CS61A NOTE14 Interpreters

Interpreters

Q1: Eval and Apply 个数

How many calls to `calc_eval` and `calc_apply` would it take to evaluate each of the following Calculator expressions?

`scm> (+ 1 2)`

For this particular prompt please list out the inputs to `calc_eval` and `calc_apply`.

eval：1 全体 + 每个操作符操作数， () 里外都计入

apply：操作符 operator

and 不算操作符

纯文本

```
4 calls to eval
 (1 entire expression+3*1 each operand and operator)

1 apply the add operator

Explicity listing out the inputs we have the following for calc_eval: , '+'
calc_eval的调用输入：'+', 1, 2
calc_apply的fn是'+'，args是(1 2)

A note is that (+ 1 2) corresponds to the following Pair, Pair('+', Pair(1,
```

scm> (+ 2 4 6 8)

纯文本

```
6 calls to eval
(1 for the entire expression, 1*5 each operator and operand)

1 call to apply the addition operator
```

scm> (+ 2 (* 4 (– 6 8)))

```
10 calls to eval:
 (1 for the whole expression + 1*9 for each of the operators and operands.括

3 calls to apply the function
```

scm> (and 1 (+ 1 0) 0)

```
7 calls to eval:
(1 for the whole expression +1*6 for each operand and operator)
其中and不算operator，1 for the first argument, 1 for (+ 1 0), 1 for the + ope

1 calls to apply to evaluate the + expression. (and不算operator)
```

scm> (and (+ 1 0) (< 1 0) (/ 1 0))

```
9 calls to eval:
 (1 for whole + 8*1 注意and不计算，且在第二处短路不执行(/ 1 0))

2 calls to apply： + and <.
```

disc&lab12 Q2: From Pair to Calculator 用 Pair 写 skeam

Write out the Calculator expression with proper syntax that corresponds to the following `Pair` constructor calls.Pair 构造函数调用相对应的 Calculator 表达式

```
>>> Pair('+', Pair(1, Pair(2, Pair(3, Pair(4, nil)))))
(+ 1 2 3 4) #与link一样，两个Pair才()

>>> Pair('+', Pair(1, Pair(Pair('*', Pair(2, Pair(3, nil))), nil)))
(+ 1 (* 2 3))

>>> Pair('and', Pair(Pair('<', Pair(1, Pair(0, nil))), Pair(Pair('/', Pair(
(and (< 1 0) (/ 1 0))
```

```
纯文本
Write out the Python expression that returns a `Pair` representing the give

Pair('+', Pair(Pair('-', Pair(2, Pair(4, nil))), Pair(6, Pair(8, nil))))

first operand of the call expression
Pair('-', Pair(2, Pair(4, nil)))
```

retriev 检索

```
纯文本
If the `Pair` you constructed in the previous part was bound to the name `p
how would you retrieve the operator?
p.first
how would you retrieve a list containing all of the operands?
p.rest
How would you retrieve only the first operand?
p.rest.first
```

Calculator Evaluation

```
纯文本
def calc_eval(exp):
    """
    >>> calc_eval(Pair("define", Pair("a", Pair(1, nil))))
    'a'
    >>> calc_eval("a")
    1
    >>> calc_eval(Pair("+", Pair(1, Pair(2, nil))))
    3
    """
    if isinstance(exp, Pair):
        operator = exp.first
        operands = exp.rest
        if operator == 'and': # and expressions
            return eval_and(operands)
        elif operator == 'define': # define expressions
            return eval_define(operands)
        else: # Call expressions
```

```
            return calc_apply(calc_eval(operator), operands.map(calc_eval))
    elif exp in OPERATORS:   # Looking up procedures
        return OPERATORS[exp]
    elif isinstance(exp, int) or isinstance(exp, bool):   # Numbers and boo
        return exp
    elif exp in bindings:   # Looking up variables
        return bindings[exp]
```

lab12 Q3: New Procedure

Suppose we want to add the `//` operation to our Calculator interpreter. Recall from Python that `//` is the floor division operation, so we are looking to add a built-in procedure `//` in our interpreter such that `(// dividend divisor)` returns dividend // divisor. Similarly we handle multiple inputs as illustrated in the following example `(// dividend divisor1 divisor2 divisor3)` evaluates to `(((dividend // divisor1) // divisor2) // divisor3)`. For this problem you can assume you are always given at least 1 divisor.

