21001715 컴파일러
Programming Assignment #2
Due: 2020년 4월 26일 23:59PM

Yacc 을 이용하여 C 소스코드를 위한 Parser 만들기

1. Token

두 번째 PA는 C언어로 된 소스코드로부터 분석된 형태소들을 이용하여 parsing tree를 생성해주는 parser를 만드는 것이다. 분석해야 할 문법의 생성규칙들은 아래 2절에 소개되어 있다. `Program`이 start symbol이며, terminal symbol은 배경을 회색으로 칠하여 구분하였다.

2. 분석해야 할 생성규칙들

```
Program → DeclarationList

DeclarationList → DeclarationList Declaration
                | Declaration

Declaration → VarDeclaration
            | FuncDeclaration

FuncDeclaration → Type ID ( Params ) CompoundStmt
                | Type ID ( Params ) ;

Params → ParamList
       | void
       | ε

ParamList → ParamList , Param
          | Param

Param → Type Value

CompoundStmt → { LocalDeclarationList StmtList }

LocalDeclarationList → LocalDeclarationList VarDeclaration
                     | ε

VarDeclaration → Type IDs ;

Type → int
     | void
     | char
     | float

IDs: IDs , Value
   | Value

Value → ID [ INTEGER ]
      | ID

StmtList → StmtList Stmt
         | ε

Stmt → MatchedStmt
     | OpenStmt

MatchedStmt → ExprStmt
            | ForStmt
```

```
            | WhileStmt
            | ReturnStmt
            | CompoundStmt
            | BreakStmt
            | SwitchStmt
            | if ( Expr ) MatchedStmt else MatchedStmt

OpenStmt → if ( Expr ) Stmt
         | if ( Expr ) MatchedStmt else OpenStmt

SwitchStmt → switch ( Expr ) { CaseList DefaultCase }

CaseList → CaseList case INTEGER : StmtList
         | case INTEGER : StmtList

DefaultCase → default : StmtList
            | ε

ReturnStmt → return Expr ;
           | return ;

BreakStmt → break ;

ExprStmt → Expr ;
         | ;

Expr → AssignExpr
     | SimpleExpr

AssignExpr → Variable = Expr
           | Variable += Expr
           | Variable -= Expr
           | Variable *= Expr
           | Variable /= Expr
           | Variable %= Expr

Variable → ID [ Expr ]
         | ID

SimpleExpr → SimpleExpr || AndExpr
           | AndExpr

AndExpr → AndExpr && RelExpr
        | RelExpr

RelExpr → RelExpr < AddExpr
        | RelExpr <= AddExpr
        | RelExpr > AddExpr
        | RelExpr >= AddExpr
        | RelExpr == AddExpr
        | RelExpr != AddExpr
        | AddExpr

AddExpr → AddExpr + Term
        | AddExpr - Term
        | Term

Term → Term * Factor
     | Term / Factor
     | Term % Factor
     | Factor

Factor → ( Expr )
       | FunctionCall
       | - Factor
       | Variable
```

```
        | Variable IncDec
        | IncDec Variable
        | NumberLiteral

NumberLiteral → INTEGER
              | REAL

IncDec → ++
       | --

WhileStmt → while ( Expr ) CompoundStmt
          | while ( Expr ) ;

ForStmt → for ( Expr ; Expr ; Expr ) CompoundStmt
        | for ( Expr ; Expr ; Expr ) ;

FunctionCall → ID ( Arguments )

Arguments → ArgumentList
          | ε

ArgumentList → ArgumentList , Expr
             | ArgumentList , STRING
             | Expr
             | STRING
```

3. 작성 방법

(0) PA1 파일 복사.
여러분들의 계정에 `PA2` 디렉토리와 `Makefile`, `test.c`, `bar.y` 파일이 있을 것이다. 여기에 PA1에서 만들었던 `foo.l` 파일을 복사해온다. 현재 위치가 `PA2` 디렉토리일 때, 복사 방법은 다음과 같다.

```
$ cp ../PA1/foo.l .
```

(1) `foo.l` 수정
- 만일 수정 사항이 있을 경우, 수정 해도 무방함.

- 다음 사항들을 지운다:
`enum tnumber {…};`
`union{ … } yylval;`
`main` 함수

- `%{`와 `%}` 사이에 `#include "y.tab.h"`를 추가한다.

- token 패턴이 발생했을 때의 Action 부분을 수정한다. 특히 정수나 문자 상수가 나왔을 때 `yylval.iVal`을 통하여 값이 넘겨질 수 있도록, 실수가 나왔을 경우 `yylval.rVal`을 통하여 값이 넘겨지게끔, `TIDENTIFIER`나 `TSTRING`이 나왔을 경우 `yylval.sVal`을 통하여 값이 넘겨질 수 있도록 코드를 수정한다.

