|  |
| --- |
| 컴파일러 01분반 |

-term project 2 –



|  |  |
| --- | --- |
| 과목명 | 컴파일러 |
| 교수명 | 김효수 교수님 |
| 제출일 | 2022.06.11 |
| 학  번 | 20200453, 20201501 |
| 학  과 | 소프트웨어학부 |
| 이  름 | 김세진, 김은솔 |

**1. CFG G**

00: S → CODE

01: CODE → VDECL CODE | FDECL CODE | ϵ

02: VDECL → vtype id semi

03: FDECL → vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace

04: ARG → vtype id MOREARGS | ϵ

05: MOREARGS → comma vtype id MOREARGS | ϵ

06: BLOCK → STMT BLOCK | ϵ

07: STMT → VDECL | id assign RHS semi

08: STMT → if lparen COND rparen lbrace BLOCK rbrace else lbrace BLOCK rbrace

09: STMT → while lparen COND rparen lbrace BLOCK rbrace

10: RHS → EXPR | literal

11: EXPR → TERM addsub EXPR | TERM

12: TERM → FACTOR multdiv TERM | FACTOR

13: FACTOR → lparen EXPR rparen | id | num

14: COND → FACTOR comp FACTOR

15: RETURN → return FACTOR semi

**Terminals (18)**

1. vtype for the types of variables and functions

2. num for signed integers

3. literal for literal strings

4. id for the identifiers of variables and functions

5. if, else, while, and return for if, else, while, and return statements respectively

6. addsub for + and - arithmetic operators

7. multdiv for \* and / arithmetic operators

8. assign for assignment operators

9. comp for comparison operators

10. semi and comma for semicolons and commas respectively

11. lparen, rparen, lbrace, and rbrace for (, ), {, and } respectively

**Non-terminals (13)**

CODE, VDECL, FDECL, ARG, MOREARGS, BLOCK, STMT, RHS, EXPR, TERM, FACTOR, COND,

RETURN

**Start symbol**: CODE

✓CFG Changes

00: S → CODE

Start Symbol로 향하는 S → CODE dummy 규칙을 00번에 추가하였다.

**2-1. First / Follow Table**

|  |  |  |
| --- | --- | --- |
| **FIRST / FOLLOW table** | | |
| **Nonterminal** | **FIRST** | **FOLLOW** |
| S | {'',vtype} | {$} |
| CODE | {'',vtype} | {$} |
| VDECL | {vtype} | {$,vtype,return,id,if,while,rbrace} |
| FDECL | {vtype} | {$,vtype} |
| ARG | {vtype,''} | {rparen} |
| MOREARGS | {comma,''} | {rparen} |
| BLOCK | {'',vtype,id,if,while} | {return,rbrace} |
| STMT | {vtype,id,if,while} | {return,vtype,id,if,while,rbrace} |
| RHS | {literal,lparen,id,num} | {semi} |
| EXPR | {lparen,id,num} | {semi,rparen} |
| TERM | {lparen,id,num} | {addsub,semi,rparen} |
| FACTOR | {lparen,id,num} | {multdiv,addsub,semi,comp,rparen} |
| COND | {lparen,id,num} | {rparen} |
| RETURN | {return} | {rbrace} |

**2-2. SLR Table[[1]](#footnote-1)**



테이블의 크기가 매우 큰 관계로, First/Follow table, SLR parsing table, SLR closure table을 SLRTable.xlsx에 첨부하였다.

**3. CODE IMPLEMENTATION**

개발 환경: Python 3.8 version

Lexical Analyzer 수정 사항: Term Project 1에서 제출한 Lexical Analyzer의 코드 중 맨 앞에 0이 와도 뒤에 있는 int를 읽는 오류가 있어 수정하였다. 또한 Syntax Analyzer terminal에 맞추기 위해 기존 Lexical Analyzer의 토큰을 모두 소문자로 변경하였고, operator로 선언한 토큰을 +,-는 addsub, \*,/은 multdiv로 나누어 선언하였다. 수정한 Lexical Analyzer 코드와 새롭게 구현한 SyntaxAnalyzer를 합쳐 하나의 anlyzer.py 파일로 제출하였다.

