/////////Assignment 7

/\*\*

\* this class Cons implements a Lisp-like Cons cell

\*

\* **@author** Gordon S. Novak Jr.

\* **@version** 29 Nov 01; 25 Aug 08; 05 Sep 08; 08 Sep 08; 12 Sep 08; 24 Sep 08

\* 06 Oct 08; 07 Oct 08; 09 Oct 08; 23 Oct 08; 27 Mar 09; 06 Aug 10

\* 30 Dec 13

\*

\* (defvar \*formulas\*

'((= s (\* 0.5 (\* a (expt t 2))))

(= s (+ s0 (\* v t)))

(= a (/ f m))

(= v (\* a t))

(= f (/ (\* m v) t))

(= f (/ (\* m (expt v 2)) r))

(= h (- h0 (\* 4.94 (expt t 2))))

(= c (sqrt (+ (expt a 2) (expt b 2))))

(= v (\* v0 (- 1.0 (exp (/ (- t) (\* r c))))))

))

(defvar \*opposites\*

'((+ -) (- +) (\* /) (/ \*) (sqrt expt) (expt sqrt) (log exp) (exp log) ) )

\*

\*

\*

\*/

**public** **class** Cons

{

// instance variables

**private** Object car;

**private** Cons cdr;

**private** Cons(Object first, Cons rest)

{ car = first;

cdr = rest; }

**public** **static** Cons cons(Object first, Cons rest)

{ **return** **new** Cons(first, rest); }

**public** **static** **boolean** consp (Object x)

{ **return** ( (x != **null**) && (x **instanceof** Cons) ); }

// safe car, returns null if lst is null

**public** **static** Object first(Cons lst) {

**return** ( (lst == **null**) ? **null** : lst.car ); }

// safe cdr, returns null if lst is null

**public** **static** Cons rest(Cons lst) {

**return** ( (lst == **null**) ? **null** : lst.cdr ); }

**public** **static** Object second (Cons x) { **return** *first*(*rest*(x)); }

**public** **static** Object third (Cons x) { **return** *first*(*rest*(*rest*(x))); }

**public** **static** **void** setfirst (Cons x, Object i) { x.car = i; }

**public** **static** **void** setrest (Cons x, Cons y) { x.cdr = y; }

**public** **static** Cons list(Object ... elements) {

Cons list = **null**;

**for** (**int** i = elements.length-1; i >= 0; i--) {

list = *cons*(elements[i], list);

}

**return** list;

}

// access functions for expression representation

**public** **static** Object op (Cons x) { **return** *first*(x); }

**public** **static** Object lhs (Cons x) { **return** *first*(*rest*(x)); }

**public** **static** Object rhs (Cons x) { **return** *first*(*rest*(*rest*(x))); }

**public** **static** **boolean** numberp (Object x)

{ **return** ( (x != **null**) &&

(x **instanceof** Integer || x **instanceof** Double) ); }

**public** **static** **boolean** integerp (Object x)

{ **return** ( (x != **null**) && (x **instanceof** Integer ) ); }

**public** **static** **boolean** floatp (Object x)

{ **return** ( (x != **null**) && (x **instanceof** Double ) ); }

**public** **static** **boolean** stringp (Object x)

{ **return** ( (x != **null**) && (x **instanceof** String ) ); }

// convert a list to a string for printing

**public** String toString() {

**return** ( "(" + *toStringb*(**this**) ); }

**public** **static** String toString(Cons lst) {

**return** ( "(" + *toStringb*(lst) ); }

**private** **static** String toStringb(Cons lst) {

**return** ( (lst == **null**) ? ")"

: ( *first*(lst) == **null** ? "()" : *first*(lst).toString() )

+ ((*rest*(lst) == **null**) ? ")"

: " " + *toStringb*(*rest*(lst)) ) ); }

**public** **boolean** equals(Object other) { **return** *equal*(**this**,other); }

// tree equality

**public** **static** **boolean** equal(Object tree, Object other) {

**if** ( tree == other ) **return** **true**;

**if** ( *consp*(tree) )

**return** ( *consp*(other) &&

*equal*(*first*((Cons) tree), *first*((Cons) other)) &&

*equal*(*rest*((Cons) tree), *rest*((Cons) other)) );

