

프로그래밍언어개론

ITEM 2 보고서

01분반		
	학번	이름
조장	201302387	김진혁
조원	201302476	전성배
조원	201302491	최광호

목차

1. 문제	
1) Project 내용	
2) Project TEST Case	
2. 문제해결	
1) Project 내용	
2) Project TEST Case	
3) define 구현	
4) lookupTable 구현	
5) Lambda 구현	
6) 전역 함수 호출이 가능	
7) 함수 내에서 전역 함수 호출 가능	
8) 변수 scope 구현	
9) Nested 함수 구현	
10) Recursion 함수 구현	
11) 함수에서 함수를 인자로 사용 가능	
3. 결과화면	
4. 역할분담 & 느낀점	
5. GIT 사용	

문제

그 동안 과제를 수행해 온 Cute 17문법에 따른 프로그램을 입력하면 결과를 출력하는 인터프리터를 설계 및 구현하라

Project 내용

Item 2. 변수의 바인딩 처리: define문으로 변수로 정의하고 사용할 수 있도록 한다.

함수의 바인딩 처리: define문으로 함수를 정의하고 사용할 수 있도록 지원한다.
(함수가 함수의 인자로 넘어 갈 수 있어야 한다. . SSR방식)

주의)

- 변수는 define 으로 값을 지정한다

예) test case 1, 2 번 등 참고

- 이름없는 함수의 정의는

(lambda (parameter_list) (optional_definition_of_local_vars)
(function_body))

와 같은 형식으로 구현한다. (optional_definition_of_local_vars) 는 없어도 된다

예) test case 8번 등 참고

- 함수가 이름을 가지게 되는 방법

define 으로 변수에 값을 지정하듯이 define으로 이름없는 함수를 지정할 수 있다.

예) test case 12 번 등 참고

- 함수는 이름을 통해 다른 함수의 인자로 전달 될 수 있다.

예) test case 18, 19번 등 참고

- 함수 내에서 함수가 정의될 수 있다.

lambda에서 define 을 활용해서 (optional_definition_of_local_vars) 내부에서 정의

예) test case 20 번 등 참고

점수 기입이 없는 것은 1점

나머지는 주석에 점수와 결과 같이 기입되어 있음.

Project TEST Case

1. (define a 1)
a ;; 1
2. (define b '(1 2 3))
b ; '(1 2 3)
3. (define c (- 5 2))
C ;; 3
4. (define d '(+ 2 3))
D ;; `(+ 2 3)
5. (define test b)
test ;; '(1 2 3) (2점)
6. (+ a 3) ;;4
7. (define a 2) ;; 앞에 a가 정의된 후에 새로 정의
(* a 4) ;;8
8. ((lambda (x) (* x -2)) 3) ;;-6
9. ((lambda (x) (/ x 2)) a) ;;1(2점)
10. ((lambda (x y) (* x y)) 3 5) ;;15(2점)
11. ((lambda (x y) (* x y)) a 5) ;;10(2점)
12. (define plus1 (lambda (x) (+ x 1))) ;; 전역 함수 구현
(plus1 3) ;;4 (3점)
13. (define mul1 (lambda (x) (* x a))) ;; 전역 함수 구현, 전역 변수 포함
(mul1 a) ;;4 (3점)
14. (define plus2
(lambda (x) (+ (plus1 x) 1))) ;;함수 내에서 전역 함수 호출 가능
(plus2 4) ;; 6 (3점)
15. (define plus3
(lambda (x) (+ (plus1 x) a))) ;;함수 내에서 전역 함수 호출 가능, 전역 변수 포함
(plus3 a) ;; 5 (3점)
16. (define mul2

```
(lambda (x) (* (plus1 x) -2)))
```

```
(mul2 7) ;; -16 (3점)
```

17. (define lastitem

```
(lambda (ls)
```

```
(cond ((null? (cdr ls)) (car ls))
```

```
(#T (lastitem (cdr ls))))) ;;Recursion 구현 (6점)
```

