# 프로그래밍언어 COMPILER PROJECT

제출 날짜 : 2018. 12.23

류호준 2016312332
이민영 2016310944
오용우 2016313958
정수림 2016310494

## 1. 문법 수정 사항

a) “vtype -> int | char | .”에서 “vtype -> int | char | void.”로 바꾸었다.
- 이는 epsilon을 넣을 시에 우리가 배운 어떤 parser로도 accept되지 않는다.

b) stat에서 epsilon으로 가는 경우를 삭제했다.
- stat->(epsilon)으로 갈 경우 slist -> slist가 되는 경우가 생겨서 무한 루프가 생길 가능성이 있다.

다음과 같이 문법을 수정함으로서 우리는 SLR로 parsing 할 수 있게 되었다.

## 2. Scanner

scanner는 원래 scanner - parser가 번갈아가면서 하는 대신, scanner가 모든 input을 통째로 tokenize하여 분류하고, 만약에 word, num에 해당하는 경우에는, 그 값까지 같이 들어있는 Input이라는 이름의 구조체를 전역변수에 해당하는 code라는 이름의 Input 구조체 배열에 넣어주는 형태를 취했다.
Non - terminal은 다음과 같이 numbering하여 구조체에 넣었다.

word : 0
( : 1
) : 2
; : 3
, : 4
int : 5
char : 6
void : 7
{ : 8
} : 9
IF : 10
THEN : 11

ELSE : 12
WHILE : 13
= : 14
RETURN: 15
> : 16
== : 17
+ : 18
* : 19
num : 20
$ : 21

$는 맨 마지막의 구조체에 넣어주는 형태를 취했다.
수정한 문법으로 우리는 char test(int alpha){ char a; int b; a = 5; b = 4; IF(a>b) THEN { a = b; } RETURN 0; }를 예시로 만들었다. 이후 우리가 구현한 Scanner로 차례로 따라가면서 유효성 검사를 하였다. 그 과정은 아래와 같다. 결론적으로 결과 값이 잘 출력되었다.
다음은 우리가 만들어낸 함수에서 Scanning 하는 과정을 나타낸 것이다. 좌측은 예시 코드고, 우측은 이를 parsing한 결과를 보여준다.
밑 comment들은 알아서 배제된 상태로 scanning이 되었음을 확인할 수 있다.

```
char test(alpha,beta){
        char a, c;
        int b;
        a = 5;#This
        b = 4;
        IF a>b THEN
        {#IS
        int d;
        d = 2;
                a = b;
        }
    ELSE { b = a; }
    c = a + b;
    #Single Line
        RETURN 0;
    #comment
}
```

```
type : 6
type : 0, string : test
type : 1
type : 0, string : alpha
type : 4
type : 0, string : beta
type : 2
type : 8
type : 6
type : 0, string : a
type : 4
type : 0, string : c
type : 3
type : 5
type : 0, string : b
type : 3
type : 0, string : a
type : 14
type : 20, value : 5
type : 3
```

# 2. PARSER

Parser는 Scanner에서부터 token을 입력으로 받아, 입력으로 들어온 스트링이 문법적으로 옳은 문장인지 확인한다. Context-free Grammar을 위한 Syntax analysis 방법으로 Top-down 방식과 Bottom-up 방식이 있다. Bottom-up방식이 Top-down 방식보다 더 강력한 방법이므로 우리는 Bottom-up 방식을 사용하였다. 따라서 SLR Parser을 생각하였다. 테이블 완성하기 위해 우리는 칠판에 FIRST와 FOLLOW를 찾고 GOTO Graph를 그려나갔다. 하지만 사진으로 남기기전에 팀원 중 한명의 실수로 테이블이 지워지는 사고가 발생하여 여기에는 과정에 대한 내용을 담지 못하였다.

## SLR table

GOTO Graph를 이용하여 다음과 같이 SLR Parsing table을 만들었다. 56x34 크기의 테이블이기 때문에 나누어서 정리하였다.

