

Continuation Passing Style

NAME: _____

March 28, 2006

Translate the following program into tail form.

```
(define product
  (lambda (lst)
    (cond
      ((null? lst) 1)
      (else (* (car lst)
                (product (cdr lst))))
    )
  )
)
```

You should define a new function `product-c` in tail form, so that the `product` function above can be redefined as

```
(define product
  (lambda (lst)
    (product-c lst (lambda (result) result))
  )
)
```

Model answer:

```
(define product-c
  (lambda (lst c)
    (cond
      ((null? lst) (c 1))
      ((zero? (car lst)) 0) ;; NOTE THIS NEW LINE throws away c
      (else (product-c (cdr lst) (lambda (result)
                                   (c (* (car lst) result))))))
    )
  )
)
```

The new line is a rudimentary form of exception handling — `product-c` is throwing an exception, caught by `product`. The corresponding Java pseudo-code might look like:

```
int product(IntList lst) {
    int result;
    try
        result = product-c(lst, IdentityFunc);
    catch (ZeroException zeroE)
        result = 0;
    return result;
}

int product-c(IntList lst, Continuation c) throws ZeroException {
    int result;
    if (lst == NULL)
        result = c(1);
    else if (lst.car == 0)
        throw new ZeroException();
    else
        result = product-c(lst.cdr, MultiplyFunc);
    return result;
}
```