Ein fiktives Kartenspiel für eine Person

(1. Prüfungsvorleistung, A107 Programmierparadigmen, korrigiert 2020-12-08)

1 Spielregeln

- Das Spiel beinhaltet einen *Kartenstapel* und einen *Zielwert*.
- Die Liste der *Handkarten* ist zu Beginn leer.
- Man führt einen *Zug* aus, indem entweder
  - die oberste Karte vom Kartenstapel *aufgenommen* und zur Liste der Handkarten hinzugefügt oder
  - eine beliebige Karte aus der Handkartenliste *abgelegt*, also entfernt wird.
- Das Spiel endet
  - wenn kein Zug mehr gewünscht ist oder
  - sobald die Summe der Werte der Handkarten den Zielwert überschreitet.
- Ziel des Spiels ist, es mit einem möglichst niedrigen *Punktestand* zu beenden.
- Der Punktestand wird wie folgt berechnet. Sei S die Summe der Werte der Handkarten und Z der Zielwert. Dann gilt:

$$P = \begin{cases} 3 \cdot (S - Z) & \text{falls } S > Z \\ Z - S & \text{falls } S \leq Z \end{cases}$$

  Der effektive Punktestand ist P, es sei denn, alle Handkarten haben die gleiche Farbe. In diesem Fall ist der Punktestand nur $P \,/\, 2$, wobei ganzzahlig dividiert wird (Operator `div`).

2 Technische Vorgaben

- Schreiben Sie Ihre Lösung in Standard ML.
- Verwenden Sie *Pattern Matching* zum Zugriff auf Daten geeigneter Typen.
- Verwenden Sie **nicht** die Standardfunktionen `null`, `hd`, `tl`.
- Verwenden Sie **nicht** die #-Notation.

3 Inhaltliche Vorgaben

1. Ihre Lösung wird ca. 50–100 Zeilen benötigen.
2. Verwenden Sie die folgenden Datentypen:

```
datatype reihe = Kreuz | Pik | Herz | Karo
datatype wert = Zahl of int | Bube | Dame | Koenig | Ass
  (* Als Zahlen kommen nur 2 .. 10 vor (nicht geprueft) *)
type karte = reihe * wert

datatype farbe = Schwarz | Rot
datatype zug = Ablegen of karte | Aufnehmen

exception IllegalerZug
```

sowie den vordefinierten algebraischen Datentyp `'a list` mit den Konstruktoren `[]` und `::` (Operator).

3. Implementieren Sie folgende Funktionen:
   - `kartenfarbe : karte -> farbe` — liefert die Farbe einer Karte. Kreuz und Pik sind schwarz, Herz und Karo sind rot.
   - `kartenwert : karte -> int` — liefert den effektiven Wert einer Karte. Zahlen zählen wie angegeben, Asse 11, alle anderen Bilder 10.
   - `entferne_karte : karte list * karte -> karte list` — nimmt eine Liste von Karten und eine einzelne Karte, und liefert eine Restliste ohne die Karte zurück. Es wird höchstens ein Exemplar aus der Liste entfernt. Kommt die gesuchte Karte nicht vor, soll die oben definierte Exception erzeugt werden (Operator **`raise`**). Karten können mit dem Operator = verglichen werden.
   - `alle_farben_gleich : karte list -> bool` — liefert `true` wenn alle Karten in der Liste die gleiche Farbe haben.
   - `kartensumme : karte list -> bool` — liefert die Summe der Werte einer Liste von Karten.
   - `punktestand : karte list * int -> int` — berechnet aus einer Liste von Karten und einem Zielwert den Punktestand wie oben spezifiziert.
   - `spielablauf : karte list * zug list * int -> int` — nimmt den Kartenstapel, eine Liste von Zügen und einen Zielwert, arbeitet die Züge der Reihe nach ab und liefert den Punktestand bei Spielende. Verwenden Sie eine lokal definierte rekursive Hilfsfunktion, deren Argumente den ganzen aktuellen Spielzustand repräsentieren, und die die Handkarten am Spielende zurückliefert.

4. Ihre Lösung sollte mindestens die folgenden Tests bestehen:

```
val test1 = kartenfarbe (Kreuz, Zahl 2) = Schwarz

val test2 = kartenwert (Kreuz, Zahl 2) = 2

val test3 = entferne_karte ([(Herz, Ass)], (Herz, Ass)) = []

val test4 = alle_farben_gleich [(Herz, Ass), (Herz, Dame)] = true

val test5 = kartensumme [(Kreuz, Zahl 2), (Kreuz, Zahl 2)] = 4

val test6 = punktestand ([(Herz, Zahl 2), (Kreuz, Zahl 4)], 10) = 4

val test7 = spielablauf ([(Herz, Zahl 2), (Kreuz, Zahl 4)],
                         [Aufnehmen], 15) = 6

val test8 = spielablauf ([(Kreuz, Ass), (Pik, Ass), (Herz, Ass),
                          (Karo, Ass)],
                         [Aufnehmen, Aufnehmen, Aufnehmen,
                          Aufnehmen, Aufnehmen], 42) = 6

fun illegal f = (case f() of _ => false)
    handle
    IllegalerZug => true

val test9 = illegal (fn () =>
    spielablauf ([(Kreuz, Bube), (Pik, Zahl(8))],
                 [Aufnehmen, Ablegen(Herz, Bube)], 42))
```