zakk0610 / hw1.sml

Created 5 years ago • Report gist

Programming Languages Coursera Course HW

hw1.sml

```
fun is_older (day1 : int*int*int, day2 : int*int*int) =
    if (#1 day1) < (#1 day2)
    then true
    else if (((#2 day1) < (#2 day2)) andalso ((#1 day1) = (#1 day2)))
    then true
    else if (((#3 day1) < (#3 day2)) andalso ((#2 day1) = (#2 day2)) andalso ((#1 day1) = (#1 day2)))
    then true
    else false



fun number_in_month  (day_list: (int*int*int) list, month : int) =
    if null day_list
    then 0
    else if #2(hd day_list) = month then 1 + number_in_month(tl day_list, month)
    else number_in_month(tl day_list, month)


fun number_in_months (day_list: (int*int*int) list, month_list: int list) =
    if null month_list
    then 0
    else number_in_month(day_list, hd month_list) + number_in_months(day_list, tl month_list)



fun dates_in_month  (day_list: (int*int*int) list, month : int) =
    if null day_list
    then []
    else if #2(hd day_list) = month then (hd day_list)::dates_in_month(tl day_list, month)
    else dates_in_month(tl day_list, month)


fun dates_in_months  (day_list: (int*int*int) list, month_list : int list) =
    if null month_list
    then []
    else dates_in_month(day_list, (hd month_list))@dates_in_months(day_list, (tl month_list))


fun get_nth (string_list: string list, n: int) =
    if n = 1 then (hd string_list)
    else if (tl string_list) = [] then ""
    else get_nth(tl string_list, n-1)


fun date_to_string(day: int*int*int) =
    let val month = ["January", "February", "March", "April",
"May", "June", "July", "August", "September", "October", "November", "December"]
    in
     get_nth(month, #3 day) ^ " " ^ Int.toString(#2 day)^ ", " ^ Int.toString(#1 day)
    end


fun number_before_reaching_sum (sum: int, num: int list) =
    let
     fun nth_sum(index: int, sum: int, num: int list) =
         if null num then index
         else if (sum - (hd num)) < 0 then index + 1
            else nth_sum(index +1, sum - (hd num), tl num)
    in
     nth_sum(0, sum, num)
```

```
    end


fun what_month(day: int) =
    let val day_of_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    in
     number_before_reaching_sum(day - 1, day_of_month)
    end


fun month_range(day: (int*int)) =
    if (#1 day) > (#2 day) then []
    else if (#1 day) = (#2 day) then [what_month(#1 day)]
    else what_month(#1 day):: month_range((#1 day) + 1, (#2 day))


fun oldest(day_list: (int*int*int) list)=
    if null day_list then NONE
    else let
     fun oldest_nonempty(day_list: (int*int*int) list) =
         if null (tl day_list) then hd day_list
         else let val older = oldest_nonempty(tl day_list)
          in
               if is_older(hd day_list, older) then hd day_list
                     else older
               end
        in
         SOME (oldest_nonempty day_list)
    end




fun is_older (day1 : int*int*int, day2 : int*int*int) =
    if (#1 day1) < (#1 day2)
    then true
    else if (#2 day1) < (#2 day2) andalso  (#1 day1) = (#1 day2)
    then true
    else if (#3 day1) < (#3 day2) andalso (#2 day1) = (#2 day2) andalso (#3 day1) = (#3 day2)
    then true
    else false



fun number_in_month  (day_list: (int*int*int) list, month : int) =
    if null day_list
    then 0
    else if #2(hd day_list) = month then 1 + number_in_month(tl day_list, month)
    else number_in_month(tl day_list, month)


fun number_in_months (day_list: (int*int*int) list, month_list: int list) =
    if null month_list
    then 0
    else number_in_month(day_list, hd month_list) + number_in_months(day_list, tl month_list)
fun dates_in_month  (day_list: (int*int*int) list, month : int) =
    if null day_list
    then []
    else if #2(hd day_list) = month then (hd day_list)::dates_in_month(tl day_list, month)
    else dates_in_month(tl day_list, month)


fun dates_in_months  (day_list: (int*int*int) list, month_list : int list) =
    if null month_list
    then []
    else dates_in_month(day_list, (hd month_list))@dates_in_months(day_list, (tl month_list))


fun get_nth (string_list: string list, n: int) =
    if n = 1 then (hd string_list)
    else if (tl string_list) = [] then ""
    else get_nth(tl string_list, n-1)


```

```
fun date_to_string(day: int*int*int) =
    let val month = ["January", "February", "March", "April",
"May", "June", "July", "August", "September", "October", "November", "December"]
    in
     get_nth(month, #2 day) ^ " " ^ Int.toString(#3 day)^ ", " ^ Int.toString(#1 day)
    end


fun number_before_reaching_sum (sum: int, num: int list) =
    let
     fun nth_sum(index: int, sum: int, num: int list) =
         if null num then index
         else if (sum - (hd num)) <= 0 then index
            else nth_sum(index +1, sum - (hd num), tl num)
    in
     nth_sum(0, sum, num)
    end


fun what_month(day: int) =
    let val day_of_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    in
     number_before_reaching_sum(day, day_of_month) + 1
    end


fun month_range(day: (int*int)) =
    if (#1 day) > (#2 day) then []
    else if (#1 day) = (#2 day) then [what_month(#1 day)]
    else what_month(#1 day):: month_range((#1 day) + 1, (#2 day))


fun oldest(day_list: (int*int*int) list)=
    if null day_list then NONE
    else let
     fun oldest_nonempty(day_list: (int*int*int) list) =
         if null (tl day_list) then hd day_list
         else let val older = oldest_nonempty(tl day_list)
          in
               if is_older(hd day_list, older) then hd day_list
                     else older
              end
        in
         SOME (oldest_nonempty day_list)
    end
```

hw2.sml

```
(* Dan Grossman, Coursera PL, HW2 Provided Code *)

(* if you use this function to compare two strings (returns true if the same
   string), then you avoid several of the functions in problem 1 having
   polymorphic types that may be confusing *)
fun same_string(s1 : string, s2 : string) =
    s1 = s2

(* put your solutions for problem 1 here *)

fun all_except_option (str, []) = NONE
  | all_except_option (str, x::xs) =
    if same_string(x, str)
      then SOME xs
    else case all_except_option(str, xs) of
                NONE   => NONE
              | SOME y => SOME (x::y)

fun get_substitutions1 ([], s) = []
  | get_substitutions1 (x::xs, s) =
    case all_except_option(s, x) of
          NONE => get_substitutions1(xs, s)
         | SOME y => y@get_substitutions1(xs, s)

fun get_substitutions2 (str_list, s) =
  let fun aux(str_list, acc) =
```

```
      case str_list of
         [] => acc
          | x::xs => case all_except_option(s, x) of
                   NONE => aux(xs, acc)
                 | SOME y => aux(xs, acc@ y)
      in
  aux(str_list, [])
end

fun similar_names (str_list, name) =
    let
        val {first=x, middle=y, last=z} = name
        fun help_similar(result_list: string list, acc) =
            case result_list of
                [] => name::acc
                 | a::ac =>
            help_similar(ac, {first=a,middle=y,last=z}::acc)
in
   help_similar(get_substitutions2(str_list, x),[])
end


(* you may assume that Num is always used with values 2, 3, ..., 10
   though it will not really come up *)
datatype suit = Clubs | Diamonds | Hearts | Spades
datatype rank = Jack | Queen | King | Ace | Num of int
type card = suit * rank

datatype color = Red | Black
datatype move = Discard of card | Draw

exception IllegalMove


(* put your solutions for problem 2 here *)

fun card_color card =
  case card of (Hearts, _)=> Red
             | (Diamonds, _) => Red
             | (_, _) => Black

fun card_value card =
    case card of(_,Ace) => 11
               | (_,Num(i)) => i
               | (_,_)=> 10

fun remove_card (cs :card list, c : card, e :exn)  =
    case cs of
        [] => raise e
      | cs::cs' => if cs = c then cs'
              else cs ::remove_card (cs', c, e)

fun all_same_color cs =
    case cs of
        [] =>false
        | c1::[] => true
        | c1::c2::cs' => if card_color(c1) = card_color(c2) then all_same_color(c2::cs')
                       else false

fun sum_cards(cs: card list) =
   let fun aux (cs: card list, acc: int) =
          case cs of
             [] => acc
             |cs::cs' => aux(cs', card_value(cs)+ acc)
in
   aux (cs, 0)
end

fun score(cs: card list, goal: int) =
   let val sum = sum_cards (cs)
   val init = if sum > goal then 3 * (sum - goal) else goal - sum
in
   if all_same_color(cs) then init div 2 else init
end
```

```
101
102
103 fun officiate(cs: card list, mv: move list, goal: int) =
104
105 let
106     fun run_turns(cs: card list, mv: move list, hel_cards: card list) =
107     if sum_cards(hel_cards) > goal then score(hel_cards , goal) else
108     case mv of
109     [] => score (hel_cards, goal)
110     | mv::mv' =>  case mv of
111                     Discard(card) => run_turns(cs, mv', remove_card(hel_cards, card, IllegalMove))
112                    | Draw => case cs of
113                       [] => score (hel_cards, goal)
114                       | cs::cs' => run_turns(cs', mv', cs::hel_cards)
115 in
116    run_turns(cs, mv, [])
117 end
118
119 fun check_tests ts =
120   List.map (fn t => if t then print "OK\n" else print "FAIL") ts
121
122 val all_except_option_t =
123   [ all_except_option ("a",[]) = NONE
124   , all_except_option ("a",["b"]) = NONE
125   , all_except_option ("a",["b","c"]) = NONE
126   , all_except_option ("a",["a"]) = SOME []
127   , all_except_option ("a",["a","b"]) = SOME ["b"]
128   , all_except_option ("a",["b","a","c"]) = SOME ["b","c"]
129   ]
130
131 val get_substitutions1_t =
132   [ get_substitutions1 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Fred") = ["Fredrick","Freddie","F"]
133   , get_substitutions1 ([[]],"Fred") = []
134   , get_substitutions1 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Stranger") = []
135   , get_substitutions1 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Fredrick") = ["Fred"]
136   , get_substitutions1 ([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]],"Jeff") = ["Jeffrey","Geoff","Jeffr
137   ]
138
139 val get_substitutions2_t =
140   [ get_substitutions2 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Fred") = ["Fredrick","Freddie","F"]
141   , get_substitutions2 ([[]],"Fred") = []
142   , get_substitutions2 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Stranger") = []
143   , get_substitutions2 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Fredrick") = ["Fred"]
144   , get_substitutions2 ([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]],"Jeff") = ["Jeffrey","Geoff","Jeffr
145   ]
146
147
148 val similar_names_t = let
149     val names1 = [["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]]
150     val names2 = [["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]]
151   in
152     [ similar_names (names1, {first="Fred", middle="W", last="Smith"} ) = [{first="Fred", last="Smith", middle="W"}, {first="Fr
153     , similar_names (names2, {first="Jeff", middle="W", last="Smith"} ) = [{first="Jeff", last="Smith", middle="W"}, {first="Je
154     , similar_names(names1, {first="Jeff", middle="W", last="Smith"} ) = [{first="Jeff", middle="W", last="Smith"}]
155     ]
156   end
157
158 val cards1 = [(Clubs,Jack),(Spades,Num(8))]
159 val cards2 = [(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)]
160 val cards3 = [(Clubs,Ace),(Diamonds,King)]
161
162 val card_color_t =
163   [ card_color ((Clubs,Jack)) = Black
164   , card_color ((Spades,Jack)) = Black
165   , card_color ((Diamonds,Ace)) = Red
166   , card_color ((Hearts,Ace)) = Red
167   ]
168
169 val card_value_t =
170   [ card_value((Clubs,Jack))=10
171   , card_value((Clubs,Queen))=10
172   , card_value((Clubs,King))=10
173   , card_value((Clubs,Ace))=11
174   , card_value((Clubs,Num(2)))=2
```

```
  , card_value((Clubs,Num(3)))=3
  , card_value((Clubs,Num(10)))=10
  ]

val remove_card_t =
  [ remove_card(cards1,(Clubs,Jack),IllegalMove)=[(Spades,Num(8))]
  , remove_card(cards2,(Spades,Ace),IllegalMove)=[(Clubs,Ace),(Clubs,Ace),(Spades,Ace)]
  , remove_card(cards2,(Clubs,Ace),IllegalMove)=[(Spades,Ace),(Clubs,Ace),(Spades,Ace)]
  , remove_card(cards1,(Spades,Num(8)),IllegalMove)=[(Clubs,Jack)]
  , (remove_card(cards2,(Spades,Num(8)),IllegalMove) handle IllegalMove => []) = []
  ]

val all_same_color_t =
[ all_same_color(cards1)=true
, all_same_color(cards2)=true
, all_same_color([(Clubs,Jack),(Spades,Num(8)),(Hearts,King)])=false
, all_same_color([(Clubs,Jack),(Hearts,King),(Spades,Num(8))])=false
, all_same_color([(Hearts,King),(Clubs,Jack),(Spades,Num(8))])=false
, all_same_color(cards3)=false
]

val sum_cards_t =
  [ sum_cards(cards1)=18
  , sum_cards(cards2)=44
  , sum_cards(cards3)=21
  ]

val score_t =
  [ score(cards3,21)=0
  , score(cards3,25)=4
  , score(cards3,17)=12
  , score(cards2,44)=0
  , score(cards2,48)=2
  , score(cards2,40)=6
  , score([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)],42)=3
  ]

val officiate_t =
  [ ( officiate([(Clubs,Jack),(Spades,Num(8))], [Draw,Discard(Hearts,Jack)] ,42) handle IllegalMove => 9999 ) = 9999
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],42)=3
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],30)=4
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],22)=16
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],100)=28
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],44)=0
  , officiate([(Diamonds,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],30)=9
  , officiate([(Clubs,Ace),(Hearts,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],22)=33
  , officiate([(Clubs,Ace),(Spades,Ace),(Diamonds,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],100)=56
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Draw,Draw,Draw,Draw],44)=0
  ,  officiate([(Clubs,Ace),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Draw],30)=8
  ,  officiate([(Clubs,Ace),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Draw],22)=0
  ,  officiate([(Clubs,Ace),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Draw],11)=33
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],11)=33
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],22)=0
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],30)=8
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],11)=16
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],22)=0
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],30)=4
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Draw,Discard(Clubs,Queen),Draw],11)=30
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Draw,Discard(Clubs,Queen),Draw],22)=0
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Draw,Discard(Clubs,Queen),Draw],30)=4
  ]


val all_tests = List.concat
  [ all_except_option_t
  , get_substitutions1_t
  , get_substitutions2_t
  , similar_names_t
  , card_color_t
  , card_value_t
  , remove_card_t
  , all_same_color_t
  , sum_cards_t
  , officiate_t
```

```
  ]

val tests = List.all (fn x => x = true) all_tests
```

