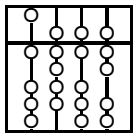
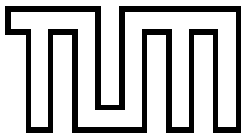


TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



Lehrstuhl für Sprachen und Beschreibungsstrukturen
Prof. Dr. Helmut Seidl
A. Berlea, F. Forster, T. Gawlitza

SoS 2007
Übungsblatt 4
Ausgegeben am: 25. Mai 2007
Abgabe bis: **8. Juni 2007**

Übungen zu Einführung in die Informatik II

Aufgabe 1 (H) Einfache Funktionen (3 Punkte)

Definieren Sie folgende Funktionen in OCaml:

a) Die Betrags-Funktion, die den Betrag eines Wertes zurückliefert.

b) Die Signum-Funktion, die für negative Werte -1, für positive Werte 1 und sonst 0 zurückliefert.

c) Die Minimum-Funktion, die für zwei Parameter jeweils den kleineren Wert zurückliefert.

d) Eine Funktion `app_4`, die eine gegebene Funktion *h* viermal nacheinander auf ein Argument *x* anwendet.

e) Eine Funktion `app_n`, die eine Funktion *h*, sowie einen Wert *x* und eine Zahl *n* als Argumente erhält und dann *h* *n*-mal auf *x* anwendet.

f) Eine Funktion `app`, die Funktionen *h* und *g*, sowie einen Wert *x* und eine Zahl *n* als Argumente erhält und dann *h* $g(n)$-mal auf *x* anwendet.

Aufgabe 2 (H) Mengen als Listen (4 Punkte)

Wir repräsentieren Mengen von Elementen vom Typ `'a` als Listen vom Typ `'a list`. Implementieren Sie folgende Funktionen:

a) `filter` mit der Signatur `'a list -> ('a -> bool) -> 'a list`. Ein Aufruf `filter l p` soll die Liste aller in `l` enthaltenen Elemente, die das Prädikat `p` erfüllen, zurückliefern;

b) `union` mit der Signatur `'a list -> 'a list -> 'a list` zur Vereinigung;

c) `intersection` mit der Signatur `'a list -> 'a list -> 'a list` zur Bildung des Durchschnitts;

d) `difference` mit der Signatur `'a list -> 'a list -> 'a list` zur Bildung der Mengendifferenz.

Aufgabe 3 Mengen als sortierte Listen

Wir repräsentieren Mengen von (vergleichbaren) Elementen vom Typ `'a` als sortierte Listen vom Typ `'a list`. Implementieren Sie Funktionen wie in der vorherigen Aufgabe.

Aufgabe 4 (H) Finanzberater **(4 Punkte)**

a) Schreiben Sie eine Funktion `rs : float * float * float * int -> float`, die als Argumente einen monatlichen Zinssatz `z`, eine Monatsrate `r`, einen Darlehenbetrag `b` und eine Anzahl von Monaten `m` erhält. Ein Aufruf `rs (z, r, b, m)` soll die Restschuld nach `m` Monaten berechnen (0 wenn die Schuld abbezahlt ist). Bezeichnet rs_k die Restschuld nach k Monaten, so ist $rs_0 = b$ und $rs_k = rs_{k-1} + rs_{k-1} \cdot z - r$ für $k \geq 1$.

b) Schreiben Sie die Funktion `rs` so um, dass sie die Signatur `float -> float -> float -> int -> float` hat.

c) Schreiben Sie folgende Funktionen:

(i) `berater1 : float -> float -> int -> float`; Ein Aufruf `berater1 r b m` soll die Restschuld bei einem monatlichen Zinssatz von `0.004` berechnen.

(ii) `berater2 : float -> float -> int -> float`; Ein Aufruf `berater2 r b m` soll die Restschuld bei einem monatlichen Zinssatz von `0.0035` berechnen.

(iii) `berater3 : float -> int -> float`; Ein Aufruf `berater3 b m` soll die Restschuld bei einem monatlichen Zinssatz von `0.004` und einer Rate von 1000 Euro berechnen.

(iv) `berater4 : float -> int -> float`; Ein Aufruf `berater4 b m` soll die Restschuld bei einem monatlichen Zinssatz von `0.004` und einer Rate von 2000 Euro berechnen.

Aufgabe 5 (H) Mergesort **(6 Punkte)**

Mergesort ist ein rekursiver Sortieralgorithmus, der gemäß dem Divide-and-Conquer-Prinzip arbeitet. Hier soll Mergesort zum Sortieren von Listen implementiert werden. Die prinzipielle Arbeitsweise des Algorithmus läßt sich wie folgt skizzieren.

- Null-/und einelementige Listen sind sortiert
- Listen mit mehr als einem Element werden in zwei möglichst gleich große Teillisten aufgeteilt. Die Teillisten werden durch einen rekursiven Aufruf von Mergesort sortiert und anschließend in einem Mischschritt zu einer sortierten Liste zusammengesetzt.

Um dieses Prinzip umzusetzen, sollen Sie folgende Funktionen definieren:

a) Eine Funktion `init : 'a list -> 'a list list`. Der Aufruf `init` $[x_1;x_2;\ldots;x_n]$ soll die Liste `[` $[x_1]$`;` $[x_2]$`;...;` $[x_n]$ `]` liefern.

b) Die Funktion `merge : 'a list -> 'a list -> 'a list` soll den Mischschritt implementieren. Als Argumente erhält sie **bereits sortierten** Listen `l1` und `l2`. Als Ergebnis liefert sie die sortierte Liste aller Elemente aus `l1` und `l2` zurück.

c) Eine Funktion `merge_list : 'a list list -> 'a list list`. Ein Aufruf `merge_list` $[l_1;l_2;\ldots;l_n]$ soll die Liste $[l_{1,2};l_{3,4};\ldots;l_{n-1,n}]$ falls n gerade ist und die Liste $[l_{1,2};l_{3,4};\ldots;l_{n-2,n-1};l_n]$ andernfalls liefern. Dabei bezeichne $l_{i,j}$ die sortierte Liste, die genau die Elemente aus l_i und l_j enthält. Das Ergebnis enthält also höchstens $\frac{n}{2}+1$ Elemente.

d) Definieren Sie schließlich die Funktion `mergesort : 'a list -> 'a list` mit Hilfe der obigen Funktionen.