### [0~9] Parsing Table : Symbols(1)

| State | word | ( | ) | ; | , | int | char | void | { | } |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | shift(3) | shift(4) | shift(5) | | |
| **1** | | | | | | | | | | |
| **2** | shift(6) | | | | | | | | | |
| **3** | reduce(*vtype*→ int) | | | | | | | | | |
| **4** | reduce(*vtype*→ char) | | | | | | | | | |
| **5** | reduce(*vtype*→ void) | | | | | | | | | |
| **6** | | shift(7) | | | | | | | | |
| **7** | shift(9) | | | | | | | | | |
| **8** | | | shift(10) | | shift(11) | | | | | |
| **9** | | | reduce(*words*→ word) | reduce(*words*→ word) | reduce(*words*→ word) | | | | | |
| **10** | reduce(*block*→ ε) | | | | | | | | shift(13) | reduce(*block*→ ε) |
| **11** | shift(14) | | | | | | | | | |
| **12** | | | | | | | | | | |
| **13** | reduce(*decls*→ ε) | | | | | reduce(*decls*→ ε) | reduce(*decls*→ ε) | reduce(*decls*→ ε) | | |
| **14** | | | reduce(*words*→ *words* ,word) | reduce(*words*→ *words* ,word) | reduce(*words*→ *words* ,word) | | | | | |
| **15** | shift(22) | | | | | shift(3) | shift(4) | shift(5) | | |
| **16** | shift(22) | | | | | | | | | shift(24) |
| **17** | reduce(*decls*→ *declsdecl*) | | | | | reduce(*decls*→ *declsdecl*) | reduce(*decls*→ *declsdecl*) | reduce(*decls*→ *declsdecl*) | | |
| **18** | reduce(*slist*→ *stat*) | | | | | | | | | reduce(*slist*→ *stat*) |
| **19** | shift(9) | | | | | | | | | |
| **20** | shift(32) | | | | | | | | | |
| **21** | shift(32) | | | | | | | | | |
| **22** | | shift(35) | | | | | | | | |
| **23** | shift(32) | | | | | | | | | |
| **24** | reduce(*block*→ { *declsslist* }) | | | | | | | | | reduce(*block*→ { *declsslist* }) |
| **25** | reduce(*slist*→ *slist stat*) | | | | | | | | | reduce(*slist*→ *slist stat*) |
| **26** | | | | shift(37) | shift(11) | | | | | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **27** | | | | | | | | | | |
| **28** | | | | | | | | | | |
| **29** | reduce(*expr*→ *term*) | | | reduce(*expr*→ *term*) | | | | | reduce(*expr*→ *term*) | reduce(*expr*→ *term*) |
| **30** | reduce(*term*→ *fact*) | | | reduce(*term*→ *fact*) | | | | | reduce(*term*→ *fact*) | reduce(*term*→ *fact*) |
| **31** | reduce(*fact*→ num) | | | reduce(*fact*→ num) | | | | | reduce(*fact*→ num) | reduce(*fact*→ num) |
| **32** | reduce(*fact*→ word) | | | reduce(*fact*→ word) | | | | | reduce(*fact*→ word) | reduce(*fact*→ word) |
| **33** | reduce(*block*→ ε) | | | | | | | | shift(13) | reduce(*block*→ ε) |
| **34** | shift(32) | | | | | | | | | |
| **35** | shift(9) | | | | | | | | | |
| **36** | | | | shift(46) | | | | | | |
| **37** | reduce(*decl*→ *vtypewords* ;) | | | | | reduce(*decl*→ *vtypewords* ;) | reduce(*decl*→ *vtypewords* ;) | reduce(*decl*→ *vtypewords* ;) | | |
| **38** | reduce(*block*→ ε) | | | | | | | | shift(13) | reduce(*block*→ ε) |
| **39** | shift(32) | | | | | | | | | |
| **40** | shift(32) | | | | | | | | | |
| **41** | shift(32) | | | | | | | | | |
| **42** | shift(32) | | | | | | | | | |
| **43** | reduce(*stat*→ WHILE*cond block*) | | | | | | | | | reduce(*stat*→ WHILE*cond block*) |
| **44** | | | | shift(52) | | | | | | |
| **45** | | | shift(53) | | shift(11) | | | | | |
| **46** | reduce(*stat*→ RETURN*expr* ;) | | | | | | | | | reduce(*stat*→ RETURN*expr* ;) |
| **47** | | | | | | | | | | |
| **48** | reduce(*cond*→ *expr* > *expr*) | | | | | | | | reduce(*cond*→ *expr* > *expr*) | reduce(*cond*→ *expr* > *expr*) |
| **49** | reduce(*cond*→ *expr* == *expr*) | | | | | | | | reduce(*cond*→ *expr* == *expr*) | reduce(*cond*→ *expr* == *expr*) |
| **50** | reduce(*expr*→ *term* + *term*) | | | reduce(*expr*→ *term* + *term*) | | | | | reduce(*expr*→ *term* + *term*) | reduce(*expr*→ *term* + *term*) |
| **51** | reduce(*term*→ *fact* * *fact*) | | | reduce(*term*→ *fact* * *fact*) | | | | | reduce(*term*→ *fact* * *fact*) | reduce(*term*→ *fact* * *fact*) |
| **52** | reduce(*stat*→ word = *expr* ;) | | | | | | | | | reduce(*stat*→ word = *expr* ;) |
| **53** | | | | shift(55) | | | | | | |
| **54** | reduce(*block*→ ε) | | | | | | | | shift(13) | reduce(*block*→ ε) |
| **55** | reduce(*stat*→ word ( *words* ) ;) | | | | | | | | | reduce(*stat*→ word ( *words* ) ;) |
| **56** | reduce(*stat*→ IF *cond* THEN *block* ELSE *block*) | | | | | | | | | reduce(*stat*→ IF *cond* THEN *block* ELSE *block*) |