(2) `bar.y` 작성
- 생성 규칙이 사용되면, 그 생성 규칙이 출력되도록 한다. 예를 들어, `Program → DeclarationList`이 사용된 경우, 다음이 출력되도록 한다.

```
Program -> DeclarationList
```

- `NumberLiteral → INTEGER`나 `NumberLiteral → REAL` 이 사용된 경우에는 INTEGER 나 REAL 대신 사용된 수가 출력되도록 한다. 예를 들어, 12 라는 정수가 C 소스코드에 있을 경우, 다음과 같이 출력되도록 한다.

```
NumberLiteral -> 12
```

- ID나 TSTRING 토큰이 등장할 때는 ID나 STRING 대신 해당 문자열이 출력되도록 한다. 예를 들어, `foo()` 라는 함수 콜 명령이 있을 경우, 다음과 같이 출력되도록 한다:

```
FunctionCall -> foo ( Arguments )
```

4. 실행 예

test.c:

```c
int foo(void);
char main(int argc, char argv[3]){
      int a;
      printf("Hello world", 4);
      return 0;
}
```

위와 같은 소스코드가 있을 때, `./parser test.c` 를 실행하면 다음과 같은 결과가 나올 것이다.

```
Type -> int
Params -> void
FuncDeclaration -> Type foo ( Params ) ;
Declaration -> FuncDeclaration
DeclarationList -> Declaration
Type -> char
Type -> int
Value -> argc
Param -> Type Value
ParamList -> Param
Type -> char
Value -> argv [ 3 ]
Param -> Type Value
ParamList -> ParamList , Param
Params -> ParamList
LocalDeclarationList -> Empty
Type -> int
Value -> a
IDs -> Value
VarDeclaraion -> Type IDs ;
LocalDeclarationList -> LocalDeclarationList VarDeclaration
StmtList -> Empty
ArgumentList -> "Hello world"
NumberLiteral -> 4
Factor -> NumberLiteral
Term -> Factor
AddExpr -> Term
RelExpr -> AddExpr
AndExpr -> RelExpr
SimpleExpr -> AndExpr
Expr -> SimpleExpr
ArgumentList -> ArgumentList , Expr
Arguments -> ArgumentList
FunctionCall -> printf ( Arguments )
Factor -> FunctionCall
Term -> Factor
AddExpr -> Term
RelExpr -> AddExpr
AndExpr -> RelExpr
SimpleExpr -> AndExpr
Expr -> SimpleExpr
ExprStmt -> Expr ;
MatchedStmt -> ExprStmt
Stmt -> MatchedStmt
StmtList -> StmtList Stmt
NumberLiteral -> 0
Factor -> NumberLiteral
Term -> Factor
AddExpr -> Term
RelExpr -> AddExpr
AndExpr -> RelExpr
```

```
SimpleExpr -> AndExpr
Expr -> SimpleExpr
ReturnStmt -> return Expr ;
MatchedStmt -> ReturnStmt
Stmt -> MatchedStmt
StmtList -> StmtList Stmt
CompoundStmt -> { LocalDeclarationList StmtList }
FuncDeclaration -> Type main ( Params ) CompoundStmt
Declaration -> FuncDeclaration
DeclarationList -> DeclarationList Declaration
Program -> DeclarationList
```

5. 제출

- 4월 26일 일요일 23:59까지. 시계는 제출용 서버의 시계를 따른다.

- 여러분의 계정에 `PA2` 디렉토리가 있고, 이 안에 `bar.y`, `Makefile`, `test.c` 파일들이 있을 것이다. 이 중 `bar.y` 파일의 내용을 채우면 된다. 그리고 PA1에서 여러분이 작성했던 `foo.l` 파일을 복사해와서 수정한다.

- 만일 위의 파일들 중 어느 하나를 지웠으면 담당 교수에게 문의할 것.

- 하루씩 Delay 될 때마다 점수의 20%를 감점한다. (예를 들어, 이틀 Delay하고 8점을 받았으면 4.8점 획득)

- Copy는 해당 PA 0점. 본인 소스코드는 반드시 본인이 모두 타이핑 하여 만들어야 함. 오해를 방지하기 위하여 둘 이상이 같이 의논하여 짰더라도 최대한 달라보이게 짤 것! (변수 바꾸기, 주석 더 넣기, 들여쓰기 바꾸기 등 만으로는 안 됨)