

PL Assignment #7 : Cute19 Parser2

과제물 부과일 : 2018-05-01(수)

Program Upload 마감일 : 2018-05-10(금) 23:59:59

문제

Cute19 문법에 따라 작성된 program이 as07.txt에 저장되어 있다. 이를 input file로 하여, 프로그램의 syntax tree를 구성하시오.(이 과정을 parsing이라고 한다.) 그리고 syntax tree를 root로부터 pre-order traverse하여 원래 입력된 프로그램과 구조가 동일한 프로그램을 파일(output07.txt)에 출력해야 한다.(이 과정을 unparsing이라고 한다.)

예를 들어, Cute19으로 작성된 program의 입력이 아래의 경우들이라면,

- (a) (+ (- 3 2) -378)
- (b) (quote (+ 2 3))
- (c) ` (+ 4 5)
- (d) (define length (lambda (x) (cond ((null? x) 0)
(#T (+ 1 (length (cdr x)))))))

이와 같은 프로그램의 출력 결과들은 다음과 같다.

- (a) ([PLUS] ([MINUS] [INT:3] [INT:2]) [INT:-378])
- (b) '([PLUS] [INT:2] [INT:3])
- (c) '([PLUS] [INT:4] [INT:5])
- (d) ([DEFINE] [ID:length] ([LAMBDA] ([ID:x]) ([COND] (([NULL_Q] [ID:X])
[INT:0]) ([#T] ([PLUS] [INT:1] ([ID:length] ([CDR] [ID:x]))))))))

Cute19의 문법

Program → Expr | “'” Expr
List → ' (' ItemList ') '
ItemList → Item ItemList | ε
Item → Expr | “'” Expr | “quote” Expr
| '+' | '- ' | '+ ' | '/' | '<' | '>' | '='
| “define” | “cond” | “not” | “lambda” | “car” | “cdr” | “cons”
| “eq?” | “atom?” | “null?”
Expr → id // id에는 define, cond, lambda, ...등의 키워드는 제외됨
| int //integer const
| “#T” | “#F”
| List

추가 설명

- 문법에서 대문자로 시작하는 이름은 non-terminal이고, 소문자로 시작하는 이름인 id와 int는 terminal이다.
- Terminal 중에서 id는 모든 identifier를 총칭하고, int는 모든 정수형 상수를 총칭한다. 따라서 id와 int는 token이라고 볼 수 있으며, token으로 처리하기 위해서 token 이름을 아래와 같이 명명할 수 있다.

```
id                TokenType.ID
```

```
int               TokenType.INT
```

```
// id와 int에는 여러 가지가 있을 수 있으므로,
```

```
// lexeme으로 구별해 주어야 한다.
```

- 특별한 의미를 가지는 keyword와 해당 token 이름은 다음과 같다. (keyword가 아니면 ID로 간주)

```
"define"          TokenType.DEFINE
```

```
"lambda"          TokenType.LAMBDA
```

```
"cond"            TokenType.COND
```

```
"quote"           TokenType.QUOTE
```

```
"not"             TokenType.NOT
```

```
"cdr"             TokenType.CDR
```

```
"car"             TokenType.CAR
```

```
"cons"            TokenType.CONDS
```

```
"eq?"             TokenType.EQ_Q
```

```
"null?"           TokenType.NULL_Q
```

```
"atom?"           TokenType.ATOM_Q
```

- Boolean 상수는 terminal이며, 해당 token은 다음과 같다.

```
#T                TokenType.TRUE
```

```
#F                TokenType.FALSE
```

- 특수 문자들은 terminal이며, 다음과 같이 token 이름을 부여할 수 있다.

```
(                TokenType.L_PAREN
```

```
)                TokenType.R_PAREN
```

```
+                TokenType.PLUS
```

```
-                TokenType.MINUS
```

```
*                TokenType.TIMES
```

```
/                TokenType.DIV
```

```
<                TokenType.LT
```

```
=                TokenType.EQ
```

```
>                TokenType.GT
```

```
`                TokenType.APOSTROPHE
```

// car, cdr, cars 등은 이번 과제에서 구현할 내용이 아닙니다. 추후 구현 예정

1. car

List의 맨 처음 원소를 리턴한다.