18. (define square (lambda (x) (* x x)))

```
(define yourfunc (lambda (x func) (func x))
```

```
(yourfunc 3 square) ;;함수에서 함수를 인자로 사용가능 (4점)
```

19. (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) ;; (4점)
```

20. (define cube

```
(lambda (n) (define sqrt (lambda (n) (* n n)))
```

```
(* (sqrt n) n))) ;;Nested 함수 구현 (3점)
```

```
(sqrt 4) ;; 오류(에러처리를 할 필요는 없음) (3점)
```

문제해결

- define 구현

```
# 함수 define 정의
def define(node):
    l_node = node.value.next # type define 이후의 순수한 노드의 값 불러오기
    result = strip_quote(l_node).value # id 추출하기
    insertTable(result, l_node.next.value) # insertTable 함수를 이용하여 idTable에 { id : value } 값 저장
```

프로그램을 실행하고 op_code의 type이 DEFINE이면,
새로 만든 define(node) 함수를 실행 시켜준다.

define(node)는 변수를 정의해주는 함수이다. 변수 l_node 에 node.value.next를 통해 type DEFINE 이후의 값을 가지게 된다. 그리고 strip_quote를 이용하여 정의할 id를 추출하게 된다. insertTable() 함수를 통해 첫 번째 매개변수 값에 변수 result를 id로, 두 번째 매개변수 값에 id 이후의 노드 값을 value로 하여 idTable에 저장해준다.

- 자료구조는 list와 dictionary를 이용
- idTable[]은 idTable에 id와 value를 저장해주는 함수
- idTable[key] = value를 실행하면 idTable은 []리스트에 key와 value가 같이 저장된다.
- save_define{}은 save_define에 id와 value를 저장해주는 함수
- save_define[]을 실행하면 save_define은 { id : value } 와 같이 저장된다.

● lookupTable 구현

```
#함수 lookupTable 정의
def lookupTable(id):
    firstTemp = id.value
    if firstTemp in idTable:
        temp = idTable[firstTemp]
        if type(temp) is int: #함수 id의 value 값의 type이 int일 경우
            return lookupTable(Node(TokenType.INT, temp))
        elif type(temp) is str: #함수 id의 value 값의 type이 str일 경우
            return lookupTable(Node(TokenType.ID, temp))
        elif temp.type is TokenType.ID: #함수 id의 value 값의 type이 ID일 경우
            return temp
    elif temp.type is TokenType.LAMBDA: #함수 id의 value 값의 type이 LAMBDA일 경우
        temp = Node(TokenType.LIST, temp) #함수 temp에 Node를 생성
        if id.next is not None:
            if id.next.type is TokenType.ID:
                temp.next = Node(lookupTable(id.next).type, lookupTable(id.next))
            else:
                temp.next = id.next
        return temp
    temp1 = run_expr(lookupTable(Node(TokenType.LIST, temp)))
    return temp1
return id
```

lookupTable(id)는

매개변수의 Type이 ID일 경우 변수의 값을 사용할 수 있도록 처리해주는 함수이다.

매개변수의 Type이 ID일 경우 변수 firstTemp에 id의 value 값을 저장하게 해준다. firstTemp의 값이 define에서 정의해 저장해주었던 idTable에 존재한다면 조건에 만족한다. 그리고 변수 temp에 idTable에 존재하는 id에 대한 value 값을 저장하여 type을 검사하여 int와 str일 경우 출력해준다.

temp의 type이 LAMBDA 일 경우 temp를 Node로하여 값을 저장하게 된다. 그리고 id의 next값의 조건문을 통해 값을 구하게 되는데 만약 next값의 type이 ID가 아닐 경우는 temp.next 의 값을 id.next값으로 지정해준다. 그리고 다시 temp 값을 return 해주고 만약 ID라면 temp.next의 값을 lookupTable을 통해 id.next값의 value 값을 찾고 type과 value 값을 저장해준다.

그리고 여기서 temp의 type이 int,str,ID,LAMBDA가 아닐경우는 계산을 해야 하는 경우인데 run_expr(lookupTable(Node(TokenType.List, temp)))를 id로 지정해주고 값이 나올 때 까지 반복하여 결과를 출력하게 해준다.

● Lambda 구현(5점)

