COMP 301 - Fall 2020 - Project 2

Mehmet Enes Erciyes, Mert Gülsün, Adar Bayan

Workload Breakdown

All of the work is done simultaneously by all the team members. We had two meetings to finish the project. We have done these meetings over Zoom and one of us shared screen and we discussed the problems and then implemented the solutions.

Part A

1. Five components of the language are
    a. Syntax and data types
    b. Values
    c. Environment
    d. Behavior specification
    e. Behavior implementation
        i. Scanning
        ii. Parsing
        iii. Evaluation
2. Racket Files
    a. Syntax and data types -> lang.rkt
    b. Values -> data_structures.rkt
    c. Environment -> environments.rkt
    d. Behavior specification -> interp.rkt
    e. Behavior implementation
        i. Scanning -> lang.rkt
        ii. Parsing -> lang.rkt
        iii. Evaluation -> interp.rkt

Part B

Addition of z: [z = 3] p
Addition of y: [y = 2, z = 3] p
Addition of x: [x = 4, y=2, z=3] p

Part C

Expressed Values: Integer + Bool + String
Denoted Values: Integer + Bool + String

Part D - (4)

For this part, we wanted to add a switch-case functionality to our language. Here is the grammar for our $switch-exp$:

$$Expression ::= switch\ (identifier)\ \{case\ Number\ :\ Expression\}^*\ default\ :\ Expression$$
$$switch-exp\ (var\ nums\ exps\ exp1)$$

The behaviour specification for our expression is as follows:

- (var x {Number}* x {Expression}* x Expression) -> Expression

switch-exp takes a variable and depending on its value, evaluates the expression that corresponds to the case for that value.

Behaviour implementation:

```
;; Custom expression -> switch-exp
(expression
 ("switch(" expression ")" (arbno "case" number ":" expression) "default:" expression) switch-exp)
))
```

(Figure 1 - Switch-exp grammar definition)

```
(switch-exp (var nums exps default)
            (let ((value (expval->num (value-of var env))))
              (let ((switch-result (switch-evaluator value nums exps env)))
              (if (car switch-result)
                  (cadr switch-result)
                  (value-of default env)))))
```

(Figure 2 - Switch-exp implementation)

```
(define switch-evaluator
  (lambda (value nums exps env)
    (if (not (null? nums))
        (if (= value (car nums))
            (list #t (value-of (car exps) env))
            (switch-evaluator value (cdr nums) (cdr exps) env))
        (list #f '()))))
```

(Figure 3 - Switch exp evaluator helper function)

Part E

```
(switch-basic "let x=3 in switch(x) case 1: 1 case 2: 2 case 3: 3 default: 'default'" 3)
(switch-with-expressions "let x=1 in switch(x) case 1: op(x,1,4) case 2: zero?(x) case 3: x default: 'default'" 0)
(switch-default-test "switch(4) case 1: op(x,1,4) case 2: zero?(x) case 3: x default: 'default'" "'default'")
(switch-unbound-var-error "switch(foo) case 1: op(x,1,4) case 2: zero?(x) case 3: x default: 'default'" error)
(switch-case-not-number-error "switch(x) case 'x': op(x,1,4) case 'y': zero?(x) case 'z': x default: 'default'" error)
(nested-switch-test "let x=3 in let y=op(x,1,1) in switch(x) case 1: op(x,1,4) case 2: zero?(x) case 3: switch(y) case 4: y default: 'inner-default' default: 'default' " 4)
```

Test cases written for our custom expression are found above, as well as in the *tests.rkt* file. All of the test cases including our custom test cases and others are successfully passed.