hw2test.sml

```
fun check_tests ts =
  List.map (fn t => if t then print "OK\n" else print "FAIL") ts

val all_except_option_t =
  [ all_except_option ("a",[]) = NONE
  , all_except_option ("a",["b"]) = NONE
  , all_except_option ("a",["b","c"]) = NONE
  , all_except_option ("a",["a"]) = SOME []
  , all_except_option ("a",["a","b"]) = SOME ["b"]
  , all_except_option ("a",["b","a","c"]) = SOME ["b","c"]
  ]

val get_substitutions1_t =
  [ get_substitutions1 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Fred") = ["Fredrick","Freddie","F"]
  , get_substitutions1 ([[]],"Fred") = []
  , get_substitutions1 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Stranger") = []
  , get_substitutions1 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Fredrick") = ["Fred"]
  , get_substitutions1 ([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]],"Jeff") = ["Jeffrey","Geoff","Jeffr
  ]

val get_substitutions2_t =
  [ get_substitutions2 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Fred") = ["Fredrick","Freddie","F"]
  , get_substitutions2 ([[]],"Fred") = []
  , get_substitutions2 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Stranger") = []
  , get_substitutions2 ([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],"Fredrick") = ["Fred"]
  , get_substitutions2 ([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]],"Jeff") = ["Jeffrey","Geoff","Jeffr
  ]


val similar_names_t = let
    val names1 = [["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]]
    val names2 = [["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]]
  in
    , similar_names (names2, {first="Jeff", middle="W", last="Smith"} ) = [{first="Jeff", last="Smith", middle="W"}, {first="Je
    , similar_names(names1, {first="Jeff", middle="W", last="Smith"} ) = [{first="Jeff", middle="W", last="Smith"}]
    ]
  end

val cards1 = [(Clubs,Jack),(Spades,Num(8))]
val cards2 = [(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)]
val cards3 = [(Clubs,Ace),(Diamonds,King)]

val card_color_t =
  [ card_color ((Clubs,Jack)) = Black
  , card_color ((Spades,Jack)) = Black
  , card_color ((Diamonds,Ace)) = Red
  , card_color ((Hearts,Ace)) = Red
  ]

val card_value_t =
  [ card_value((Clubs,Jack))=10
  , card_value((Clubs,Queen))=10
  , card_value((Clubs,King))=10
  , card_value((Clubs,Ace))=11
  , card_value((Clubs,Num(2)))=2
  , card_value((Clubs,Num(3)))=3
  , card_value((Clubs,Num(10)))=10
  ]

val remove_card_t =
  [ remove_card(cards1,(Clubs,Jack),IllegalMove)=[(Spades,Num(8))]
  , remove_card(cards2,(Spades,Ace),IllegalMove)=[(Clubs,Ace),(Clubs,Ace),(Spades,Ace)]
  , remove_card(cards2,(Clubs,Ace),IllegalMove)=[(Spades,Ace),(Clubs,Ace),(Spades,Ace)]
  , remove_card(cards1,(Spades,Num(8)),IllegalMove)=[(Clubs,Jack)]
  , (remove_card(cards2,(Spades,Num(8)),IllegalMove) handle IllegalMove => []) = []
  ]

val all_same_color_t =
```

```
[ all_same_color(cards1)=true
, all_same_color(cards2)=true
, all_same_color([(Clubs,Jack),(Spades,Num(8)),(Hearts,King)])=false
, all_same_color([(Clubs,Jack),(Hearts,King),(Spades,Num(8))])=false
, all_same_color([(Hearts,King),(Clubs,Jack),(Spades,Num(8))])=false
, all_same_color(cards3)=false
]

val sum_cards_t =
  [ sum_cards(cards1)=18
  , sum_cards(cards2)=44
  , sum_cards(cards3)=21
  ]

val score_t =
  [ score(cards3,21)=0
  , score(cards3,25)=4
  , score(cards3,17)=12
  , score(cards2,44)=0
  , score(cards2,48)=2
  , score(cards2,40)=6
  , score([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)],42)=3
  ]

val officiate_t =
  [ ( officiate([(Clubs,Jack),(Spades,Num(8))], [Draw,Discard(Hearts,Jack)] ,42) handle IllegalMove => 9999 ) = 9999
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],42)=3
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],30)=4
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],22)=16
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],100)=28
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],44)=0
  , officiate([(Diamonds,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],30)=9
  , officiate([(Clubs,Ace),(Hearts,Ace),(Clubs,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],22)=33
  , officiate([(Clubs,Ace),(Spades,Ace),(Diamonds,Ace),(Spades,Ace)], [Draw,Draw,Draw,Draw,Draw],100)=56
  , officiate([(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Draw,Draw,Draw,Draw],44)=0
  ,  officiate([(Clubs,Ace),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Draw],30)=8
  ,  officiate([(Clubs,Ace),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Draw],22)=0
  ,  officiate([(Clubs,Ace),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Draw],11)=33
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],11)=33
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],22)=0
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Clubs,Ace),(Hearts,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],30)=8
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],11)=16
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],22)=0
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Discard(Clubs,Queen),Draw,Draw],30)=4
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Draw,Discard(Clubs,Queen),Draw],11)=30
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Draw,Discard(Clubs,Queen),Draw],22)=0
  ,  officiate([(Clubs,Queen),(Diamonds,Ace),(Hearts,Ace),(Diamonds,Ace)], [Draw,Draw,Discard(Clubs,Queen),Draw],30)=4
  ]


val all_tests = List.concat
  [ all_except_option_t
  , get_substitutions1_t
  , get_substitutions2_t
  , similar_names_t
  , card_color_t
  , card_value_t
  , remove_card_t
  , all_same_color_t
  , sum_cards_t
  , score_t
  , officiate_t
  ]

val tests = List.all (fn x => x = true) all_tests
```

hw3.sml

```
(* Coursera Programming Languages, Homework 3, Provided Code *)

exception NoAnswer

datatype pattern = Wildcard
         | Variable of string
```

```
                | UnitP
                | ConstP of int
                | TupleP of pattern list
                | ConstructorP of string * pattern

datatype valu = Const of int
              | Unit
              | Tuple of valu list
              | Constructor of string * valu

fun g f1 f2 p =
    let
        val r = g f1 f2
    in
        case p of
            Wildcard          => f1 ()
          | Variable x        => f2 x
          | TupleP ps         => List.foldl (fn (p,i) => (r p) + i) 0 ps
          | ConstructorP(_,p) => r p
          | _                 => 0
    end

(**** for the challenge problem only ****)

datatype typ = Anything
             | UnitT
             | IntT
             | TupleT of typ list
             | Datatype of string

(**** you can put all your code here ****)
fun only_capitals L =
    List.filter (fn (s) => Char.isUpper(String.sub(s, 0))) L

fun longest_string1 L =
    foldl (fn (x, y) => if String.size(x) > String.size(y) then x else y) "" L

fun longest_string2 L =
    foldl (fn (x, y) => if String.size(x) >= String.size(y) then x else y) "" L

fun longest_string_helper f L =
    foldl (fn (x, y)=> if f(String.size(x), String.size(y)) then x else y) "" L

val longest_string3 = longest_string_helper (fn (x, y) => x > y)

val longest_string4 = longest_string_helper (fn (x, y) => x >= y)

val longest_capitalized = longest_string1 o only_capitals

val rev_string = String.implode o List.rev o String.explode

fun first_answer f [] = raise NoAnswer
  | first_answer f(x::xs') = case f x of
                                NONE => first_answer f xs'
                              | SOME v => v;
fun all_answers f xs =
let fun help_answer [] acc = SOME acc
      | help_answer (x::xs') acc = case f x of
                                     NONE => NONE
                                   | SOME v => help_answer xs' (acc@v)
in
    help_answer xs []
end

val count_wildcards = g (fn() => 1) (fn _ => 0)

val count_wild_and_variable_lengths = g (fn() => 1) String.size

fun count_some_var (s, p) =
    g (fn() => 0) (fn(s') => if s = s' then 1 else 0) p

fun check_pat pat =
  let
    fun get_vars pat =
```

```
        case pat of
            Variable x => [x]
          | ConstructorP(_, p) => get_vars p
          | TupleP ps => List.concat (map get_vars ps)
          | _  => []
    fun check_repeat [] = true
      | check_repeat (x::xs') =
        if List.exists (fn(x') => x = x') xs'
        then false
        else check_repeat xs'
in
 check_repeat(get_vars (pat))
end

fun match vp =
    case vp of
        (_,Wildcard) => SOME []
      | (v, Variable s) => SOME [(s,v)]
      | (Unit, UnitP) => SOME []
      | (Const a, ConstP a') => if a = a'
                                then SOME []
                                else NONE
      | (Tuple ps, TupleP vs) =>
        if (length ps) = (length vs)
        then all_answers match (ListPair.zip(ps, vs))
        else NONE
      | (Constructor(s1, p), ConstructorP(s2, v)) => if s1 = s2
                                                     then match (p, v)
                                                     else NONE
      | _ => NONE;

fun first_match v pat =
    SOME (first_answer (fn(pat) => match (v, pat)) pat)
    handle NoAnswer => NONE;


