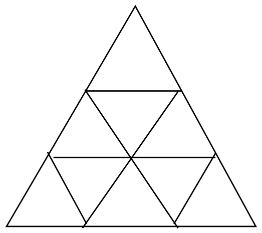
End-Semester Examination

CS 152 – Abstractions and Paradigms in Programming

Date - 25th April 2012 Max. Marks – 75

1. Consider the diagram shown below. Assume all triangles in the diagram are equilateral. The outermost triangle is of height 3. It contains 9 triangles of height 1, of which 3 are inverted. It also contains 3 triangles of height 2 of which none are inverted. Thus a triangle of height 3 contains 9+3+1=13 triangles including itself. If `(triangles h)` is a function which gives the total number of triangles contained in a triangle of height h, then `(triangles 3)` = 13. Similarly, you can verify for yourself that `(triangles 4)` = 27.



   (a) What is `(triangles 5)`?

   (b) Define `triangles`. You get 50% credit if your program can calculate `(triangles 35)` in less than a minute and 100% credit if it can find `(triangles 10000000)` in less than a minute. You cannot use memoization for this part. (2+10 Marks)

2. Assume that we have defined expression trees using the `structs`:

```
(struct node(op ltree rtree) #:transparent)
(struct leaf (num) #:transparent)
```

   Write a function `convert` which will take an expression represented as a tree and converts it into string of parenthesized expression. Assume that the only operators are `*` and `+`. As an example, `convert` applied to a tree on the left gives the string of parenthesized expression on the right. *Note that the parenthesized expression should use minimum number of parenthesis.*

```
       *
     /   \
    +     *        =>    "(5+2)*3*4"
   / \   / \
  5   2 3   4
```

   In other words:

```
(convert (node '* (node '+ (leaf 5) (leaf 2))
                  (node '* (leaf 3) (leaf 4))))
```

   returns `"(5+2)*3*4"` (5 Marks)

3. Consider a general tree defined by the following struct:

```
(struct gnode  (val  lst) #:transparent)
```

   A node of a general tree is at level n if the path from the root to the node has length $n-1$. The root node is at level 1. Write a function `(atlevel t n)` to collect all nodes at a given level n in a tree t. If n is greater than the height of the tree, the `atlevel` returns the null list. (3 Marks)

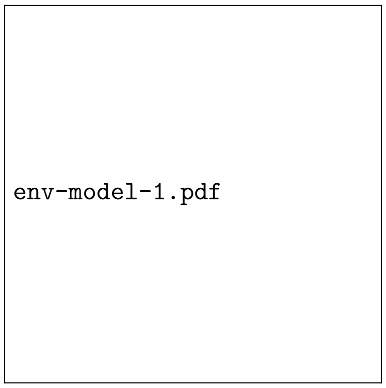
4. Consider the program shown below:

```
(define x 2)
(define (recurse i q)
  (define (p) (display i))
  (if (> i 0) (recurse (- i 1) p)
      (begin  (p) (q))))
(define (dummy) (display ""))
(define (main) (recurse x dummy))
(main)
```

We want to add code to the program so that apart from normal values, it prints the environment on entering each function. For example, just after entering main, it will print:

```
{{main_frame:} .
    #0={{global_frame: {main main #0#}
                       {dummy dummy #0#}
                       {recurse recurse #0#}
                       {x 2}}}}
```

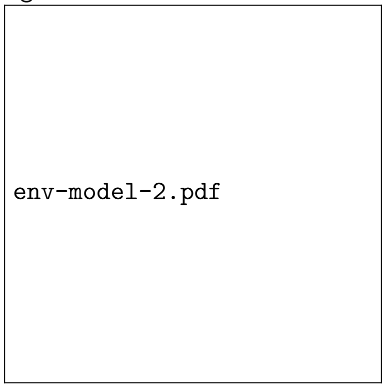
This corresponds to the diagram:



After entering `recurse` and reaching the `if` expression, we want the program to print:

```
#0={{recurse_frame:
     {p p #0#}
     {i 2}
     {q dummy #1={{global_frame:
                   {main main #1#}
                   {dummy dummy #1#}
                   {recurse recurse #1#}
                   {x 2}}}}} .
                             #1#}
```

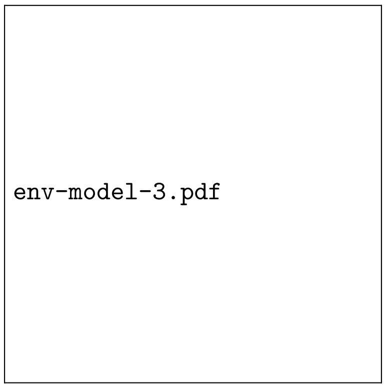
This can be represented by the diagram:



After entering `recurse` for the second time, it will print:

```
#0={{recurse_frame:
     {p p #0#}
     {i 1}
     {q p #1={{recurse_frame:
               {p p #1#}
               {i 2}
               {q dummy #2={{global_frame:
                             {main main #2#}
                             {dummy dummy #2#}
                             {recurse recurse #2#}
                             {x 2}}}}} .
                                       #2#}}} .
                                              #2#}
```

Represented by:



Your problem is to add code to the original program so that it will display the environment after entering each function. The code with your addition should work if I replace `(define x 2)` by `(define x 10)` - it should show many more environments. (25 Marks)

5. Let us design a processor, which, in my honour, will be called as2012. Here is a sample program in the assembly language of as2012.

```
(define sample-prog (list
    (list ’load ’r2   15)   ;Assign to register r2 the number 15
    (list ’load ’r1    5)   ;Assign to register r1 the number 5
    (list ’add  ’r1  ’r2)   ;Add r1 and r2, the result goes to r1
    (list ’load ’r3    2)   ;Assign to register r3 the number 2
    (list ’incr ’r3)        ;Increment contents of register r3 by 1
    (list ’mul  ’r3  ’r2)   ;Multiply r3 and r2, the result goes to r3
    (list ’add  ’r1  ’r3)   ;Add r1 and r3, the result goes to r1
    (list ’store  1  ’r1))) ;Store r1 in memory location 1
```

You could find the effect of running the program by:

```
(send as2012 execute sample-prog)
```

The output shows the registers and the memory after execution of each instruction. Note carefully how they are represented.

To continue the story, you create as2012 from a class called `4r-processor` standing for 4-register processors. Here is a skeleton of how as2012 is created. You have to fill in the missing parts:

```
(define as2012
  (new 4r-processor%
       [mem-size 32]
       [inst-set (list  (list ’add  (lambda (reg1 reg2)
                                      (let ((val (+ (send reg1 read)
                                                    (send reg2 read))))
                                        (send reg1 write val))))
                        (list ’mul  ...)
                        (list ’incr ...))
                        (list ’load (lambda (reg val) ...))
                        (list ’store (lambda (... ... ...) ...))]))
```

We shall now design a class for registers and another for memory. Since they are similar, we shall inherit them from a class called `storable%`. Here is `storable%`.

```
(define storable%
  (class object%
    (super-new)
    (define/public (read) (error "Should be overridden"))
    (define/public (write) (error "Should be overridden"))
    (define/public (print) (error "Should be overridden"))))
```

Now define the classes `register%` and `memory%` by inheriting from `storable%`. [1].

Now all that remains is to define the class `4r-processor%`. Do it by filling the following template:

```
(define 4r-processor%
  (class object%
    (super-new)

    \% Do the initializations here

    (define/public (execute prog)
      (if (null? prog)   "Done"
             ...
             ...
            (execute (cdr prog))))))
```

(15 Marks)

6. On the left bank of a river, there are 3 cannibals and 3 missionaries. There is also a boat on the left bank which can carry at most 2 persons across the river. If any bank has more cannibals than missionaries, then it is a dangerous situation, since the cannibals may eat the missionaries. Needless to say, missionaries do not eat cannibals; they prefer biryani. The problem is to determine how they can all cross the river without anyone getting eaten up. You can attempt any one of the following two versions of the problem:

   (a) Use the fact that there is a solution which involves no more than 11 crossings. (50% Marks)

   (b) Obtain the solution without making any assumptions. (100% Marks).

   In either case you must have a predicate `goal(Ans)` such that at the end of the execution, `Ans` must be bound to a list which looks like `[[3, 3, left], [2, 2, right], [3, 2, left], [3, 0, right], ...]`. Here an element `[M, C, B]` of the list represents the number of missionaries (`M`) and cannibals (`C`) on the left bank and the boat position (`B`). As you can see, in the first crossing, a missionary and a cannibal crossed over from the left bank to the right bank. You can use a built-in predicate called `reverse(X,Y)` which is true if `Y` is the reverse of `X`.

   Note that prolog sometime does not print a list completely but indicates the elements towards the end by dots (...). To force full printing of `L`, use the predicate `write(L)`. (15 Marks)

[1]Yes I know that none of the instruction require to read from memory, but think of the processor as2013 that is going to come out around the same time next year. This may have such an instruction