3-1. Rules 정의

Reduce를 수행하기 위해서는 생성 규칙을 정의할 필요가 있다. 생성 규칙의 순서를 key로, 생성 규칙을 value로 갖는 dictionary를 이용하여 규칙을 정의하였다.

    # *rules*

    RULES = {'0': 'S → CODE',

             '1': 'CODE → VDECL CODE',

             '2': 'CODE → FDECL CODE',

             '3': 'CODE → epsilon',

             '4': 'VDECL → vtype id semi',

             '5': 'FDECL → vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace',

             '6': 'ARG → vtype id MOREARGS',

             '7': 'ARG → epsilon',

             '8': 'MOREARGS → comma vtype id MOREARGS',

             '9': 'MOREARGS → epsilon',

             '10': 'BLOCK → STMT BLOCK',

             '11': 'BLOCK → epsilon',

             '12': 'STMT → VDECL',

             '13': 'STMT → id assign RHS semi',

             '14': 'STMT → if lparen COND rparen lbrace BLOCK rbrace else lbrace BLOCK rbrace',

             '15': 'STMT → while lparen COND rparen lbrace BLOCK rbrace',

             '16': 'RHS → EXPR',

             '17': 'RHS → literal',

             '18': 'EXPR → TERM addsub EXPR',

             '19': 'EXPR → TERM',

             '20': 'TERM → FACTOR multdiv TERM',

             '21': 'TERM → FACTOR',

             '22': 'FACTOR → lparen EXPR rparen',

             '23': 'FACTOR → id',

             '24': 'FACTOR → num',

             '25': 'COND → FACTOR comp FACTOR',

             '26': 'RETURN → return FACTOR semi'}

3-2. SLR Table 정의

SLR Table은 Shift, GOTO, Reduce operation을 포함하고 있으며, 그 외의 부분은 Error로 처리한다. 위에 언급한 CFG에 의하면 Shift, GOTO, Reduce operation은 SLR Table의 일부만 차지하므로, 빈 부분을 list에 저장하는 경우 메모리 낭비가 심해진다. 따라서 list를 사용하여 SLR Table을 표현하되, Action과 GOTO Table의 열을 key 값으로 하는 Dictionary를 이용하여 구현하였다.

Action table의 열에 해당하는 terminal 들에는 Shift, Reduce operation을 str로 저장하고, GOTO table의 열에 해당하는 생성 규칙들은 생성 규칙의 번호를 value로 저장하였다.

# *SLR table*

    SLR\_TABLE = [{'vtype': 's4', '$': 'r3', 'CODE': 1, 'VDECL': 2, 'FDECL': 3},

                 {'$': 'acc'},

                 {'vtype': 's4', '$': 'r3', 'CODE': 5, 'VDECL': 2, 'FDECL': 3},

                 {'vtype': 's4', '$': 'r3', 'CODE': 6, 'VDECL': 2, 'FDECL': 3},

                 {'id': 's7'},

                 {'$': 'r1'},

                 {'$': 'r2'},

                 {'semi': 's8', 'lparen': 's9'},

                 {'vtype': 'r4', 'id': 'r4', 'rbrace': 'r4', 'if': 'r4', 'while': 'r4', 'return': 'r4', '$': 'r4'},

                 {'vtype': 's11', 'rparen': 'r7', 'ARG': 10},

                 {'rparen': 's12'},

                 {'id': 's13'},

                 {'lbrace': 's14'},

                 {'rparen': 'r9', 'comma': 's16', 'MOREARGS': 15},

                 {'vtype': 's23', 'id': 's20', 'rbrace': 'r11', 'if': 's21', 'while': 's22', 'return': 'r11', 'VDECL': 19, 'BLOCK': 17, 'STMT': 18},

                 {'rparen': 'r6'},

                 {'vtype': 's24'},

                 {'return': 's26', 'RETURN': 25},

                 {'vtype': 's23', 'id': 's20', 'rbrace': 'r11', 'if': 's21', 'while': 's22', 'return': 'r11', 'VDECL': 19, 'BLOCK': 27, 'STMT': 18},