```

hw3test.sml

```
(* HW3 Tests *)
fun t_responder s = if String.sub(s, 0) = #t then SOME s else NONE;
fun chars_responder s = if String.size s  0 then SOME (explode s) else NONE;

val pat1 = TupleP([ConstP 12, Variable var1, ConstructorP(constr1, Wildcard)]);
val pat2 = TupleP([Variable var, Wildcard, TupleP([Variable var, Wildcard, TupleP([Variable var, Wildcard])])]);
val pat3 = TupleP([Variable var1, Wildcard, TupleP([Variable var2, Wildcard, TupleP([Variable var3, Wildcard])])]);
val val1ok1 = Tuple([Const 12, Constructor(blah, Unit), Constructor(constr1, Tuple([]))]);
val val1ok2 = Tuple([Const 12, Const 13, Constructor(constr1, Const 14)]);
val val1ko1 = Tuple([Const 12, Constructor(blah, Unit), Constructor(constr2, Tuple([]))]);
val val1ko2 = Tuple([Const 13, Constructor(blah, Unit), Constructor(constr1, Tuple([]))]);
val val1ko3 = Tuple([Const 13, Constructor(blah, Unit), Unit]);
val val3ok1 = Tuple([Const 1, Unit, Tuple([Const 2, Unit, Tuple([Const 3, Unit])])]);
val val3ok2 = Tuple([Unit, Const 1, Tuple([Unit, Const 2, Tuple([Unit, Const 3])])]);
val val3ko1 = Tuple([Const 1, Unit, Tuple([Const 2, Unit, Tuple([Const 3])])]);

val _ = print nAssertionsn;
val a0101 = only_capitals([Cap,small]) = [Cap];
val a0201 = longest_string1([]) = ;
val a0202 = longest_string1([a,bb,cc]) = bb;
val a0301 = longest_string2([a,bb,cc]) = cc;
val a0401 = longest_string3([]) = ;
val a0402 = longest_string3([a,bb,cc]) = bb;
val a0403 = longest_string4([a,bb,cc]) = cc;
val a0501 = longest_capitalized([Short,longbutsmall,Longer]) = Longer;

val a0601 = rev_string(sdrawkcab) = backwards;
val a0701 = first_answer t_responder [one, two, three] = two;
val a0702 = (first_answer t_responder [one, other] handle NoAnswer = none) = none;
val a0801 = all_answers chars_responder [one, two] = SOME [#o,#n,#e,#t,#w,#o];
val a0802 = all_answers chars_responder [one, two, ] = NONE;

val a09a1 = count_wildcards pat1 = 1;
val a09a2 = count_wildcards UnitP = 0;
val a09a3 = count_wildcards pat2 = 3;
```

```
val a09b1 = count_wild_and_variable_lengths pat1 = 5;
val a09b2 = count_wild_and_variable_lengths UnitP = 0;
val a09b3 = count_wild_and_variable_lengths pat2 = 12;


val a09c1 = count_some_var(var1, pat1) = 1;
val a09c2 = count_some_var(whatever, UnitP) = 0;
val a09c3 = count_some_var(var, pat2) = 3;


val a1001 = check_pat UnitP;
val a1002 = check_pat pat1;
val a1003 = check_pat pat3;

val a1004 = not (check_pat pat2);
val a1101 = match(Unit, UnitP) = SOME [];
val a1102 = match(val1ok1, pat1) = SOME [(var1, Constructor(blah, Unit))];
val a1103 = match(val1ok2, pat1) = SOME [(var1, Const 13)];
val a1104 = match(val1ko1, pat1) = NONE;
val a1105 = match(val1ko2, pat1) = NONE;
val a1106 = match(val1ko3, pat1) = NONE;
val a1107 = match(val3ok1, pat3) = SOME [(var1, Const 1), (var2, Const 2), (var3, Const 3)];
val a1108 = match(val3ok2, pat3) = SOME [(var1, Unit), (var2, Unit), (var3, Unit)];
val a1109 = match(val3ko1, pat3) = NONE;


val a1201 = first_match val1ok1 [pat2, pat1] = SOME [(var1, Constructor(blah, Unit))];
val a1202 = first_match val1ok1 [Wildcard, pat1] = SOME [];
val a1203 = first_match val1ko1 [pat1, pat2, pat3, UnitP] = NONE;
val a1204 = first_match val3ok1 [pat1, UnitP, pat3] = SOME [(var1, Const 1), (var2, Const 2), (var3, Const 3)];
```

hw4.rkt

```
#lang racket

(provide (all-defined-out)) ;; so we can put tests in a second file

;; put your code below

(define (sequence low high stride)
  (cond [(<= low high)
         (cons low (sequence (+ low stride) high stride))]
        [#t null]))

(define (string-append-map xs suffix)
  (map (lambda (xs) (string-append xs suffix)) xs))

(define (list-nth-mod xs n)
  (cond [(< n 0) (error "list-nth-mod: negative number")]
        [(null? xs) (error "list-nth-mod: empty list")]
        [#t (car (list-tail xs (remainder n (length xs))))]))

(define (stream-for-n-steps s n)
  (if (<= n 0)
      null
      (cons (car (s)) (stream-for-n-steps (cdr (s)) (- n 1)))))

(define funny-number-stream
   (letrec ([f (lambda (x)
                 (cons (if (= (remainder x 5) 0) (- x) x)
                       (lambda () (f (+ x 1)))))])
    (lambda () (f 1))))

(define dan-then-dog
  (letrec ([f (lambda (x)
                (cons x (lambda () (f (if (eq? x "dog.jpg") "dan.jpg" "dog.jpg")))))])
           (lambda () (f "dan.jpg"))))

(define (stream-add-zero s)
  (letrec ([f (lambda (x)
                (cons (cons 0 (car (x)))
```

```
                     (lambda () (f (cdr (x))))))])
    (lambda () (f s))))

(define (cycle-lists xs ys)
  (letrec ([f (lambda (n)
                (cons (cons (list-nth-mod xs n) (list-nth-mod ys n))
                      (lambda () (f (+ n 1)))))])
    (lambda () (f 0))))

(define (vector-assoc v vec)
  (letrec ([f (lambda (n)
                (if (>= n (vector-length vec)) #f
                    (let ([vi (vector-ref vec n)])
                      (cond [(not (pair? vi)) (f (+ n 1))]
                            [(equal? (car vi) v) vi]
                            [#t (f (+ n 1))]))))])
    (f 0)))

(define (cached-assoc xs n)
  (letrec ([cache-vec (make-vector n #f)]
           [next 0]
           [find (lambda (x)
                (let ([ans (vector-assoc x cache-vec)])
                 (if ans
                     ans
                     (let ([new-ans (assoc x xs)])
                       new-ans
                       (begin
                         (vector-set! cache-vec next new-ans)
                         (set! next (remainder (+ next 1) n))
                         new-ans)))))])
  find))


```

hw4tests.rkt

```
#lang racket

(require "hw4.rkt")

(require rackunit)

;; A simple library for displaying a 2x3 grid of pictures: used
;; for fun in the tests below (look for "Tests Start Here").

(require (lib "graphics.rkt" "graphics"))

(open-graphics)

(define window-name "Programming Languages, Homework 4")
(define window-width 700)
(define window-height 500)
(define border-size 100)

(define approx-pic-width 200)
(define approx-pic-height 200)
(define pic-grid-width 3)
(define pic-grid-height 2)

(define (open-window)
  (open-viewport window-name window-width window-height))

(define (grid-posn-to-posn grid-posn)
  (when (>= grid-posn (* pic-grid-height pic-grid-width))
    (error "picture grid does not have that many positions"))
  (let ([row (quotient grid-posn pic-grid-width)]
        [col (remainder grid-posn pic-grid-width)])
    (make-posn (+ border-size (* approx-pic-width col))
               (+ border-size (* approx-pic-height row)))))

(define (place-picture window filename grid-posn)
  (let ([posn (grid-posn-to-posn grid-posn)])
    ((clear-solid-rectangle window) posn approx-pic-width approx-pic-height)
```

```
    ((draw-pixmap window) filename posn)))

(define (place-repeatedly window pause stream n)
  (when (> n 0)
    (let* ([next (stream)]
           [filename (cdar next)]
           [grid-posn (caar next)]
           [stream (cdr next)])
      (place-picture window filename grid-posn)
      (sleep pause)
      (place-repeatedly window pause stream (- n 1)))))

;; Tests Start Here

; These definitions will work only after you do some of the problems
; so you need to comment them out until you are ready.
; Add more tests as appropriate, of course.

(define nums (sequence 0 5 1))

(define files (string-append-map
               (list "dan" "dog" "curry" "dog2")
               ".jpg"))

(define funny-test (stream-for-n-steps funny-number-stream 16))

; a zero-argument function: call (one-visual-test) to open the graphics window, etc.
(define (one-visual-test)
  (place-repeatedly (open-window) 0.5 (cycle-lists nums files) 27))

; similar to previous but uses only two files and one position on the grid
(define (visual-zero-only)
  (place-repeatedly (open-window) 0.5 (stream-add-zero dan-then-dog) 27))


(check-equal? (sequence 0 5 1)
              '(0 1 2 3 4 5) "sequence #1")

(check-equal? (sequence 3 11 2)
              '(3 5 7 9 11) "sequence #2")

(check-equal? (sequence 3 8 3)
              '(3 6) "sequence #3")

(check-equal? (sequence 3 2 1)
              '() "sequence #4")

(check-equal? (string-append-map '("a" "b" "c") "-1")
              '("a-1" "b-1" "c-1") "string-append-map #1" )

(check-equal? (string-append-map '("a") "-1")
              '("a-1") "string-append-map #2" )

(check-equal? (string-append-map null "-1")
              '() "string-append-map #3" )


(check-equal? (list-nth-mod '("a" "b" "c") 0)
              "a" "list-nth-mod #1")
(check-equal? (list-nth-mod '("a" "b" "c") 2)
              "c" "list-nth-mod #2")
(check-equal? (list-nth-mod '("a" "b" "c") 4)
              "b" "list-nth-mod #3")

(check-exn (regexp "list-nth-mod: negative number")
           (lambda () (list-nth-mod '("a" "b" "c") -1) ) "not a 'list-nth-mod: negative number' thrown #5")

(check-exn (regexp "list-nth-mod: empty list")
           (lambda () (list-nth-mod '() 0)) "not a 'list-nth-mod: empty list' thrown #6")

(define nats-for-test
  (letrec ([f (lambda (x) (cons x (lambda () (f (+ x 1)))))])
    (lambda () (f 1))))