## [10~21] Parsing Table : Symbols(2)

| IF | THEN | ELSE | WHILE | = | RETURN | > | == | + | * | num | $ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| | | | | | | | | | | | accept |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| reduce( *block*→ ε) | | reduce( *block*→ ε) | reduce( *block*→ ε) | | reduce( *block*→ ε) | | | | | | reduce( *block*→ ε) |
| | | | | | | | | | | | |
| | | | | | | | | | | | reduce( *prog*→ *vtype*word ( *words* ) *block*) |
| reduce( *decls*→ ε) | | | reduce( *decls*→ ε) | | reduce( *decls*→ ε) | | | | | | |
| | | | | | | | | | | | |
| shift(20 ) | | | shift(21 ) | | shift(23 ) | | | | | | |
| shift(20 ) | | | shift(21 ) | | shift(23 ) | | | | | | |
| reduce( *decls*→ *declsdecl*) | | | reduce( *decls*→ *declsdecl*) | | reduce( *decls*→ *declsdecl*) | | | | | | |
| reduce( *slist*→ *stat*) | | | reduce( *slist*→ *stat*) | | reduce( *slist*→ *stat*) | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | shift(31 ) | |
| | | | | | | | | | | shift(31 ) | |
| | | | | shift(34 ) | | | | | | | |
| | | | | | | | | | | shift(31 ) | |
| reduce( *block*→ { *declsslist* }) | | reduce( *block*→ { *declsslist* }) | reduce( *block*→ { *declsslist* }) | | reduce( *block*→ { *declsslist* }) | | | | | | reduce( *block*→ { *declsslist* }) |
| reduce( *slist*→ *slist stat*) | | | reduce( *slist*→ *slist stat*) | | reduce( *slist*→ *slist stat*) | | | | | | |
| | | | | | | | | | | | |
| | shift(38 ) | | | | | | | | | | |
| | | | | | | shift(39 ) | shift(40 ) | | | | |
| reduce( *expr*→ *term*) | reduce( *expr*→ *term*) | | reduce( *expr*→ *term*) | | reduce( *expr*→ *term*) | reduce( *expr*→ *term*) | reduce( *expr*→ *term*) | shift(41 ) | | | |
| reduce( *term*→ *fact*) | reduce( *term*→ *fact*) | | reduce( *term*→ *fact*) | | reduce( *term*→ *fact*) | reduce( *term*→ *fact*) | reduce( *term*→ *fact*) | reduce( *term*→ *fact*) | shift(42 ) | | |
| reduce( *fact*→ num) | reduce( *fact*→ num) | | reduce( *fact*→ num) | | reduce( *fact*→ num) | reduce( *fact*→ num) | reduce( *fact*→ num) | reduce( *fact*→ num) | reduce( *fact*→ num) | | |
| reduce( *fact*→ word) | reduce( *fact*→ word) | | reduce( *fact*→ word) | | reduce( *fact*→ word) | reduce( *fact*→ word) | reduce( *fact*→ word) | reduce( *fact*→ word) | reduce( *fact*→ word) | | |
| reduce( *block*→ ε) | | reduce( *block*→ ε) | reduce( *block*→ ε) | | reduce( *block*→ ε) | | | | | | reduce( *block*→ ε) |
| | | | | | | | | | | shift(31 ) | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| reduce( *decl*→ *vtypewords* ;) | | | reduce( *decl*→ *vtypewords* ;) | | reduce( *decl*→ *vtypewords* ;) | | | | | | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| reduce(*block*→ ε) | | reduce(*block*→ ε) | reduce(*block*→ ε) | | reduce(*block*→ ε) | | | | | | reduce(*block*→ ε) |
| | | | | | | | | | | shift(31) | |
| | | | | | | | | | | shift(31) | |
| | | | | | | | | | | shift(31) | |
| | | | | | | | | | | shift(31) | |
| reduce(*stat*→ WHILE *cond block*) | | | reduce(*stat*→ WHILE *cond block*) | | reduce(*stat*→ WHILE *cond block*) | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| reduce(*stat*→ RETURN *expr* ;) | | | reduce(*stat*→ RETURN *expr* ;) | | reduce(*stat*→ RETURN *expr* ;) | | | | | | |
| | | shift(54) | | | | | | | | | |
| reduce(*cond*→ *expr* > *expr*) | reduce(*cond*→ *expr* > *expr*) | | reduce(*cond*→ *expr* > *expr*) | | reduce(*cond*→ *expr* > *expr*) | | | | | | |
| reduce(*cond*→ *expr* == *expr*) | reduce(*cond*→ *expr* == *expr*) | | reduce(*cond*→ *expr* == *expr*) | | reduce(*cond*→ *expr* == *expr*) | | | | | | |
| reduce(*expr*→ *term* + *term*) | reduce(*expr*→ *term* + *term*) | | reduce(*expr*→ *term* + *term*) | | reduce(*expr*→ *term* + *term*) | reduce(*expr*→ *term* + *term*) | reduce(*expr*→ *term* + *term*) | | | | |
| reduce(*term*→ *fact* ∗ *fact*) | reduce(*term*→ *fact* ∗ *fact*) | | reduce(*term*→ *fact* ∗ *fact*) | | reduce(*term*→ *fact* ∗ *fact*) | reduce(*term*→ *fact* ∗ *fact*) | reduce(*term*→ *fact* ∗ *fact*) | reduce(*term*→ *fact* ∗ *fact*) | | | |
| reduce(*stat*→ word = *expr* ;) | | | reduce(*stat*→ word = *expr* ;) | | reduce(*stat*→ word = *expr* ;) | | | | | | |
| | | | | | | | | | | | |
| reduce(*block*→ ε) | | reduce(*block*→ ε) | reduce(*block*→ ε) | | reduce(*block*→ ε) | | | | | | reduce(*block*→ ε) |
| reduce(*stat*→ word ( *words* ) ;) | reduce(*stat*→ word ( *words* ) ;) | | | reduce(*stat*→ word ( *words* ) ;) | | reduce(*stat*→ word ( *words* ) ;) | | | | | |
| reduce(*stat*→ IF *cond* THEN *block* ELSE *block*) | reduce(*stat*→ IF *cond* THEN *block* ELSE *block*) | | | reduce(*stat*→ IF *cond* THEN *block* ELSE *block*) | | reduce(*stat*→ IF *cond* THEN *block* ELSE *block*) | | | | | |