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

```
2
```

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

```
2
```

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
```

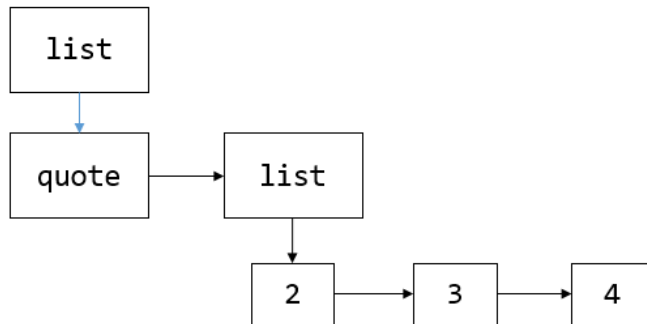
```
> (cdr '((2 3) (4 5) 6))
```

```
4
```

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

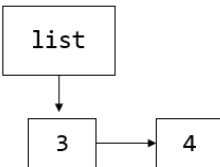
```
5
```

위의 두 `car`, `cdr`의 예를 그림으로 나타내면 다음과 같다.



이러한 노드가 있다고 하면

`car`는 

`cdr`는 

를 리턴한다.

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)
```

Cute19의 특징

괄호를 써서 프로그램이 표현되는 Cute19은 list가 기본 표현이다. 또한 아래와 같이 각 list의 맨 첫번째 원소를 연산자나 함수 호출로 보고 리스트의 나머지를 피연산자로 간주한 후 evaluate 하게 된다. (다음 예는 > 를 prompt 로 사용하고 있는 인터프리터를 보여준다. 이후 과제에서 진행 예정인 부분이다.)

```
> (+ 2 3)
5
> (* (+ 3 3) 2)
12
```

따라서 만일 상수 list (즉, 프로그램이 아닌 데이터 list) 를 표현하고자 할 때는 특별한 표시를 해야한다. 이 때 사용되는 것이 연산자 “\” 와 키워드 quote이다.

```
> (+ 1 2)
3
> \(+ 1 2)
(+ 1 2)
> (quote (+ 1 2))
(+ 1 2)
```

연산자 “\” 와 키워드 quote가 문자나 문자열에 적용되면 문자나 문자열 상수를 의미한다. 그렇지 않은 경우는 변수를 의미한다.

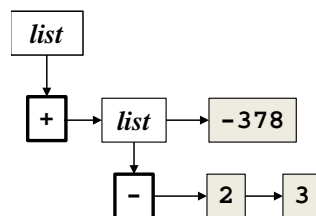
```
> `a
a
> `abc
abc
```

만일 quote 후에 여러 item이 나오면 첫 번째 것만 상수로 취하고 나머지는 무시한다. 그러나 편의상, 본 과제에서는 이러한 입력 등 문법오류가 있는 입력은 없다고 가정한다.

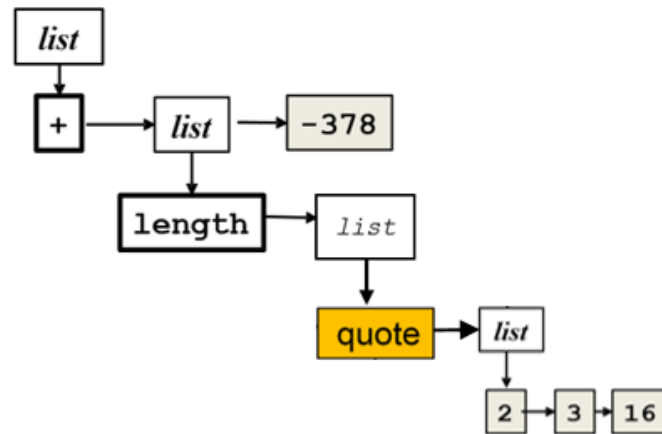
```
> (quote + 1 2)
+
```

다음과 같은 프로그램이 있다고 가정하면, parse tree 는 다음과 같이 된다.

(+ (- 2 3) -378)



(+ (length `(2 3 16)) -378)



linked list 의 맨 앞 노드는 연산자나 함수 이름으로 인식한다. 그러나 위 '(2 3 16)과 같이 list 앞에 ' 표시가 있거나 (QUOTE (2 3 16)) 과 같이 표현되면 상수 리스트 (데이터)로 인식한다. id 나 기타 expression에 대한 quote는 quote노드의 value 로 표현된다. 다음은 a 일 때(왼쪽)와 'a 일 때(오른쪽)의 노드 모양이다.



