CS496 – Parameter Passing Methods
Exercise Booklet 7

Exercise 1

Consider the following code. Assume parameters are passed using call-by-reference.

```
let b=3
in let p=proc (x) { proc (y) {
           begin
             set x = 4;
             y
           end
          }}
in ((p b) b)
```

1. What is the result of this expression?
2. Draw a line by line trace of the evaluation of this program. For each line indicate the value of the environment and the store.

Exercise 2

Consider the following code.

```
let a=3
in let p=proc (x) { set a = x }
in let b=a
in (p 2)
```

1. What is the result of this expression?
2. Draw the environment and store extant at the breakpoint for both call-by-value and call-by-reference.

Exercise 3

Indicate the result of executing the following program in call-by-value, call-by-reference, call-by-name and call-by-need.

```
letrec inf (x) = (inf x)
in let f = proc (z) { 11 }
in (f (inf 0))
```

Exercise 4

Write a program to test whether a particular interpreter is using call-by-name or call-by-need.

Exercise 5

What is the difference between a thunk and a closure?

Exercise 6

Consider the following program.

```
let a = 3
in let g = proc(x) {
               proc(y) {
                 if zero?(x) then y else x-y }}
in ((g a) a)
```

1. Depict the environment and store at the breakpoint assuming call-by-reference.
2. Depict the environment and store at the breakpoint assuming call-by-need

Exercise 7

Consider the following program.

```
let a = 3
in let g = proc(x) {
               proc(y) {
                 if zero?(x) then y else x-y  } }
in ((g a) (a-1))
```

1. Depict the environment and store at the breakpoint assuming call-by-reference.
2. Depict the environment and store at the breakpoint assuming call-by-need

Exercise 8

Consider the following program where a box indicates a breakpoint.

```
let a = 3
in let g = proc(x) {
               proc(y) {
                 (if zero?(x)
                 then y
                 else x-y)+3 } }
in ((g (a-1)) a)
```

1. Depict the environment and store at the breakpoint assuming call-by-name.
2. Depict the environment and store at the breakpoint assuming call-by-need

Exercise 9

Consider the following program:

```
let swap = proc (x) {
                proc (y) {
                       let temp = x
                       in begin
                            set x = y;
                            set y = temp
                           end }}
in let a = 33
in let b = 44
in begin
    ((swap a) b);
    a-b
   end
```

1. Write a trace of the evaluation of this program in call-by-value
2. Write a trace of the evaluation of this program in call-by-reference

Exercise 10 (*Ex.4.35 from the book*)

We can get some of the benefits of call-by-reference without leaving the call-by-value framework. Extend the language IMPLICIT-REFS by adding a new expression

```
<Expression> ::= ref <Identifier>
```

Note that references now become expressed values. Thus we also need `deref` and `setref` operations.

The resulting language differs from the language EXPLICIT-REFS however, since references are only of variables. This allows us to write familiar programs such as swap within our call-by-value language. What should be the value of this expression?

```
let a = 3
in let b = 4
   in let swap = proc (x) {
                    proc (y) {
                      let temp = deref(x)
                      in begin
                           setref(x,deref(y));
                           setref(y,temp)
                          end }}
      in begin
           ((swap (ref a)) (ref b));
           a-b
         end
```

What are the expressed and denoted values of this language?