[22~33] : Parsing Table : GOTO Table

| prog | vtype | words | block | decls | decl | slist | stat | cond | expr | term | fact |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | 8 | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | 12 | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | 15 | | | | | | | |
| | | | | | | | | | | | |
| | 19 | | | | 17 | 16 | 18 | | | | |
| | | | | | | | 25 | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | 26 | | | | | | | | | |
| | | | | | | | | 27 | 28 | 29 | 30 |
| | | | | | | | | 33 | 28 | 29 | 30 |
| | | | | | | | | | | | |
| | | | | | | | | | 36 | 29 | 30 |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | 43 | | | | | | | | |
| | | | | | | | | | 44 | 29 | 30 |
| | | 45 | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | 47 | | | | | | | | |
| | | | | | | | | | 48 | 29 | 30 |
| | | | | | | | | | 49 | 29 | 30 |
| | | | | | | | | | | 50 | 30 |
| | | | | | | | | | | | 51 |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | 56 | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |

# 3. Code Generator

Parser를 통해서 도출된 AST를 기반으로 Assembly code를 작성하였다. 기본적으로, 변수가 등장할 때마다 Register에 저장하게 하였다. 그 이유로는 Conditional statement에서 변수를 가리키는 Register가 루프 혹은 조건에 따라 달라질 수 있기 때문이다.

