Download the accompanying zip file from Blackboard. Solve each problem below using the indicated languages on the cs-parallel server. When you are ready to submit, compress your solution files into a zip file, and upload to Blackboard. Double-check that you have submitted all the files you intended.

1. Write these two related Scheme functions. Do not use "set!" or "set-car!" or "set-cdr!"

   a. (fun f g p id L) applies unary function g to each element of list L, then determines which of those results satisfy predicate p, and then combines all such results using binary associative function f. However, if none of those results satisfy p, then return identity element id. Example:
   (fun + square (lambda (x) (> x 10)) 0 '(2 6 3 5 -3 -4 1 -2)) returns $6^2 + 5^2 + (-4)^2 = 77$.
   b. (make h L), where h is a unary function that takes a list and returns a list, and L is a list of lists. (make h L) applies h to each element of L, and appends together all the resulting lists whose lengths are odd and greater than 4. Example:
   (make (lambda (z) (cons 0 z)) '((1 2)(3 4 5)(6 7 8 9)(10 11 12 13 14)(15 16 17 18 19 20)))
   returns (0 6 7 8 9 0 15 16 17 18 19 20).

2. Write a Scheme function (point x y z) that returns a new function that implements a 3D point ADT and that behaves as shown below. The new function that is returned accepts either one or two parameters: the first denotes a movement direction ('east, 'west, 'north, 'south, 'up, or 'down), and the second denotes an optional distance operand which has default value 1 if not present. Example:

```
(define p1 (point 10 20 30))
(define p2 (point 40 50 60))
(begin (p1 'east) (p1 'north) (p1 'up))          ; returns (11 21 31)
(begin (p2 'west) (p2 'south) (p2 'down))        ; returns (39 49 59)
(begin (p1 'east 2) (p1 'north 3) (p1 'up 4))    ; returns (13 24 35)
(begin (p2 'west 5) (p2 'south 6) (p2 'down 7))  ; returns (34 43 52)
```

3. Write these two related Haskell functions.

   a. (fun f g p id xs) applies unary function g to each element of list xs, then determines which of those results satisfy predicate p, and then combines all such results using binary associative function f. However, if none of those results satisfy p, then return identity element id. Example: fun (+) (^2) (\x -> x>10) 0 [2,6,3,5,-3,-4,1,-2] returns $6^2 + 5^2 + (-4)^2 = 77$.
   b. (make h xs), where h is a unary function that takes a list and returns a list, and xs is a list of lists. (make h xs) applies h to each element of xs, and appends together all the resulting lists whose lengths are odd and greater than 4. Example:
   make (\z -> 0:z) [[1,2],[3,4,5],[6,7,8,9],[10,11,12,13,14],[15,16,17,18,19,20]] returns [0,6,7,8,9,0,15,16,17,18,19,20].

4. Write these two related Haskell functions, which both use this new datatype:
   data List a = Nil | Cons a (List a) deriving (Eq, Ord, Read, Show)

