프로그래밍 언어 개론

- item2.py -

분 반		01	
담당교수		조은선	
학 과		컴퓨터공학과	
조 장	학 번	201102400	김 남 형
조 원	이 름	201302492	최 규 호

1. 문제 조건

변수에 대한 define 문을 처리한다.

- 예를 들어 (define a 1) 이라고 하면, a 의 값이 1 임을 저장하는 심벌테이블을 만든다.
- (define b '(1 2 3)) 등도 마찬가지로 처리한다. 즉, 테이블에 b가 '(1 2 3) 임을 저장한다. insertTable(id, value) 와 같은 함수를 만들어 사용한다.
- (define c (- 5 2)) 와 같은 경우도 가능하다. 이때에는 insertTable에 삽입하기 전에 runExpr을 통해서 (- 5 2) 대신 결과값 3으로 바꾸어 저장한다.
- 결과적으로 quoted 된 리스트나 상수만 테이블에 저장된다고 가정한다.
- Python의 자료구조를 적절히 이용할 것(찾아서 하기)

변수의 값을 사용할 수 있도록 한다.

- 예를 들어 (+ a 3) 은 먼저 a의 값인 1을 꺼내 온다.
  Node lookupTable(String id) 과 같은 함수를 만들어 사용한다.a 대신 1을 사용하므로 결과로 4를 얻는다. (+ a 3)에서 a 대신 loopkupTable()의 결과로 대체한 후 runExpr()의 인자로 넣고 수행하여 결과로 4를 얻는다.
- 같은 방식으로 (car b) 는 1 이 된다.
- 오류가 있는 프로그램은 없다고 가정한다. (즉, 예를 들어 (+ b 1) 등의 input은 없다고 가정한다.)
- 변수는 전역으로 정의된다고 가정한다.
- 같은 변수를 두 번 이상 define 하면 앞에 define 된 것이 없어진다고 가정한다.

함수 호출을 처리한다. (이하는 처리 예임)

- '(' 다음에 나오는 첫번째 원소는 그 다음에 두번째 원소, 즉 인자가 있을 때만 현재와 같이 수행한다. (기존 과제 ~07 과 동일)
  1. (car '(a b c)) ➔ a
  2. (+ 1 2) ➔ 3
- '(' 다음에 나오는 첫번째 원소가 lambda 일 때
  1. 두번째 이후의 원소가 없다면 그대로 return 된다.
     - (lambda (x) (+ x 1)) ➔ (lambda (x) (+ x 1))
  2. 두번째 이후의 원소가 있다면 아래와 같이 수행한다.
     - lambda 키워드 다음에 오는 (x) 의 x에 actual parameter를 바인딩하여 임시로 변수 테이블에 넣어둔다.
       e.g. ((lambda (x) (+ x 1)) 10) 이라면 x에 10을 바인딩
     - lambda (x) 다음에 오는 리스트가 함수의 내용이므로 이것을 expression으로 생각하고 수행한다.
       e.g. (+ x 1)
     - 함수 수행 후에는 임시로 x에 10을 바인딩 한 부분을 테이블에서 제거한다.

함수의 정의도 define 문으로 가능하므로 변수 바인딩과 함께 테이블에 저장한다.

- 예를 들어 plus1이 아래와 같이 정의되어 있다면, plus1이 (lambda …) 임을 테이블에 저장한다.

  ex)

  ```
  (define plus1 (lambda (x) (+ x 1)))
  (plus1 2)
  … 3
  ```

- 이와 같은 방식으로 구현할 수도 있지만, 함수의 인자 x를 따로 뽑아내어 테이블에 분리 저장함으로써 추후 호출 시간을 단축시킬 수 있다. 이러한 경우 별도의 추가점은 없다.
- 변수와 마찬가지로 함수도 전역으로 정의된다. (즉, 중첩 정의 되지 않는다.) **단, 수업 중 배운 Symbol Table 처리 방법 중 하나를 사용해서 처리한다면 추가점이 있다. (보고서에 명시할 것!)**
- 함수 내에 변수 선언(즉, 지역변수 선언)은 없다고 가정한다. **단, 수업 중 배운 Symbol Table 처리 방법 중 하나를 사용해서 처리한다면 추가점이 있다. (보고서에 명시할 것!)**