                 {'vtype': 'r12', 'id': 'r12', 'rbrace': 'r12', 'if': 'r12', 'while': 'r12', 'return': 'r12'},

                 {'assign': 's28'},

                 {'lparen': 's29'},

                 {'lparen': 's30'},

                 {'id': 's31'},

                 {'id': 's32'},

                 {'rbrace': 's33'},

                 {'id': 's36', 'lparen': 's35', 'num': 's37', 'FACTOR': 34},

                 {'rbrace': 'r10', 'return': 'r10'},

                 {'id': 's36', 'lparen': 's35', 'literal': 's40', 'num': 's37', 'RHS': 38, 'EXPR': 39, 'TERM': 41, 'FACTOR': 42},

                 {'id': 's36', 'lparen': 's35', 'num': 's37', 'FACTOR': 44, 'COND': 43},

                 {'id': 's36', 'lparen': 's35', 'num': 's37', 'FACTOR': 44, 'COND': 45},

                 {'semi': 's8'},

                 {'rparen': 'r9', 'comma': 's16', 'MOREARGS': 46},

                 {'vtype': 'r5', '$': 'r5'},

                 {'semi': 's47'},

                 {'id': 's36', 'lparen': 's35', 'num': 's37', 'EXPR': 48, 'TERM': 41, 'FACTOR': 42},

                 {'semi': 'r23', 'rparen': 'r23', 'addsub': 'r23', 'multdiv': 'r23', 'comp': 'r23'},

                 {'semi': 'r24', 'rparen': 'r24', 'addsub': 'r24', 'multdiv': 'r24', 'comp': 'r24'},

                 {'semi': 's49'},

                 {'semi': 'r16'},

                 {'semi': 'r17'},

                 {'semi': 'r19', 'rparen': 'r19', 'addsub': 's50'},

                 {'semi': 'r21', 'rparen': 'r21', 'addsub': 'r21', 'multdiv': 's51'},

                 {'rparen': 's52'}, {'comp': 's53'}, {'rparen': 's54'},

                 {'rparen': 'r8'},

                 {'rbrace': 'r26'},

                 {'rparen': 's55'},

                 {'vtype': 'r13', 'id': 'r13', 'rbrace': 'r13', 'if': 'r13', 'while': 'r13', 'return': 'r13'},

                 {'id': 's36', 'lparen': 's35', 'num': 's37', 'EXPR': 56, 'TERM': 41, 'FACTOR': 42},

                 {'id': 's36', 'lparen': 's35', 'num': 's37', 'TERM': 57, 'FACTOR': 42},

                 {'lbrace': 's58'},

                 {'id': 's36', 'lparen': 's35', 'num': 's37', 'FACTOR': 59},

                 {'lbrace': 's60'},

                 {'semi': 'r22', 'rparen': 'r22', 'addsub': 'r22', 'multdiv': 'r22', 'comp': 'r22'},

                 {'semi': 'r18', 'rparen': 'r18'},

                 {'semi': 'r20', 'rparen': 'r20', 'addsub': 'r20'},

                 {'vtype': 's23', 'id': 's20', 'rbrace': 'r11', 'if': 's21', 'while': 's22', 'return': 'r11', 'VDECL': 19, 'BLOCK': 61, 'STMT': 18},

                 {'rparen': 'r25'},

                 {'vtype': 's23', 'id': 's20', 'rbrace': 'r11', 'if': 's21', 'while': 's22', 'return': 'r11', 'VDECL': 19, 'BLOCK': 62, 'STMT': 18},

                 {'rbrace': 's63'},

                 {'rbrace': 's64'},

                 {'else': 's65'},

                 {'vtype': 'r15', 'id': 'r15', 'rbrace': 'r15', 'if': 'r15', 'while': 'r15', 'return': 'r15'},

                 {'lbrace': 's66'},

                 {'vtype': 's23', 'id': 's20', 'rbrace': 'r11', 'if': 's21', 'while': 's22', 'return': 'r11', 'VDECL': 19, 'BLOCK': 67, 'STMT': 18},

                 {'rbrace': 's68'},

                 {'vtype': 'r14', 'id': 'r14', 'rbrace': 'r14', 'if': 'r14', 'while': 'r14', 'return': 'r14'}]

✓문장의 끝을 의미하는 END\_MARK는 ‘$’로 선언해 주었다.