그리고 이들 프로그램의 각 출력결과는 다음과 같다. (키워드 quote를 사용한 경우와 "\'" 를 사용한 경우는 동일하게 출력한다.)

```

(+ (- 3 2) -378)
([PLUS] ([MINUS] [INT:3] [INT:2] ) [INT:-378] )

(+ (length '(2 3 16)) -378)
([PLUS] ([ID:length] '([INT:2] [INT:3] [INT:16] ) ) [INT:-378] )

a
[ID:a]

'a
'[ID:a]

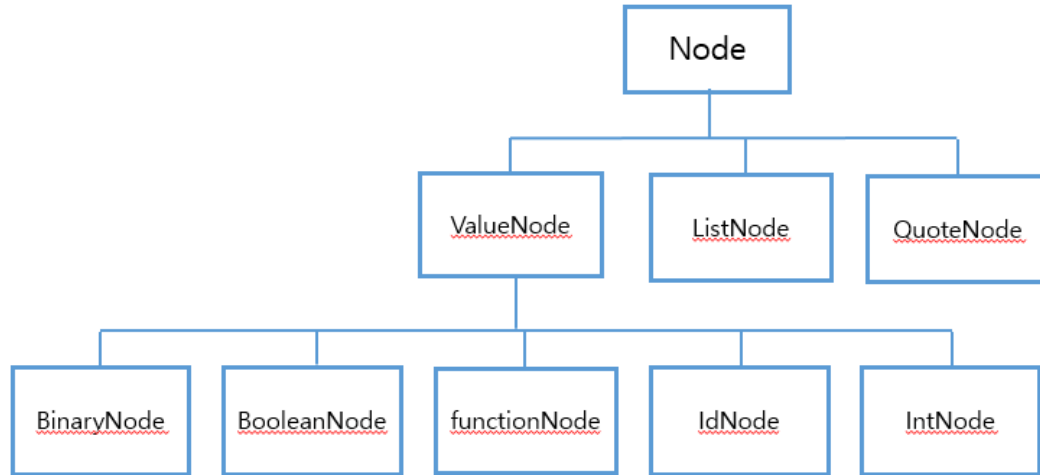
(quote (1 2 3) )
'([INT:1] [INT:2] [INT:3] )

```

Programming

1. 노드의 자료구조

QuoteNode, ValueNode 클래스 추가



***ValueNode의 밑에 있는 node들은 ValueNode를 implements하게 수정해야 함**

```
public interface Node {  
    // 새로 수정된 Node Class  
}  
  
public class QuoteNode implements Node {  
    // 새로 추가된 QuoteNode Class  
    Node quoted;  
  
    public QuoteNode(Node quoted) {  
        this.quoted = quoted;  
    }  
    @Override  
    public String toString(){  
        return quoted.toString();  
    }  
    public Node nodeInside() {  
        // TODO Auto-generated method stub  
        return quoted;  
    }  
}  
  
...  
public interface ValueNode extends Node {  
    // 새로 추가된 ValueNode Class  
}  
  
... public class BooleanNode implements ValueNode {  
    // 새로 수정된 BooleanNode Class  
    public static BooleanNode FALSE_NODE = new BooleanNode(false);  
    public static BooleanNode TRUE_NODE = new BooleanNode(true);  
    Boolean value;  
  
    private BooleanNode(Boolean b) {  
        value = b;  
    }  
}
```

```

@Override
public String toString() {
    return value ? "#T" : "#F";
}
}
public class IdNode implements ValueNode {
    // 새로 수정된 IdNode Class
    String idString;

    public IdNode(String text) {
        idString = text;
    }
    @Override
    public String toString(){
        return "ID:" + idString;
    }
}
public class IntNode implements ValueNode {
    // 새로 수정된 IntNode
    private Integer value;
    @Override
    public String toString(){
        return "INT:" + value;
    }
    public IntNode(String text) {
        this.value = new Integer(text);
    }
}
public interface ListNode extends Node {
    // 새로 수정된 ListNode
    static ListNode EMPTYLIST = new ListNode() {
        @Override
        public Node car() {
            return null;
        }
        @Override
        public ListNode cdr() {
            return null;
        }
    };
    static ListNode ENDLIST = new ListNode() {
        @Override
        public Node car() {
            return null;
        }
        @Override
        public ListNode cdr() {
            return null;
        }
    };
    static ListNode cons(Node head, ListNode tail) {
        return new ListNode() {
            @Override
            public Node car() {

```



```

        return head;
    }
    @Override
    public ListNode cdr() {
        return tail;
    }
};

}
Node car();
ListNode cdr();
}