> *Hint:* You will need to modify both the `calc_eval` and `floor_div` methods for this question!需要同时修改 calc_eval 和 floor_div 方法

> *Hint:* Make sure that every element in a `Pair` (from operator to operands) will be `calc_eval`-uated once, so that we can correctly apply the relevant Python operator to operands! You may find the `map` method of the `Pair` class useful for this.确保 Pair 中的每个元素(从操作数到操作数)都将被 calc_eval-uated 一次，这样我们就可以正确地将相关的 Python 操作符应用到操作数上，map 方法在这方面很有用

纯文本

```
def floor_div(expr):
    """
    >>> floor_div(Pair(100, Pair(10, nil)))
    10
    >>> floor_div(Pair(5, Pair(3, nil)))
    1
    >>> floor_div(Pair(1, Pair(1, nil)))
    1
    >>> floor_div(Pair(5, Pair(2, nil)))
    2
    >>> floor_div(Pair(23, Pair(2, Pair(5, nil))))
```

```
    2
    >>> calc_eval(Pair("//", Pair(4, Pair(2, nil))))
    2
    >>> calc_eval(Pair("//", Pair(100, Pair(2, Pair(2, Pair(2, Pair(2, Pair
    3
    """
    # BEGIN SOLUTION Q3
    divided=expr.first
    expr=expr.rest
    while expr!=nil:
        divisor=expr.first
        divided=divided//divisor
        expr=expr.rest
    return divided
```

lab12 Q4: New Form

Suppose we want to add handling for `and` expressions to our Calculator interpreter as well as introduce the Scheme boolean expressions `#t` and `#f`. These should work the same way they do in Scheme. (The examples below assumes conditional operators (e.g. `<`, `>`, `=`, etc) have already been implemented, but you do not have to worry about them for this question.)

In a typical call expression, we first evaluate the operator, then evaluate the operands, and finally apply the operator to the evaluated operands (just like you did for `floor_div` in the previous question). However, since `and` is a special form that short circuits on the first false-y operand, we cannot handle these expressions the same way we handle regular call expressions. We need to add special handling for combinations that don't evaluate all the operands. Complete the implementation below to handle `and` expressions.

纯文本

```
def eval_and(operands):
    """
    >>> calc_eval(Pair("and", Pair(1, nil)))
    1
    >>> calc_eval(Pair("and", Pair(False, Pair("1", nil))))
    False
    >>> calc_eval(Pair("and", Pair(1, Pair(Pair("//", Pair(5, Pair(2, nil))
    2
    >>> calc_eval(Pair("and", Pair(Pair('+', Pair(1, Pair(1, nil))), Pair(3
```

```
    3
    >>> calc_eval(Pair("and", Pair(Pair('-', Pair(1, Pair(0, nil))), Pair(P
    2.5
    >>> calc_eval(Pair("and", Pair(0, Pair(1, nil))))
    1
    >>> calc_eval(Pair("and", nil))
    True
    """
    # BEGIN SOLUTION Q4
    oper,result=operands,True
    while oper is not nil:  #用is not
        result=calc_eval(oper.first)
        if result is False: #用is
            return False
        oper=oper.rest
    return result
```

lab12 Q5: Saving Values 保存值

In the last few questions we went through a lot of effort to add operations so we can do most arithmetic operations easily. However it's a real shame we can't store these values. So for this question let's implement a `define` special form that saves values to variable names. This should work like variable assignment in Scheme; this means that you should expect inputs of the form `(define <variable_name> <value>)` and these inputs should return the symbol corresponding to the variable name.

纯文本

```
def eval_define(expr):
    """
    >>> eval_define(Pair("a", Pair(1, nil)))
    'a'
    >>> eval_define(Pair("b", Pair(3, nil)))
    'b'
    >>> eval_define(Pair("c", Pair("a", nil)))
    'c'
    >>> calc_eval("c")
    1
    >>> calc_eval(Pair("define", Pair("d", Pair("//", nil))))
```

```
    'd' #用列表来保存名字和存储值
    >>> calc_eval(Pair("d", Pair(4, Pair(2, nil))))
    2
    """
    # BEGIN SOLUTION Q5
    name,save=expr.first,calc_eval(expr.rest.first)
    bindings[name] = save
    return name
```

disc Q7: Longest increasing subsequence

Write the procedure `longest-increasing-subsequence`, which takes in a list `lst` and returns the longest subsequence in which all the terms are increasing. 返回最长的上升列 *Note: the elements do not have to appear consecutively in the original list*. For example, the longest increasing subsequence of `(1 2 3 4 9 3 4 1 10 5)` is `(1 2 3 4 9 10)`. Assume that the longest increasing subsequence is unique.

纯文本

```
; helper function
; returns the values of lst that are bigger than x
; e.g., (larger-values 3 '(1 2 3 4 5 1 2 3 4 5)) --> (4 5 4 5)#选大的出来用fi
(define (larger-values x lst)
    (filter (lambda (y) (> y x) lst))
    #学(filter (lambda (x) (not (= x item))) lst)
)

(define (longest-increasing-subsequence lst)
    (if (null? lst)
        nil
        (begin
            (define first (car lst))
            (define rest (cdr lst))
            (define large-values-rest
                (larger-values first rest))
            (define with-first
                (cons first (longest-increasing-subsequence large-values-re
                #一开始写的下面这行，skeam里list默认link list要用cons等方法建立
                #(+ first longest-increasing-subsequence large-values-rest)
            (define without-first
                (longest-increasing-subsequence rest))
```

```
            (if (> (length with-first) (length without-first))
                with-first
                without-first)
        )
    )
)
```