```

```
(check-equal? (stream-for-n-steps nats-for-test 5)
              '(1 2 3 4 5) "should return 5 elements in list")

(check-equal? (stream-for-n-steps funny-number-stream 16)
              '(1 2 3 4 -5 6 7 8 9 -10 11 12 13 14 -15 16) "should return 16 numbers")

(check-equal? (stream-for-n-steps funny-number-stream 0)
              '() "should return empty list")

(check-equal? (stream-for-n-steps funny-number-stream 1)
              '(1) "should return list with 1 element")


(check-equal? (stream-for-n-steps dan-then-dog 4)
              '("dan.jpg" "dog.jpg" "dan.jpg" "dog.jpg") "should return dan.jpg dog.jpg ... of 4 item list")

(check-equal? (stream-for-n-steps dan-then-dog 1)
              '("dan.jpg") "should return dan.jpg item in list")

(check-equal? (stream-for-n-steps dan-then-dog 2)
              '("dan.jpg" "dog.jpg") "should return dan.jpg and dog.jpg items in list")

(check-equal? (stream-for-n-steps (stream-add-zero dan-then-dog) 2)
              '((0 . "dan.jpg") (0 . "dog.jpg")) "should return 2 pairs (0 . 'dan.jpg') and (0 . 'dog.jpg')")

(check-equal? (stream-for-n-steps (stream-add-zero dan-then-dog) 4)
              '((0 . "dan.jpg") (0 . "dog.jpg") (0 . "dan.jpg") (0 . "dog.jpg"))  "should return 4 pairs (0 . 'dan.jpg') and (0

(check-equal? (stream-for-n-steps (stream-add-zero dan-then-dog) 0)
              '() "should return empty list")

(check-equal? (stream-for-n-steps (cycle-lists '(1 2 3) '("a" "b") ) 4)
              '((1 . "a") (2 . "b") (3 . "a") (1 . "b")) "should return mixed lists 4 pairs")

(check-equal? (vector-assoc 5 (list->vector '((1 . "a") (2 . "b") (3 . "c") (4 . "d") (5 . "e"))))
              (cons 5 "e") "should return pair ( 5 . 'e' )" )

(check-equal? (vector-assoc 6 (list->vector '((1 . "a") (2 . "b") (3 . "c") (4 . "d") (5 . "e"))))
              #f "should return pair with '5' in field" )

(check-equal? (vector-assoc 5 (list->vector '(1 2 3 4 5)))
              #f "should return #f for non paired items vector" )

(check-equal? (vector-assoc 7 (list->vector '(1 2 3 4 5 (7 . 8))))
              (cons 7 8) "should return pair with '7' in field" )

(check-equal? (vector-assoc 3 (list->vector '(1 2 (3 . 7) 4 5 (7 . 8))))
              (cons 3 7) "should return pair with '7' in field" )

(define ctf (cached-assoc '((1 . 2) (3 . 4) (5 . 6) (7 . 8) (9 . 10)) 3 ))

(check-equal? (ctf 3) (cons 3 4) "should return (3 . 4)")
(check-equal? (ctf 5) (cons 5 6) "should return (5 . 6)")
(check-equal? (ctf 9) (cons 9 10) "should return (9 . 10)")
(check-equal? (ctf 11) #f "should return #f for  v=11")
```

hw5.rkt

```
;; Programming Languages, Homework 5

#lang racket
(provide (all-defined-out)) ;; so we can put tests in a second file

;; definition of structures for MUPL programs - Do NOT change
(struct var  (string) #:transparent)  ;; a variable, e.g., (var "foo")
(struct int  (num)    #:transparent)  ;; a constant number, e.g., (int 17)
(struct add  (e1 e2)  #:transparent)  ;; add two expressions
(struct ifgreater (e1 e2 e3 e4)    #:transparent) ;; if e1 > e2 then e3 else e4
(struct fun  (nameopt formal body) #:transparent) ;; a recursive(?) 1-argument function
(struct call (funexp actual)       #:transparent) ;; function call
(struct mlet (var e body) #:transparent) ;; a local binding (let var = e in body)
(struct apair (e1 e2)     #:transparent) ;; make a new pair
(struct fst  (e)    #:transparent) ;; get first part of a pair
(struct snd  (e)    #:transparent) ;; get second part of a pair
```

```
(struct aunit ()    #:transparent) ;; unit value -- good for ending a list
(struct isaunit (e) #:transparent) ;; evaluate to 1 if e is unit else 0

;; a closure is not in "source" programs; it is what functions evaluate to
(struct closure (env fun) #:transparent)

;; Problem 1
(define (racketlist->mupllist xs)
  (if (null? xs)
      (aunit)
      (apair (car xs) (racketlist->mupllist (cdr xs)))))

;; Problem 2

(define (mupllist->racketlist xs)
  (if (aunit? xs)
      null
      (cons (apair-e1 xs) (mupllist->racketlist (apair-e2 xs)))))

;; lookup a variable in an environment
;; Do NOT change this function
(define (envlookup env str)
  (cond [(null? env) (error "unbound variable during evaluation" str)]
        [(equal? (car (car env)) str) (cdr (car env))]
        [#t (envlookup (cdr env) str)]))

;; Do NOT change the two cases given to you.
;; DO add more cases for other kinds of MUPL expressions.
;; We will test eval-under-env by calling it directly even though
;; "in real life" it would be a helper function of eval-exp.
(define (eval-under-env e env)
  (cond [(var? e)
         (envlookup env (var-string e))]
        [(add? e)
         (let ([v1 (eval-under-env (add-e1 e) env)]
               [v2 (eval-under-env (add-e2 e) env)])
           (if (and (int? v1)
                    (int? v2))
               (int (+ (int-num v1)
                       (int-num v2)))
               (error "MUPL addition applied to non-number")))]
        [(int? e) e]
        [(ifgreater? e)
         (let ([v1 (eval-under-env (ifgreater-e1 e) env)]
               [v2 (eval-under-env (ifgreater-e2 e) env)])
           (if (and (int? v1) (int? v2))
               (if (> (int-num v1) (int-num v2))
               (eval-under-env (ifgreater-e3 e) env)
               (eval-under-env (ifgreater-e4 e) env))
               ((error "MUPL ifgreater applied to non-number"))))]
        [(fun? e) (closure env e)]
        [(mlet? e)                          ;; a local binding (let var = e in body)
         (let ([eVal (eval-under-env (mlet-e e) env)])
          (eval-under-env (mlet-body e) (cons (cons (mlet-var e) eVal) env)))]
        [(apair? e)
         (apair (eval-under-env(apair-e1 e) env)
          (eval-under-env(apair-e2 e) env))]
        [(fst? e)
         (let ([frst (eval-under-env (fst-e e) env)])
               (cond [(apair? frst) (apair-e1 frst)]
                     [#t (error "MUPL fst to non-apair")]))]
        [(snd? e)
         (let ([scnd (eval-under-env (snd-e e) env)])
               (cond [(apair? scnd) (apair-e2 scnd)]
                     [#t (error "MUPL snd to non-apair")]))]
        [(isaunit? e)
         (let ([v (eval-under-env (isaunit-e e) env)])
            (cond [(aunit? v) (int 1)]
                  [#t (int 0)]))]
        [(aunit? e) e]
        [(closure? e) e]
        [(call? e)(let* ((c (eval-under-env (call-funexp e) env))
                         (arg (eval-under-env (call-actual e) env))
                         (f (cond ((closure? c)(closure-fun c))
```

```
                                    (error "MUPL call applied to non-closure")))
                            (env-temp (cons (cons (fun-formal f) arg) (closure-env c)))
                            (env (cond [(equal? (fun-nameopt f) #f) env-temp]
                                       [#t (cons (cons (fun-nameopt f) c) env-temp)])))
                       (eval-under-env (fun-body f) env))]
        ;; CHANGE add more cases here
        [#t (error "bad MUPL expression")]))

;; Do NOT change
(define (eval-exp e)
  (eval-under-env e null))

;; Problem 3

(define (ifaunit e1 e2 e3)
   (ifgreater (isaunit e1) (int 0) e2 e3))

(define (mlet* lstlst e2)
  (if (null? lstlst) e2
      (let ([v (car lstlst)])
        (mlet (car v) (cdr v) (mlet* (cdr lstlst) e2)))))

(define (ifeq e1 e2 e3 e4)
  (mlet* (list (cons "_x" e1) (cons "_y" e2))
         (ifgreater (var "_x") (var "_y") e4
                    (ifgreater (var "_y") (var "_x") e4 e3))))

;; Problem 4

(define mupl-map
  (fun #f "fun"
       (fun "map" "list"
            (ifaunit (var "list") (aunit)
                     (apair (call (var "fun") (fst (var "list")))
                            (call (var "map") (snd (var "list"))))))))


(define mupl-mapAddN
  (mlet "map" mupl-map
        (fun #f "i"
             (call (var "map") (fun #f "x" (add (var "x") (var "i")))))))

;; Challenge Problem

(struct fun-challenge (nameopt formal body freevars) #:transparent) ;; a recursive(?) 1-argument function

;; We will test this function directly, so it must do
;; as described in the assignment
(define (compute-free-vars e) "CHANGE")

;; Do NOT share code with eval-under-env because that will make
;; auto-grading and peer assessment more difficult, so
;; copy most of your interpreter here and make minor changes
(define (eval-under-env-c e env) "CHANGE")

;; Do NOT change this
(define (eval-exp-c e)
  (eval-under-env-c (compute-free-vars e) null))
```

hw5tests.rkt

```
#lang racket

(require "hw5.rkt")
(require rackunit)



; a test case that uses problems 1, 2, and 4
; should produce (list (int 10) (int 11) (int 16))
(define test1
  (mupllist->racketlist
   (eval-exp (call (call mupl-mapAddN (int 7))
                   (racketlist->mupllist
```

```
                    (list (int 3) (int 4) (int 9)))))))



(define test1_1 (racketlist->mupllist (list (int 1) (int 2) (int 3) (int 4))))
(check-equal? test1_1
              (apair (int 1) (apair (int 2) (apair (int 3) (apair (int 4) (aunit)))))
              "Testing racketlist->mupllist")
(check-equal? (mupllist->racketlist test1_1)
              (list (int 1) (int 2) (int 3) (int 4))
              "Testing mupllist->racketlist")

(check-equal? (eval-exp (ifgreater (int 1) (int 2) (add (var "crashifevaluated") (int 3)) (int 42)))
              (int 42)
              "Testing ifgreater 1 2")
(check-equal? (eval-exp (ifgreater (int 2) (int 1) (int 42) (add (var "crashifevaluated") (int 3))))
              (int 42)
              "Testing ifgreater 2 1")
(define f2_1 (eval-exp (fun "myFct" "nb" (add (int 42) (var "nb")))))
(check-equal? (eval-exp (call f2_1 (int 3)))
              (int 45)
              "Testing call")

(define f2_2 (eval-exp (mlet "ref" (int 42) (fun "myFct" "nb" (add (var "ref") (var "nb"))))))
(check-equal? (eval-exp (call f2_2 (int 3)))
              (int 45)
              "Testing mlet+call")
(check-equal? (eval-exp (mlet "ref" (int 2) (call f2_2 (int 3))))
              (int 45)
              "Testing unused mlet with call")

(define p2 (eval-exp (apair (int 7) (int 8))))
(check-equal? (eval-exp (fst p2)) (int 7) "Testing fst")
(check-equal? (eval-exp (snd p2)) (int 8) "Testing snd")

(define f2_sumall (eval-exp (fun "sumall" "nb" (ifgreater (var "nb")
                                                          (int 0)
                                                          (add (var "nb") (call (var "sumall") (add (int -1) (var "nb"))))
                                                          (int 0)))))
(check-equal? (eval-exp (call f2_sumall (int 10)))
              (int 55)
              "Testing recursive function")

(check-equal? (eval-exp (ifaunit (int 6) (add (var "crashifeval") (int 1)) (int 42)))
              (int 42)
              "Testing ifaunit 6")
(check-equal? (eval-exp (ifaunit (aunit) (int 42) (add (var "crashifeval") (int 1))))
              (int 42)
              "Testing ifaunit aunit")

(check-equal? (eval-exp (mlet* (list (cons "a" (int 5)) (cons "b" (int 6))) (add (var "b") (var "a"))))
              (int 11)
              "Testing mlet*")

(check-equal? (eval-exp (ifeq (int 5) (int 5) (int 42) (add (int 0) (var "crashifeval"))))
              (int 42)
              "Testing ifeq 5 5")
(check-equal? (eval-exp (ifeq (int 6) (int 5) (add (int 0) (var "crashifeval")) (int 42)))
              (int 42)
              "Testing ifeq 6 5")
(check-equal? (eval-exp (ifeq (int 5) (int 6) (add (int 0) (var "crashifeval")) (int 42)))
              (int 42)
              "Testing ifeq 5 6")

(define nums (racketlist->mupllist (list (int 1) (int 2) (int 3) (int 4))))
(check-equal? (eval-exp (call (call mupl-mapAddN (int 10)) nums))
              (racketlist->mupllist(list (int 11) (int 12) (int 13) (int 14)))
              "Testing mupl-map and mupl-mapAddN")
```