완성된 Assembly code는 Chaitin 알고리즘을 사용하여 사용될 register의 개수를 줄였다. Chaitin의 k를 선택하는 방법은 이전의 Assembly code에서 사용된 register의 개수를 N이라고 할 때, 1과 N 사이에서 Chaitin 알고리즘을 만족하는 최소의 k를 이진 탐색을 통해 찾았다.

더불어, 2가지의 instruction을 추가하였다.

1) expr == expr 의 경우를 위해 "EQ Reg#1 Reg#2 Reg#3" 를 추가했다. 이는 Reg#2가 Reg#3과 같다면 Reg#1이 0이 되고, 그렇지 않다 면 Reg#1는 1이 되는 Instruction이다.

2) RETURN의 경우를 맞추어 주기 위해서 RET Reg#1 이라는 instruction을 추가했다. 이는 Reg#1의 값을 return하는 instruction이다.

다음 그림은 위의 test 함수 예제 code에서 assembly로 변환한 후, chaitin algorithm을 통해서 사용될 register의 개수를 최소화한 그림이다. 레지스터 출력 후, 마지막 줄에는 pseudo-assembly code에 사용된 register의 개수를 알려준다.

```
BEGIN test
  LD    Reg#1, 5
  ST    Reg#1, a
  LD    Reg#1, 4
  ST    Reg#1, b
  LD    Reg#3, a
  LD    Reg#2, b
  LT    Reg#1, Reg#2, Reg#3
  JUMPF Reg#1   2
  LD    Reg#1, 2
  ST    Reg#1, d
  LD    Reg#1, b
  ST    Reg#1, a
  JUMP      1
LABEL:2
  LD    Reg#1, a
  ST    Reg#1, b
LABEL:1
  LD    Reg#3, a
  LD    Reg#2, b
  ADD   Reg#1, Reg#3, Reg#2
  ST    Reg#1, c
  LD    Reg#1, 0
  RET   Reg#1
END test

# number of registers: 3
```

위의 결과 코드와 요구 사항에 비추어 설명하면 다음과 같이 요약 할 수 있다.

1) BEGIN <function name> ~ END <function name>을 확인 할 수 있다.

2) 방금 전 언급된 Chaitin algorithm으로 사용되는 register의 개수를 최소로 만들었다.

3) 방금 전 언급한 대로 EQ와 RET instruction이 추가되었다.

사용된 예제 코드의 symbol table은 다음과 같이 출력된다.

```
type:char
name:test       type:function    scope:global
name:alpha      type:variable    scope:global->test
name:beta       type:variable    scope:global->test
type:char
name:a type:variable     scope:global->test
name:c type:variable     scope:global->test
type:int
name:b type:variable     scope:global->test
type:int
name:d type:variable     scope:global->test->IF_THEN
```

type : char / type : int는 밑의 범주의 변수 자료형을 나타낸 것이다.
그 사이의 각각의 줄은 name은 변수명, type은 함수인지 변수인지 표시하고, scope는 이 변수의 유효 범위를 나타내준다.

# 4. Conclusion

전체적으로 간단히 요약하자면 다음과 같은 방식으로 진행되었다.

1) 무한루프의 가능성을 해결하고, SLR(1) / LR(1) / LALR(1) 중 하나라도 되도록 문법을 변경했다.
2) scanner는 미리 입력값을 한꺼번에 tokenize해서 배열에 저장한다.
FIRST / FOLLOW 입력 후, 그래프를 그렸고, 이를 기반으로 parsing table을 완성하였다. SLR parser를 사용하였다.
3) 완성된 parsing table로 AST를 만들었고, 이에 기반하여 Assembly code를 작성하였다.
4) Chaitin algorithm을 사용하여 Assembly에서 사용될 register의 개수를 줄였다.

더불어, 다음과 같은 어려움이 있었다.
1) Assembly에서의 register와 변수를 mapping하는 과정에서 많은 혼란을 겪었다.
2) Parsing table 및 AST를 구현하는 과정에서의 잔실수로 확인하는데 약간의 혼란을 겪었다.

아쉬운 점은 다음과 같다.
1) 변수 선언이 안 되어 있는데도 compiler에서 이를 갖다가 사용한 것을 발견하지 못했다.
2) Assembly line 들을 최적화 할 수 있었으면 훨씬 더 좋은 결과가 있지 않았을까 생각한다.