CS305 – Programming Languages

Spring 2018-2019

Homework 6

Interpreting Scheme using Scheme

1 Introduction

In this homework you will implement a Scheme interpreter, similar to the ones we saw in the lectures. However, the subset of Scheme that is handled by the interpreter will be larger.

2 The Scheme subset `HW6`

The syntax of the Scheme subset that will be covered by the interpreter that you will implement for this homework is as follows.

Grammar for the subset HW6

```
            <HW6> -> <expr>
                  |  <define>

            <expr> -> NUMBER
                    | IDENT
                    | <if>
                    | <let>
                    | <letstar>
                    | <lambda>
                    | <application>

          <define> -> ( define IDENT <expr> )

              <if> -> ( if <expr> <expr> <expr> )

             <let> -> ( let ( <var_binding_list> ) <expr> )

         <letstar> -> ( let* ( <var_binding_list> ) <expr> )

          <lambda> -> ( lambda ( <formal_list> ) <expr> )

     <application> -> ( <operator> <operand_list> )

        <operator> -> <built_in_operator>
                    |  <lambda>
                    |  IDENT

<built_in_operator> -> + | * | - | /

    <operand_list> -> <expr> <operand_list>
                    |  empty

 <var_binding_list> -> ( IDENT <expr> ) <var_binding_list>
                    |  ( IDENT <expr> )

     <formal_list> -> IDENT <formal_list>
                    |  IDENT
```

Note that, compared to the subsets we handled in the class, this grammar allows much more liberal expressions to be used: we will be able to apply a lambda expression directly within an interaction as

```
((lambda (n) (+ n 2)) 5)
```

and we will also be able bind a lambda expression to a variable as

```
(define inc2 (lambda (n) (+ n 2)))
```

and apply it later as

```
(inc2 5)
```

Note that, if there is a variable in procedure, it checks the value when the procedure is applied. You can see the example for binding variables to some values in Section 3.

We are now also able to use "if" expressions.

```
(if ((lambda (n) (+ n 2)) 2) (+ 1 2) (- 3 5))
```

Since we do not have the boolean data type in our subset, we use "if" expression with the following semantics. When the first `<expr>` of an `<if>` expression evaluates to 0, then the value of the `<if>` expression is to be taken as the value of the third `<expr>` of the `<if>` expression. Otherwise, (i.e. when the first `<expr>` of an `<if>` expression evaluates to a value other than 0), the value of the `<if>` expression is to be taken as the value of the second `<expr>` of the `<if>` expression.

You should pay attention to the different semantics of `let` and `let*`. In the following sequence of expressions

```
(define x 5)
(let ((x 3)(y x)) (+ x y))
(let* ((x 3)(y x)) (+ x y))
```

the `let` expression should produce the value 8, whereas the `let*` expression should produce the value 6.

A note on the number of items in `<operand_list>` in an `<application>`.

- If the `<operator>` is a lambda expression, then the number of items in the `<operand_list>` and the number of formal parameters in the lambda expression must match. If they are different, then an error should be produced.

- If the `<operator>` is the addition operator, then `<operand_list>` can have any number of items. When there are 0 arguments, it should evaluate to 0.
- If the `<operator>` is the multiplication operator, then `<operand_list>` can have any number of items. When there are 0 arguments, it should evaluate to 1.
- If the `<operator>` is the subtraction operator, then `<operand_list>` must have at least two items. This operator is left associative.
- If the `<operator>` is the division operator, then `<operand_list>` must have at least two items. This operator is left associative.

3 The procedure `cs305`

You should declare a procedure named `cs305` which will start the show when called. It should not take any arguments.

In every iteration of your REPL, you should print out the prompt given below in "Scheme Interaction" sample given below, then accept an input from the user, then evaluate the value of the input expression, and finally print the value evaluated by using a value prompt. The following is a sample on how the interaction with your interpreter must look like.

Scheme Interaction

```
1 ]=> (cs305)
cs305> 3
cs305: 3

cs305> (define x 5)
cs305: x

cs305> x
cs305: 5

cs305> ((lambda (n) (+ n 2)) 5)
cs305: 7

cs305> (define inc2 (lambda (n) (+ n 2)))
cs305: inc2

cs305> (inc2 5)
cs305: 7

cs305> (define incx (lambda (n) (+ n x)))
cs305: incx

cs305> (define x 3)
cs305: x

cs305> (incx 1)
cs305: 4

cs305> (define x 1)
cs305: x

cs305> (incx 1)
cs305: 2
```

In the example, `incx` increments the argument `n` by `x`. Since `x` is evaluated when the procedure is called, `x` is not 5. `x` is not bound when the procedure is defined, but when the procedure is applied/called.

4 How to Submit

Submit a single file which must be named as:

`id-hw6.scm`

where `id` is your student id.

In order to test your submission, we will start the MIT Scheme interpreter on `flow.sabanciuniv.edu`, and load your file in the interpreter as follows:

```
1 ]=> (load "id-hw6.scm")
;Loading "id-hw6.scm" -- done
;Value: ....
1 ]=> (cs305)
```

After loading your file, we will start the `cs305` procedure, and start to interact with your interpreter.

5 Notes

- **Important**: SUCourse's clock may be off a couple of minutes. Take this into account to decide when to submit.
- No homework will be accepted if it is not submitted using SUCourse.
- Note that, you may be able to find Scheme interpreters for Windows. Although it is discouraged, you may use them. However, we want to remind you that, your homework will be evaluated on `flow.sabanciuniv.edu`. Hence we recommend that you, at least, test your implementation on this before submitting.
- Start working on the homework immediately.