**return** *eql*(tree, other); }

// simple equality test

**public** **static** **boolean** eql(Object tree, Object other) {

**return** ( (tree == other) ||

( (tree != **null**) && (other != **null**) &&

tree.equals(other) ) ); }

// member returns null if requested item not found

**public** **static** Cons member (Object item, Cons lst) {

**if** ( lst == **null** )

**return** **null**;

**else** **if** ( item.equals(*first*(lst)) )

**return** lst;

**else** **return** *member*(item, *rest*(lst)); }

**public** **static** Cons union (Cons x, Cons y) {

**if** ( x == **null** ) **return** y;

**if** ( *member*(*first*(x), y) != **null** )

**return** *union*(*rest*(x), y);

**else** **return** *cons*(*first*(x), *union*(*rest*(x), y)); }

**public** **static** **boolean** subsetp (Cons x, Cons y) {

**return** ( (x == **null**) ? **true**

: ( ( *member*(*first*(x), y) != **null** ) &&

*subsetp*(*rest*(x), y) ) ); }

**public** **static** **boolean** setEqual (Cons x, Cons y) {

**return** ( *subsetp*(x, y) && *subsetp*(y, x) ); }

// combine two lists: (append '(a b) '(c d e)) = (a b c d e)

**public** **static** Cons append (Cons x, Cons y) {

**if** (x == **null**)

**return** y;

**else** **return** *cons*(*first*(x),

*append*(*rest*(x), y)); }

// look up key in an association list

// (assoc 'two '((one 1) (two 2) (three 3))) = (two 2)

**public** **static** Cons assoc(Object key, Cons lst) {

**if** ( lst == **null** )

**return** **null**;

**else** **if** ( key.equals(*first*((Cons) *first*(lst))) )

**return** ((Cons) *first*(lst));

**else** **return** *assoc*(key, *rest*(lst)); }

**public** **static** **int** square(**int** x) { **return** x\*x; }

**public** **static** **int** pow (**int** x, **int** n) {

**if** ( n <= 0 ) **return** 1;

**if** ( (n & 1) == 0 )

**return** *square*( *pow*(x, n / 2) );

**else** **return** x \* *pow*(x, n - 1); }

**public** **static** Cons *formulas* =

*list*( *list*( "=", "s", *list*("\*", **new** Double(0.5),

*list*("\*", "a",

*list*("expt", "t", **new** Integer(2))))),

*list*( "=", "s", *list*("+", "s0", *list*( "\*", "v", "t"))),

*list*( "=", "a", *list*("/", "f", "m")),

*list*( "=", "v", *list*("\*", "a", "t")),

*list*( "=", "f", *list*("/", *list*("\*", "m", "v"), "t")),

*list*( "=", "f", *list*("/", *list*("\*", "m",

*list*("expt", "v", **new** Integer(2))),

"r")),

*list*( "=", "h", *list*("-", "h0", *list*("\*", **new** Double(4.94),

*list*("expt", "t",

**new** Integer(2))))),

*list*( "=", "c", *list*("sqrt", *list*("+",

*list*("expt", "a",

**new** Integer(2)),

*list*("expt", "b",

**new** Integer(2))))),

*list*( "=", "v", *list*("\*", "v0",

*list*("-", **new** Double(1.0),

*list*("exp", *list*("/", *list*("-", "t"),

*list*("\*", "r", "c"))))))

);

// Note: this list will handle most, but not all, cases.

// The binary operators - and / have special cases.

**public** **static** Cons *opposites* =

*list*( *list*( "+", "-"), *list*( "-", "+"), *list*( "\*", "/"),

*list*( "/", "\*"), *list*( "sqrt", "expt"), *list*( "expt", "sqrt"),

*list*( "log", "exp"), *list*( "exp", "log") );

**public** **static** **void** printanswer(String str, Object answer) {

System.*out*.println(str +

((answer == **null**) ? "null" : answer.toString())); }

// \*\*\*\*\*\* your code starts here \*\*\*\*\*\*

**public** **static** Cons findpath(Object item, Object cave)

{

Cons pathf, pathr;

**if** (cave == **null**)

**return** **null**;

**else** **if** (*consp*(cave))

{

Object fCave = *first*((Cons)cave);

Object rCave = (Object)*rest*((Cons)cave);

pathf = *findpath*(item,fCave);

pathr = *findpath*(item,rCave);