```

2. 프로그램을 parsing하는 클래스 구현 예시

```

public class CuteParser {
    private Iterator<Token> tokens;
    private static Node END_OF_LIST = new Node({}); // 새로 추가된 부분
    ... // 이전 소스코드는 지난 과제와 동일
    case ID:
        return new IdNode(tLexeme);
    case INT:
        if (tLexeme == null)
            System.out.println("???");
        return new IntNode(tLexeme);
    //새로 구현된 BooleanNode Case
    case FALSE:
        return BooleanNode.FALSE_NODE;
    case TRUE:
        return BooleanNode.TRUE_NODE;
    //새로 구현된 L_PAREN, R_PAREN Case
    case L_PAREN:
        return parseExprList();
    case R_PAREN:
        return END_OF_LIST ;
    //새로 추가된 APOSTROPHE, QUOTE
    case APOSTROPHE:
        QuoteNode quoteNode = new QuoteNode(parseExpr());
        ListNode listNode = ListNode.cons(quoteNode, ListNode.ENDLIST);
        return listNode;
    case QUOTE:
        return new QuoteNode(parseExpr());
    default:
        System.out.println("Parsing Error!");
        return null;
}

}
private ListNode parseExprList() {
    Node head = parseExpr();
    if (head == null)
        return null;
    if (head == END_OF_LIST) // if next token is RPAREN
        return ListNode.ENDLIST;
    ListNode tail = parseExprList();
    if (tail == null)
        return null;
    return ListNode.cons(head, tail);
}
}

```

```

}

public class NodePrinter {
    private final String OUTPUT_FILENAME = "output07.txt";
    private StringBuffer sb = new StringBuffer();
    private Node root;
    private NodePrinter(Node root) {
        this.root = root;
    }

    // ListNode, QuoteNode, Node에 대한 printNode 함수를 각각 overload 형식으로 작성
    private void printList(ListNode listNode) {
        if (listNode == ListNode.EMPTYLIST) {
            sb.append("( )");
            return;
        }
        if (listNode == ListNode.ENDLIST) {
            return;
        }
        // 이후 부분을 주어진 출력 형식에 맞게 코드를 작성하시오.
    }
    private void printNode(QuoteNode quoteNode) {
        if (quoteNode.nodeInside() == null)
            return;
        // 이후 부분을 주어진 출력 형식에 맞게 코드를 작성하시오.
    }
    private void printNode(Node node) {
        if (node == null)
            return;
        // 이후 부분을 주어진 출력 형식에 맞게 코드를 작성하시오.
    }
    public void prettyPrint() {
        printNode(root);
        try (FileWriter fw = new FileWriter(OUTPUT_FILENAME);
            PrintWriter pw = new PrintWriter(fw)) {
            pw.write(sb.toString());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

3. 테스트

```

public static final void main(String... args) throws Exception {
    ClassLoader cloader = ParserMain.class.getClassLoader();
    File file = new File(cloader.getResource("parser/as07.txt").getFile());