```
def run_lambda(node):
    a = strip_quote(node)
    b = strip_quote(node.value)
    if b.type is TokenType.LAMBDA:
        insertTable(b.next.value.value, node.next.value.next.value)
        return run_expr(node.next.value.value.next.next)
    if b.value.next is not None:
        aa=strip_quote(a)
        secondVariable= b.value.next.value # 두번째 변수 값
        if b.value.next.next is None:
            secondValue = a.next.value # 두번째 저장될 값
            insertTable(secondVariable, secondValue)
        else:
            secondValue = aa.value.value # 두번째 저장될 값 (세번째 변수 있을 경우)
            insertTable(secondVariable, secondValue)
            insertTable(b.value.next.next.value, aa.value.next.value) # 세번째 변수와 값
            firstVariable=strip_quote(b.value).value # 첫번째 입력될 변수
            firstValue=a.value.value # 첫번째 저장될 값
            insertTable(firstVariable, firstValue)
            temp6 = run_expr(firstValue.next.next)
            insertTable(b.value.next.next.value, temp6)
            secondValue.next.next.next = temp6
            return run_expr(secondValue.next.next)
    a = run_expr(a)
    insertTable(b.value.value, a.value)
    if lookupTable(b.next.value).type is TokenType.LIST:
        first = lookupTable(b.next.value)
        if lookupTable(b.next.value.next).type is TokenType.LIST:
            second = lookupTable(b.next.value.next)
            first.set_last_next(second)
            makeList = Node(TokenType.LIST, first)
            makeList = run_expr(makeList)
            return makeList
        else:
            second = b.next.value.next
            second = lookupTable(second)
            first.set_last_next(second)
            makeList = Node(TokenType.LIST, first)
            makeList = run_expr(makeList)
            return makeList

is_define = b.next #Nested 구현을 할 때, 실행부분에 define이 있는 경우 처리하기 위한 is_define변수
nested_define_Table = [] #Nested로 define 된 것이 내부에서만 define 되도록 지역변수 배열 만들어줌
save_define = {} #미리 전역변수로 선언된 경우, 저장해두기 위한 dictionary자료구조
while is_define.value.type is TokenType.DEFINE: #TokenType.DEFINE이라면 처리해줌
    save_data = is_define.value.next.value #define의 key 값
    if save_data in idTable: #key값이 idTable에 이미 존재한다면
        save_define[save_data] = idTable[save_data] #save_define dictionary에 값을 저장
    nested_define_Table.append(save_data) #nested_define_Table에 해당하는 value 값을 똑같이 넣어줌
    run_expr(is_define) # define을 해줌
    is_define = is_define.next #next에 define Token이 또 있는지 확인하기 위해 전진

return_val = run_expr(is_define)

while nested_define_Table.__len__() is not 0: #nested_define_Table이 빌때까지
    del idTable[nested_define_Table.pop()] #idTable에서 nested로 Define된 key, Value를 지워줌

for save_define_data in save_define: # save_define에 data가 있다면
    idTable[save_define_data] = save_define[save_define_data] #그 data를 다시 idTable에 바인딩

