Design and Development of Compilerfor C- Language

Phase 2: Design and Implementation of LALR Parser

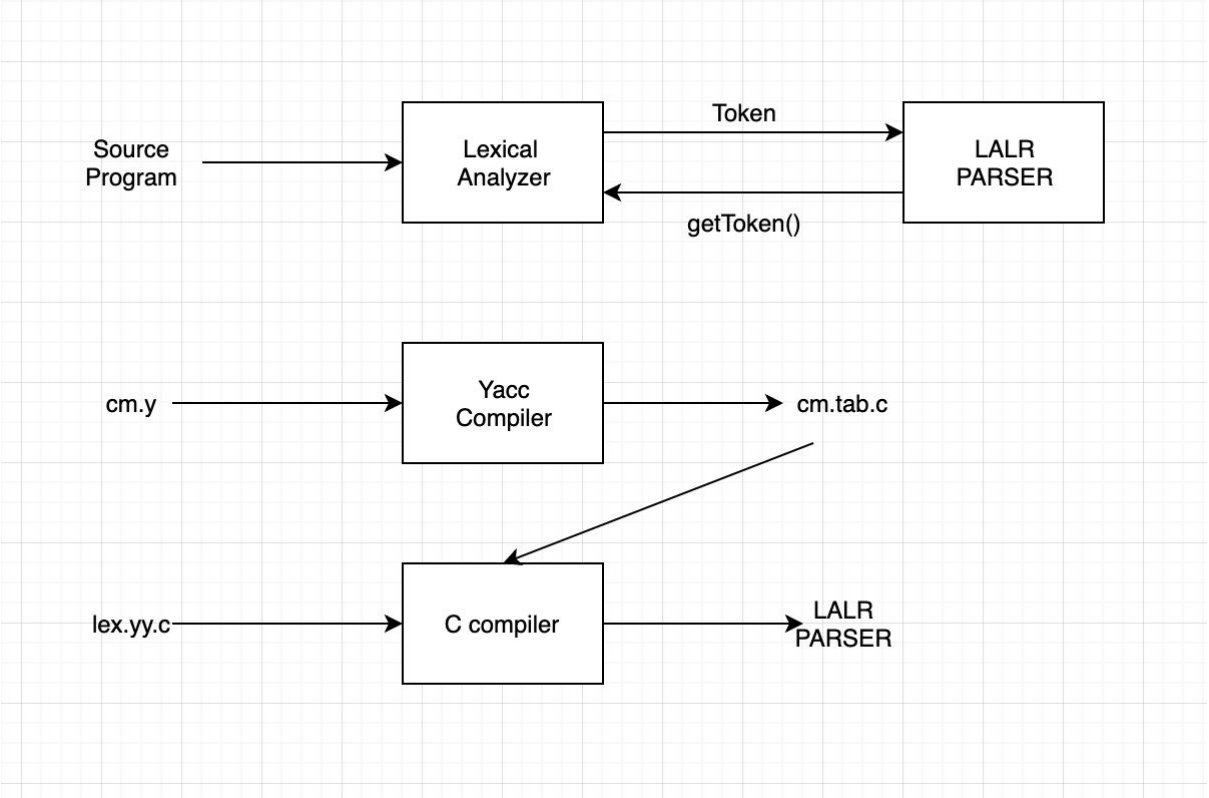
Proposal of Phase 2

과목명: [CSE4120] 기초 컴파일러 구성
담당교수: 서강대학교 컴퓨터공학과 정 성 원
개발자: 20141510 김민우

I. 개발 목표

본 프로젝트는 C- Language를 위한 Parser를 개발하는 것을 목표로 한다. 본 프로젝트에서는 Syntax Analyzer를 구현하여 C- Language의 코드로부터 생성된 token들을 이용하여 Abstract Syntax Tree를 구현한다. 구현하는 Parser는 LALR(1) 문법을 이용해 Abstract Syntax Tree를 구축하도록 한다. 구현하는 parser는 LALR(1) 문법을 따르며, Syntax Tree를 구현하는 과정에 있어서 문법에 어긋나는 오류가 있다면, syntax error가 발생한 line number와 error message를 출력한다.

2. flow chart for LALR parser generator



3. How to run my program

A) tar -xvf project2_20141510.tar
B) cd project2_20141510
C) make
D) /20141510 filename

4. precedence directives

주어진 grammar를 살펴보면,

simple_exp -> additive_exp relop additive_exp | additive_exp
additive_exp -> additive_exp addop term | term
addop -> + | -
term -> term mulop factor | factor
mulop -> * | /

factor -> (exp) | number

비교 연산자가 덧셈과 뺄셈보다 syntax tree에서 높은 위치를 차지하고, 덧셈과 뺄셈이 다음 높은 위치를 차지한다. 다음으로는 곱셈과 나눗셈, 마지막으로는 괄호와 상수 순으로 높이가 조절된다.
token이 위치하는 tree의 높이에 따라서 연산자 우선순위가 정해진다. 이는, 우리가 일상 생활에서 사용하는 수학과 같은 연산자 우선순위이므로 별도의 precedence directive가 필요하지 않다.
그리고 additive_exp와 term은 left recursion이므로, 덧셈, 뺄셈, 곱셈, 나눗셈 모두 left associative하다.