hw6assignment.rb

```
# University of Washington, Programming Languages, Homework 6, hw6runner.rb

# This is the only file you turn in, so do not modify the other files as
```

```
# part of your solution.

class MyTetris < Tetris
  # your enhancements here
  def initialize
  super
  end

  def key_bindings
        super
        @root.bind('u', proc {@board.rotate_180_degree})
        @root.bind('c', proc {@board.cheating})
  end

  def set_board
  @canvas = TetrisCanvas.new
  @board = MyBoard.new(self)
  @canvas.place(@board.block_size * @board.num_rows + 3,
                @board.block_size * @board.num_columns + 6, 24, 80)
  @board.draw
  end
end

class MyPiece < Piece
  # The constant All_My_Pieces should be declared here
  All_My_Pieces = Piece::All_Pieces.concat([
                                  rotations([[0, 0], [-1, 0], [-1, -1], [0, -1], [1, -1]]),
                             [[[0, 0], [-1, 0], [1, 0], [2, 0], [-2, 0]],
                 [[0, 0], [0, -1], [0, 1], [0, 2], [0, -2]]],
                             rotations([[0, 0], [1, 0], [0, 1]])])
  Cheat_piece = [[[0, 0]]]

  def initialize (point_array, board)
    super(point_array, board)
  end
  # your enhancements here

  def self.next_piece (board)
    MyPiece.new(All_My_Pieces.sample, board)
  end

  def self.next_cheat_piece(board)
        MyPiece.new(Cheat_piece, board)
  end

end

class MyBoard < Board
  def initialize (game)
    @grid = Array.new(num_rows) {Array.new(num_columns)}
    @current_block = MyPiece.next_piece(self)
    @score = 0
    @game = game
    @delay = 500
        @cheating = false
  end
  # your enhancements here
  def rotate_180_degree
    if !game_over? and @game.is_running?
      @current_block.move(0, 0, -2)
    end
    draw
  end

  def cheating
        if @score >= 100 && @cheating == false
                @score -= 100
                @cheating = true
        end

  end

 # gets the next piece
  def next_piece
```

```
        if @cheating
                @current_block = MyPiece.next_cheat_piece(self)
                @cheating = false
        else
                @current_block = MyPiece.next_piece(self)
        end
    @current_pos = nil
  end

  def store_current
    locations = @current_block.current_rotation
    displacement = @current_block.position
    (0..(locations.size-1)).each{|index|
      current = locations[index];
      @grid[current[1]+displacement[1]][current[0]+displacement[0]] =
      @current_pos[index]
    }
    remove_filled
    @delay = [@delay - 2, 80].max
  end


end
```

hw6graphics.rb

```
# University of Washington, Programming Languages, Homework 6, hw6graphics.rb

# This file provides an interface to a wrapped Tk library. The auto-grader will
# swap it out to use a different, non-Tk backend.

require 'tk'

class TetrisRoot
  def initialize
    @root = TkRoot.new('height' => 615, 'width' => 205,
              'background' => 'lightblue') {title "Tetris"}
  end

  def bind(char, callback)
    @root.bind(char, callback)
  end

  # Necessary so we can unwrap before passing to Tk in some instances.
  # Student code MUST NOT CALL THIS.
  attr_reader :root
end

class TetrisTimer
  def initialize
    @timer = TkTimer.new
  end

  def stop
    @timer.stop
  end

  def start(delay, callback)
    @timer.start(delay, callback)
  end
end

class TetrisCanvas
  def initialize
    @canvas = TkCanvas.new('background' => 'grey')
  end

  def place(height, width, x, y)
    @canvas.place('height' => height, 'width' => width, 'x' => x, 'y' => y)
  end

  def unplace
    @canvas.unplace
  end
```

```
49
50    def delete
51      @canvas.delete
52    end
53
54    # Necessary so we can unwrap before passing to Tk in some instances.
55    # Student code MUST NOT CALL THIS.
56    attr_reader :canvas
57  end
58
59  class TetrisLabel
60    def initialize(wrapped_root, &options)
61      unwrapped_root = wrapped_root.root
62      @label = TkLabel.new(unwrapped_root, &options)
63    end
64
65    def place(height, width, x, y)
66      @label.place('height' => height, 'width' => width, 'x' => x, 'y' => y)
67    end
68
69    def text(str)
70      @label.text(str)
71    end
72  end
73
74  class TetrisButton
75    def initialize(label, color)
76      @button = TkButton.new do
77        text label
78        background color
79        command (proc {yield})
80      end
81    end
82
83    def place(height, width, x, y)
84      @button.place('height' => height, 'width' => width, 'x' => x, 'y' => y)
85    end
86  end
87
88  class TetrisRect
89    def initialize(wrapped_canvas, a, b, c, d, color)
90      unwrapped_canvas = wrapped_canvas.canvas
91      @rect = TkcRectangle.new(unwrapped_canvas, a, b, c, d,
92                               'outline' => 'black', 'fill' => color)
93    end
94
95    def remove
96      @rect.remove
97    end
98
99    def move(dx, dy)
100     @rect.move(dx, dy)
101   end
102
103 end
104
105 def mainLoop
106   Tk.mainloop
107 end
108
109 def exitProgram
110   Tk.exit
111 end
```

hw6provided.rb

```
1  # University of Washington, Programming Languages, Homework 6, hw6provided.rb
2
3  require_relative './hw6graphics'
4
5  # class responsible for the pieces and their movements
6  class Piece
7
8    # creates a new Piece from the given point array, holding the board for
```

```
  # determining if movement is possible for the piece, and gives the piece a
  # color, rotation, and starting position.
  def initialize (point_array, board)
    @all_rotations = point_array
    @rotation_index = (0..(@all_rotations.size-1)).to_a.sample
    @color = All_Colors.sample
    @base_position = [5, 0] # [column, row]
    @board = board
    @moved = true
  end

  def current_rotation
    @all_rotations[@rotation_index]
  end

  def moved
    @moved
  end

  def position
    @base_position
  end

  def color
    @color
  end

  def drop_by_one
    @moved = move(0, 1, 0)
  end

  # takes the intended movement in x, y and rotation and checks to see if the
  # movement is possible.  If it is, makes this movement and returns true.
  # Otherwise returns false.
  def move (delta_x, delta_y, delta_rotation)
    # Ensures that the rotation will always be a possible formation (as opposed
    # to nil) by altering the intended rotation so that it stays
    # within the bounds of the rotation array
    moved = true
    potential = @all_rotations[(@rotation_index + delta_rotation) % @all_rotations.size]
    # for each individual block in the piece, checks if the intended move
    # will put this block in an occupied space
    potential.each{|posns|
      if !(@board.empty_at([posns[0] + delta_x + @base_position[0],
                            posns[1] + delta_y + @base_position[1]]));
        moved = false;
      end
    }
    if moved
      @base_position[0] += delta_x
      @base_position[1] += delta_y
      @rotation_index = (@rotation_index + delta_rotation) % @all_rotations.size
    end
    moved
  end

  # class method to figures out the different rotations of the provided piece
  def self.rotations (point_array)
    rotate1 = point_array.map {|x,y| [-y,x]}
    rotate2 = point_array.map {|x,y| [-x,-y]}
    rotate3 = point_array.map {|x,y| [y,-x]}
    [point_array, rotate1, rotate2, rotate3]
  end

  # class method to choose the next piece
  def self.next_piece (board)
    Piece.new(All_Pieces.sample, board)
  end

  # class array holding all the pieces and their rotations
  All_Pieces = [[[[0, 0], [1, 0], [0, 1], [1, 1]]],  # square (only needs one)
                rotations([[0, 0], [-1, 0], [1, 0], [0, -1]]), # T
                [[[0, 0], [-1, 0], [1, 0], [2, 0]], # long (only needs two)
                [[0, 0], [0, -1], [0, 1], [0, 2]]],
```

```
                rotations([[0, 0], [0, -1], [0, 1], [1, 1]]), # L
                rotations([[0, 0], [0, -1], [0, 1], [-1, 1]]), # inverted L
                rotations([[0, 0], [-1, 0], [0, -1], [1, -1]]), # S
                rotations([[0, 0], [1, 0], [0, -1], [-1, -1]])] # Z

  # class array
  All_Colors = ['DarkGreen', 'dark blue', 'dark red', 'gold2', 'Purple3',
                'OrangeRed2', 'LightSkyBlue']
end


# Class responsible for the interaction between the pieces and the game itself
class Board

  def initialize (game)
    @grid = Array.new(num_rows) {Array.new(num_columns)}
    @current_block = Piece.next_piece(self)
    @score = 0
    @game = game
    @delay = 500
  end

  # both the length and the width of a block, since it is a square
  def block_size
    15
  end

  def num_columns
    10
  end

  def num_rows
    27
  end

  # the current score
  def score
    @score
  end

  # the current delay
  def delay
    @delay
  end

  # the game is over when there is a piece extending into the second row
  # from the top
  def game_over?
    @grid[1].any?
  end

  # moves the current piece down by one, if this is not possible stores the
  # current piece and replaces it with a new one.
  def run
    ran = @current_block.drop_by_one
    if !ran
      store_current
      if !game_over?
        next_piece
      end
    end
    @game.update_score
    draw
  end

  # moves the current piece left if possible
  def move_left
    if !game_over? and @game.is_running?
      @current_block.move(-1, 0, 0)
    end
    draw
  end

  # moves the current piece right if possible
```

```
  def move_right
    if !game_over? and @game.is_running?
      @current_block.move(1, 0, 0)
    end
    draw
  end

  # rotates the current piece clockwise
  def rotate_clockwise
    if !game_over? and @game.is_running?
      @current_block.move(0, 0, 1)
    end
    draw
  end

  # rotates the current piece counterclockwise
  def rotate_counter_clockwise
    if !game_over? and @game.is_running?
      @current_block.move(0, 0, -1)
    end
    draw
  end

  # drops the piece to the lowest location in the currently occupied columns.
  # Then replaces it with a new piece
  # Change the score to reflect the distance dropped.
  def drop_all_the_way
    if @game.is_running?
      ran = @current_block.drop_by_one
      while ran
        @current_pos.each{|block| block.remove}
        @score += 1
        ran = @current_block.drop_by_one
      end
      draw
      store_current
      if !game_over?
        next_piece
      end
      @game.update_score
      draw
    end
  end

  # gets the next piece
  def next_piece
    @current_block = Piece.next_piece(self)
    @current_pos = nil
  end

  # gets the information from the current piece about where it is and uses this
  # to store the piece on the board itself.  Then calls remove_filled.
  def store_current
    locations = @current_block.current_rotation
    displacement = @current_block.position
    (0..3).each{|index|
      current = locations[index];
      @grid[current[1]+displacement[1]][current[0]+displacement[0]] =
      @current_pos[index]
    }
    remove_filled
    @delay = [@delay - 2, 80].max
  end

  # Takes a point and checks to see if it is in the bounds of the board and
  # currently empty.
  def empty_at (point)
    if !(point[0] >= 0 and point[0] < num_columns)
      return false
    elsif point[1] < 1
      return true
    elsif point[1] >= num_rows
      return false
    end
```

```
    @grid[point[1]][point[0]] == nil
  end

  # removes all filled rows and replaces them with empty ones, dropping all rows
  # above them down each time a row is removed and increasing the score.
  def remove_filled
    (2..(@grid.size-1)).each{|num| row = @grid.slice(num);
      # see if this row is full (has no nil)
      if @grid[num].all?
        # remove from canvas blocks in full row
        (0..(num_columns-1)).each{|index|
          @grid[num][index].remove;
          @grid[num][index] = nil
        }
        # move down all rows above and move their blocks on the canvas
        ((@grid.size - num + 1)..(@grid.size)).each{|num2|
          @grid[@grid.size - num2].each{|rect| rect && rect.move(0, block_size)};
          @grid[@grid.size-num2+1] = Array.new(@grid[@grid.size - num2])
        }
        # insert new blank row at top
        @grid[0] = Array.new(num_columns);
        # adjust score for full flow
        @score += 10;
      end}
    self
  end

  # current_pos holds the intermediate blocks of a piece before they are placed
  # in the grid.  If there were any before, they are sent to the piece drawing
  # method to be removed and replaced with that of the new position
  def draw
    @current_pos = @game.draw_piece(@current_block, @current_pos)
  end
end

class Tetris

  # creates the window and starts the game
  def initialize
    @root = TetrisRoot.new
    @timer = TetrisTimer.new
    set_board
    @running = true
    key_bindings
    buttons
    run_game
  end

  # creates a canvas and the board that interacts with it
  def set_board
    @canvas = TetrisCanvas.new
    @board = Board.new(self)
    @canvas.place(@board.block_size * @board.num_rows + 3,
                  @board.block_size * @board.num_columns + 6, 24, 80)
    @board.draw
  end

  def key_bindings
    @root.bind('n', proc {self.new_game})

    @root.bind('p', proc {self.pause})

    @root.bind('q', proc {exitProgram})

    @root.bind('a', proc {@board.move_left})
    @root.bind('Left', proc {@board.move_left})

    @root.bind('d', proc {@board.move_right})
    @root.bind('Right', proc {@board.move_right})

    @root.bind('s', proc {@board.rotate_clockwise})
    @root.bind('Down', proc {@board.rotate_clockwise})

    @root.bind('w', proc {@board.rotate_counter_clockwise})
```

```
    @root.bind('Up', proc {@board.rotate_counter_clockwise})

    @root.bind('space' , proc {@board.drop_all_the_way})
  end

  def buttons
    pause = TetrisButton.new('pause', 'lightcoral'){self.pause}
    pause.place(35, 50, 90, 7)

    new_game = TetrisButton.new('new game', 'lightcoral'){self.new_game}
    new_game.place(35, 75, 15, 7)

    quit = TetrisButton.new('quit', 'lightcoral'){exitProgram}
    quit.place(35, 50, 140, 7)

    move_left = TetrisButton.new('left', 'lightgreen'){@board.move_left}
    move_left.place(35, 50, 27, 536)

    move_right = TetrisButton.new('right', 'lightgreen'){@board.move_right}
    move_right.place(35, 50, 127, 536)

    rotate_clock = TetrisButton.new('^_)', 'lightgreen'){@board.rotate_clockwise}
    rotate_clock.place(35, 50, 77, 501)

    rotate_counter = TetrisButton.new('(_^', 'lightgreen'){
      @board.rotate_counter_clockwise}
    rotate_counter.place(35, 50, 77, 571)

    drop = TetrisButton.new('drop', 'lightgreen'){@board.drop_all_the_way}
    drop.place(35, 50, 77, 536)

    label = TetrisLabel.new(@root) do
      text 'Current Score: '
      background 'lightblue'
    end
    label.place(35, 100, 26, 45)
    @score = TetrisLabel.new(@root) do
      background 'lightblue'
    end
    @score.text(@board.score)
    @score.place(35, 50, 126, 45)
  end

  # starts the game over, replacing the old board and score
  def new_game
    @canvas.unplace
    @canvas.delete
    set_board
    @score.text(@board.score)
    @running = true
    run_game
  end

  # pauses the game or resumes it
  def pause
    if @running
      @running = false
      @timer.stop
    else
      @running = true
      self.run_game
    end
  end

  # alters the displayed score to reflect what is currently stored in the board
  def update_score
    @score.text(@board.score)
  end

  # repeatedly calls itself so that the process is fully automated.  Checks if
  # the game is over and if it isn't, calls the board's run method which moves
  # a piece down and replaces it with a new one when the old one can't move any
  # more
  def run_game
```

```
    if !@board.game_over? and @running
      @timer.stop
      @timer.start(@board.delay, (proc{@board.run; run_game}))
    end
  end

  # whether the game is running
  def is_running?
    @running
  end

  # takes a piece and optionally the list of old TetrisRects corresponding
  # to it and returns a new set of TetrisRects which are how the piece is
  # visible to the user.
  def draw_piece (piece, old=nil)
    if old != nil and piece.moved
      old.each{|block| block.remove}
    end
    size = @board.block_size
    blocks = piece.current_rotation
    start = piece.position
    blocks.map{|block|
    TetrisRect.new(@canvas, start[0]*size + block[0]*size + 3,
                        start[1]*size + block[1]*size,
                        start[0]*size + size + block[0]*size + 3,
                        start[1]*size + size + block[1]*size,
                        piece.color)}
  end
end

# To help each game of Tetris be unique.
srand
```

