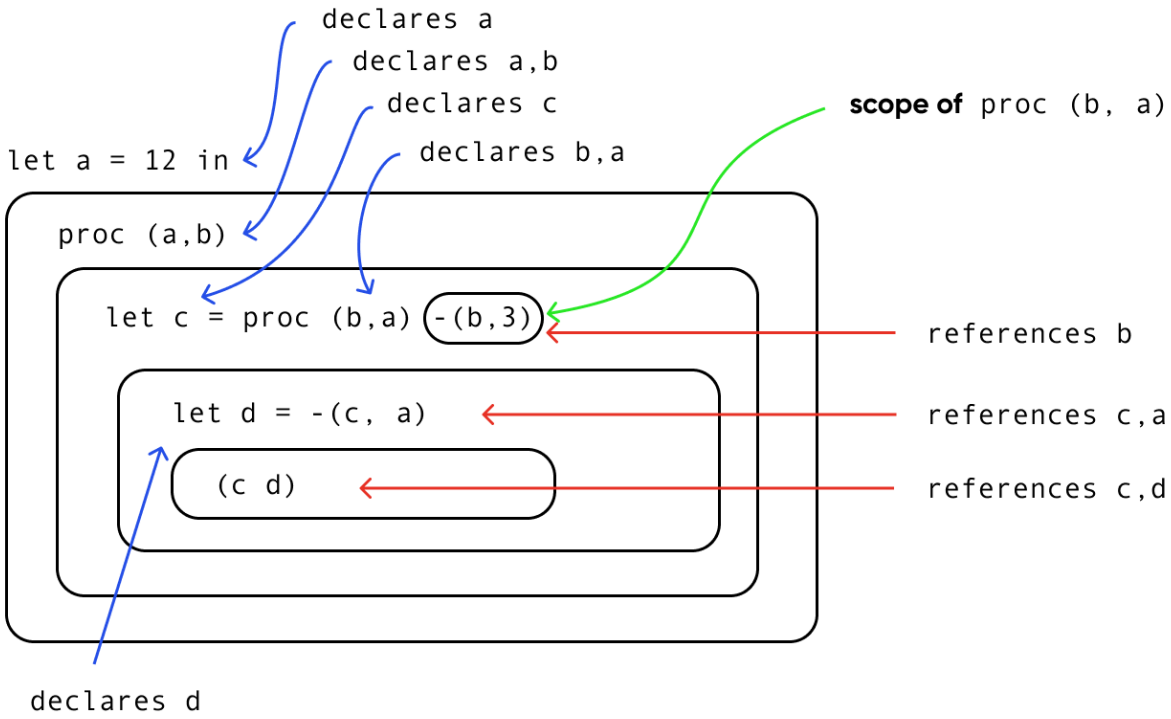
Problem 1A.



Problem 2B.

```
(value-of <<let z = 4 in
                letrec f(x) = if zero?(x) then 1 else (f -(x, 1)) in
                     (f z)>> ρ0)

= (value-of <<letrec f(x) = if zero?(x) then 1 else (f -(x, 1)) in
                     (f z)>>
          (extend-env-rec z (value-of <<4>>) ρ0))


Let ρ1 = (extend-env-rec z (value-of <<4>>) ρ0)
       = [z=[4]]ρ0


= (value-of <<(f z)>>
         (extend-env-rec f << if zero?(x) then 1 else (f -(x, 1))>> ρ1))


Let ρ2 = (extend-env-rec f << if zero?(x) then 1 else (f -(x, 1))>> ρ1)


= (apply-procedure (value-of <<f>>) (value-of <<z>>) ρ2)

= (apply-procedure (procedure x <<if zero?(x) …>> ρ2) [4] ρ2)

= (value-of <<if zero?(x) …>> [x=[4]]ρ2)

… = (value-of <<(f -(x, 1))>> [x=[4]]ρ2)
```