3-3. SyntaxAnalyzer 생성자 (\_\_init\_\_)

Lexical Analyzer의 결과 list를 저장하는 analyzer\_table, error가 난 부분을 저장하기 위한 error\_table, 에러가 발생한 줄을 저장할 error\_row를 정의한다.

SyntaxAnalyzer 객체를 생성하면 analyzer\_table 요소의 첫 번째 부분만 analyzer\_table에 추가한다. END\_MARK를 표현하기 위해 analyzer\_table 마지막 부분에 END\_MARK를 삽입한다.

error\_table, error\_row의 값도 함께 초기화한다.

vtype int

id main

lparen (

Lexical Analyzer 결과 값의 일부

analyzer\_table = []   # *lexical analyzer result table*

    error\_table = []   # *error table*

    error\_row = 1   # *에러가 발생하는 줄 저장*

    def \_\_init\_\_(self, analyzer\_table):

*for* row *in* analyzer\_table:   # *lexical analyzer result table 파싱*

            self.analyzer\_table.append(row[0])

        self.analyzer\_table.append(self.END\_MARK)   # *end mark 추가*

        self.error\_table = list(self.analyzer\_table)    # *error table 초기화*

3-4. Splitter, Stack 초기화

analyzer\_table의 길이가 1인 경우는 END\_MARK만 포함되어 있는 경우이므로 True를 리턴한다.

syntax\_stack은 list로 구현하였으며, start state인 0을 삽입한다. splitter\_position도 0으로 초기화한다.

    def *parse*(self):

*if* len(self.analyzer\_table) == 1:   # *end mark만 존재하는 경우 True*

*return* True

        syntax\_stack = [0]  # *slr stack*

        splitter\_position = 0   # *splitter position*

3-5. Error, Accept 판정

current\_state에는 항상 syntax\_stack[-1]을 저장하여 현재 state를 저장한다.

Next Symbol이 SLR\_TABLE에 존재하지 않는 경우는 Error로 판정하여 False를 리턴한다. SLR\_TABLE[current\_state][next\_terminal]이 ‘acc’인 경우는 dummy start symbol인 S 전까지 reduce 되었다는 뜻이므로 True를 리턴한다.

*while* True:

            current\_state = syntax\_stack[-1]  # *현재 위치*

            next\_terminal = self.analyzer\_table[splitter\_position]  # *다음 symbol 가져오기*

*if* next\_terminal not in self.SLR\_TABLE[current\_state].keys():   # *SLR\_TABLE에 없는 경우 False*

*return* False

*if* self.SLR\_TABLE[current\_state][next\_terminal] == 'acc':   # *acc에 도달하면 True*

*return* True

3-6. Shift

SLR Table의 값이 ‘s12’ 같은 형식인 경우 Shift를 수행한 후, 12를 stack에 넣으라는 것을 의미한다. Splitter를 한 칸 이동시키고, stack에 ‘s’ 뒤 숫자를 삽입한다.

*elif* self.SLR\_TABLE[current\_state][next\_terminal][0] == 's':  # *Shift 수행*

                splitter\_position += 1  # *splitter 이동*

                self.error\_row += 1  # *error\_row 증가*

                syntax\_stack.append(int(self.SLR\_TABLE[current\_state][next\_terminal][1:]))  # *stack에 next state 추가*

3-7 Reduce

SLR Table의 값이 ‘r7’ 같은 형식인 경우, 생성 규칙 7번으로 Reduce를 수행하라는 의미이다. 생성 규칙의 우단이 epsilon이 아닌 경우에만 생성 규칙에 포함된 값들을 analyzer\_table에서 제거한다. 생성 규칙의 우단이 epsilon인 경우 stack에서 pop을 수행하지 않는다. Reduce를 수행한 생성 규칙의 좌측 Nonterminal을 analyzer\_table에 추가하여 parsing을 수행한다.

current\_state를 재조정하고, 만약 Reduce를 수행한 Nonterminal이 SLR\_TABLE에 포함되지 않는다면 False를 리턴한다.

