

PL Assignment #7: Cute17 Built-in Function 구현

과제물 부과일 : 2017-04-27 (목)

Program Upload 마감일 : 2017-05-03 (수) 23:59:59(추가 제출 기한 없음)

문제

Cute17의 문법에 맞게 작성된 프로그램을 interpret하는 프로그램을 구현 하시오.

Cute17의 built-in function

이번 과제에서 구현해야하는 built-in function들은 다음과 같다.

- 리스트 연산

car | cdr | cons

- 노트 연산

null? | atom? | eq?

1. car

List의 맨 처음 원소를 리턴한다.

```
> (car '(2 3 4))
```

```
2
```

```
> (car '((2 3) (4 5) 6))
```

```
'(2 3)
```

주의: car는 list가 아닌 데이터나 ()이 인자로 주어지는 경우 error 를 내게 된다. 그러나, 본 과제에서는 이러한 error 발생 경우에 대해서는 고려하지 않고, 모든 입력이 올바르게 되어 주어진다고 가정한다. (이것은 car 외의 다른 함수에도 동일하게 적용한다.)

2. cdr

list의 맨 처음 원소를 제외한 나머지 list를 리턴한다. list가 아닌 데이터에 대해서는 error 를 낸다.

```
> (cdr '(2 3 4))
```

```
'(3 4)
```

```
> (cdr '((2 3) (4 5) 6))
```

```
'((4 5) 6)
```

```
> (cdr '(2))
```

```
'()
```

3. cons

한 개의 원소(head)와 한 개의 리스트(tail)를 붙여서 새로운 리스트를 만들어 리턴한다.

```
> (cons 1 '(2 3 4))
'(1 2 3 4)
> (cons '(2 3) '(4 5 6))
'((2 3) 4 5 6)
> (cons 2 '())
'(2)
```

4. null?

리스트가 NULL 인지 검사한다. 즉, () 인지 검사한다.

```
> (null? '())
#T
> (null? '(1 2))
#F
> (null? '(()))
#F
```

5. atom?

list가 아니면 모두 atom 이다. 따라서 list인 경우는 false, list 가 아닌 경우는 true를 리턴한다. 주의 : null list은 atom 으로 취급된다.

```
> (atom? 'a)
#T
> (atom? '(1 2))
#F
> (atom? ())
#T
```

6. eq?

숫자 또는 문자열이 같은지 비교하여 true, false를 리턴한다. 만약 parameter가 숫자가 아닐 경우에는 전부 #F를 결과로 낸다. 현 과제에서는 문자열에 대해서는 구현하지 않았기 때문에 node가 INT일 경우에 대해서만 구현한다.

```
> (eq? 'a 'a)
#F
```

```
> (eq? 'a 'b)
```

```
#F
```

```
> (eq? 3 3)
```

```
#T
```

Programming

cdr, cons, atom?, null? 등을 처리하는 built-in 함수 처리를 위한 함수를 작성한다.

```
def run_list(root_node):
    """
    :type root_node: Node
    """
    op_code_node = root_node.value
    return run_func(op_code_node)(root_node)

def run_func(op_code_node):
    """
    :type op_code_node: Node
    """
    def quote(node):
        return node

    def strip_quote(node):
        """
        :type node: Node
        """
        if node.type is TokenType.LIST:
            if node.value is TokenType.QUOTE or TokenType.APOSTROPHE:
                return node.value.next
            if node.type is TokenType.QUOTE:
                return node.next
            return node

    def car(node):
        l_node = run_expr(node.value.next)
        result = strip_quote(l_node).value
        if result.type is not TokenType.LIST:
            return result
        return create_new_quote_list(result)

    def cdr(node):
        """
        :type node: Node
        """
        #Fill Out

    def cons(node):
        """
        :type node: Node
        """
        #Fill Out

    def null_q(node):
        #Fill Out

    def atom_q(node):
        #Fill Out

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

```

new_l_node = strip_quote(run_expr(l_node))
new_r_node = strip_quote(run_expr(r_node))

if (new_l_node.type or new_r_node.type) is not TokenType.INT:
    return Node(TokenType.FALSE)
if new_l_node.value == new_r_node.value:
    return Node(TokenType.TRUE)
return Node(TokenType.FALSE)

def create_new_quote_list(value_node, list_flag=False):
    """
    :type value_node: Node
    """
    quote_list = Node(TokenType.QUOTE, 'quote')
    wrapper_new_list = Node(TokenType.LIST, quote_list)
    if value_node is None:
        pass
    elif value_node.type is TokenType.LIST:
        if list_flag:
            inner_l_node = Node(TokenType.LIST, value_node)
            quote_list.next = inner_l_node
        else:
            quote_list.next = value_node
        return wrapper_new_list
    new_value_list = Node(TokenType.LIST, value_node)
    quote_list.next = new_value_list
    return wrapper_new_list