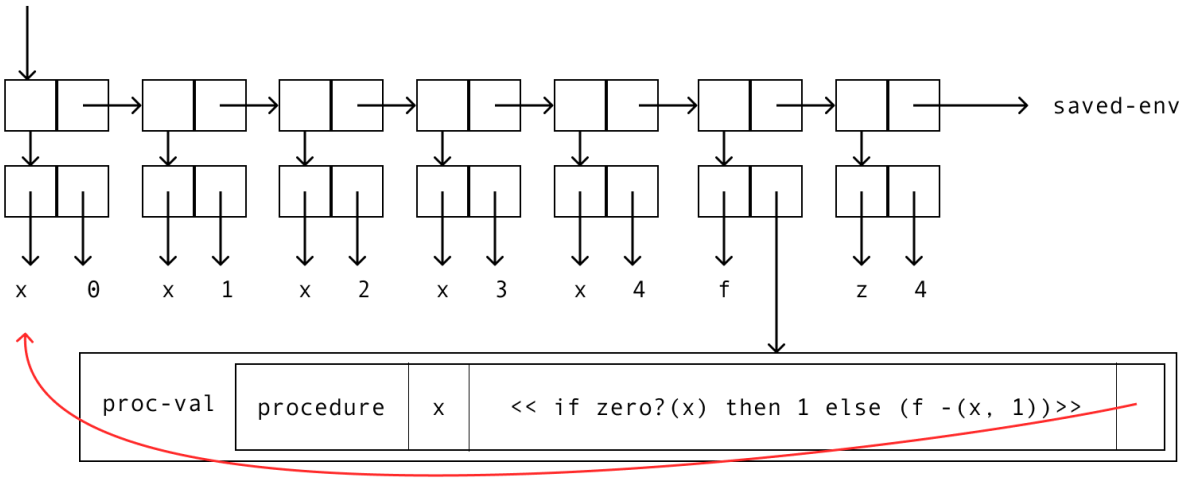
```
… = (apply-procedure (procedure x <<if zero?(x) …>> [x=[4]]ρ2)
                     [3] [x=[4]]ρ2)

… = (value-of <<(f -(x, 1))>> [x=[3]][x=[4]]ρ2)

… = (value-of <<(f -(x, 1))>> [x=[2]][x=[3]][x=[4]]ρ2)

… = (value-of <<(f -(x, 1))>> [x=[1]][x=[2]][x=[3]][x=[4]]ρ2)

… = (value-of <<1>> [x=[0]][x=[1]][x=[2]][x=[3]][x=[4]]ρ2)
```



Problem 2A.

```
%let %nameless-var 6 in
      %let %nameless-var 4 in
            %let %nameless-var -(#2,#1) in
                  %let %nameless-var -(#2,#1) in
                        %nameless-var %lexproc %lexproc %lexproc -(#2,-(#1,#0))
                              in (((#0 #1) #3) #2)
```

Problem 2B.

```
let a = 13 in
      let b = 7 in
            let c = 12 in
                  let f = proc (x) -(b,x) in
                        let d = 5 in
                              proc (x) -(x,-(b,c))
```

Problem 3A.

Main function for `letrec-m`

```
(define apply-extend-env-rec*
  (lambda (env p-names b-vars bodies saved-env search-var)
    (let ((procs (member search-var
                         (map list p-names b-vars bodies)
                         (lambda (key proc) (eqv? key (car proc))))))
      (if procs
          (proc-val (procedure (cadar procs) (caddar procs) env))
          (apply-env saved-env search-var)))))
```

Problem 3B.

Yes, certainly! But, it depends on the requirements. “letrec” is essentially, “letrec-m” with only one procedure declaration, which means “letrec-m” can easily replace “letrec”. In its current implementation, “letrec-m” extends the environment by declaring all the procedures at once within the same scope, which makes it possible to have mutually recursive procedures.

Depending on the requirement, letrec-m can easily be modified to bind procvals either in parallel, much like scheme’s “let”, or bind sequentially like “let*”.

This is how a parallel binding would look:

```
(let-exp (vars exps body)
              (let ((vals (map (lambda (exp) (value-of exp env)) exps)))
                 (value-of body (append-env vars vals env))))
```

append-env => binds pre-evaluated values to the environment.

This is how a sequential binding would look:

```
(let*-exp (vars exps body)
                (value-of body (fold-env vars exps env)))
```

fold-env => iteratively evaluates and binds values to the environment.

Depending on the requirements, a binding mechanism can be implemented.