컴파일러의 기초 Project 2

전기정보공학부

2016-13343

유상윤

기존 grammar에 아래와 같은 shift/reduce conflict 9개 존재한다.

1개: dangling else problem

4개, 4개: unary operator (unary ‘!’, unary ‘\*’, unary ‘&’, unary ‘-‘) 다음 unary 다음 ‘[‘ or ‘(‘

‘Dangling Else Problem’ Conflict 해결

해결 방향: Else를 가장 가까이 있는 if에 matching 되도록 해 conflict를 해결해야한다.

해결 방법: Yacc의 Precedence를 활용해서 우선순위를 지정해 해결하였다.

%nonassoc IFX

%nonassoc ELSE

…

stmt:

…

| IF '(' test ')' stmt  %prec IFX   {

            REDUCE("stmt->IF '(' test ')' stmt");

        }

| IF '(' test ')' stmt ELSE stmt    {

            REDUCE("stmt->IF '(' test ')' stmt ELSE stmt");

        }

Open If statement에 IFX라는 precedence를 지정하고, 이를 ELSE보다 낮은 우선순위를 지정해 만약 if statement 다음 else statement가 존재할 경우 먼저 parsing되도록 한다.

이를 통해 else가 항상 가까운 if와 matching 되도록 할 수 있다.

unary operator (unary ‘!’, unary ‘\*’, unary ‘&’, unary ‘-‘) 다음 unary 다음 ‘[‘, ‘(‘

해결 방향: int\* a[10] 같은 형태는 우리 grammer에서 받지 않는다. 따라서 array of pointer declaration은 고려하지 않아도 된다.

‘[]’의 경우 !a[10], \*a[10], &a[10], -a[10] 모두 a[10]을 먼저 구한 후 그 값에 unary operator를 취한 것을 의미한다. 따라서 unary ‘[‘ ‘]’ 를 먼저 parsing하고 그 다음 unary operator가 parsing되도록 해야한다.

‘()’의 경우 !foo(10), \*foo(10), &foo(10), -foo(10) 모두 foo(10)을 먼저 구한 후 그 값에 unary operator를 취한 것을 의미한다. 따라서 unary ‘(‘ ‘)’ 를 먼저 parsing하고 그 다음 unary operator가 parsing되도록 해야한다.

해결 방법: Yacc의 Precedence를 활용해서 우선순위를 지정해 해결하였다.

%right  PLUS\_PLUS MINUS\_MINUS UNARY

%nonassoc '[' '('

…

unary:

…

    | '-' unary %prec UNARY  {

            REDUCE("unary->'-' unary");

        }

| '!' unary %prec UNARY {

            REDUCE("unary->'!' unary");

        }

| '&' unary %prec UNARY {

            REDUCE("unary->'&' unary");

        }

    | '\*' unary %prec UNARY   {

            REDUCE("unary->'\*' unary");

        }

Dangling else를 해결한 것과 마찬가지로 conflict가 일어날 수 있는 unary operator들에게 UNARY라는 precedence를 지정하고, 우선순위를 ‘[‘, ‘(‘보다 낮게 두어 해결하였다.