CS 571

HOMEWORK 2

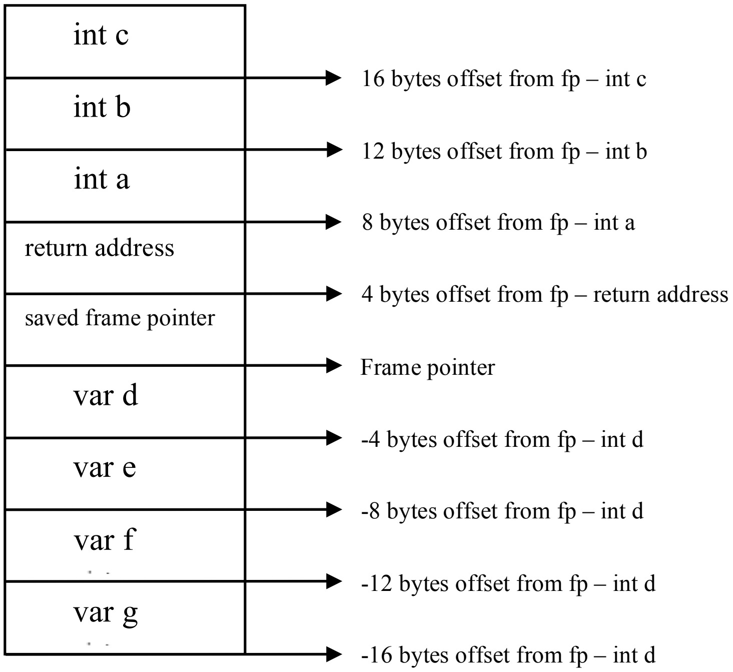
LOVELIN ANAND EDWARD PAUL

B00669954

1.

a) Formula in terms of n and m for the size of a stack frame : 4(m + n + 2)

b)


int c
int b
int a
return address
saved frame pointer
var d
var e
var f
var g
16 bytes offset from fp – int c
12 bytes offset from fp – int b
8 bytes offset from fp – int a
4 bytes offset from fp – return address
Frame pointer
-4 bytes offset from fp – int d
-8 bytes offset from fp – int d
-12 bytes offset from fp – int d
-16 bytes offset from fp – int d

2.

```
f(a, b) {                                   //1
        var x = ...;                        //2
        g(a, x) {                           //3
                var x = ...;                //4
                h(b) {                      //5
                        var a = ...;        //6
                        return a + b * x;   //refs to a, b, x.
                                            a -> 6
                                            b -> 5
                                            x -> 4

                }
                return b + h(a)*x;          //refs to a, b, x.
                                            a -> 3
                                            b -> 1
                                            x -> 4

        }
        return a*b + x;                     //refs to a, b, x.
                                            a-> 1
                                            b->1
```

```
                                                    x->2
      }
```

3.

```
lambda(x){
        return (x + static1 * static2 +b);
}
```

x => 5
static1 =>10
static2 =>8
b => 3

Output => 88

4.

n10 => '(e ()) => '(e)
n9 => '(n9 ()) => '(e ()) => '((e))
n8 => '(d n9) => '(d (e))
n7 => '(n8 ()) => '(d (e) ()) => '(d (e))
n6 => '(n7 ()) => '((d (e)) ()) => '((d (e)))
n5 => '(() n6) => '(() ((d (e))))
n4 => '(c n5) => '(c (() ((d (e))))) => '(c () ((d (e))))
n3 => '(b ()) => '(b)
n2 => '(a n3) => '(a b)
n1 => '(n2 n4) =< '((a b) c () ((d (e))))

Hence it is proved that,
n1 = ' ((a b) c () ((d (e))))

5.

```
(define (count-non-pairs ls)
        (if (not (pair? ls))
                1
                ( +  (count-non-pairs (car ls))
                        (count-non-pairs (cdr ls)))))
```

- For this above function, depending on the input, it works recursively.
- The function will be called recursively only when the given input is a pair.
- It will terminate when the input is not a pair.
- Since every list can be considered as pair, except empty list, the function will be executed recursively.
- If the list is a proper list, the list will be terminated by an empty list and if the list is an improper list, the list will be terminated by a pair.

- So when the list is a proper list, the function will be recursively executed until the list becomes an empty list. Since empty list is not a pair, the function will get terminated.
- When the list is an improper list, the function will be recursively called until the list is terminated by a pair. A pair will be consisting of two non pair items which will terminate the function.

  So, it is shown that count-non-pairs function will always terminate.

6.

In Scheme, there is a feature named delayed evaluation, which can delay the execution of a function. Whenever delay primitive is used over a function, a promise will be returned. This promise can be executed whenever needed using the primitive force.

```
( delay (+ 8 9)                    => #<promise>
(let ((delayed (delay (+ 8 9))))   => 17
        (force delayed))
```

So to generate an infinite list, every tail of a list can be delayed to return a promise.

```
(define inf-list ( cons ( init ( delay ( next init)))))
```

The accessor function can be modified to execute the promise in the tail of the list.

```
(define inf-car ( car inf-list))
(define inf-cdr ( force (cdr inf-list)))
```

7.

a) Valid - Scopes into which names from external scopes must be explicitly imported are called closes scopes. In modules, names have to be explicitly called from other scopes.

b) Invalid - Ex. In Java class, private variables are not in scope but they do exist.

c) Invalid – The language must support first class functions to support without destructive assignment.

d) Invalid – Scheme does support destructive assignment using (set! var exp).

e) Invalid – Every list is a pair, except empty list. So (list? ‘()) will be true and (pair? ‘()) will be false.