hw6runner.rb

```
# University of Washington, Programming Languages, Homework 6, hw6runner.rb

require_relative './hw6provided'
require_relative './hw6assignment'

def runTetris
  Tetris.new
  mainLoop
end

def runMyTetris
  MyTetris.new
  mainLoop
end

if ARGV.count == 0
  runMyTetris
elsif ARGV.count != 1
  puts "usage: hw6runner.rb [enhanced | original]"
elsif ARGV[0] == "enhanced"
  runMyTetris
elsif ARGV[0] == "original"
  runTetris
else
  puts "usage: hw6runner.rb [enhanced | original]"
end

```

hw7.rb

```
# University of Washington, Programming Languages, Homework 7, hw7.rb
# (See also ML code.)

# a little language for 2D geometry objects

# each subclass of GeometryExpression, including subclasses of GeometryValue,
#  needs to respond to messages preprocess_prog and eval_prog
#
# each subclass of GeometryValue additionally needs:
```

```
#   * shift
#   * intersect, which uses the double-dispatch pattern
#   * intersectPoint, intersectLine, and intersectVerticalLine for
#       for being called by intersect of appropriate clases and doing
#       the correct intersection calculuation
#   * (We would need intersectNoPoints and intersectLineSegment, but these
#      are provided by GeometryValue and should not be overridden.)
#   *  intersectWithSegmentAsLineResult, which is used by
#      intersectLineSegment as described in the assignment
#
# you can define other helper methods, but will not find much need to

# Note: geometry objects should be immutable: assign to fields only during
#       object construction

# Note: For eval_prog, represent environments as arrays of 2-element arrays
# as described in the assignment

class GeometryExpression
  # do *not* change this class definition
  Epsilon = 0.00001
end

class GeometryValue < GeometryExpression
  # do *not* change methods in this class definition
  # you can add methods if you wish

  private
  # some helper methods that may be generally useful
  def real_close(r1,r2)
    (r1 - r2).abs < GeometryExpression::Epsilon
  end
  def real_close_point(x1,y1,x2,y2)
          real_close(x1,x2) && real_close(y1,y2)
  end
  # two_points_to_line could return a Line or a VerticalLine
  def two_points_to_line(x1,y1,x2,y2)
      if real_close(x1,x2)
        VerticalLine.new x1
      else
        m = (y2 - y1).to_f / (x2 - x1)
        b = y1 - m * x1
        Line.new(m,b)
      end
  end

  public
  # we put this in this class so all subclasses can inherit it:
  # the intersection of self with a NoPoints is a NoPoints object
  def intersectNoPoints np
    np # could also have NoPoints.new here instead
  end

  # we put this in this class so all subclasses can inhert it:
  # the intersection of self with a LineSegment is computed by
  # first intersecting with the line containing the segment and then
  # calling the result's intersectWithSegmentAsLineResult with the segment
  def intersectLineSegment seg
    line_result = intersect(two_points_to_line(seg.x1,seg.y1,seg.x2,seg.y2))
    line_result.intersectWithSegmentAsLineResult seg
  end
end

class NoPoints < GeometryValue
  # do *not* change this class definition: everything is done for you
  # (although this is the easiest class, it shows what methods every subclass
  # of geometry values needs)

  # Note: no initialize method only because there is nothing it needs to do
  def eval_prog env
    self # all values evaluate to self
  end
  def preprocess_prog
    self # no pre-processing to do here
```

```ruby
  end
  def shift(dx,dy)
    self # shifting no-points is no-points
  end
  def intersect other
    other.intersectNoPoints self # will be NoPoints but follow double-dispatch
  end
  def intersectPoint p
    self # intersection with point and no-points is no-points
  end
  def intersectLine line
    self # intersection with line and no-points is no-points
  end
  def intersectVerticalLine vline
    self # intersection with line and no-points is no-points
  end
  # if self is the intersection of (1) some shape s and (2)
  # the line containing seg, then we return the intersection of the
  # shape s and the seg.  seg is an instance of LineSegment
  def intersectWithSegmentAsLineResult seg
    self
  end
end


class Point < GeometryValue
  # *add* methods to this class -- do *not* change given code and do not
  # override any methods

  # Note: You may want a private helper method like the local
  # helper function inbetween in the ML code
  attr_reader :x, :y
  def initialize(x,y)
    @x = x
    @y = y
  end

  def eval_prog env
    self # all values evaluate to self
  end

  def preprocess_prog
    self # no pre-processing to do here
  end

  def shift(dx, dy)
    Point.new(dx+@x,dy+@y)
  end

  def intersect other
    other.intersectPoint self
  end

  def intersectPoint p
  if real_close_point(@x,@y,p.x,p.y)
        then self
    else NoPoints.new
    end
  end

  def intersectLine line
        line.intersectPoint self
  end

  def intersectVerticalLine vline
    vline.intersectPoint self
  end

  def inbetween(v, end1, end2)
        (end1 - GeometryExpression::Epsilon <= v  and v <= end2 + GeometryExpression::Epsilon) or (end2 - GeometryExpression::E
  end

  def intersectWithSegmentAsLineResult seg
        if  inbetween(x,seg.x1,seg.x2) and inbetween(y,seg.y1,seg.y2)
```

```
                then Point.new(@x, @y)
        else NoPoints.new
        end
  end
end

class Line < GeometryValue
  # *add* methods to this class -- do *not* change given code and do not
  # override any methods
  attr_reader :m, :b
  def initialize(m,b)
    @m = m
    @b = b
  end

  def eval_prog env
    self # all values evaluate to self
  end

  def preprocess_prog
    self # no pre-processing to do here
  end

  def shift(dx,dy)
    Line.new(m,b+dy-m*dx)
  end

  def intersect other
    other.intersectLine self
  end

  def intersectPoint p
    if real_close(p.y,@m*p.x+@b) then p
    else NoPoints.new
    end
  end

  def intersectLine line
    if real_close(@m,line.m) then
      if real_close(@b, line.b) then self
      else NoPoints.new
      end
    else Point.new((line.b-@b)/(@m-line.m),@m*(line.b-@b)/(@m-line.m)+@b)
    end
  end

  def intersectVerticalLine vline
    vline.intersectLine self
  end

  def intersectWithSegmentAsLineResult seg
        seg
  end
end

class VerticalLine < GeometryValue
  # *add* methods to this class -- do *not* change given code and do not
  # override any methods
  attr_reader :x
  def initialize x
    @x = x
  end

  def eval_prog env
    self # all values evaluate to self
  end

  def preprocess_prog
    self # no pre-processing to do here
  end

  def shift(dx,dy)
    VerticalLine.new(@x+dx)
  end
```

```
232
233   def intersect other
234     other.intersectVerticalLine self
235   end
236
237   def intersectPoint p
238     if real_close(p.x,@x) then p
239     else NoPoints.new
240     end
241   end
242
243   def intersectLine line
244         Point.new(@x,line.m*@x+line.b)
245   end
246
247   def intersectVerticalLine vline
248     if real_close(@x, vline.x) then self
249     else NoPoints.new
250     end
251   end
252
253   def intersectWithSegmentAsLineResult seg
254     seg
255   end
256 end
257
258 class LineSegment < GeometryValue
259   # *add* methods to this class -- do *not* change given code and do not
260   # override any methods
261   # Note: This is the most difficult class.  In the sample solution,
262   #  preprocess_prog is about 15 lines long and
263   # intersectWithSegmentAsLineResult is about 40 lines long
264   attr_reader :x1, :y1, :x2, :y2
265   def initialize (x1,y1,x2,y2)
266     @x1 = x1
267     @y1 = y1
268     @x2 = x2
269     @y2 = y2
270   end
271
272   def eval_prog env
273     self # all values evaluate to self
274   end
275
276   def preprocess_prog
277         s_close = real_close(@x1,@x2)
278         e_close = real_close(@y1,@y2)
279         if (real_close_point(@x1,@y1,@x2,@y2)) then Point.new(@x1,@y1)
280         elsif ((@x1>@x2) and (not s_close)) then LineSegment.new(@x2,@y2,@x1,@y1)
281     elsif ((@y1>@y2) and (not e_close)) then LineSegment.new(@x2,@y2,@x1,@y1)
282     else self
283         end
284   end
285
286   def shift(dx,dy)
287     LineSegment.new(@x1+dx,@y1+dy,@x2+dx,@y2+dy)
288   end
289
290   def intersect other
291     other.intersectLineSegment self
292   end
293
294   def intersectPoint p
295     p.intersectLineSegment self
296   end
297   def intersectLine line
298     line.intersectLineSegment self
299   end
300   def intersectVerticalLine vline
301     vline.intersectLineSegment self
302   end
303
304   def return_real_close_helper(aXstart,aYstart,aXend,aYend,bXstart,bYstart,bXend,bYend)
305         if real_close(aYend,bYstart) then Point.new(aXend,aYend)
```

```
        elsif aYend < bYstart then NoPoints.new
        elsif aYend > bYend then LineSegment.new(bXstart,bYstart,bXend,bYend)
        else LineSegment.new(bXstart,bYstart,aXend,aYend)
        end
  end

  def return_not_real_close_helper(aXstart,aYstart,aXend,aYend,bXstart,bYstart,bXend,bYend)
        if real_close(aXend,bXstart) then Point.new(aXend,aYend)
        elsif aXend < bXstart then NoPoints.new
        elsif aXend > bXend then LineSegment.new(bXstart,bYstart,bXend,bYend)
        else LineSegment.new(bXstart,bYstart,aXend,aYend)
        end
  end

  def intersectWithSegmentAsLineResult seg
    if real_close(@x1, @x2) then
                if @y1 < seg.y1 then
                  return_real_close_helper(@x1,@y1,@x2,@y2,seg.x1,seg.y1,seg.x2,seg.y2)
                else
                  return_real_close_helper(seg.x1,seg.y1,seg.x2,seg.y2,@x1,@y1,@x2,@y2)
                end
        else
                if @x1 < seg.x1 then
                  return_not_real_close_helper(@x1,@y1,@x2,@y2,seg.x1,seg.y1,seg.x2,seg.y2)
                else
                  return_not_real_close_helper(seg.x1,seg.y1,seg.x2,seg.y2,@x1,@y1,@x2,@y2)
                end
        end
  end
end

# Note: there is no need for getter methods for the non-value classes

class Intersect < GeometryExpression
  # *add* methods to this class -- do *not* change given code and do not
  # override any methods
  def initialize(e1,e2)
    @e1 = e1
    @e2 = e2
  end

  def preprocess_prog
    Intersect.new(@e1.preprocess_prog, @e2.preprocess_prog)
  end

  def eval_prog env
    @e1.eval_prog(env).intersect(@e2.eval_prog env)
  end
end

class Let < GeometryExpression
  # *add* methods to this class -- do *not* change given code and do not
  # override any methods
  def initialize(s,e1,e2)
    @s = s
    @e1 = e1
    @e2 = e2
  end

  def preprocess_prog
    Let.new(@s, @e1.preprocess_prog, @e2.preprocess_prog)
  end

  def eval_prog env
    @e2.eval_prog env.unshift([@s, (@e1.eval_prog env)])
    #@e2.eval_prog([@s,@e1.eval_prog(env)]+env)
  end
end

class Var < GeometryExpression
  # *add* methods to this class -- do *not* change given code and do not
  # override any methods
  def initialize s
    @s = s
```

```ruby
  end

  def preprocess_prog
    self
  end

  def eval_prog env
    env.assoc(@s)[1]
  end
end

class Shift < GeometryExpression
  # *add* methods to this class -- do *not* change given code and do not
  # override any methods
  def initialize(dx,dy,e)
    @dx = dx
    @dy = dy
    @e = e
  end

  def preprocess_prog
    Shift.new(@dx, @dy, @e.preprocess_prog)
  end

  def eval_prog env
    e = @e.eval_prog env
    e.shift(@dx, @dy)
  end
end
```