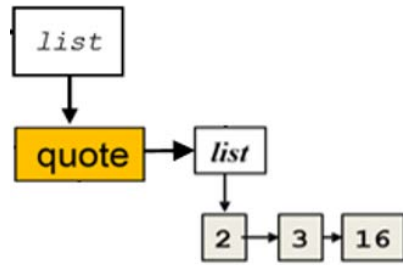
    CuteParser cuteParser = new CuteParser(file);
    NodePrinter nodePrinter = new NodePrinter(cuteParser.parseExpr());
    nodePrinter.prettyPrint();
}

```

유의사항

- "\'\"와 quote로 만들어진 노드는 구조가 같아야 한다.
 - ex) '(2 3 16) 과 (quote (2 3 16)) 의 노드 구조는 같아야 한다.

구조는 아래와 같다.

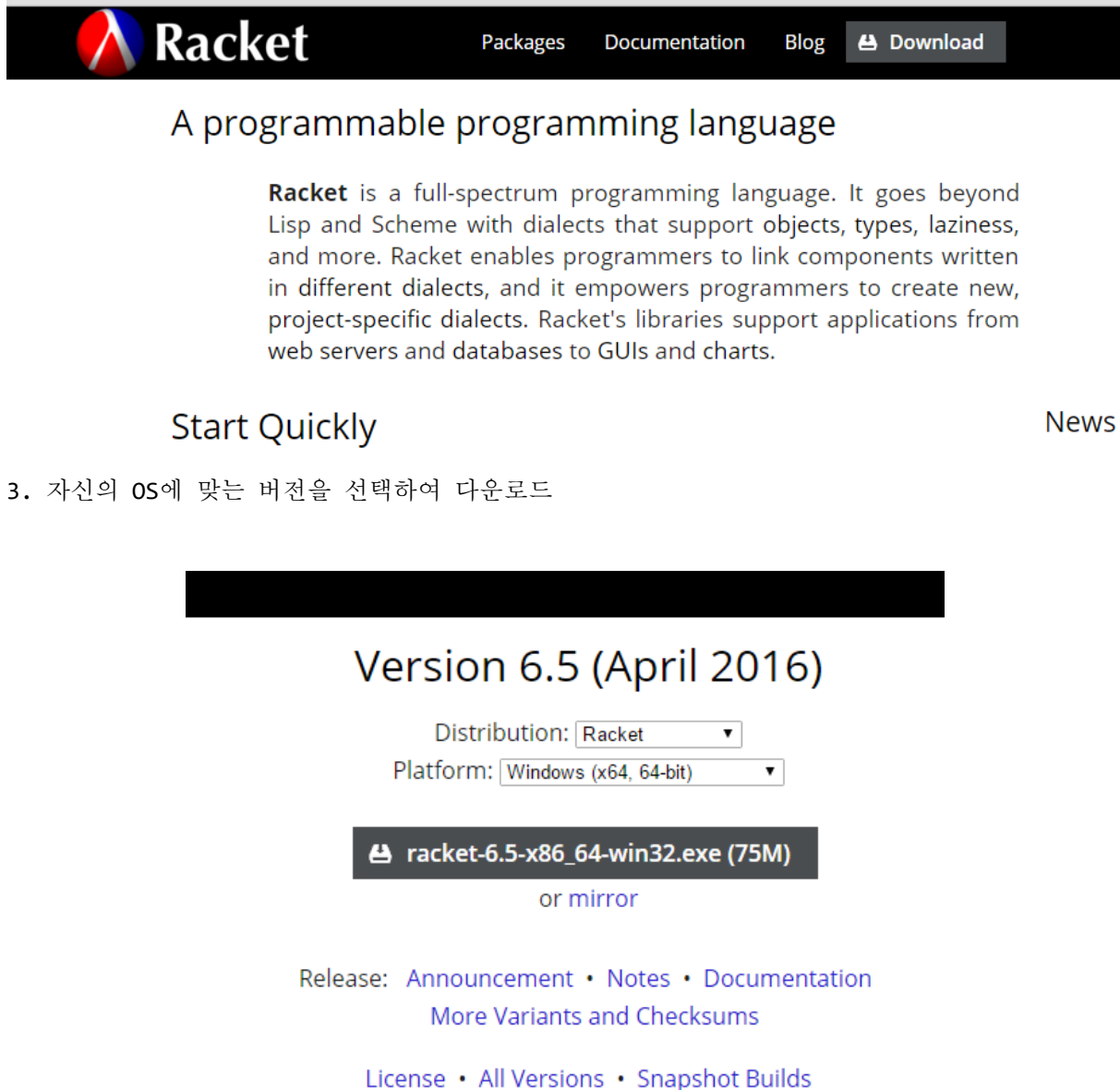


- 결과는 '([INT: 2] [INT: 3] [INT: 16])' 와 같이 같은 결과가 나와야 한다.
즉 "\'"의 경우 ListNode가 곁에 있다고 생각하고 내부 구조를 만든다.
구조는 (quote(...)) 이지만 출력은 \'(...) 이어야 한다.
- 파일읽기 부분은 편의대로 작성 가능
- Input file은 직접 작성하여 사용한다.
- 쪽지 시험 문제로 나올 수 있음
- **과제물 추가 제출 기간은 없음**

보충 자료

Scheme 인터프리터를 사용하여 자신이 만든 인터프리터와 비교하기.

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



The screenshot shows the Racket website's download page. At the top is a navigation bar with the Racket logo, 'Packages', 'Documentation', 'Blog', and a 'Download' button. Below the navigation bar is the heading 'A programmable programming language' followed by a paragraph describing Racket as a full-spectrum programming language. To the right of this paragraph is a 'News' link. Below the paragraph is a 'Start Quickly' link. Further down is a section for 'Version 6.5 (April 2016)' which includes dropdown menus for 'Distribution' (set to 'Racket') and 'Platform' (set to 'Windows (x64, 64-bit)'). Below these is a large button to download 'racket-6.5-x86_64-win32.exe (75M)' with a link to a 'mirror'. At the bottom are links for 'Release: Announcement • Notes • Documentation More Variants and Checksums' and 'License • All Versions • Snapshot Builds'.

Racket

Packages Documentation Blog [Download](#)

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:

[Download](#) **racket-6.5-x86_64-win32.exe (75M)**

or [mirror](#)

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

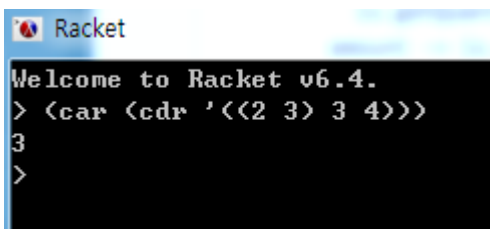
[License](#) • [All Versions](#) • [Snapshot Builds](#)

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
>
```