-

  - Lambda 구현(5점)
  - 전역 함수 호출이 가능(7점)
  - 함수 내에서 전역 함수 호출 가능(5점)
  - 변수 scope 구현(8점) (구현 방법에 대해 보고서에 자세히 쓸 것!)
  - Nested 함수 구현 가능(3점)
  - Recursion 함수 구현 가능(2점)
  - 함수에서 함수를 인자로 사용가능(3점)

2. 솔루션

▷ Define

```
def define(node):
    l_node = node.value.next
    r_node = node.value.next.next

    if r_node.type is TokenType.ID:
        r_node = lookupDefine(r_node.value)
    else:
        r_node = run_expr(r_node)
    insertDefine(l_node.value, r_node)
```

- define을 Table에 추가해 놓고 def define(node)함수를 선언해서 define이 조건에 걸리면 define(node)함수에 들어오게 만든다.
좌측 node를 l_node로 받고 우측 node를 r_node로 받아서 insertDefine()에 넣어준다.
만약 우측 node의 타입이 ID값이면 value로 바꾸어 주어야 하기 때문에,
lookupdefine()함수를 호출한다.

▷ InsertDefine

```
def insertDefine(id, value):
    if value.type is TokenType.INT:
        intNode = Node(TokenType.INT, value.value)
        define_add[id] = intNode
    elif value.type is TokenType.QUOTE:
        define_add[id] = value
    elif value.type is TokenType.LIST:
        # new_value = run_expr(value)
        # print "CC"
        # define_add[id] = new_value
        define_add[id] = value
    else:
        define_add[id] = lookupDefine(value.value)
```

-insertDefine()의 역할은 전역변수로 선언 되어 있는 define_add에 키값으로 l_node의 id와 value를 넣고 관리한다.

▷ LookUpDefine

```
def lookupDefine(id):

    temp = define_add[id]
    if temp is not None:
        if temp.type is TokenType.INT:
            return temp
        elif temp.type is TokenType.QUOTE:
            return temp
        elif temp.type is TokenType.LIST:
            return temp
    return None
```

- lookupDefine()함수는 define_add 테이블에 id가 존재하는지 확인을 하고 있다면 return해준다.

▷ backup/load Define

```
def backupDefine():
    tmpdefine = define_add.copy()

def loadDefine():
    define_add = tmpdefine
```

- lambda함수가 실행 될 때, 그 변수의 값이 다른 lambda 함수의 변수와 중복이 되지 않게 하기 위하여, lambda함수 실행 전의 define_add정보를 backupDefine()함수를 통해 저장해 두고, 실행 후 에 loadDefine()함수로 그 전의 정보를 복구한다.

▷ Lambda

```
def lam(node):
    backupDefine()

    if node.next is None:
        return node

    if node.type is TokenType.LIST:
        parameter = node
        while node.value.next.value:
            parameter = parameter.next
            insertDefine(node.value.next.value.value, run_expr(parameter))
            if node.value.next.value.next is None:
                break
            node.value.next.value = node.value.next.value.next
        node = node.value
    result = run_expr(node.next.next)
    while True:
        if node.next.next.next is not None:
            result = run_expr(node.next.next.next)
            node = node.next
        else:
            break
    loadDefine()

    return result
```

- 람다가 인풋에 존재 할 때 람다 뒤의 노드가 없다면 전역 함수로 하기 위해 노드를 return해 준다.
만약 노드의 타입이 리스트라면 전역 변수 parameter에 노드를 저장하고 노드의 변수가 더 이상 존재 하지 않을 때까지 while문을 통해 insertDefine()에 정보들을 저장하고 parameter를 전진시키는 것을 반복한다.
더 이상 노드의 변수가 존재하지 않으면 break를 통해 while문을 중지시킨다.
특히 parameter에 run_expr()함수를 호출함으로써 재귀 함수가 실행된다.
또한 매개변수와 파라미터를 위치를 매치시키기 위해 노드의 값의 위치를 전진시킨다.
그리고 계산식을 run_expr()을 통해 계산하고 그 값을 result변수에 저장한다.
또, while반복문을 통해 계산식이 더 이상 존재하지 않을 때 까지 계산식을 반복해서 계산하고, 더 이상 존재하지 않는다면 break로 while문을 중지 시킨다.