return return_val
```


‘(’ 다음에 나오는 첫 번째 원소가 lambda일 때 함수 run_lambda(node)가 실행된다. 변수 a 에는 strip_quote(node)를 통해 가장 node.value.next에 있는 값을 저장하고 변수 b 에는 strip_quote(node.value) 값을 통해 node.value.value.next 값을 저장하며 시작이 된다. 그리고 변수 b의 타입검사를 해주며 만약 조건을 만족한다면 insertTable에 값을 저장하고 return을 해준다. 그리고 b.value.next가 none이 아닐 조건을 들어가게 된다. b.value.next값이 none 이라면 a 값을 run_expr를 통해 다시 저장해주게 되고 아니라면 두 번째, 변수값을 저장하고 다시 그 조건의 next 값이 none 일때까지 조건을 줘서 insertTable에 저장하고 return 해주게된다. 그리고 변수가 다시 수정된다면 다시 insertTable에 저장을 해주며 진행된다.

그리고 다음 조건으로 lookupTable을 통하여 Type을 검사해주게 되고 Type이 List라면 List를 만들어 return 해주게 된다. 만약에 조건에 들어가 기존의 값의 .next 값이 List가 아니라면 한번 더 .next 값을 변수로 하여 List 만들어 주게된다.

마지막으로 nested_define_Table을 만들어주게 된다. 그리고 save_define도 만들어 준다. 여기서 반복문을 사용하게 되는데 위에서 저장한 변수 is_define(=b.next) 가 Type이 Define일 때 만약 여기서 저장한 데이터(save_date)가 idTable에 존재한다면 만들었던 save_data[]에다가 value 값을 저장하고 또한 nested_define_Table에도 append해주면서 is_define.next를 통해 이어가게 된다.

두 번째 반복문은 nested_define_table의 .len이 0이 아닐 때 까지 pop해주게 된다. 그리고 save_define_data가 save_define 안에 있다면 idTable에 저장을 해주면서 return 되는 원리다.

- 전역 함수 호출이 가능(7점)

```
table = {}
table['define'] = define
table['cons'] = cons
table['"'] = quote
table['quote'] = quote
table['cdr'] = cdr
table['car'] = car
table['eq?'] = eq_q
table['null?'] = null_q
table['atom?'] = atom_q
table['not'] = not_op
table['+'] = plus
table['-'] = minus
table['*'] = multiple
table['/'] = divide
table['<'] = lt
table['>'] = gt
table['='] = eq
table['cond'] = cond
table['lambda'] = run_lambda
if op_code_node.type is TokenType.LIST:
    return table[op_code_node.value.value]
if op_code_node.value in idTable:
    return run_expr(Node(TokenType.LIST, idTable[lookupTable(op_code_node)]))
return table[op_code_node.value]
```

이 전까지 define 함수에 key 값에 변수, value 값에 값에 해당하는 value 값을 저장하며 바인딩하였다. ‘전역 함수 호출’ 또한 크게 다르지 않은데, define 함수를 이용하여 key는 함수이름, value는 함수내용으로 idTable에 저장된다. 저장을 하는 부분에서는 이제 key, value일 때와 크게 다르지 않은데,

호출을 할 때 호출을 한 것이 LIST인 경우이고 List의 첫 Token이 바인딩된 key인 경우, lookupTable을 실행시켜 그 key에 바인딩된 value가 함수라면 그 함수를 불러와 run_expr을 실행하여 전역 함수가 호출이 가능하도록 구현했다.

- 함수 내에서 전역 함수 호출 가능(5점)

함수 내에서 전역 함수 호출이 가능한 부분 또한, 위에서 전역 함수를 호출 가능하게 한 부분에서 해결이 가능하게 되었다. plus, minus 등의 keyword 등에서 run_expr이 쓰이는 것은 이미 구현이 되어있으므로 run_expr을 실행을 하면서 table에 해당하는 keyword를 체크 할 때 조건문에서 해당하는 key가 idTable에 속하는지 확인하여 속한다면 lookupTable을 실행해서 value인 함수를 불러오고 그 함수를 기존의 next에 나오는 값들과 함께 묶어 다시 List를 만들어 return하여 연산이 가능하도록 한다.

- 변수 scope 구현(8점) (구현 방법에 대해 보고서에 자세히 쓸 것!)

변수의 scope를 구현하기 위해서는 시각적으로 봤을 때의 다음 token이 next로 연결되어 있는지 value로 연결되어있는지에 대해 이해해야한다. value는 해당 LIST의 값을 나타내고 next는 다음 LIST를 나타내다고 할 수 있다. 때문에 scope를 구별하기 위해선 LIST안에 LIST가 있을 경우 안쪽에 있는 innerLIST에 대해 먼저 계산해주고 바깥에 있는 outerLIST를 계산해 주는 방식으로 한다. 구현은 앞에 나오는 함수부터 접근하여 해당 value에 어떤 값들이 있나 확인을 한다. 예를들어 계산식이 사칙연산과 같다면 2개의 값을 각각 어떤 값인지 확인한다. 두 값중에 한 개라도 LIST로 존재하여 scope를 형성하고 있다면 recursive를 이용하여 안쪽에 있는 값을 먼저 계산하여 return값을 받아와 이전에 진행 하던 계산을 진행할 수 있도록 한다. lambda와 같은 경우에도 마찬가지로 lambda 계산을 하는 도중에 scope를 반전하면 recursive를 이용하여 해당값을 먼저 return받아 나머지 계산을 해주는 방식이다.

● Nested 함수 구현 가능(3점)

