Technische Universität München
Fakultät für Informatik
Prof. Dr. H. Seidl

SoS 2006
Übungsblatt 6
07. Juni 2006

Übungen zu Einführung in die Informatik II

Aufgabe 1 Mini-Java-Interpreter

Erweitern Sie das Programm aus Aufgabe 12 um Statements. Gehen Sie dabei wie folgt vor:

a) Definieren Sie zunächst 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:

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

b) Definieren Sie ähnlich zur Aufgabe 12 einen Type `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.

c) Definieren Sie einen Typ `stmt` für Mini-Java-Statements. Beschränken Sie sich dabei auf Zuweisungen (`Assign`), bedingte Verzweigung (`If`), `while`-Schleifen (`While`) sowie auf Blöcke (`Block`). Ein Block ist dabei eine Folge von Statements, die in Mini-Java mit `{` eingeleitet und mit `}` abgeschlossen wird.

d) Definieren Sie schließlich eine Funktion `run`, die Mini-Java-Statements ausführt. Diese erhält als Parameter ein Statement sowie eine *Variablen-Umgebung* und liefert eine *Variablen-Umgebung* zurück. Der Rückgabewert entspricht der *Variablen-Umgebung* nach Ausführung des Mini-Java-Statements.

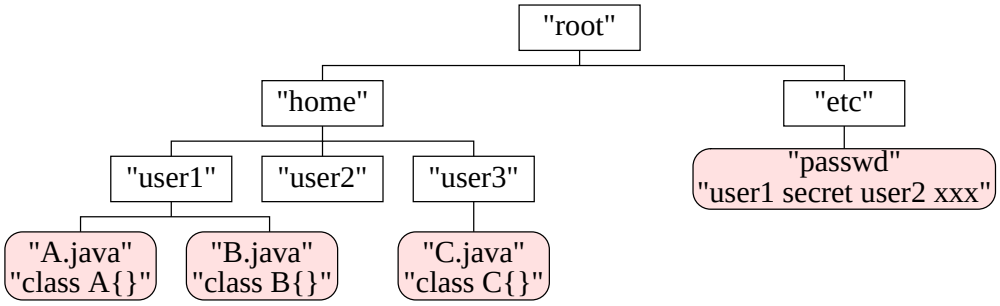
e) Testen Sie Ihren Mini-Java-Interpreter anhand des folgenden Statements:

```
while (x < 10000)
{
  x = x + 1;
  y = y + x;
}
```

Aufgabe 2 Filesystem

a) Definieren Sie einen OCaml-Typ `node`, mit dem sich Dateien und Verzeichnisse als Knoten eines Filesystem-Baums darstellen lassen. Eine Datei besteht aus einem Namen vom Typ `string` und einem Inhalt, der (der Einfachheit halber) auch vom Typ `string` ist. Ein Verzeichnis besteht aus einem Namen vom Typ `String` und aus einer (möglicherweise leeren) Liste von Unterverzeichnissen und Dateien.

Ein Beispiel-Filesystem ist unten abgebildet. Verzeichnisse sind durch Rechtecke und Dateien durch abgerundete graue Rechtecke dargestellt:



b) Instantiieren Sie einen Beispiel-Filesystem-Baum (z.B. den oben abgebildeten).

c) Definieren Sie eine Funktion `get_name : node -> string`, die den Namen eines Konten zurückliefert.

d) Definieren Sie eine Funktion `find`, die als Argumente eine Liste von Knoten `l`, einen Namen `name` bekommt und dann einen Knoten aus der Liste `l` mit Namen `name` zurückliefert.

e) Implementieren Sie eine Funktion `get_node : node -> string list -> node`, die als Argumente ein Verzeichnis d und einen Pfad *path* (als Liste von `strings`) erhält. Wenn `path` zu einem Knoten n aus dem Verzeichnis d führt, soll n zurückgeliefert werden.

f) * Schreiben Sie eine Funktion `add_node : node -> string list -> node -> node`, die ein Verzeichnis d, einen Pfad p und einen Knoten n als Argumente bekommt. Bezeichnet der Pfad p ein Unterverzeichnis d' von d, so soll der Knoten n im Unterverzeichnis d' zu dem Verzeichnis-Baum hinzugefügt werden und der veränderte Verzeichnis-Baum zurückgegeben werden.