**if** (pathf != **null**)

**return** *cons* ("first", pathf);

**else** **if** (pathr != **null**)

**return** *cons*("rest",pathr);

**else** **return** **null**;

}

**else** **if** (item.equals(cave))

**return** *cons*("done",**null**);

**else** **if** (!item.equals(cave))

**return** **null**;

**else**

**return** **null**;

}

**public** **static** Object follow(Cons path, Object cave) {

**if** (*consp*(cave))

{

**if** ("rest".equals(*first*((Cons)path)))

**return** *follow*(*rest*((Cons)path),*rest*((Cons)cave));

**else** **if** ("first".equals(*first*((Cons)path)))

**return** *follow*(*rest*((Cons)path),*first*((Cons)cave));

**else** **if** ("done".equals(*first*((Cons)path)))

**return** *first*((Cons)cave);

**else**

**return** **null**;

}

**else** **if** ("done".equals(*first*(path)))

**return** cave;

**else**

**return** **null**;

}

**public** **static** Object corresp(Object item, Object tree1, Object tree2) {

Object of, or;

**if** (*consp*(*first*((Cons)tree1)))

{

of = *corresp*(item,*first*((Cons)tree1),*first*((Cons)tree2));

}

**else** **if** ( *lhs*((Cons)tree1) != **null** && *lhs*((Cons)tree1).equals(item))

of = *lhs*((Cons)tree2);

**else**

of = **null**;

**if** (*consp*(*rest*((Cons)tree1)))

{

or = *corresp*(item,*rest*((Cons)tree1),*rest*((Cons)tree2));

}

**else** **if** ( *lhs*((Cons)tree1) != **null** && *lhs*((Cons)tree1).equals(item))

{

or = *lhs*((Cons)tree2);

}

**else**

or = **null**;

**if** ( of != **null** && or == **null**)

**return** of;

**else** **if** (of == **null** && or != **null**)

**return** or;

**else** **return** **null**;

}

**public** **static** Cons solve(Cons e, String v) {

Cons inverse = *list* (*list*("+", "-"), *list*("-" , "+"), *list*("\*","/"), *list*("/", "\*"), *list*("sqrt", "expt"),

*list*("expt","sqrt"), *list*("exp","log"), *list*("log","exp"), *list* ("=", "="));

System.*out*.println("e is " + e.toString() + " and v is " + v);

**if** (*lhs*((Cons)e).equals(v))

**return** e;

**else** **if** ( *rhs*((Cons)e) != **null** && *rhs*((Cons)e).equals(v))

**return** **new** Cons(*op*((Cons)e),**new** Cons(*rhs*((Cons)e), **new** Cons(*lhs*((Cons)e),**null**)));

**else** **if** (*consp*(*rhs*((Cons)e)))

{

**if** (*lhs*((Cons)*rhs*(e)).equals(v))

{

String inv = (String)*lhs*(*assoc*(*op*((Cons)*rhs*((Cons)e)),inverse));

e = *list*("=", v , *list*(inv , *lhs*((Cons)e) , *rhs*((Cons)*rhs*(e))));

**return** *solve*(e,v);

}

**else** **if** (*rhs*((Cons)*rhs*(e)) != **null** && *rhs*((Cons)*rhs*(e)).equals(v))

{

String inv = (String)*lhs*(*assoc*(*op*((Cons)*rhs*((Cons)e)),inverse));

e = *list* ("=", v , *list*(inv, *lhs*((Cons)e), *lhs*((Cons)*rhs*(e))));

**return** *solve*(e,v);

}

**else**

{

String inv = (String)*lhs*(*assoc*(*op*((Cons)*rhs*((Cons)e)),inverse));

e = *list*("=", *list* (inv, *lhs*((Cons)e), *lhs*((Cons)*rhs*(e))), *rhs*((Cons)*rhs*(e)));

**return** *solve*(e,v);

}

}

**else** **if** (*rhs*((Cons)e) != **null** && !*consp*(*rhs*((Cons)e)) && !*rhs*((Cons)e).equals(v))

{

e = *list*("=", *rhs*((Cons)e), *lhs*((Cons)e));

**return** *solve*(e,v);

}

**else**

**return** **null**;

}

/\*

public static Double solveit (Cons equations, String var, Cons values) {

}

\*/

// Include your functions vars and eval from the previous assignment.

// Modify eval as described in the assignment.

