프로그래밍 언어 개론 hw item2

학번/이름:

정성욱(팀장)	201404376
김주성	201701996
김기범	201502019

분반: 00반 조은선 교수님 분반

문제 해결 방법

문제개요

-DEFINE함수를 작성해서 변수 정의,재정의, 다른 함수 연산 및 사칙연산, 논리연산에서도 호출해서 사용할 수 있도록 작성.

문제 해결 방법

DEFINE 시

1. **디파인을 case문에 구현한다. 프로그램 실행시 들어오는 스트링값에 대해 구문 분석을 해서 미리 한번 연산을 한 후 그 결과값을 첫번째 매개변수에 저장한다. Ex((define a (> 3 2)) 일시 (> 3 2) 를 하나의 리스트 노드로 보고 runExpr를 실행해서 그 결과값을 리턴하는 식으로 함.**
2. **리턴된 값은 Static ArrayList로 정의된** *DF_Table_list에* DF_Table의 클래스 객체로 저장됨. DF_Table는 `class DF_Table<String,Node>` 로 String과 Node 두개의 튜플을 가짐 생성자를 만들 시 아예 key와 value를 주고 생성함.
   이 *DF_Table_list는* Table_List 클래스 내부에 static으로 생성 되어있음.
3. **DEFINE에서** Table_List객체를 따로 정의해서 2.에서 생성한 객체를 Table_List의 add_DF_table함수를 이용하여 *DF_Table_list에 추가함.*
4. **그후 추가한 노드 자체를 리턴함.**

DEFINE 하고 실행할 때

(ex (define a 4)를 하고 다음 문장에 a만 입력함 그럴 경우 스트링 a가 아니라 4가 나오는 경우임)

1. **define으로 a를 4로 정의함.**
2. **DEFINE 으로 들어가서** `DF_Table_list`**에 저장됨.**
3. **다음 인터프리터 라인에 a 입력 받을 시, 4가 출력되야하는데 parsing tree를 만들고 그것에 대한 유도된 값에 대한 최종 결과값이 IdNode :a 일 것인데 여기에** `DF_Table_list`**에 정의가 된 String값인지 검사를 함. 만약 일치하는 Key 값이 있다면, Node로 대치함.**
4. **대치된 노드가 Nodeprinter로 가면서 변경된 값인 4가 나온다.**

사전에 정의된 변수를 함수연산에 사용할 때

1. **(define a '(1 2 3))을 정의 함.**
   디파인 테이블

String a, '(123)

2. **인터프리터에서 (car a) 를 입력받음.**
3. **처리를하려고 runFunction을 들어갔는데 operand인 a가 IdNode임 그에 따라 디파인 테이블 리스트에 정의된 변수일 가능성이 있으니, 디파인 이 IdNode의 값을 매개변수로 디파인 테이블 리스트의 키값들을 조사하고 일치하는 값이 있다면 키 값이 가진 디파인 테이블이 가진 노드로 대체함.**
4. **'(1 2 3)으로 대체되고 결국 (car '(1 2 3))의 연산이 되고 1이 나옴.**
5. **BinaryOPNode도 같은 원리로 진행함**

디파인에서 정의된 키,노드를 튜플로
저장하는 디파인 테이블 객체

```
class DF_Table<String,Node> {
    //튜플 형태로 스트링,노드를 저장함.
    private String key;
    public Node value;

    public DF_Table(String k, Node v) {
       this.key = k;
       this.value = v;
     }
    //가진 키값리턴함.
    public String getKey() {
        return key;
    }
    //현재 클래스가 가진 노드 값 리턴
    public Node getNode() {
        return value;
    }
}
```

디파인 테이블 리스트를 관리하는 클래스임 내부에 스태틱으로 디파인 테이블 리스트가 정의되어 있어서 이 객체의 생성 유무에 관계없이 프로그램 실행시 처음부터 힙에 잡혀있음.

```
class Table_List{
    //DEFINE을 호출해서 정의된 변수들을 프로그램 종료될때까지 저장하기위한 Define_table의 집합임.
    static ArrayList<DF_Table> DF_Table_list = new ArrayList<DF_Table> ();
    //현재 리스트에 몇개의 변수가 정의되어있는지 쓸거같아서 정의해뒀는데 여기서는 안쓰게됨
    static int DF_Table_List_max_index_num =0;
    //변수 재정의 그 변수가 재정의 된 위치를 찾기위한 함수임
    //string k값고 같은 노드의 위치를 리턴함
    public int find_table_index(String k) {
        DF_Table sort_res = null;
        int idx = 0;
        for (DF_Table one : DF_Table_list) {
            String tmp_str = (String) one.getKey();
            if(tmp_str.equals(k)) {
                return idx;
            }
            idx++;
        }
        return -1;
    }
    //디파인 테이블리스트에 디파인 테이블을 만들어서 추가하는 노드임
     public void add_DF_table(String k, Node v) {
         //디파인 테이블 리스트가 비었다면, 첫부분 헤드를 만들어줌
         //max값은 헤드 index가 0 이므로 증가 시키지 않음.
         if(DF_Table_list.isEmpty()) {
             DF_Table head= new DF_Table(k,v);
             DF_Table_list.add(head);
             //디파인 테이블 리스트가 비어있지않다면
             //추가해줌
         }else {
             //만약 키값이 중복된 노드일 경우 재정의하는 상태이므로
             // 테이블리스트에 해다인덱스를 찾아서 제거하고
             Node duple = find_id_Node(k);
             if(duple!=null) {
                int duple_idx= find_table_index(k);
                DF_Table_list.remove(duple_idx);
                DF_Table_list.add(duple_idx,new DF_Table(k,v));
             }else{
```

