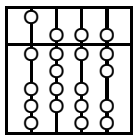
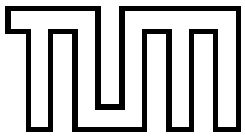


TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



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

SoS 2007
Übungsblatt 5
Ausgegeben am: 8. Juni 2007
Abgabe bis: **22. Juni 2007**

Übungen zu Einführung in die Informatik II

Aufgabe 1 (H) Terminkalender (5 Punkte)

a) Ein Termin besteht aus einem Wochentag, einem Ort und einer Beschreibung. Definieren Sie geeignete Datentypen für Wochentage und Termine.

b) Schreiben Sie eine Funktion terminkalender, die eine Liste von Terminen aufsteigend nach dem Wochentag sortiert zurückgibt. Die Liste darf bei der Berechnung nur einmal durchlaufen werden.

Aufgabe 2 (H) Java-Interpreter (15 Punkte)

Ziel dieser Aufgabe ist es, einen Interpreter für eine Java-ähnliche Sprache in OCaml zu implementieren. Gehen Sie dabei wie folgt vor:

a) Definieren Sie zunächst einen Typ `expr`, der *konstante* Ausdrücke beschreibt, d.h. diese Ausdrücke können die Grundrechenarten („+“, „−“, „∗“ und „/“) und `int` Zahlen enthalten. Außerdem benötigen Sie eine Funktion

```
val eval: expr -> int,
```

die Ihre Ausdrücke auswertet.

b) Definieren Sie nun einen Typ `expr`, der zusätzlich auch Variablen enthalten kann. Sie können allerdings für die Aufgabe davon ausgehen, dass nur die Variablen *x*, *y* und *z* vorkommen können. Ändern Sie die Funktion `eval` entsprechend.

Bedenken Sie, dass Sie nun zum Auswerten eine *Variablen-Umgebung* ρ benötigen, in der die aktuellen Werte der drei Variablen nachgeschlagen werden können. (Tipp: Sie können ρ als Funktion an `eval` übergeben!)

c) Definieren Sie einen Typ `bool_expr` für boolsche Ausdrücke und eine Funktion `eval_bool` zur Auswertung solcher Ausdrücke. Unterstützen Sie dabei die Operatoren $\wedge$ (`And`) und $\neq$ (`Not`) sowie die Relationen $=$ (`Eq`) und $<$ (`Lt`). Beachten Sie, dass boolsche Ausdrücke Variablen enthalten können. Die Funktion `eval_bool` erhält also als Parameter eine *Variablen-Umgebung* und einen boolschen Ausdruck.

d) Um den Status des Programs verändern zu können, müssen wir Variablen-Umbegungen „verändern“ können. Definieren Sie dazu eine Funktion `update`, die als Parameter eine *Variablen-Umgebung* ρ, eine Variable *v* und einen Integer-Wert *x* erhält. Zurückliefern soll diese Funktion eine *Variablen-Umgebung* ρ′, die wie folgt definiert ist:

$$\rho'(v') = \begin{cases} x & \text{falls } v' = v \\ \rho(v') & \text{sonst} \end{cases}$$

e) Definieren Sie einen Typ `stmt` für Statements. Beschränken Sie sich dabei auf Zuweisungen (`Assign`), bedingte Verzweigungen (`If`), `while`-Schleifen (`While`), Print-Statements (`Print`) und Read-Statements (`Read`).

f) Definieren Sie eine Funktion `run`, die eine Liste von Statements ausführt. Diese erhält als Parameter eine Liste von Statements sowie eine *Variablen-Umgebung* und liefert eine *Variablen-Umgebung* zurück. Der Rückgabewert entspricht der *Variablen-Umgebung* nach Ausführung des Statements.

g) Testen Sie Ihren Interpreter anhand eines Wertes der das folgende Programm repräsentiert:

```
x = read();
y = 1;
z = 0;
while (y < x) {
  y = y + 1;
  z = z + y;
}
write(z);
```

Hinweis: Verwenden Sie die OCaml-Funktionen `string_of_int`, `print_string` und `read_int`.