wochentag((monatstage(1, 0) - 1) mod 7)
    else zahl_zu_wochentag(monatstage(1, 0) mod 7)

let anzahlTageSeit1_1_1900(t, m, j) =
  let rec zaehleMonate(m, anz, j) = match m with
  | 0 -> anz + t - 1
  | _ -> anz + zaehleMonate(m - 1, monatslaenge(zahl_zu_monat(m), j), j) in
    let rec zaehleJahre(j, ende, anz) = match j with
    | jahr when (jahr = ende) -> zaehleMonate(monat_zu_zahl(m) - 1, anz, j)
```

```
    | _ -> anz + zaehleJahre(j + 1, ende, anzahlTageImJahr(j))
    in zaehleJahre(1900, j, 0);;

let anzahlTageSeit1_1_1900zuDatum(z) =
  let rec zaehleMonate(j, monatstage, monate) = match monatstage with
  | schranke1 when (schranke1 < monatslaenge(zahl_zu_monat(monate), j))
    -> [welcherWochentag (monatstage + 1, monate, j),
        Datum(monatstage + 1, zahl_zu_monat(monate), j)]
  | _ -> zaehleMonate(j, monatstage - monatslaenge(zahl_zu_monat(monate), j), monate + 1) in
    let rec zaehleJahre(j, anz) = match anz with
    | schranke2 when (schranke2 < anzahlTageImJahr(j)) -> zaehleMonate(j, schranke2, 1)
    | _ -> zaehleJahre(j + 1, anz - anzahlTageImJahr(j))
    in zaehleJahre(1900, z);;

let kalender(t1, m1, j1, t2, m2, j2) =
  let anz1 = anzahlTageSeit1_1_1900(t1, m1, j1)
  and anz2 = anzahlTageSeit1_1_1900(t2, m2, j2)
  in
  let rec iteriereTage(tag, datumliste) = match tag with
  | ende when (ende = anz2) -> datumliste
  | _ -> iteriereTage(tag + 1, datumliste @ anzahlTageSeit1_1_1900zuDatum(tag))
  in iteriereTage(anz1, []);;

kalender(30, Dezember, 1912, 2, Maerz, 1913);;
```