**public** **static** Cons vars (Object expr) {

Cons leaf = **null**;

**if** (*consp*(expr) && *rest*((Cons)expr) != **null**)

{

**return** *union* (*vars*(*first*((Cons)expr)), *vars*(*rest*((Cons)expr)));

}

**else** **if** (*consp*(expr) && *rest*((Cons)expr) == **null**)

{

Object last = *first*((Cons)expr);

**return** *vars*(last);

}

**else** **if** (!*numberp*(expr) && expr != "expt"

&& expr != "exp"

&& expr != "+"

&& expr != "-"

&& expr != "\*"

&& expr != "/"

&& expr != "=")

{

leaf = *cons*(expr,**null**);

}

**return** leaf;

}

**public** **static** Double eval (Object tree, Cons bindings) {

**if** (*consp*(tree))

{

**if** (*op*((Cons)tree) == "+")

{

**return** *eval* (*lhs*((Cons)tree),bindings) + *eval*(*rhs*((Cons)tree), bindings);

}

**else** **if** (*op*((Cons)tree) == "-")

{

**if** (*rhs*((Cons)tree) == **null**)

{

**if** (*numberp*(*lhs*((Cons)tree)))

{

Double negNum = -1 \* (Double)*lhs*((Cons)tree);

**return** negNum;

}

**else**

{

Cons tupple = *assoc*(*lhs*((Cons)tree),bindings);

Double associatedNum = -1 \* (Double)*lhs*((Cons)tupple);

**return** associatedNum;

}

}

**else**

**return** *eval* (*lhs*((Cons)tree), bindings) - *eval*(*rhs*((Cons)tree), bindings);

}

**else** **if** (*op*((Cons)tree) == "\*")

{

**return** *eval* (*lhs*((Cons)tree), bindings) \* *eval*(*rhs*((Cons)tree), bindings);

}

**else** **if** (*op*((Cons)tree) == "/")

{

**return** *eval* (*lhs*((Cons)tree), bindings) / *eval*(*rhs*((Cons)tree), bindings);

}

**else**

{

**return** Math.*pow*(*eval* (*lhs*((Cons)tree), bindings) , *eval*(*rhs*((Cons)tree), bindings));

}

}

**else**

{

**if** (*integerp*(tree))

{

**return** (Double)tree;

}

**else**

{

Cons tupple = *assoc*(tree,bindings);

Double associatedNum = (Double)*lhs*((Cons)tupple);

**return** associatedNum;

}

}

}

// \*\*\*\*\*\* your code ends here \*\*\*\*\*\*

**public** **static** **void** main( String[] args ) {

Cons cave = *list*("rocks", "gold", *list*("monster"));

Cons path = *findpath*("gold", cave);

// printanswer("cave = " , cave);

// printanswer("path = " , path);

// printanswer("follow = " , follow(path, cave));

Cons caveb = *list*(*list*(*list*("green", "eggs", "and"),

*list*(*list*("ham"))),

"rocks",

*list*("monster",

*list*(*list*(*list*("gold", *list*("monster"))))));

Cons pathb = *findpath*("gold", caveb);

//printanswer("caveb = " , caveb);

// printanswer("pathb = " , pathb);

//printanswer("follow = " , follow(pathb, caveb));

Cons treea = *list*(*list*("my", "eyes"),

*list*("have", "seen", *list*("the", "light")));

Cons treeb = *list*(*list*("my", "ears"),

*list*("have", "heard", *list*("the", "music")));

//printanswer("treea = " , treea);

// printanswer("treeb = " , treeb);

// printanswer("corresp = " , corresp("light", treea, treeb));

System.*out*.println("formulas = ");

Cons frm = *formulas*;

Cons vset = **null**;

**while** ( frm != **null** ) {

*printanswer*(" " , ((Cons)*first*(frm)));

vset = *vars*((Cons)*first*(frm));

**while** ( vset != **null** ) {

*printanswer*(" " ,

*solve*((Cons)*first*(frm), (String)*first*(vset)) );

vset = *rest*(vset); }

frm = *rest*(frm); }

Cons bindings = *list*( *list*("a", (Double) 32.0),

*list*("t", (Double) 4.0));

//printanswer("Eval: " , rhs((Cons)first(formulas)));

//printanswer(" bindings " , bindings);

// printanswer(" result = " , eval(rhs((Cons)first(formulas)), bindings));

/\*

\* printanswer("Tower: " , solveit(formulas, "h0",

list(list("h", new Double(0.0)),

list("t", new Double(4.0)))));

printanswer("Car: " , solveit(formulas, "a",

list(list("v", new Double(88.0)),

list("t", new Double(8.0)))));

printanswer("Capacitor: " , solveit(formulas, "c",

list(list("v", new Double(3.0)),

list("v0", new Double(6.0)),

list("r", new Double(10000.0)),

list("t", new Double(5.0)))));

printanswer("Ladder: " , solveit(formulas, "b",

list(list("a", new Double(6.0)),

list("c", new Double(10.0)))));

\*/

}

}