디파인 테이블을 만들어서 디파인 테이블 리스트에 추가하는 add_DF_table 함수가 있음.

```
            }else {
                //만약 키값이 중복된 노드일 경우 재정의하는 상태이므로
                // 테이블리스트에 해다인덱스를 찾아서 제거하고
                Node duple = find_id_Node(k);
                if(duple!=null) {
                   int duple_idx= find_table_index(k);
                   DF_Table_list.remove(duple_idx);
                   DF_Table_list.add(duple_idx,new DF_Table(k,v));
                }else{
                DF_Table tmpnxt = new DF_Table(k,v);
                DF_Table_list.add(tmpnxt);
                DF_Table_List_max_index_num++;
                }
            }
        }
        //id에 대응되는 key값이 있는지 확인하기 위해 static arrayList인 DF_table를 순회하며 찾는다
        //있을시 key가 가르치는 실제 value값 ,Node를 리턴한다. 없으면 null임
        public Node find_id_Node(String id) {
           for (DF_Table one : DF_Table_list) {
               String tmp_str = (String) one.getKey();
               if(tmp_str.equals(id)) {
                   return (Node) one.getNode();
               }

           }
           return null;
        }

        //테이블 노드가 비었는지 확인 어레이리스트 내장함수 쓴다.
        public boolean isTableNull(){
            return (DF_Table_list.isEmpty());
        }

    }
```

car 내부에 있는 operand가 Id노드일 때 처리하는 과정

```
if(head instanceof IdNode &&!(head instanceof QuoteNode)) {
    Node tmp = lookupTable(head.toString());
    //tmp가 null이 아닌 것은 정의된 값이 있다는 것이므로 그값을 대체해줌
    if(tmp !=null) {
        head =  ((ListNode)tmp);
        FunctionNode fnode = new FunctionNode();
        //( car a ) a = '( 1 2 3 4 5 )  이런상황 이므로 그냥 리턴하면서재귀로 한번더 호출해줌.
        fnode.setValue(FunctionNode.FunctionType.CAR.tokenType());
        return runFunction(fnode,(ListNode)head);
    }
    }
```

대부분 다 비슷함. 함수에 따라 재귀 호출이 있을 수도 있고 없을 수도 있음.

구현결과

```
$ (define a 4 )
...> 4
$ a
...> 4
$ (+ a 4 )
...> 8
$ (define a '( a b c ) )
...> '( a b c )
$ a
...> '( a b c )
$ (car a )
...> 'a
$ (cdr a )
...> '( b c )
$ ( car '( a b c d e ) )
...> 'a
$ (car '(cdr '( 1 2 3 ) ) )
...> 'CDR
$ a
...> '( a b c )
$ (define b (+ 3 4 ) )
...> 7
$ b
...> 7
$ (cons b a )
...> '( 7 a b c )
$ (- b 44 )
...> -37
$ ( > b 44 )
...> #F
$ ( eq? b 44 )
...> #F
$ ( eq? b 7 )
...> #T
$ (eq? 'b '7 )
...> #F
$ b
...> 7
```

```
$ (define b 7 )
...> 7
$ b
...> 7
$ (atom? 'b )
...> #T
$ (define c '( ) )
...> '( )
$ (null? c )
...> #T
$ (define a '( a b c ) )
...> '( a b c )
$ a
...> '( a b c )
$ (atom? 'a )
...> #T
$ (atom? a )
...> #F
$ b
...> 7
$ (cond ( (< b 10 ) b ) ( ( > 1 2 ) #F ) )
...> 7
$ ( + ( - b 44 ) b )
...> -30
```

```
$ (define ct ( car '( 4 3 2 ) ) )
...> 4
$ (define cd ( cdr '( 4 3 2 ) ) )
...> '( 3 2 )
$ ( cons ct cd )
...> '( 4 3 2 )
```

느낀점:

정말 뿌듯했습니다. 변수를 정의하고 사용한다는 게 이렇게 어려운 일인지 잘 몰랐습니다.

다시한번 사용하는 언어들을 되돌아보는 시간이었습니다.