```
is_define = b.next #Nested 구현을 할 때, 실행부분에 define이 있는 경우 처리하기 위한 is_define 변수
nested_define_Table = [] #Nested로 define 된 것이 내부에서만 define 되도록 지역변수 배열 만들어줌
save_define = {} #미리 전역변수로 선언된 경우, 저장해두기 위한 dictionary 자료구조
while is_define.value.type is TokenType.DEFINE: #TokenType.DEFINE이라면 처리해줌
    save_data = is_define.value.next.value #define의 key 값
    if save_data in idTable: #key값이 idTable에 이미 존재한다면
        save_define[save_data] = idTable[save_data] #save_define_dictionary에 값을 저장
    nested_define_Table.append(save_data) #nested_define_Table에 해당하는 value 값을 똑같이 넣어줌
    run_expr(is_define) # define을 해줌
    is_define = is_define.next #next에 define Token이 또 있는지 확인하기 위해 전진

return_val = run_expr(is_define)

while nested_define_Table.__len__() is not 0: #nested_define_Table이 빌때까지
    del idTable[nested_define_Table.pop()] #idTable에서 nested로 Define된 key, Value를 지워줌

for save_define_data in save_define: # save_define에 data가 있다면
    idTable[save_define_data] = save_define[save_define_data] #그 data를 다시 idTable에 바인딩

return return_val
```

Nested함수는 define 된 함수 내에서 define이 또 존재하는 경우 그 define 된 함수 내부에서만 실행이 define하고 그 함수가 끝난다면 마찬가지로 사라지게 구현해야했다. 그로 인해 이 부분은 lambda를 실행 할 때 구현을 해줘야했는데 실행을 하는 도중 define이 나타난다면 이 define을 idTable에 넣어줌으로서 그 뒤에 나오는 Nested 함수에 대해서 실행이 가능하도록 하고, 이 함수로 인한 실행이 끝났다면 바인딩 된 부분을 지워줘야하므로 새로운 List 자료구조를 가지고 있는 nested_define_Table을 선언하여 이 List에 저장하고 실행이 끝난다면 이 nested_define_Table을 다시 확인하여 해당하는 함수명에 바인딩 된 함수를 idTable에서 제거하도록 구현하였다. 여기에 추가적으로 Nested에서 define된 함수가 이 전에 전역으로 선언이 되어있을 때 처리를 해주기 위해서 save_define이라는 dictionary를 만들어서 nested define함수를 처리해주기 전 기존에 있던 함수의 value를 저장해주고 실행이 끝나고 Nested에서 define된 함수를 idTable에서 제거를 해준 뒤, save_define dictionary를 찾아서 다시 idTable에 바인딩을 해줌으로서 구현하였다.

● Recursion 함수 구현 가능(2점)

우리는 Recursion를 구현하기 위해

lookuptable, cdr, run_cond, run_lambda 를 이용하였다.

cdr 함수

$l_node = run_expr(l_node) \rightarrow l_node = run_expr(lookupTable(l_node))$ 변경

run_cond 함수

$return\ node.value.next \rightarrow return\ run_expr(node.value.next)$ 변경

Recursion 은 함수의 매개변수가 계속 반복되기 때문에 변수의 scope를 구현해줘야 한다. 예를 들어 '(cdr ls)' 같은 문장이 계속되면 ls는 바뀌지 않기 때문에 예를 다시 한번 사용하자면 '(cdr ls)' 자체를 넘겨주면 안 되고 run_expr()를 통하여 결과 값을 넘겨주어야한다.

● 함수에서 함수를 인자로 사용가능(3점)

함수에서 함수를 인자로 사용하는 부분은 run_lambda함수를 수정하였다. 인자로 들어가는 값을 우선 table에 저장하고, 실행 할 때 table을 체크하여 해당하는 결과가 List 즉, 함수로 나온다면 그 인자를 함수로 바꾸어 리스트를 만들고 새롭게 만든 리스트를 run_expr을 통해 실행하여 return하도록 한다.

결과화면

