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

SoS 2005
Übungsblatt 6
3. Juni 2005

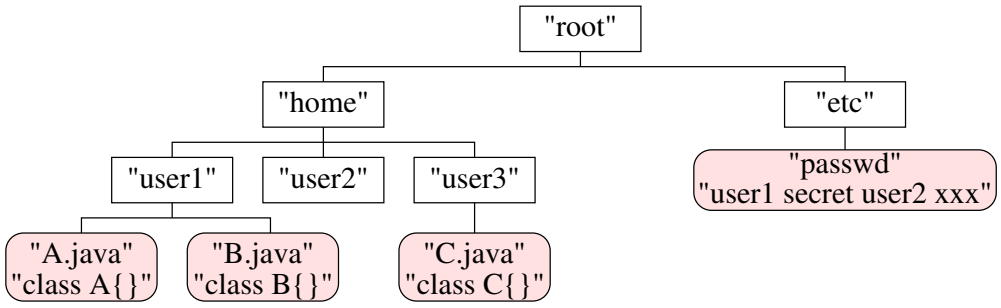
Übungen zu Einführung in die Informatik II

Hinweis: Die Anmeldung für die mündlichen Prüfungen Info2 für die Diplomstudenten findet am 6., 7. und 8. Juni 2005 (je nach Anfangsbuchstabe des Famliennamens) statt. Mehr Info unter
`http://www.in.tum.de/servlet/Mitteilung_Query_PA/index.html?cmd=listitems&mode=detail&itemid=2791`

Aufgabe 12 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) Schreiben Sie eine Funktion `addNode : node -> node -> node`, die ein Verzeichnis d_1 und einen Knoten n als Argumente bekommt und das Verzeichnis d_2 zurückliefert, in dem n zu dem Inhalt von d_1 hinzugefügt wird.

c) Implementieren Sie eine Funktion `changeDir : node -> string list -> node`, die als Argumente ein Verzeichnis d und einen Pfad *path* (als Liste von `strings`) erhält. Wenn `path` zu einem Unterverzeichnis d_1 aus dem Verzeichnis d führt, soll d_1 zurückgeliefert werden.

Aufgabe 13 Arithmetische Ausdrücke

Schreiben Sie ein OCaml Programm, mit dem man arithmetische Ausdrücke auswerten kann.

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

```
val simple_eval: simple_expr -> int,
```

die Ihre Ausdrücke auswertet.

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

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

Aufgabe 14 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;
}
```