1. Show how the meaning of each of the following expressions and given states are derived from the semantic rules given in 8.2.3

(a) M((x+2) * y, {<x,2>,<y,-3>,<z,75>})

(b) M(2 * x+3/y-4), {<x,2>,<y,-3>,<z,75>})

(c) M(1, {<x,2>,<y,-3>,<z,75>})

M: Expression x State → Value

a - M((x+2) * y, {<x,2>,<y,-3>,<z,75>}) = -4

* + x 2 y

Binary expression MR 8.7:3 → 8.8

M((x+2) * y, {<x,2>,<y,-3>,<z,75>}) = Apply Binary (+, A, B)

where A = M(x, {<x,2>,<y,-3>,<z,75>})

B = M(2 * y, {<x,2>,<y,-3>,<z,75>})

The value of A is the value of x in the state A = 2

And B = M(2*y, {<x,2>,<y,-3>,<z,75>})

Binary Expression MR 8.7:3 → 8.8

B. ApplyBinary (*, C, D) where C = M(2, {<x,2>,<y,-3>,<z,75>})

D = M(y, {<x,2>,<y,-3>,<z,75>})

B = C = 2, D = -3

⇒ M(2*y, {<x,2>,<y,-3>,<z,75>}) = Apply Binary (*, 2, -3)

= -6

⇒ M(x+2 * y, {<x,2>,<y,-3>,<z,75>}) = Apply Binary(+, 2, -6)

= -4

---

b - M(2* x+3/y - 4, {<x,2>,<y,-3>,<z,75>})

Binary Expression MR 8.7:3 → 8.8

ApplyBinary (-, A, B) A = M(2 * x+3/y, {<x,2>,<y,-3>,<z,75>})

= (-, 3, 4) = -1 B = M(4, {<x,2>,<y,-3>,<z,75>})

A. ApplyBinary (+, C, D) C = M(2*x, {<x,2>,<y,-3>,<z,75>})

= (+, 4, -1) = 3 D = M(3/y, {<x,2>,<y,-3>,<z,75>})

C. ApplyBinary (*, E, F) E = M(2, {<x,2>,<y,-3>,<z,75>})

F = M(x, {<x,2>,<y,-3>,<z,75>})

= (*, 2, 2) = 4 F = M(2, {<x,2>,<y,-3>,<z,75>})

D. Apply Binary (/, G, H) G = M(3, {<x,2>,<y,-3>,<z,75>})

H = M(y, {<x,2>,<y,-3>,<z,75>})

H = M(-3, {<x,2>,<y,-3>,<z,75>})

= (/, 3, -3) = -1 -1

c - M(1, {<x,2>,<y,-3>,<z,75>}) = 1

Given meaning rule 8.7:1

2. Give the stack activation record generated by the call gcd(24,10) and subsequent recursive calls to gcd. Include in each stack activation record the return value as well as the argument values for its call.

```
int rem(int x, int y){
   return x-x/y*y; }

int gcd (int x, int y) {
   if (y == 0) return x;
   else return gcd (y, rem(x,y)); }

int main() {
   int ans;
   ans = gcd (24, 10)
```

main | ans undef

call gcd | x 24, y 10, gcd undef

call rem | x 24, y 10, rem undef

return rem | x 24, y 10, rem 4

call gcd | x 10, y 4, gcd undef

call rem | x 10, y 4, rem undef

return rem | x 10, y 4, rem 2

call gcd | x 4, y 2, gcd undef

call rem | x 4, y 2, rem undef

return rem | x 4, y 2, rem ∅

call gcd | x 2, y ∅, gcd undef

return gcd | x 2, y ∅, gcd 2

return gcd | x 4, y 2, gcd 2

return gcd | x 10, y 4, gcd 2

return gcd | x 24, y 10, gcd 2

main | ans 2

3. Can the Meaning rule 10.1 be revised so the functions with sideeffects are disallowed? Explain why or why not

Yes, rule 10.1 could be altered to disallow functions with side effects. Altering the rules would be a matter of forcibly redirecting a call's activation record. Some kind of error designated by the type checker that checked the stack activation record for any unsuspecting global variables who's states were altered by a function call.

4. What are the trade-offs (in time and space) when allocation of dynamic arrays occurs on the runtime stack rather than the heap?

As it relates to garbage collection within the heap, considering the the number of active heap blocks and the ratio of R to the heap size n-h, efficiency is greater following copy collection than mark sweep when (n-h)/2. Mark sweep is notably slower but has no overhead. Its active when the heap becomes full. If you allocate this memory dynamically, possible memory leaks occur, because the data is pushed on the stack. Manual deallocation becomes necessary.

5. Write λ-calculus terms to implement the following over ℕ:

(a) equals: returns true when two natural numbers are equal

(b) lt: returns true when two natural numbers are in the less-than relation

(c) Write a definition of subtract:

sub m n {
  if (m<=n) then n-m
  else m-n

ex. add λn. λm.(λf.(λx. n f (m f x)))

let True = λa.λb.a succ λx.λs.λz.s(x s z)
False = λa.λb.b id (λa.a)
zero = λs.λz. z.
one = λs.λz. s(z)
two = λs.λz. s(s(z))
three = λs.λz. s(s(z))

IF = λp.λa.λb.pab
Eq = λx.λy.IF x=y then True else False 1 1

λx.λy.(λp.λa.λb

sub = λm.λn. n PRED m
pred = λn.λf.λx.n(λg.λh. h(g f)) (λu.x) (λu.u)

sub 1 2
(λm.λn.n PRED m) 1 2
(λm. λn.n PRED m) (λs.λz.s(z)) (λs.λz.s(s(z)))
λs.λz. 2 (λf.λg.g (f s)) (λg.z) (λa.a)
λs.λz.(λi0.(λf.λg.g [f s])([λf.λg.g (f s)] i0)) (λg.z) (λa.a)
λs.λz.(λf.λg.g (f s)) ((λf.λg. g [f s]) (λg.z)) (λa.a)
λs.λz.(λg.g([λf.λi0.i0(f s)][λi0.z]s)) (λa.a)
λs.λz. (λa.a) ((λf.λg.g [f s]) (λg.z)s)
λs.λz.(λf.λg.g (f s)) (λg.z) s
λs.λz. (λg.g([λi0.z]s)) s
λs.λz.s ((λg.z) s) one.

LEQ λm.λn. ISZERO (sub m n) 1 2 ISZERO λn.n (λ FALSE) TRUE

λs.λz. one (λf.λg.g [f s]) (λg.z) (λa.a) (λf.λg.g(f [λx.false])) (λg.True) (λa
λz. one (λf.λg.g [f(λi0. λi one. i one (i0 {λx. false}))]) (λg.z) (λa.a) (λg.true)
one (λf.λg.g (f [λi0. λi one. i one (i0 (λx. false))]) z) (λg.λi0.true) (λa.a) (λa.a)
(λz.(λf.λg. g [f (λi0.λi one. i one (i0 {λx. false}))]) z) (λg.λi0.true) (λa.a) (λa.a)
(λf.λg.g (f [λi0.λi one. i one (i0 (λx. false))])) (λg.λi0.true) (λa.a) (λa.a)
(λg.g([λi0.λi one. true] [λf.λi0.i0 (f (λx.false))])) (λa.a) (λa.a)
(λa.a) ((λg.λi0.true) (λf.λg g [f (λx. false)])) (λa.a)
(λg.λi0. true) (λf.λg.g(f[λx. false])) (λa.a)
(λg. true) (λa.a)
true