hw7.sml

```
(* University of Washington, Programming Languages, Homework 7, hw7.sml
   (See also Ruby code.)
*)

(* Do not make changes to this code except where you see comments containing
   the word CHANGE. *)

(* expressions in a little language for 2D geometry objects
   values: points, lines, vertical lines, line segments
   other expressions: intersection of two expressions, lets, variables,
                      (shifts added by you)
*)
datatype geom_exp =
           NoPoints
   | Point of real * real (* represents point (x,y) *)
         | Line of real * real (* represents line (slope, intercept) *)
         | VerticalLine of real (* x value *)
         | LineSegment of real * real * real * real (* x1,y1 to x2,y2 *)
         | Intersect of geom_exp * geom_exp (* intersection expression *)
         | Let of string * geom_exp * geom_exp (* let s = e1 in e2 *)
         | Var of string
         | Shift of real * real * geom_exp
(* CHANGE add shifts for expressions of the form Shift(deltaX, deltaY, exp *)

exception BadProgram of string
exception Impossible of string

(* helper functions for comparing real numbers since rounding means
   we should never compare for equality *)

val epsilon = 0.00001

fun real_close (r1,r2) =
    (Real.abs (r1 - r2)) < epsilon

(* notice curried *)
fun real_close_point (x1,y1) (x2,y2) =
    real_close(x1,x2) andalso real_close(y1,y2)

(* helper function to return the Line or VerticalLine containing
   points (x1,y1) and (x2,y2). Actually used only when intersecting
   line segments, but might be generally useful *)
```

```
fun two_points_to_line (x1,y1,x2,y2) =
    if real_close(x1,x2)
    then VerticalLine x1
    else
        let
            val m = (y2 - y1) / (x2 - x1)
            val b = y1 - m * x1
        in
            Line(m,b)
        end

(* helper function for interpreter: return value that is the intersection
   of the arguments: 25 cases because there are 5 kinds of values, but
   many cases can be combined, especially because intersection is commutative.
   Do *not* call this function with non-values (e.g., shifts or lets)
 *)
fun intersect (v1,v2) =
    case (v1,v2) of

       (NoPoints, _) => NoPoints (* 5 cases *)
     | (_, NoPoints) => NoPoints (* 4 additional cases *)

     |  (Point p1, Point p2) => if real_close_point p1 p2
                                then v1
                                else NoPoints

      | (Point (x,y), Line (m,b)) => if real_close(y, m * x + b)
                                     then v1
                                     else NoPoints

      | (Point (x1,_), VerticalLine x2) => if real_close(x1,x2)
                                           then v1
                                           else NoPoints

      | (Point _, LineSegment seg) => intersect(v2,v1)

      | (Line _, Point _) => intersect(v2,v1)

      | (Line (m1,b1), Line (m2,b2)) =>
        if real_close(m1,m2)
        then (if real_close(b1,b2)
              then v1 (* same line *)
              else  NoPoints) (* parallel lines do not intersect *)
        else
            let (* one-point intersection *)
                val x = (b2 - b1) / (m1 - m2)
                val y = m1 * x + b1
            in
                Point (x,y)
            end

      | (Line (m1,b1), VerticalLine x2) => Point(x2, m1 * x2 + b1)

      | (Line _, LineSegment _) => intersect(v2,v1)

      | (VerticalLine _, Point _) => intersect(v2,v1)
      | (VerticalLine _, Line _)  => intersect(v2,v1)

      | (VerticalLine x1, VerticalLine x2) =>
        if real_close(x1,x2)
        then v1 (* same line *)
        else NoPoints (* parallel *)

      | (VerticalLine _, LineSegment seg) => intersect(v2,v1)

      | (LineSegment seg, _) =>
        (* the hard case, actually 4 cases because v2 could be a point,
           line, vertical line, or line segment *)
        (* First compute the intersection of (1) the line containing the segment
           and (2) v2. Then use that result to compute what we need. *)
        (case intersect(two_points_to_line seg, v2) of
            NoPoints => NoPoints
          | Point(x0,y0) => (* see if the point is within the segment bounds *)
            (* assumes v1 was properly preprocessed *)
```

```
            let
                fun inbetween(v,end1,end2) =
                    (end1 - epsilon <= v andalso v <= end2 + epsilon)
                    orelse (end2 - epsilon <= v andalso v <= end1 + epsilon)
                val (x1,y1,x2,y2) = seg
            in
                if inbetween(x0,x1,x2) andalso inbetween(y0,y1,y2)
                then Point(x0,y0)
                else NoPoints
            end
          | Line _ => v1 (* so segment seg is on line v2 *)
          | VerticalLine _ => v1 (* so segment seg is on vertical-line v2 *)
          | LineSegment seg2 =>
            (* the hard case in the hard case: seg and seg2 are on the same
               line (or vertical line), but they could be (1) disjoint or
               (2) overlapping or (3) one inside the other or (4) just touching.
               And we treat vertical segments differently, so there are 4*2 cases.
             *)
            let
                val (x1start,y1start,x1end,y1end) = seg
                val (x2start,y2start,x2end,y2end) = seg2
            in
                if real_close(x1start,x1end)
                then (* the segments are on a vertical line *)
                    (* let segment a start at or below start of segment b *)
                    let
                        val ((aXstart,aYstart,aXend,aYend),
                             (bXstart,bYstart,bXend,bYend)) = if y1start < y2start
                                                              then (seg,seg2)
                                                              else (seg2,seg)
                    in
                        if real_close(aYend,bYstart)
                        then Point (aXend,aYend) (* just touching *)
                        else if aYend < bYstart
                        then NoPoints (* disjoint *)
                        else if aYend > bYend
                        then LineSegment(bXstart,bYstart,bXend,bYend) (* b inside a *)
                        else LineSegment(bXstart,bYstart,aXend,aYend) (* overlapping *)
                    end
                else (* the segments are on a (non-vertical) line *)
                    (* let segment a start at or to the left of start of segment b *)
                    let
                        val ((aXstart,aYstart,aXend,aYend),
                             (bXstart,bYstart,bXend,bYend)) = if x1start < x2start
                                                              then (seg,seg2)
                                                              else (seg2,seg)
                    in
                        if real_close(aXend,bXstart)
                        then Point (aXend,aYend) (* just touching *)
                        else if aXend < bXstart
                        then NoPoints (* disjoint *)
                        else if aXend > bXend
                        then LineSegment(bXstart,bYstart,bXend,bYend) (* b inside a *)
                        else LineSegment(bXstart,bYstart,aXend,aYend) (* overlapping *)
                    end
            end
          | _ => raise Impossible "bad result from intersecting with a line")
      | _ => raise Impossible "bad call to intersect: only for shape values"

(* interpreter for our language:
   * takes a geometry expression and returns a geometry value
   * for simplicity we have the top-level function take an environment,
     (which should be [] for the whole program
   * we assume the expression e has already been "preprocessed" as described
     in the homework assignment:
         * line segments are not actually points (endpoints not real close)
         * lines segment have left (or, if vertical, bottom) coordinate first
*)

fun eval_prog (e,env) =
    case e of
        NoPoints => e (* first 5 cases are all values, so no computation *)
      | Point _  => e
      | Line _   => e
```

```
      | VerticalLine _ => e
      | LineSegment _  => e
      | Var s =>
        (case List.find (fn (s2,v) => s=s2) env of
             NONE => raise BadProgram("var not found: " ^ s)
           | SOME (_,v) => v)
      | Let(s,e1,e2) => eval_prog (e2, ((s, eval_prog(e1,env)) :: env))
      | Intersect(e1,e2) => intersect(eval_prog(e1,env), eval_prog(e2, env))
      | Shift(dx,dy,e) =>
        let
          val result = eval_prog(e, env)
        in
          case result of
              NoPoints => NoPoints
            | Point(x,y)=> Point(x+dx,y+dy)
            | Line(s,i)=>  Line(s,i+dy-s*dx)
            | VerticalLine(x) => VerticalLine(x+dx)
            | LineSegment(x1,y1,x2,y2) => LineSegment(x1+dx,y1+dy,x2+dx,y2+dy)
        end

(* CHANGE: Add a case for Shift expressions *)

(* CHANGE: Add function preprocess_prog of type geom_exp -> geom_exp *)

fun preprocess_prog e=
    case e of
      LineSegment(s1,e1,s2,e2) =>
      let val s_close = real_close(s1,s2)
          val e_close = real_close(e1,e2)
      in
        if (real_close_point(s1,e1) (s2,e2)) then Point(s1,e1)
        else if ((s1>s2) andalso (not s_close)) then LineSegment(s2,e2,s1,e1)
        else if ((e1>e2) andalso (not e_close)) then LineSegment(s2,e2,s1,e1)
        else LineSegment(s1,e1,s2,e2)
      end
      | Intersect(x,y) => Intersect(preprocess_prog(x),preprocess_prog(y))
      | Let(s,e1,e2) => Let(s, preprocess_prog(e1),preprocess_prog(e2))
      | Shift(dx,dy,e) => Shift(dx,dy,preprocess_prog(e))
      | _ => e
```