table = {}
table['cons'] = cons
table['""'] = quote
table['cdr'] = cdr
table['car'] = car
table['eq?'] = eq_q
table['null?'] = null_q
table['atom?'] = atom_q

return table[op_code_node.value]

def run_expr(root_node):
    """
    :type root_node : Node
    """
    if root_node is None:
        return None

    if root_node.type is TokenType.ID:
        return root_node
    elif root_node.type is TokenType.INT:
        return root_node
    elif root_node.type is TokenType.TRUE:
        return root_node
    elif root_node.type is TokenType.FALSE:
        return root_node
    elif root_node.type is TokenType.LIST:
        return run_list(root_node)
    else:
        print 'Run Expr Error'
        return None

```

Print_node

```
def print_node(node):
    """
    "Evaluation 후 결과를 출력하기 위한 함수"
    "입력은 List Node 또는 atom"
    :type node: Node
    """
    def print_list(node):
        """
        "List노드의 value에 대해서 출력"
        "( 2 3 )이 입력이면 2와 3에 대해서 모두 출력함"
        :type node: Node
        """
        if node.type is TokenType.LIST:
            if node.value.type is TokenType.QUOTE:
                return print_node(node.value)
            return "("+print_list(node.value)+")"
        else:
            if node.next is None:
                return print_node(node)
            else:
                return print_node(node)+" "+print_list(node.next)

    if node is None:
        return ""
    if node.type in [TokenType.ID, TokenType.INT]:
        return node.value
    if node.type is TokenType.TRUE:
        return "#T"
    if node.type is TokenType.FALSE:
        return "#F"
    if node.type is TokenType.PLUS:
        return "+"
    if node.type is TokenType.MINUS:
        return "-"
    if node.type is TokenType.TIMES:
        return "*"
    if node.type is TokenType.DIV:
        return "/"
    if node.type is TokenType.GT:
        return ">"
    if node.type is TokenType.LT:
        return "<"
    if node.type is TokenType.EQ:
        return "="
    if node.type is TokenType.LIST:
        return print_list(node)
    if node.type is TokenType.ATOM_Q:
        return "atom?"
    if node.type is TokenType.CAR:
        return "car"
    if node.type is TokenType.CDR:
        return "cdr"
    if node.type is TokenType.COND:
        return "cond"
    if node.type is TokenType.CONS:
        return "cons"
    if node.type is TokenType.LAMBDA:
```

```

        return "lambda"
    if node.type is TokenType.NULL_Q:
        return "null?"
    if node.type is TokenType.EQ_Q:
        return "eq?"
    if node.type is TokenType.NOT:
        return "not"
    if node.type is TokenType.QUOTE:
        return "'" + print_node(node.next)

```

테스트

```

def Test_method(input):
    test_cute = CuteScanner(input)
    test_tokens = test_cute.tokenize()
    test_basic_paser = BasicPaser(test_tokens)
    node = test_basic_paser.parse_expr()
    cute_inter = run_expr(node)
    print print_node(cute_inter)

```

```

def Test_All():
    Test_method("(car '(2 3 4))")
    Test_method("(car '((2 3) (4 5) 6))")
    Test_method("(cdr '(2 3 4))")
    Test_method("(cdr '((2 3) (4 5) 6))")
    Test_method("(cdr '(2))")
    Test_method("(cons 1 '(2 3 4))")
    Test_method("(cons '(2 3) '(4 5 6))")
    Test_method("(cons 2 '())")
    Test_method("(null? '())")
    Test_method("(null? '(1 2))")
    Test_method("(null? '())")
    Test_method("(atom? 'a)")
    Test_method("(atom? '(1 2))")
    Test_method("(atom? '())")
    Test_method("(eq? 'a 'a)")
    Test_method("(eq? 'a 'b)")
    Test_method("(eq? '3 '3)")
    Test_method("(eq? '3 '4)")

```

Test_All()

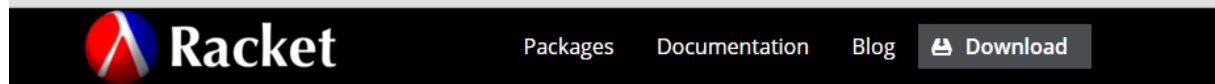
결과화면