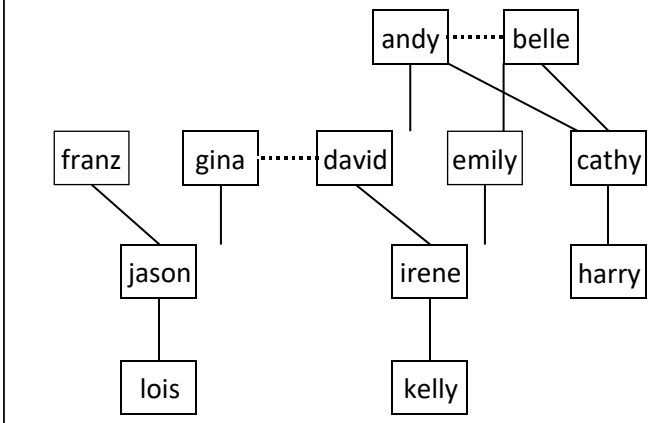
   a. (fun f g p id xs) applies unary function g to each element of List xs, then determines which of those results satisfy predicate p, and then combines all such results using binary associative function f. However, if none of those results satisfy p, then return identity element id. Example: fun (+) (^2) (\x -> x>10) 0
   (Cons 2 (Cons 6 (Cons 3 (Cons 5 (Cons (-3) (Cons (-4) (Cons 1 (Cons (-2) Nil))))))))) returns $6^2 + 5^2 + (-4)^2 = 77$.
   b. (make h xs), where h is a unary function that takes a List and returns a List, and xs is a List of Lists. (make h xs) applies h to each element of xs, and appends together all the resulting Lists whose lengths are odd and greater than 4. Example: If list =
   (Cons (Cons 1 (Cons 2 Nil))
   (Cons (Cons 3 (Cons 4 (Cons 5 Nil)))
   (Cons (Cons 6 (Cons 7 (Cons 8 (Cons 9 Nil))))
   (Cons (Cons 10 (Cons 11 (Cons 12 (Cons 13 (Cons 14 Nil)))))
   (Cons (Cons 15 (Cons 16 (Cons 17 (Cons 18 (Cons 19 (Cons 20 Nil))))))
   Nil))))) then
   (make (\z -> Cons 0 z) list) returns Cons 0 (Cons 6 (Cons 7 (Cons 8 (Cons 9 (Cons 0 (Cons 15 (Cons 16 (Cons 17 (Cons 18 (Cons 19 (Cons 20 Nil)))))))))))).

5. Write these two related Prolog predicates.

   a. fun(F, G, P, Id, L, R) applies G to each element of list L, then determines which of those results satisfy predicate P, and then combines all such results using F. However, if none of those results satisfy P, then return identity element Id. Either way, return the final answer in R. Example: If plus(A,B,C) :- C is A+B, and square(A,B) :- B is A*A, and gt10(A) :- A>10, then fun(plus, square, gt10, 0, [2,6,3,5,-3,-4,1,-2], R) yields R = $6^2 + 5^2 + (-4)^2 = 77$.
   b. make(H, L, R) applies H to each element of L, which is a list of lists, and appends together all the resulting lists whose lengths are odd and greater than 4, and returns the final answer in R. Example: If cons0(Z, [0|Z]), then
   make(cons0, [[1,2],[3,4,5],[6,7,8,9],[10,11,12,13,14],[15,16,17,18,19,20]], R) yields R = [0,6,7,8,9,0,15,16,17,18,19,20].

6. Suppose a Prolog database defines family relations of the form sibling(X, Y), spouse(X, Y), and parent(X, Y), which means that X is a parent of Y. Define the new predicate step_cousin(X, Y). Note: your step-cousin might be your step-parent's nephew or niece, or your uncle or aunt's step-child; another possibility is that you and your step-cousin each have a parent who are step-siblings. Example: Here harry and jason are step-cousins; also kelly and lois are step-cousins.

```
parent(andy,cathy).
parent(belle,cathy).
parent(andy,david).
parent(belle,david).
parent(cathy,harry).
parent(david,irene).
parent(emily,irene).
parent(franz,jason).
parent(gina,jason).
parent(irene,kelly).
parent(jason,lois).
married(andy,belle).
married(david,gina).
spouse(X,Y) :- married(X,Y).
spouse(X,Y) :- married(Y,X).
sibling(X,Y) :-
        parent(Z,X), parent(Z,Y), X\=Y.
```