▷ 변수 scope

scope는 영역을 나타낸다.
이에 따라서 전역 변수와 지역 변수로 나뉘게 되는데,
지역 변수는 변수가 선언된 지점에서만 사용 할 수 있다. 그에 반에 전역 변수는 global로 선언을 하게 되면 전 지역에서 사용 할 수 있다.
이번과제에서는 전역변수로 사용된 변수가 global parameter가 있습니다.
parameter는 각 input에서 입력된 파라미터들을 전역의 변수로 저장해 각 함수들에서 사용이 가능합니다.

3.실행 결과

```
> (define a 1)
> a
… 1
> (define b '(1 2 3))
> b
… '(1 2 3)
> (define c (- 5 2))
> c
… 3
> (define d '(+ 2 3))
> d
… '(+ 2 3)
> (define test b)
> test
… '(1 2 3)
> (+ a 3)
… 4
> (define a 2)
> (* a 4)
… 8
> ((lambda (x) (* x -2)) 3)
… -6
> ((lambda (x) (/ x 2)) a)
… 1
> ((lambda (x y) (* x y)) 3 5)
… 15
> ((lambda (x y) (* x y)) a 5)
… 10
> (define plus1 (lambda (x) (+ x 1)))
> (plus1 3)
… 4
> (define mul1 (lambda (x) (* x a)))
> (mul1 a)
… 4
> (define plus2 (lambda (x) (+ (plus1 x) 1)))
> (plus2 4)
… 6
> (define plus3 (lambda (x) (+ (plus1 x) a)))
> (plus3 a)
… 5
> (define mul2 (lambda (x) (* (plus1 x) -2)))
> (mul2 7)
… -16
> (define lastitem (lambda (ls) (cond ((null? (cdr ls)) (car ls)) (#T (lastitem (cdr ls))))))
> (lastitem '(1 2 3 4))
… 4
> (define square (lambda (x) (* x x)))
> (define yourfunc (lambda (x func) (func x))
> (yourfunc 3 square)
… 9
> (define square (lambda (x) (* x x)))
> (define mul_two (lambda (x) (* 2 x)))
> (define new_fun (lambda (fun1 fun2 x) (fun2 (fun1 x))))
> (new_fun square mul_two 10)
… 200
> (define cube (lambda (n) (define sqrt (lambda (n) (* n n))) (* (sqrt n) n)))
> (sqrt 4)

Error
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/unittest/case.py", lin
    testMethod()
  File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/unittest/loader.py", l
moments ago)                                                    101:1  CRLF  UTF-8
```

4. 느낀점

201102400 김남형 : 전체적인 구조를 이해하는게 어려웠습니다. 처음 과제를 시작하고 define 을 구현하는 데만 거의 하루가 걸렸습니다. 그 후 람다를 구현하면서 구조를 좀 더 잘 이해할 수 있었고 네스티드와 전역함수를 구현했을때 정말 뿌듯했습니다. 전문가들만 접근할 수 있다고 생각했던 인터프리터를 직접 만져보면서 한층 더 가까워진 것을 느꼈습니다.

201302492 최규호 : 처음에 과제가 나왔을 때 어떻게 해야 할지 너무 막막했습니다.
파이썬이라는 언어를 프로그래밍 언어 개론 과목에서 처음으로 접해 봤는데, 익숙하지 않은 언어로 과제를 해나감에 있어서 어려운 점이 많았지만, 결국 인터프리터를 완성했다는 점에 있어서 뿌듯함을 느낍니다.
또한 파이썬 과제가 매 주차를 거듭하면서 매 주마다 뜬금 없는 과제로 계속 주제가 바뀌는 것이 아니고 인터프리터라는 큰 틀 안에서 쉬운 것부터 단계적으로 과제가 진행 되었다는 점이 좋았습니다.
이것 저것 잡다하게 과제를 했다면, 최종 인터프리터를 못 만들었을 것 같습니다.
하지만 단계학습으로 인터프리터의 개념을 이해하게 되었고, 자바만 조금 할 줄 알았는데 새로운 언어를 배워간다는 점에서 수업이 유익했던 것 같습니다.
이렇게 최종 과제를 제출하고 유종의 미를 거둘 수 있었던 점에서 조은선 교수님께 감사합니다.