```
(define-syntax when
    (syntax-rules ()
        ((when test) test)
        ((when test expr ...)
            (if test (begin expr ...) #f)
        )
    )
)
```

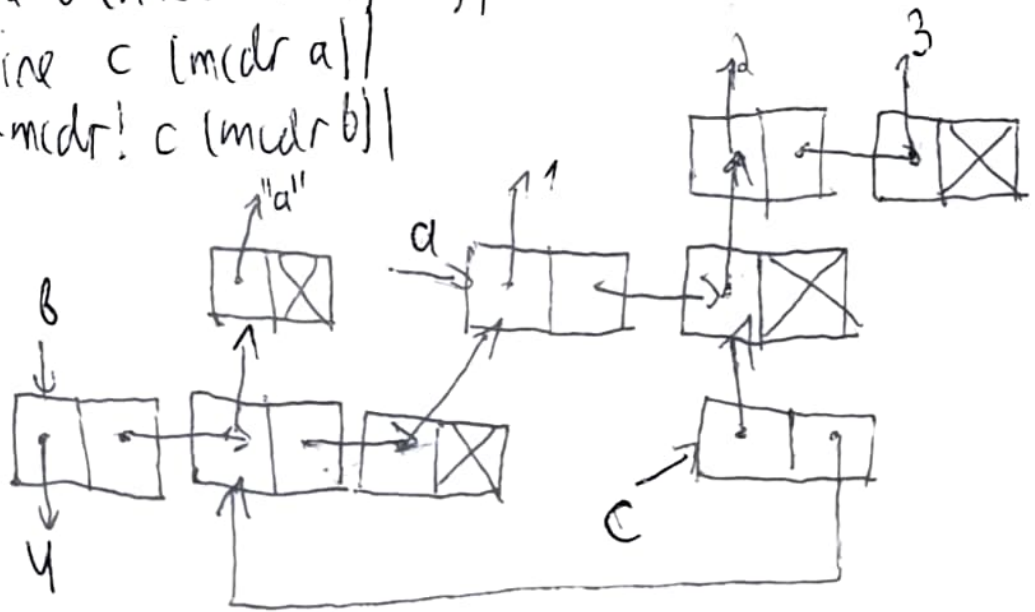
```
(define (filter1 f lst)
  (reverse (foldl (lambda (x y) ( if (f x) (cons x y) y )) null lst)))

(define (filter2 f lst)
  (foldr (lambda (x y) (if (f x) (append (list x) y) y)) null lst))
```

(λz. (λx. ((λy. (x z)) ((λy. y y) (λy. y y z))))) a b
(λz. (λx. ((λy. (x z)) ((λt. t t) (λk. k k z))))) a b
(λx. ((λy. (x a)) ((λt. t t) (λk. k k a)))) b
((λy. (b a)) ((λt. t t) (λk. k k a)))
вместо y подставляется ((λt. t t) (λk. k k a))
но в (b a) нету y, поэтому ответом будет просто (b a)
Ответ: (b a) - нормальная форма

```
(define (nthbit n) (
    let loop ((x 1)) ( if (= x n) 1 (
        if (> x n) 0 ( loop (* x 3))
    ))
))
```

5)

```
(define a (mlist 1 (mcons 2 (mlist 3))))
(define b (mlist 4 'a a))
(define c (mcdr a))
(set-mcdr! c (mcdr b))
```



```
(define (merge-streams s1 s2)
  (stream-match s1
    [('() s2)]
    ((else (stream-match s2
             [('() s1)]
             ((else (let ((a (stream-first s1))
                          (b (stream-first s2)))
                      (if (< a b)
                          (stream-cons a (merge-streams (stream-rest s1) s2))
                          (stream-cons b (merge-streams s1 (stream-rest s2))))))))))))
))

(define (scale-stream stream factor)
  (stream-map (lambda (x) (* x factor)) stream)
)

(define stream-of-3s (stream-cons 1 (scale-stream stream-of-3s 3)))
(define stream-of-5s (stream-cons 1 (scale-stream stream-of-5s 5)))

(define (merge-powers m-stream n-stream)
  (if (stream-empty? m-stream)
      n-stream
      (merge-streams (scale-stream n-stream (stream-first m-stream))
                     (merge-powers (stream-rest m-stream) n-stream))))

(define stream3^m5^n (merge-powers stream-of-3s stream-of-5s))
```