

CS 101 Computer Science I (Spring 2001)

Assignment 8

1. Write a procedure called **"switch"** that takes a predicate **selector?** and two procedures **fn1** and **fn2**, as arguments, and returns a procedure (**switch selector? fn1 fn2**) as a result. When the procedure (**switch selector? fn1 fn2**) is applied to an argument **arg**, it returns (**fn1 arg**) whenever the value of (**selector? arg**) is **#t**. Otherwise, it returns the value of (**fn2 arg**).

```
(define increment (lambda (x) (+ x 1)))
(define identity (lambda (x) x))
(define make-even (switch even? identity increment))
(make-even 3)    ==> 4
(make-even 4)    ==> 4
```

2. Write a procedure called **"maximize"** that takes a function **fn** as an argument and returns a procedure (**maximize fn**) as a result. The procedure (**maximize fn**) takes a non-empty list **lst** as argument and returns a largest value of (**fn n**) for any member **n** of **lst**.

```
(define square (lambda (x) (* x x)))
((maximize square) '(2 4 -6)) ==> 36
((maximize identity) '(2 4 -6)) ==> 4
```

3. Write a procedure called **"greater-results"** that takes a procedure **fn1** and two integers **low** and **high** as arguments, and returns a procedure (**greater-results fn1 low high**) as a result. When the procedure (**greater-results fn1 low high**) is applied to a function **fn2**, it returns **#t** if (**fn2 int**) is greater than (**fn1 int**) for each integer **int** from **low** to **high**, including the integers **low** and **high** themselves. Otherwise, the procedure (**greater-results fn1 low high**) returns **#f**. Write the procedure **greater-results** in such a way that the procedure **fn1** is called when (**greater-results fn1 low high**) is evaluated, but **fn1** is not called each time (**greater-results fn1 low high**) is applied to a new argument. Hint: Use the procedure **each-matched-pair?** that you wrote in a previous assignment.

```
(define greater-than-identity? (greater-results identity 1 100))
(greater-than-identity? square) ==> #f
(greater-than-identity? increment) ==> #t
```

4. Write a procedure called **"cross-op"** that takes a procedure **fn** as an argument and returns a procedure (**cross-op fn**) as a result. The procedure **fn** takes two arguments. The procedure (**cross-op fn**) takes two lists **lst1** and **lst2** as arguments. The expression (**cross-op fn lst1 lst2**) evaluates to a list consisting of the results of applying **fn** to all possible combinations of a member of **lst1** and a member of **lst2**. Your solution should use the **dist-op** procedure from the most recent lab.

```
((cross-op +) '(0 1) '(10 15 20)) ==> (10 15 20 11 16 21)
((cross-op *) '(1 2) '(10 15 20)) ==> (10 15 20 20 30 40)
((cross-op list) '(1 2) '(10 20)) ==> ((1 10)(1 20)(2 10)(2 20))
```

Due Dates

- Section 51 (Professor Welty): Monday April 9, 2001
- Section 52 (Professor Ellman): Tuesday April 10, 2001