5. List of all conflicts with the explanations of how each conflict are resolved

```
State 99

   31 selection_stmt: IF LPAREN expression RPAREN statement .
   32               | IF LPAREN expression RPAREN statement . ELSE statement

    ELSE  shift, and go to state 102

    ELSE      [reduce using rule 31 (selection_stmt)]
    $default  reduce using rule 31 (selection_stmt)
```

cm.y 파일에 별다른 처리를 하지 않을 경우, 위와 같은 shift / reduce conflict가 발생한다.
%nonassoc RPAREN
%nonassoc ELSE
와 같이 명시적으로 처리함으로써, ELSE의 우선순위를 RPAREN보다 높여서 ELSE가 shift를 우선적으로 하게 된다.
하지만, bison의 경우에는 shift / reduce conflict가 발생하였을 경우에는 shift를 우선적으로 실행하므로 별다른 처리를 하지 않아도 괜찮다.

6. explanation on the implementations including semantic action part

주어진 grammar에 대한 Abstract Syntax Tree를 구축하려면, Enum Type을 적절하게 추가해야 한다.
본 프로젝트에서는 statement_node, expression_node, declaration_node 3 가지 노드를 정의하여 parser를 구현하였다.

printTree :

주어진 트리의 노드가 어디에 속해있는지 enum type을 이용해서 분류한 다음, 주어진 노드의 종류와 attribute를 출력한다. 이후에는 주어진 노드의 자식 노드들에 대해서 모두 출력하고 다음 sibling에 대해서 출력한다.

만약, tree->nodekind == StmtK라면, 해당 노드가 어떠한 statement인지 출력하고 자식 노드들에 대해서 출력을 수행한다. 특히 If의 경우, else문이 존재한다면 if의 child가 3개가

존재한다. else를 따로 출력하지 않고, 3개의 자식들이 출력되는데, 3번째 statement는 조건문이 거짓일 경우 수행한다.

만약, tree->nodekind == ExpK라면, 각 expression의 종류와 관련된 attribute를 출력한다. 배열의 크기를 나타내기 위해서 ArrSizeK를 선언했고, 단일 변수와 배열 변수에 대입인지 구분하기 위해서
ArrVarK를 선언하였다.

만약, tree->nodekind == DecK라면, 각 선언의 종류에 대하여 출력한다. 여러 종류의 DecK type의 node를 구분하여 지역변수인지, 함수의 parameter가 배열인지 등도 다양하게 알 수 있다.
특히, 함수 선언의 경우 함수의 이름을 함수 선언과 같은 줄에 출력하도록 했다.

만일 어디에도 속하지 않는다면, error message가 출력될 것이다.

cm.y :

semantic action의 경우, 주어진 grammar에서 필요 없는 부분을 제거하고 필요한 부분만 노드에 저장하는 방식으로 AST를 구축한다.
보통 새로운 노드를 생성할 때는 new*Node함수를 호출하여 새로운 노드를 만들고, 값을 $$(left hand side의 attribute)에 저장한 다음, 노드의 attribute와 child의 값을 설정하여 새로 생성된 노드가 어느 노드와 연관이 있는지를 나타낼 수 있다.
ex) $$ = newStmtNode(CompK);
$$ -> child[0] = $1;
$$ -> child[1] = $2;

해당하는 노드와 유사한 종류의 트리를 계속 연결할 때는 아래와 같은 알고리즘을 사용한다.

```
YYSTYPE t = $1;
if ( t != NULL )
{
        while ( t -> sibling != NULL )
                t = t -> sibling;
        t -> sibling = $2;
        $$ = $1;
}
else
        $$ = $2;
```

input token이 충분하지 못한 경우에는 계속 shift를 해서 stack에 쌓아올리다가 충분히 input token이 존재해서 특정한 의미를 나타내는 경우에는 stack에서 pop시킴과 동시에 reduce하여 하나의 nonterminal로 변한다. nonterminal로 변할 때, 관련된 node를 연결함으로써 계속 연결해나가는 방식으로 프로그램에 대한 하나의 큰 트리를 구성한다.

7. source Codes and output & two test C- programs

구현한 parser는 직접 구현한 errorfree.c, error.c에 대하여 다음과 같이 동작한다

```
[cse20141510@cspro:~/compilerProject/project2_20141510$ ./20141510 errorfree.c
Syntax tree:
  Function Declaration : main
    Type: void
    Compound Expression
      Array Declaration :
        Type: int
        ID : arr
        Constant : 11111
      If
        Op: <=
          ID : x
          ID : y
        Compound Expression
          While
            Op: >=
              Array Subscript
                ID : arr
                ID : i
              Constant : 0
            Compound Expression
              Op: =
                ID : a
                Op: +
                  ID : a
                  ID : b
        Compound Expression
          Function Call
            ID : fun
            Op: +
              ID : x
              Op: /
                ID : y
                Constant : 4
[cse20141510@cspro:~/compilerProject/project2_20141510$ cat errorfree.c
void main(void)
{
        int arr[11111];
        if (x <= y)
        {
                while ( arr[i] >= 0 )
                {
                        a = a + b;
                }
        }
        else
        {
                fun(x + y / 4);
        }
}
```

문법이 올바른 경우 (errorfree.c)

```
[cse20141510@cspro:~/compilerProject/project2_20141510$ ./20141510 error.c
Syntax error at line 6: syntax error
Current token:  ]
Syntax tree:
[cse20141510@cspro:~/compilerProject/project2_20141510$ cat error.c
void main(void)
{
        int arr[11111];
        if (x <= y)
        {
                while ( arr[] >= 0 )
                {
                        a = a + b;
                }
        }
        else
        {
                fun(x + y / 4);
        }
}
```

6번째 줄 arr[]에 index값이 없어서 syntax error 발생! (error.c)