```
hw06_revise_version.py
C:\Python27\python.exe C:/Users/hyungken/PycharmProjects/PL2017/PL/
2
'(2 3)
'(3 4)
'((4 5) 6)
'(')
'(1 2 3 4)
'((2 3) 4 5 6)
'(2)
#T
#F
#F
#T
#F
#T
#F
#F
#T
#F
```


보충 자료

Scheme 인터프리터를 사용하여 자신이 만든 인터프리터와 비교하기.

1. <https://racket-lang.org/>에 접속
2. Download 클릭



A programmable programming language

Racket is a full-spectrum programming language. It goes beyond Lisp and Scheme with dialects that support objects, types, laziness, and more. Racket enables programmers to link components written in different dialects, and it empowers programmers to create new, project-specific dialects. Racket's libraries support applications from web servers and databases to GUIs and charts.

Start Quickly


News

3. 자신의 OS에 맞는 버전을 선택하여 다운로드

Version 6.5 (April 2016)

Distribution:

Platform:

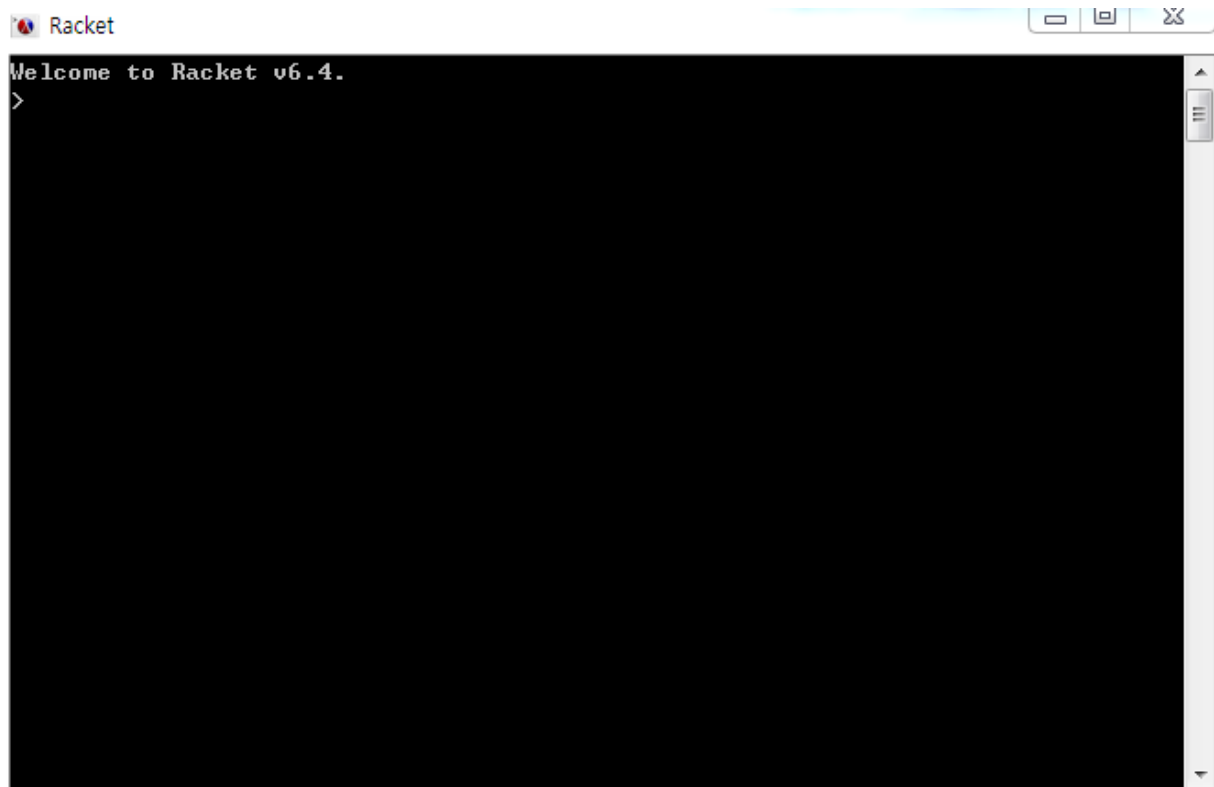
 **racket-6.5-x86_64-win32.exe (75M)**

or [mirror](#)

Release: [Announcement](#) • [Notes](#) • [Documentation](#)
[More Variants and Checksums](#)

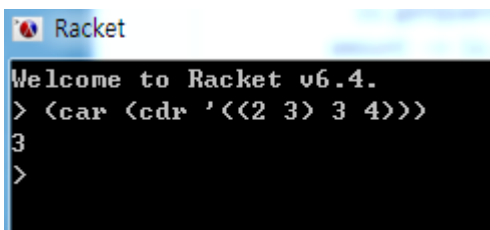
[License](#) • [All Versions](#) • [Snapshot Builds](#)

4. 설치 후 실행



```
Racket
Welcome to Racket v6.4.
>
```

5. 확인하고자 하는 statement 작성 후 결과 확인



```
Racket
Welcome to Racket v6.4.
> <car <cdr '<(2 3) 3 4>>>
3
>
```