hw7testsprovided.rb

```
# University of Washington, Programming Languages, Homework 7,
# hw7testsprovided.rb

require "./hw7.rb"

#  Will not work completely until you implement all the classes and their methods

# Will print only if code has errors; prints nothing if all tests pass

# These tests do NOT cover all the various cases, especially for intersection

#Constants for testing
ZERO = 0.0
ONE = 1.0
TWO = 2.0
THREE = 3.0
FOUR = 4.0
FIVE = 5.0
SIX = 6.0
SEVEN = 7.0
TEN = 10.0

#Point Tests
a = Point.new(THREE,FIVE)
if not (a.x == THREE and a.y == FIVE)
  puts "Point is not initialized properly"
end
if not (a.eval_prog([]) == a)
        puts "Point eval_prog should return self"
end
if not (a.preprocess_prog == a)
        puts "Point preprocess_prog should return self"
```

```
end
a1 = a.shift(THREE,FIVE)
if not (a1.x == SIX and a1.y == TEN)
        puts "Point shift not working properly"
end

a2 = a.intersect(Point.new(THREE,FIVE))
if not (a2.x == THREE and a2.y == FIVE)
        puts "Point intersect1 not working properly"
end
a3 = a.intersect(Point.new(FOUR,FIVE))
if not (a3.is_a? NoPoints)
        puts "Point intersect2 not working properly"
end

#Line Tests
b = Line.new(THREE,FIVE)
if not (b.m == THREE and b.b == FIVE)
        puts "Line not initialized properly"
end
if not (b.eval_prog([]) == b)
        puts "Line eval_prog should return self"
end
if not (b.preprocess_prog == b)
        puts "Line preprocess_prog should return self"
end

b1 = b.shift(THREE,FIVE)
if not (b1.m == THREE and b1.b == ONE)
        puts "Line shift not working properly"
end

b2 = b.intersect(Line.new(THREE,FIVE))
if not (((b2.is_a? Line)) and b2.m == THREE and b2.b == FIVE)
        puts "Line intersect not working properly"
end
b3 = b.intersect(Line.new(THREE,FOUR))
if not ((b3.is_a? NoPoints))
        puts "Line intersect not working properly"
end

#VerticalLine Tests
c = VerticalLine.new(THREE)
if not (c.x == THREE)
        puts "VerticalLine not initialized properly"
end

if not (c.eval_prog([]) == c)
        puts "VerticalLine eval_prog should return self"
end
if not (c.preprocess_prog == c)
        puts "VerticalLine preprocess_prog should return self"
end
c1 = c.shift(THREE,FIVE)
if not (c1.x == SIX)
        puts "VerticalLine shift not working properly"
end
c2 = c.intersect(VerticalLine.new(THREE))
if not ((c2.is_a? VerticalLine) and c2.x == THREE )
        puts "VerticalLine intersect not working properly"
end
c3 = c.intersect(VerticalLine.new(FOUR))
if not ((c3.is_a? NoPoints))
        puts "VerticalLine intersect not working properly"
end

#LineSegment Tests
d = LineSegment.new(ONE,TWO,-THREE,-FOUR)
if not (d.eval_prog([]) == d)
        puts "LineSegement eval_prog should return self"
end
d1 = LineSegment.new(ONE,TWO,ONE,TWO)
d2 = d1.preprocess_prog
if not ((d2.is_a? Point)and d2.x == ONE and d2.y == TWO)
```

```
        puts "LineSegment preprocess_prog should convert to a Point"
        puts "if ends of segment are real_close"
end

d = d.preprocess_prog
if not (d.x1 == -THREE and d.y1 == -FOUR and d.x2 == ONE and d.y2 == TWO)
        puts "LineSegment preprocess_prog should make x1 and y1"
        puts "on the left of x2 and y2"
end

d3 = d.shift(THREE,FIVE)
if not (d3.x1 == ZERO and d3.y1 == ONE and d3.x2 == FOUR and d3.y2 == SEVEN)
        puts "LineSegment shift not working properly"
end

d4 = d.intersect(LineSegment.new(-THREE,-FOUR,ONE,TWO))
if not (((d4.is_a? LineSegment)) and d4.x1 == -THREE and d4.y1 == -FOUR and d4.x2 == ONE and d4.y2 == TWO)
        puts "LineSegment intersect1 not working properly"
end

d5 = d.intersect(LineSegment.new(TWO,THREE,FOUR,FIVE))
if not ((d5.is_a? NoPoints))
        puts "LineSegment intersect2 not working properly"
end

#Intersect Tests
i = Intersect.new(LineSegment.new(-ONE,-TWO,THREE,FOUR), LineSegment.new(THREE,FOUR,-ONE,-TWO))
i1 = i.preprocess_prog.eval_prog([])
if not (i1.x1 == -ONE and i1.y1 == -TWO and i1.x2 == THREE and i1.y2 == FOUR)
        puts "Intersect eval_prog should return the intersect between e1 and e2"
end

#Var Tests
v = Var.new("a")
v1 = v.eval_prog([["a", Point.new(THREE,FIVE)]])
if not ((v1.is_a? Point) and v1.x == THREE and v1.y == FIVE)
        puts "Var eval_prog is not working properly"
end
if not (v1.preprocess_prog == v1)
        puts "Var preprocess_prog should return self"
end

#Let Tests
l = Let.new("a", LineSegment.new(-ONE,-TWO,THREE,FOUR),
             Intersect.new(Var.new("a"),LineSegment.new(THREE,FOUR,-ONE,-TWO)))
l1 = l.preprocess_prog.eval_prog([])
if not (l1.x1 == -ONE and l1.y1 == -TWO and l1.x2 == THREE and l1.y2 == FOUR)
        puts "Let eval_prog should evaluate e2 after adding [s, e1] to the environment"
end

#Let Variable Shadowing Test
l2 = Let.new("a", LineSegment.new(-ONE, -TWO, THREE, FOUR),
              Let.new("b", LineSegment.new(THREE,FOUR,-ONE,-TWO), Intersect.new(Var.new("a"),Var.new("b"))))
l2 = l2.preprocess_prog.eval_prog([["a",Point.new(0,0)]])
if not (l2.x1 == -ONE and l2.y1 == -TWO and l2.x2 == THREE and l2.y2 == FOUR)
        puts "Let eval_prog should evaluate e2 after adding [s, e1] to the environment"
end


#Shift Tests
s = Shift.new(THREE,FIVE,LineSegment.new(-ONE,-TWO,THREE,FOUR))
s1 = s.preprocess_prog.eval_prog([])
if not (s1.x1 == TWO and s1.y1 == THREE and s1.x2 == SIX and s1.y2 == 9)
        puts "Shift should shift e by dx and dy"
end
```

hw7testsprovided.sml

```
(* University of Washington, Programming Languages, Homework 7
   hw7testsprovided.sml *)
(* Will not compile until you implement preprocess and eval_prog *)
```

```
4
5  (* These tests do NOT cover all the various cases, especially for intersection *)
6
7  use "hw7.sml";
8
9  (* Must implement preprocess_prog and Shift before running these tests *)
10
11 fun real_equal(x,y) = Real.compare(x,y) = General.EQUAL;
12
13 (* Preprocess tests *)
14 let
15   val Point(a,b) = preprocess_prog(LineSegment(3.2,4.1,3.2,4.1))
16         val Point(c,d) = Point(3.2,4.1)
17 in
18         if real_equal(a,c) andalso real_equal(b,d)
19         then (print "preprocess converts a LineSegment to a Point successfully\n")
20         else (print "preprocess does not convert a LineSegment to a Point succesfully\n")
21 end;
22
23 let
24         val LineSegment(a,b,c,d) = preprocess_prog (LineSegment(3.2,4.1,~3.2,~4.1))
25         val LineSegment(e,f,g,h) = LineSegment(~3.2,~4.1,3.2,4.1)
26 in
27         if real_equal(a,e) andalso real_equal(b,f) andalso real_equal(c,g) andalso real_equal(d,h)
28         then (print "preprocess flips an improper LineSegment successfully\n")
29         else (print "preprocess does not flip an improper LineSegment successfully\n")
30 end;
31
32 (* eval_prog tests with Shift*)
33 let
34         val Point(a,b) = (eval_prog (preprocess_prog (Shift(3.0, 4.0, Point(4.0,4.0))), []))
35         val Point(c,d) = Point(7.0,8.0)
36 in
37         if real_equal(a,c) andalso real_equal(b,d)
38         then (print "eval_prog with empty environment worked\n")
39         else (print "eval_prog with empty environment is not working properly\n")
40 end;
41
42 (* Using a Var *)
43 let
44         val Point(a,b) = (eval_prog (Shift(3.0,4.0,Var "a"), [("a",Point(4.0,4.0))]))
45         val Point(c,d) = Point(7.0,8.0)
46 in
47         if real_equal(a,c) andalso real_equal(b,d)
48         then (print "eval_prog with 'a' in environment is working properly\n")
49         else (print "eval_prog with 'a' in environment is not working properly\n")
50 end;
51
52
53 (* With Variable Shadowing *)
54 let
55         val Point(a,b) = (eval_prog (Shift(3.0,4.0,Var "a"), [("a",Point(4.0,4.0)),("a",Point(1.0,1.0))]))
56         val Point(c,d) = Point(7.0,8.0)
57 in
58         if real_equal(a,c) andalso real_equal(b,d)
59         then (print "eval_prog with shadowing 'a' in environment is working properly\n")
60         else (print "eval_prog with shadowing 'a' in environment is not working properly\n")
61 end;
```

maxfriedrich42 commented on Sep 30, 2017

Dear zakk0610, could you please delete this gist? You are giving away the solutions to graded assignments in the Coursera "Programming Languages" Course (which is currently offered again in self-paced mode). This makes it very easy to cheat and reduces the learning effect for other learners. Many thanks in advance for your collaboration in making the course more effective for everyone!