7. Extend Smalltalk's predefined Collection class by completing these two methods.

   a. (c fun1: f fun2: g fun3: p fun4: id) applies block g to each element of the collection c, then determines which of those results satisfy block p, and then combines all such results using block f. However, if none of those results satisfy p, then return identity element id. Examples: If plus := [:a :b | a + b], square := [:a | a * a], and gt10 := [:a | a > 10], then each of the fun1:fun2:fun3:fun4: message sends below returns $6^2 + 5^2 + (-4)^2 = 77$.
      array := #(2 6 3 5 -3 -4 1 -2).
      (array fun1: plus fun2: square fun3: gt10 fun4: 0) printNl.
      set := Set new add: 2; add: 6; add: 3; add: 5; add: -3; add: -4; add: 1; add: -2; yourself.
      (set fun1: plus fun2: square fun3: gt10 fun4: 0) printNl.
   b. (c make: h), where h is a unary block that takes a collection and returns a collection, and c is a collection whose elements are also collections. (c make: h) applies h to each element of c, and appends together all the resulting collections whose lengths are odd and greater than 4. Example:
      coll := #((1 2) (3 4 5) (6 7 8 9) (10 11 12 13 14) (15 16 17 18 19 20)).
      (coll make: [:z | #(0) , z]) printNl. " returns (0 6 7 8 9 0 15 16 17 18 19 20) "

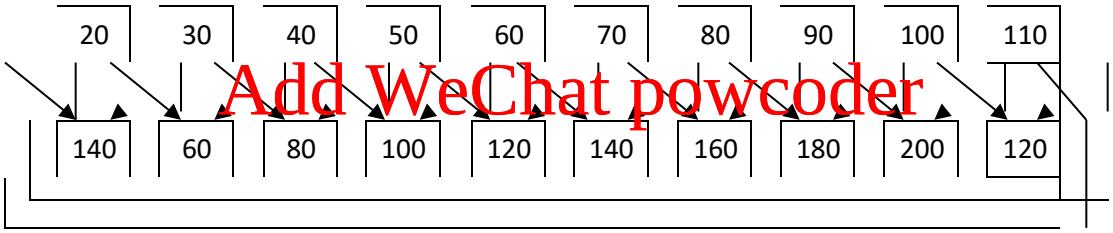
```
Collection extend [
   fun1: f fun2: g fun3: p fun4: id [ ]
   make: h [ ]
]
```

8. Complete all the method bodies for the Smalltalk classes Point1D, Point2D, Point3D. For this problem, no class is permitted to directly access any inherited instance variable. Also, do not add any additional methods. The expected behavior is shown in the client code and output below.

Object subclass: Point1D [
  |x|
  x: a [ ]
  east: a [ ]
  west: a [ ]
  display [ ]
].
	Point1D subclass: Point2D [
	  |y|
	  x: a y: b [ ]
	  north: b [ ]
	  south: b [ ]
	  display [ ]
	].
		Point2D subclass: Point3D [
		  |z|
		  x: a y: b z: c [ ]
		  up: c [ ]
		  down: c [ ]
		  display [ ]
		].

Client code: p := Point3D new x: 40 y: 50 z: 60. p east: 5; west: 3; north: 9; south: 2; up: 8; down: 5. p display.		Output: x=42 y=57 z=63

9. Write this Cilk/Cilk++ function two different ways, each making efficient use of parallelism. Function sum (int *A, int n) takes an array A of size n, and replaces each element of A by the sum of the elements to its immediate left and to its immediate right. Use wrap-around indexing to treat the array as if it were circular. Example:
int A[10] = {20,30,40,50,60,70,80,90,100,110};
sum (A, 10);



Now array A should contain the values {140,60,80,100,120,140,160,180,200,120}.

a. Write the sum function using cilk_for, but do not use cilk_spawn or cilk_sync.
b. Write the sum function using cilk_spawn and cilk_sync, but do not use cilk_for.

10. Determine whether each of the two statements below about the Cilk/Cilk++ language extension is true or false, and provide a convincing explanation for each answer. Type your responses in the provided file problem10.txt in the accompanying zip file, and remember to include this file along with your all your other exam files when you compress into a zip file and submit.

a. Every cilk_for statement that iterates through the elements of an array can be replaced by an equivalent parallel function that uses cilk_spawn and cilk_sync.
b. Every recursive function that uses cilk_spawn and cilk_sync to implement divide-and-conquer on an array can be replaced by one or more equivalent cilk_for statements.