Assignment 6(Memory Management)

Submission of assignments is optional. Submitted assignments will not be graded. If your solution differs from the posted solution, please submit it and indicate clearly why you think your solution is correct.

1. For the following Algol-like program, write the number printed by executing the program under each of the listed parameter passing mechanisms.

```
begin
   integer i;
   procedure pass ( x, y )
    begin
      integer x, y; // types of the formal parameters begin
      x := x + 1;
      y := x + 1;
      x := y;
      i := i + 1
    end;

   i := 1;
   pass (i, i);
   print i
end
```

   (a) pass-by-value
   (b) pass-by-reference
   (c) pass-by-value-result

Note: in pass-by-value-result, also called call-by-value-result and copy-in/copy-out, parameters are passed by value, with an added twist. More specifically, suppose a function f with a pass- by-value-result parameter u is called with actual parameter v. The activation record for f will contain a location for formal parameter u that is initialized to the R-value of v. Within the body of f, the identifier u is treated as an assignable variable. On return from the call to f, the actual parameter v is assigned the R-value of u.

The following pseudo-Algol code illustrates the main properties of pass-by-value-result.

```
var x : integer;
x := 0;
procedure p(value-result y : integer)
  begin
    y := 1;
    x := 0;
  end;

p(x);
```

With pass-by-value-result, the final value of x will be 1: since y is given a new location distinct from x, the assignment to x does not change the local value of y. When the function returns, the value of y is assigned to the actual parameter x. If the parameter were passed by reference, then x and y would be aliases and the assignment to x would change the value of y to 0. If the parameter were passed by value, the assignment to y in the body of p would not change the global variable x and the final value of x would also be 0.

2.
```
let val x = 5
 in let fun f y = (x+y) -1;
        fun g h = let val x = 7 in h x end
     in
        let val x = 10 in g f end
    end
end ;
```

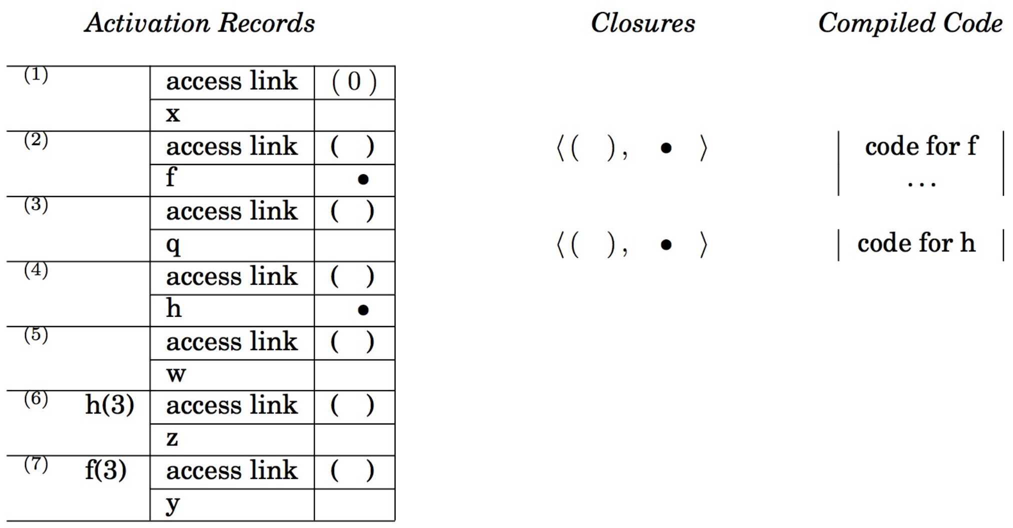
   (a) What is the value of g(f) in the code example?
   (b) The call g(f) in the code example causes the expression (x+y)-1 to be evaluated. What are the values of x and y that are used to produce the value you gave in part (a)?
   (c) Explain how the value of y is set in the sequence of calls that occur before (x+y)-1 is evaluated.
   (d) Explain why x has the value you gave in part (b) when (x+y)-1 is evaluated.

3. Consider the following piece of ML code:

```
let val x = 1
 in let fun f y = y + x;
   in  let val q = 2
        in let fun h z = (f z ) * q;
           in let val w = h 3 in w
               end
           end
        end
   end
end
```

(a) What is the value of w?

(b) Fill in the missing parts in the following diagram of the run-time structures for execution of this code up to the point where the call inside h(3) is about to return. The drawing uses Closures, however, for this exercise a function can point directly to the compiled code. You do not need to make use of Closures. The activation records are numbered 1 − 7, from the top.



4. Consider the following ML program:

```
fun foo x =  let fun bar f = fn x =>  f ( f x);
             in   bar (fn y => y + x)
             end ;
```

(a) what is the value of foo(3)(2) according to the standard (statically scoped) semantics of ML?

Now, suppose we try to optimize the above function by first inlining the bar function

```
fun foo x = fn x => (fn y => y + x) ((fn y => y + x) x);
```

beta reducing

```
fun foo x = fn x => (fn y => y + x) (x + x);
```

and beta reducing again

```
fun foo x = fn x => x + x + x;
```

(b) What does foo(3)(2) evaluate to using the "optimized" version of foo?

(c) What was the mistake we made in our attempted optimization?

(d) What happens if using the original, unoptimized, definition of foo we evaluate foo(3)(2) but under dynamic instead of static scoping?

5. Ex 7.12 from Mitchell
6. Ex 7.13 from Mitchell
7. Ex 7.14 from Mitchell
8. Ex 7.15 from Mitchel