컴파일러 java CUP 과제

컴퓨터정보공학부 2019202009 서여지

1. Parser의 동작

   입력으로 주어진 계산식의 계산 결과를 출력한다. 여러 개의 문장이 입력되었을 때 각각의 결과 값을 순서대로 출력한다.

2. Grammar

   여러 문장을 ; 으로 구분하는 사칙연산(+,-,*,/), 괄호(()), 대입(=)연산을 다음의 Grammar를 이용하여 파싱한다.

   StmtList -> empty | StmtList Stmt //여러 문장

   Stmt -> IDENT = Expr ; | Expr ; //대입문과 계산문

   Expr -> Expr + Term | Expr – Term | Term

   Term -> Term * Factor | Term / Factor | Factor

   Factor -> (Expr) | NUMBER | IDENT

   Terminal: +, -, *, /, =, ;

   Terminal: NUMBER=[0-9]+, IDENT=[A-Za-z][A-Za-z0-9]*

   Non-Terminal: StmtList, Stmt, Expr, Term, Factor

3. Java CUP 입력 소스

```
import java_cup.runtime.*;
import java.util.*; //hashtable

action code {:
    Hashtable table = new Hashtable(); // IDENT 저장
:}

parser code  {:
// 사용자 정의 코드
    Scanner scanner; //lexical analyzer
    parser(String expr){
        this(); //생성자
        scanner = new Scanner (new java.io.StringReader(expr));
    }

:};

scan with {: return scanner.next_token(); :}
```

대입문 처리를 위한 Hashtable 사용, lexical analyzer 할당

Scan에 next_token메소드 사용

```
// terminals & non terminals
terminal PLUS, MINUS, DIV, MULT, LPAREN, RPAREN, SEMICOLON, ASSIGN;
terminal Integer NUMBER;
terminal String IDENT;

non terminal Integer StmtList, Stmt, Expr, Term, Factor;
```

사용한 terminal, non terminal 선언

이하 grammar

```
StmtList ::=    //EMPTY
        |
            StmtList:stmtlist Stmt:stmt
            {:
            :}
        ;
```

StmtList: empty ~ 여러 개의 문장 인식

```
Stmt ::=    IDENT:ident ASSIGN Expr:expr SEMICOLON
            {:
            table.put(ident, expr);
            :}
        |
            Expr:expr SEMICOLON
            {:
            RESULT = expr;
            System.out.println(RESULT);
            :}
        ;
```

대입문 구조 IDENT = Expr ; hashtable에 식별자와 값을 저장한다.

계산문 구조 Eexpr ; 계산문의 계산 결과를 출력한다.

```
Expr ::=    Expr:expr PLUS Term:term
            {:
            RESULT = (expr + term);
            :}
        |
            Expr:expr MINUS Term:term
            {:
            RESULT = (expr - term);
            :}
        |
            Term:term
            {:
            RESULT = term;
            :}
        ;
```

덧셈, 뺄셈 계산과 Term을 그대로 받는 동작

```
Term ::=    Term:term MULT Factor:factor
            {:
            RESULT = (term * factor);
            :}
        |
            Term:term DIV Factor:factor
            {:
            RESULT = (term / factor);
            :}
        |
            Factor:factor
            {:
            RESULT = factor;
            :}
        ;
```

곱셈, 나눗셈 계산과 Factor를 그대로 받는 동작

```
80  Factor::=   LPAREN Expr:expr RPAREN
81              {:
82              RESULT = expr;
83              :}
84          |
85              NUMBER:value
86              {:
87              RESULT = value;
88              :}
89          |
90              IDENT:ident
91              {:
92              Integer value = (Integer) table.get(ident);
93              if(value==null){
94                  parser.report_error("Undeclared Identifier"+ident,
95                  new Symbol(sym.IDENT, identleft, identright, ident));
96                  value = (Integer)0;
97              }
98              RESULT = value;
99              :}
100         ;
```

(괄호)동작과 NUMBER의 값을 받는 동작

IDENT의 값을 Hashtable에서 읽어오거나, 새로 저장하는 동작

4. Jflex 입력 소스

```
import java_cup.runtime.*;

%%

%class Scanner
%cup

```

동작에 필요한 파일 import, lexical analyzer의 class 이름을 Scanner로 정함, java cup을 사용함

```
%{
// 사용자 정의 코드
  private Symbol symbol(int type) {
    return new Symbol(type, yyline+1, yycolumn+1);
  }

  private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline+1, yycolumn+1, value);
  }

%}
```

Symbol을 할당하여 반환하는 함수 정의 – type만 있는 경우와 type과 value가 모두 있는 경우

```
//토큰 정의
NUMBER = [0-9]+
IDENT = [A-Za-z][A-Za-z0-9]*

SEMICOLON = ";"
ASSIGN = "="
PLUS = "+"
MINUS = "-"
MULT = "*"
DIV = "/"
LPAREN = "("
RPAREN = ")"
```

사용한 토큰 정의

```
35  %%
36  //인식했을 때 동작
37  <YYINITIAL> {
38    {NUMBER}  { return symbol(sym.NUMBER, Integer.valueOf(yytext())); }
39    {IDENT}   { return symbol(sym.IDENT, String.valueOf(yytext())); }
40
41    ";"       { return symbol(sym.SEMICOLON); }
42    "="       { return symbol(sym.ASSIGN); }
43    "+"       { return symbol(sym.PLUS); }
44    "-"       { return symbol(sym.MINUS); }
45    "*"       { return symbol(sym.MULT); }
46    "/"       { return symbol(sym.DIV); }
47    "("       { return symbol(sym.LPAREN); }
48    ")"       { return symbol(sym.RPAREN); }
49  }
```

각각의 토큰을 인식했을 때 위에서 정의한 symbol함수를 호출하여 동작함

5. 실행 예

```
C:₩Program_A₩java-cup-bin-11b-20160615>java Main
3+5;x=4;4-(x+3)*2;
8
-10

C:₩Program_A₩java-cup-bin-11b-20160615>java Main
a=365;b=a/5;b;
73

C:₩Program_A₩java-cup-bin-11b-20160615>java Main
a=31;b=13;a+3-(2+4*3)/2-1;b*3-2;
26
37

C:₩Program_A₩java-cup-bin-11b-20160615>java Main


C:₩Program_A₩java-cup-bin-11b-20160615>java Main
3;
3
```

[참고자료]

Chapter4 Parsing Using Java CUP

https://www.cs.auckland.ac.nz/courses/compsci330s1c/lectures/330ChaptersPDF/Chapt4.pdf

CUP User's Manual

http://www2.cs.tum.edu/projects/cup/docs.php

JFlex User's Manual

https://jflex.de/manual.html