P. Thiemann, S. Wehr Wintersemester 2006/2007

Compilerbau

http://proglang.informatik.uni-freiburg.de/teaching/compilerbau/2006ws/

Übungsblatt 1

31.10.2006

Aufgabe 1 (Listen und Bäume in OCaml; 6 Punkte)

(a) Schreibe mittels expliziter Rekursion eine Funktion `inc_list_list` mit Typ `int list list -> int list list`, welche alle Elemente einer Liste um 1 inkrementiert. Schreibe dieselbe Funktion durch zweimaliges Benutzen von `List.map`.

(b) Schreibe mittels expliziter Rekursion eine Funktion `concat` vom Typ `string -> string list -> string`. Der Aufruf `concat sep l` soll die Strings in `l` konkatenieren; `sep` ist dabei das Trennzeichen. Beispiele:

- `concat ", " ["a"; "b"]` liefert `"a, b"`
- `concat ", " ["a"; "b"; ""]` liefert `"a, b, "`

Implementiere dieselbe Funktion mit Hilfe von `List.fold_right`.

(c) Erweitere die in der Vorlesung vorgestellte Datenstruktur für binäre Suchbäume, so dass nun nicht mehr nur einzelne Werte, sondern Schlüssel–Wert Paare in einem Suchbaum gespeichert werden können. Der Datentyp für binäre Suchbäume sieht also wie folgt aus:

```
type ('a, 'b) tree = Node of ('a * 'b * ('a, 'b) tree * ('a, 'b) tree)
                   | Leaf
```

Implementiere nun die folgenden beiden Operationen:

```
insert : 'a -> 'b -> ('a, 'b) tree -> ('a, 'b) tree
find : 'a -> ('a, 'b) tree -> 'b option
```

Aufgabe 2 (Arithmetische Ausdrücke mit OCaml; 6 Punkte)

In dieser Aufgabe geht es um arithmetische Ausdrücke. Ein arithmetischer Ausdruck ist dabei entweder eine ganze Zahl oder ein binärer Operator angewendet auf zwei Operanden. Als binäre Operatoren stehen Addition, Subtraktion, Multiplikation und ganzzahlige Division ohne Rest zur Verfügung.

(a) Schreibe einen algebraischen Datentyp `expr` der obige Definition von arithmetischen Ausdrücken implementiert. Gebe die Repräsentation der Term $t_1 \equiv 4 + 6 * (8 - 3)$ und $t_2 \equiv (4+6)/(4-4)$ an. Schreibe dann eine Evaluierungsfunktion `eval` vom Typ `expr -> int`, welche einen arithmetischen Ausdrück auswertet.

(b) Verbessere deine Evaluierungsfunktion so dass die Ausnahme `Division_by_zero`, welche z.B. beim Auswerten von t_2 auftritt, verhindert wird. Die verbesserte Evaluierungsfunktion soll den Typ `expr -> int option` haben.

(c) Die Definition von arithmetischen Ausdrücken soll um zwei Konstrukte erweitert werden. Eine Variable ist nun auch ein arithmetischer Ausdruck. Außerdem wird ein Bindungsmechanismus für Variablen eingefürt. Die abstrakte Syntax von arithmetischen Ausdrücken sieht nun also wie folgt aus (dabei bezeichnet x eine Variable und n eine ganze Zahl):

$$\begin{aligned} op &::= + \mid - \mid * \mid / \\ e &::= n \mid e\ op\ e \mid x \mid \texttt{let}\ x\ \texttt{be}\ e\ \texttt{in}\ e \end{aligned}$$

Erweitere deinen Datentyp für arithmetische Ausdrücke um die zwei Konstrukte. Variablen sollen dabei als Strings repräsentiert werden. Passe auch die Evaluierungsfunktion an, die jetzt den Typ `(String, int) tree -> expr -> int option` haben soll. Die Variablen sollen dabei in dem Suchbaum vom Typ `(String, int) tree` an ihre Werte gebunden werden. Teste deine Implementierung an dem Ausdruck

$$\begin{array}{l} \texttt{let}\ x\ \texttt{be}\ 4+5\ \texttt{in} \\ \texttt{let}\ y\ \texttt{be}\ 2*3\ \texttt{in} \\ \quad x+y+3 \quad . \end{array}$$

Aufgabe 3 (T–Diagramme; 4 Punkte)
Auf einem Flohmarkt in den USA findest Du zwei Magnetbänder, die die obskure Aufschrift "Multics MACLisp Compiler" tragen. Du ahnst gleich, dass Du hier einen historischen Schatz in den Händen hälst, gehst in das nächste Rechenzentrum damit und inspizierst den Fund genauer. Wie Du bald herausfindest, handelt es sich um die letzten beiden existierenden Kopien eines Compilers für den legendären Lisp-Dialekt MACLisp für den Multics-Rechner GE-635[1]. Das erste Band enthält die Quelldateien des Compilers, geschrieben in *PL/I* und *MACLisp*, das zweite Band enthält den Compiler selber als ausführbares Programm. Das PL/I-Programm ist nur ein Treiber-Programm des Compilers und erzeugt selber keinen Zielcode.

Um diesen einmaligen Schatz für immer für die Menschheit zu kultivieren und wieder einsetzbar zu machen, beschließt Du sofort, den Multics MACLisp-Compiler auf Deinen Linux-Rechner zu portieren. Zum Glück hat Dein Nachbar, ein altes IBM-Urgestein, noch einen PL/I-Compiler für seinen PC zu Hause, den er Dir gerne zur Verfügung stellt.

Welchen Programmcode musst Du zusätzlich noch erstellen bzw. Dir besorgen, welchen musst Du adaptieren, damit dieses Unterfangen gelingen kann? Begründe Deine Lösung mit Hilfe eines Schaubilds von T-Diagrammen.

Abgabe: 8.11.2006
Die Abgabe erfolgt vor der Übungsstunde. Code soll in gedruckter Form abgegeben werden. Für Plagiate werden keine Punkte vergeben.

[1] `http://www.multicians.org/lcp.html`