program: ext\_def\_list

ext\_def\_list: ext\_def\_list ext\_def

| /\* empty \*/

ext\_def: opt\_specifier ext\_decl\_list ';'

| opt\_specifier funct\_decl compound\_stmt

ext\_decl\_list: ext\_decl

| ext\_decl\_list ',' ext\_decl

ext\_decl: var\_decl

| funct\_decl

opt\_specifier: type\_specifier

| /\* empty \*/

<= When the type specifier is omitted, the default type is 'int'

type\_specifier: TYPE

| struct\_specifier

struct\_specifier: STRUCT opt\_tag '{' def\_list '}'

| STRUCT ID

<= In the second case, the struct must have been defined before.

opt\_tag: ID

| /\* empty \*/

<= In the second case, the struct becomes anonymous.

var\_decl: ID

| ID '[' ']'

| ID '[' INTEGER\_CONST ']'

| '\*' ID

<= ID[] is the same as \*ID. (pointer type)

funct\_decl: ID '(' ')'

| ID '(' var\_list ')'

<= When we declare a function with ID(),

we want to have a function which has no parameter.

var\_list: param\_decl

| var\_list ',' param\_decl

param\_decl: type\_specifier var\_decl

def\_list: def\_list def

| /\* empty \*/

def: type\_specifier decl\_list ';'

decl\_list: decl\_list ',' decl

| decl

decl: funct\_decl

| var\_decl

compound\_stmt: '{' local\_defs stmt\_list '}'

local\_defs: def\_list

stmt\_list: stmt\_list stmt

| /\* empty \*/

stmt: expr ';'

| compound\_stmt

| RETURN ';'

| RETURN expr ';'

| ';'

| IF '(' test ')' stmt

| IF '(' test ')' stmt ELSE stmt

| WHILE '(' test ')' stmt

| FOR '(' opt\_expr ';' test ';' opt\_expr ')' stmt

| BREAK ';'

| CONTINUE ';'

test: expr

| /\* empty \*/

opt\_expr: expr

| /\* empty \*/

expr: expr ASSINGOP expr

| expr '=' expr

| or\_expr

or\_expr: or\_list

or\_list: or\_list LOGICAL\_OR and\_expr

| or\_list ‘|’ and\_expr

| and\_expr

and\_expr: and\_list

and\_list: and\_list LOGICAL\_AND binary

| and\_list ‘&’ binary

| binary

binary: binary RELOP binary

| binary EQUOP binary

| binary '\*' binary

| binary '/' binary

| binary '%' binary

| binary '+' binary

| binary '-' binary

| unary

unary: '(' expr ')'

| INTEGER\_CONST

| CHAR\_CONST

| ID

| STRING

| '-' unary

| '!' unary

| unary PLUS\_PLUS

| unary MINUS\_MINUS

| '&' unary

| '\*' unary <= The type of unary is pointer.

| unary '[' expr ']' <= The type of expr is integer.

| unary STRUCTOP ID <= The type of unary is a struct.

| unary '(' args ')' <= The type of unary is a function.

| unary '(' ')'

args: expr

| args ',' expr