Reduce가 완료되면 SLR Table[current\_state]에서 reduce\_cfg\_rule[0]에 해당하는 값으로 GOTO를 진행한다.

*elif* self.SLR\_TABLE[current\_state][next\_terminal][0] == 'r':  # *Reduce 수행*

                reduce\_num = self.SLR\_TABLE[current\_state][next\_terminal][1:]  # *Reduce number 파싱*

                reduce\_cfg\_rule = self.RULES[reduce\_num].split()  # *해당 state의 Rule*

*for* i *in* range(len(reduce\_cfg\_rule) - 2):  # *Ex) A -> B인 경우 B만 사용*

*if* reduce\_cfg\_rule[2] != 'epsilon':  # *생성 규칙이 epsilon이 아닌 경우*

*del* self.analyzer\_table[splitter\_position - i - 1]  # *table에서 해당하는 요소 삭제*

                        syntax\_stack.pop()  # *stack에서 삭제*

*if* reduce\_cfg\_rule[2] != 'epsilon':  # *생성 규칙이 epsilon이 아닌 경우*

                    splitter\_position -= len(reduce\_cfg\_rule) - 3  # *splitter position 재조정*

*else*:  # *생성 규칙이 epsilon인 경우*

                    splitter\_position += 1  # *splitter position 1 증가*

                self.analyzer\_table.insert(splitter\_position - 1, reduce\_cfg\_rule[0])  # *reduce 수행*

                current\_state = syntax\_stack[-1]  # *current\_state 위치 수정*

*if* reduce\_cfg\_rule[0] not in self.SLR\_TABLE[current\_state].keys():  # *reduce한 rule이 SLR\_TABLE에 존재하지 않는 경우 False*

*return* False

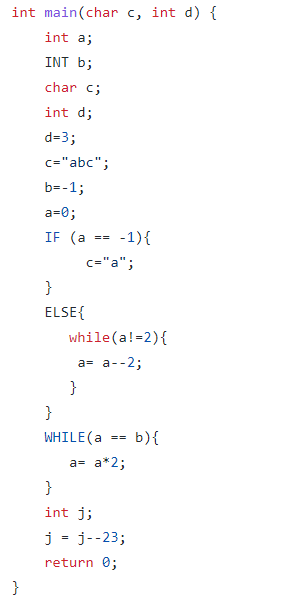
                syntax\_stack.append(self.SLR\_TABLE[current\_state][reduce\_cfg\_rule[0]])  # *GOTO(current\_state, reduce\_cfg\_rule[0]) 삽입*

4. Test Input File & Output

Lexical Analyzer, Syntax Analyzer 테스트를 위해 오류가 없는 hard\_test.c 파일을 대상으로 테스트를 진행하였다.

Linux Terminal에 “Python3 analyzer.py hard\_test.c”라고 입력하면, 문제가 없는 프로그램의 경우 hard\_test\_lexical.out이 생성되고 Terminal 창에 “This Program is accepted”라고 뜨는 것을 볼 수 있다.

Input File



텍스트, 스크린샷, 모니터, 노트북이(가) 표시된 사진

자동 생성된 설명

위 Input File의 두 번째 줄에 “int a;” 부분을 “int a=1;”로 변경하여 다시 테스트를 진행하였다. 과제 Description에 의해 변수는 초기화 없이 선언되어야 하므로 Syntax Rule에 어긋나는 경우이다.

텍스트, 스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

Lexical Analyzer에 의해서는 오류가 탐지되지 않으므로 정상적으로 hard\_test\_lexical.out이 생성되지만 Syntax Analyzer의 Rule에 맞지 않으므로 “This Program is Rejected”라는 메시지와 hard\_test\_error.out이 생성되는 것을 발견할 수 있다. 원인이 되는 Lexical Analyzer의 token 위치를 Error line으로 표시하였다.

1. <http://jsmachines.sourceforge.net/machines/slr.html> [↑](#footnote-ref-1)