```
Launching unittest with arguments python -m unittest discover -s C:/Users/user/Desktop/2017-1-6-(3-r-1-6)/alpha-  
... 1  
... '(1 2 3)  
... 3  
... '(+ 2 3)  
... '(1 2 3)  
... 4  
... 8  
... -6  
... 1  
... 15  
... 10  
... 4  
... 4  
... 6  
... 5  
... -15  
... 4  
... 9  
... 200  
  
Error  
Traceback (most recent call last):  
File "C:\Python27\lib\unittest\case.py", line 329, in run  
testMethod()  
File "C:\Python27\lib\unittest\loader.py", line 32, in testFailure  
raise exception  
ImportError: Failed to import test module: iteml  
Traceback (most recent call last):  
File "C:\Python27\lib\unittest\loader.py", line 254, in _find_tests  
module = self._get_module_from_name(name)  
File "C:\Python27\lib\unittest\loader.py", line 232, in _get_module_from_name  
__import__(name)  
File "C:\Users\User\Desktop\2017-1-6-(3-r-1-6)\alpha- 응용프로젝트_1\item1.py", line 853, in <module>  
Test_All()  
File "C:\Users\User\Desktop\2017-1-6-(3-r-1-6)\alpha- 응용프로젝트_1\item1.py", line 843, in Test_All  
Test_method("sort 4")  
File "C:\Users\User\Desktop\2017-1-6-(3-r-1-6)\alpha- 응용프로젝트_1\item1.py", line 794, in Test_method  
cute_inter = run_expr(node)  
File "C:\Users\User\Desktop\2017-1-6-(3-r-1-6)\alpha- 응용프로젝트_1\item1.py", line 711, in run_expr  
return run_list(root_node)  
File "C:\Users\User\Desktop\2017-1-6-(3-r-1-6)\alpha- 응용프로젝트_1\item1.py", line 356, in run_list  
return run_func(op_code_node)(root_node)  
File "C:\Users\User\Desktop\2017-1-6-(3-r-1-6)\alpha- 응용프로젝트_1\item1.py", line 665, in run_func  
return table[op_code_node.value]  
KeyError: 'sort'
```


역할분담 & 느낀점

김진혁(팀장) : 인터프리터가 생각보다 더욱 구현하기 힘든 과정이라는 것을 알게되었고, 인터프리터가 지금까지 배웠던 이론들과 연계가 되는 것이 조금 신기하였다. 그리고 깃허브를 지금까지 배워왔지만 제대로 사용하지 못했는데 이번 기회에 제대로 써보고 깃허브의 중요성을 알 수 있는 기회를 가지게 되었다.

전성배 : JAVA와 C를 통해 메소드 여러개를 만들어 프로그램을 구성해보았는데 이번학기에 처음 접해본 python을 통해 interpreter라는 큰 프로그램을 만들어보니 큰 project를 하는 것이 쉽지 않다는 걸 충분히 깨달았다. 하지만 git이라는 프로그램을 통해 3명의 팀원이 코드를 쉽게 공유하고 수정을 용이하게 할 수 있었다. 명령어만 알고있었는데 실제로 사용해보니 git에대한 숙련도가 쌓여 앞으로는 효율적으로 사용할 수 있을 것 같다. 이번 interpreter에서는 프로그램이 어떻게 돌아가고 python이 어떻게 구동되는지에 대해 잘 알 수 있는 계기였고, 한 학기동안 실습을 진행하면서 조금씩 배워나가 여기까지 오게되어 뿌듯한 마음이다.

최광호 : 우선 팀프로젝트에서 깃허브 사용의 중요성을 확실히 느꼈고 처음에 사용법에 대해 힘들었지만 익히고 사용을 해보니 매우 유용한 것 같다. 앞으로도 깃을 많이 사용해야 할 것 같다. 그리고 과제를 하면서 나의 코딩실력에 대해서 많은 자괴감도 빠지게 되었지만 팀원들 덕분에 과제를 더욱 잘 해결할 수 있었고 많이 배웠던 과제였던 것 같다. 디버그 실력 뿐만 아니라 코드실력, 알고리즘 이해에 대해 나 자신이 더욱 성숙해진 것 같다.

GIT 사용

 This repository Search Pull requests Issues Marketplace Gist

JeonSeongBae / ProgramingLanguage_08

Unwatch 3 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

PL Assignment #8: Cute17 Project

77 commits 4 branches 0 releases 3 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

persona5173 committed on GitHub Merge pull request #28 from JeonSeongBae/jinhyeokKim Latest commit 9283371 2 hours ago

idea	nested 완성, testCode, Input 분리	2 hours ago
.gitignore	gitignore 생성	5 days ago
ITEM2.hwp	nested 완성, testCode, Input 분리	2 hours ago
Project.pdf	temp	5 days ago
README.md	README내용추가 및 python 빈파일 생성	5 days ago
project item2-1.py	run_lambda 변수제거	a day ago
project item2.py	nested 완성, testCode, Input 분리	2 hours ago
project_testcase%28수정%29.pdf	PDF파일 추가	5 days ago

README.md

ProgramingLanguage_08

PL Assignment #8: Cute17 Project

PL Assignment #8: Cute17 Project 과제물 부과일 : 2017-05-11 (목) Program Upload 마감일 : 각 아이템에 따라 다른 문제 그 동안 과제를 수행해 온 Cute 17문법에 따른 프로그램을 입력하면 결과를 출력하는 인터프리터를 설계 및 구현하라 Project 내용 아래와 같이 2개의 item으로 나뉜다. 아래 일정에 따라 수행하며, 각 아이템 별로 보고서를 제출 한다. 조를 구성한 경우 각 조원 간의 역할 분담도 명확하게 되어야 한다. Item 1. 프로그램의 interpretation 환경 구현, Item 2. 변수의 바인딩 처리: define문은

JeonSeongBae / ProgramingLanguage_08

Unwatch 3 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

Branch: master ProgramingLanguage_08 / project item2.py Find file Copy path

persona5173 nested 완성, testCode, Input 분리 44b8c27 2 hours ago

3 contributors

853 lines (746 sloc) 25.8 KB Raw Blame History

```
1 # -*- coding: utf-8 -*-
2 from string import letters, digits, whitespace
3
4
5 class CuteType:
6     INT = 1
7     ID = 4
8
9     MINUS = 2
10    PLUS = 3
11
12    L_PAREN = 5
13    R_PAREN = 6
14
15    TRUE = 8
16    FALSE = 9
17
18    TIMES = 10
19    DIV = 11
20
21    LT = 12
22    GT = 13
23    EQ = 14
24    APOSTROPHE = 15
25
26    DEFINE = 20
27    LAMBDA = 21
28    CONID = 22
29    QUOTE = 23
30    NOT = 24
```