C 语言文法 ( C Language Grammar )

%token IDENTIFIER CONSTANT STRING\_LITERAL SIZEOF

%token TYPE\_NAME

%token TYPEDEF EXTERN STATIC AUTO REGISTER

%token CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE CONST VOLATILE VOID

%token STRUCT UNION ENUM

%token CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK RETURN

%start translation\_unit

%%

translation\_unit

: external\_declaration

| translation\_unit external\_declaration

;

external\_declaration

: function\_definition

| declaration

;

function\_definition

: declaration\_specifiers declarator declaration\_list compound\_statement

| declaration\_specifiers declarator compound\_statement

| declarator declaration\_list compound\_statement

| declarator compound\_statement

;

declaration\_list

: declaration

| declaration\_list declaration

;

declaration

: declaration\_specifiers ';'

| declaration\_specifiers init\_declarator\_list ';'

;

declaration\_specifiers

: storage\_class\_specifier

| storage\_class\_specifier declaration\_specifiers

| type\_specifier

| type\_specifier declaration\_specifiers

| type\_qualifier

| type\_qualifier declaration\_specifiers

;

specifier\_qualifier\_list

: type\_specifier specifier\_qualifier\_list

| type\_specifier

| type\_qualifier specifier\_qualifier\_list

| type\_qualifier

type\_qualifier\_list

: type\_qualifier

| type\_qualifier\_list type\_qualifier

;

type\_qualifier

: CONST

| VOLATILE

;

storage\_class\_specifier

: TYPEDEF

| EXTERN

| STATIC

| AUTO

| REGISTER

;

type\_specifier

: VOID

| CHAR

| SHORT

| INT

| LONG

| FLOAT

| DOUBLE

| SIGNED

| UNSIGNED

| struct\_or\_union\_specifier

| enum\_specifier

| TYPE\_NAME

;

struct\_or\_union\_specifier

: struct\_or\_union IDENTIFIER '{' struct\_declaration\_list '}'

| struct\_or\_union '{' struct\_declaration\_list '}'

| struct\_or\_union IDENTIFIER

;

struct\_or\_union

: STRUCT

| UNION

;

struct\_declaration\_list

: struct\_declaration

| struct\_declaration\_list struct\_declaration

;

struct\_declaration

: specifier\_qualifier\_list struct\_declarator\_list ';'

;

struct\_declarator\_list

: struct\_declarator

| struct\_declarator\_list ',' struct\_declarator

;

struct\_declarator

: declarator

| ':' constant\_expression

| declarator ':' constant\_expression

;

enum\_specifier

: ENUM '{' enumerator\_list '}'

| ENUM IDENTIFIER '{' enumerator\_list '}'

| ENUM IDENTIFIER

;

enumerator\_list

: enumerator

| enumerator\_list ',' enumerator

;

enumerator

: IDENTIFIER

| IDENTIFIER '=' constant\_expression

;

init\_declarator\_list

: init\_declarator

| init\_declarator\_list ',' init\_declarator

;

init\_declarator

: declarator

| declarator '=' initializer

;

initializer\_list

: initializer

| initializer\_list ',' initializer

;

initializer

: assignment\_expression

| '{' initializer\_list '}'

| '{' initializer\_list ',' '}'

;

parameter\_type\_list

: parameter\_list

| parameter\_list ',' '...'

;

parameter\_list

: parameter\_declaration

| parameter\_list ',' parameter\_declaration

parameter\_declaration

: declaration\_specifiers declarator

| declaration\_specifiers abstract\_declarator

| declaration\_specifiers

;

identifier\_list

: IDENTIFIER

| identifier\_list ',' IDENTIFIER

;

type\_name

: specifier\_qualifier\_list

| specifier\_qualifier\_list abstract\_declarator

;

abstract\_declarator

: pointer

| direct\_abstract\_declarator

| pointer direct\_abstract\_declarator

;

direct\_abstract\_declarator

: '(' abstract\_declarator ')'

| '[' ']'

| '[' constant\_expression ']'

| direct\_abstract\_declarator '[' ']'

| direct\_abstract\_declarator '[' constant\_expression ']'

| '(' ')'

| '(' parameter\_type\_list ')'

| direct\_abstract\_declarator '(' ')'

| direct\_abstract\_declarator '(' parameter\_type\_list ')'

;

declarator

: pointer direct\_declarator

| direct\_declarator

;

direct\_declarator

: IDENTIFIER

| '(' declarator ')'

| direct\_declarator '[' constant\_expression ']'

| direct\_declarator '[' ']'

| direct\_declarator '(' parameter\_type\_list ')'

| direct\_declarator '(' identifier\_list ')'

| direct\_declarator '(' ')'

;

pointer

: '\*'

| '\*' type\_qualifier\_list

| '\*' pointer

| '\*' type\_qualifier\_list pointer

;

statement

: labeled\_statement

| compound\_statement

| expression\_statement

| selection\_statement

| iteration\_statement

| jump\_statement

;

labeled\_statement

: IDENTIFIER ':' statement

| CASE constant\_expression ':' statement

| DEFAULT ':' statement

;

compound\_statement

: '{' '}'

| '{' statement\_list '}'

| '{' declaration\_list '}'

| '{' declaration\_list statement\_list '}'

;

statement\_list

: statement

| statement\_list statement

;

expression\_statement

: ';'

| expression ';'

;

selection\_statement

: IF '(' expression ')' statement

| IF '(' expression ')' statement ELSE statement

| SWITCH '(' expression ')' statement

;

iteration\_statement

: WHILE '(' expression ')' statement

| DO statement WHILE '(' expression ')' ';'

| FOR '(' expression\_statement expression\_statement ')' statement

| FOR '(' expression\_statement expression\_statement expression ')' statement

;

jump\_statement

: GOTO IDENTIFIER ';'

| CONTINUE ';'

| BREAK ';'

| RETURN ';'

| RETURN expression ';'

;

expression

: assignment\_expression

| expression ',' assignment\_expression

;

assignment\_expression

: conditional\_expression

| unary\_expression assignment\_operator assignment\_expression

;

assignment\_operator

: '='

| '\*='

| '/='

| '%='

| '+='

| '-='

| '<<='

| '>>='

| '&='

| '^='

| '|='

;

constant\_expression

: conditional\_expression

;

conditional\_expression

: logical\_or\_expression

| logical\_or\_expression '?' expression ':' conditional\_expression

;

logical\_or\_expression

: logical\_and\_expression

| logical\_or\_expression '||' logical\_and\_expression

;

logical\_and\_expression

: inclusive\_or\_expression

| logical\_and\_expression '&&' inclusive\_or\_expression

;

inclusive\_or\_expression

: exclusive\_or\_expression

| inclusive\_or\_expression '|' exclusive\_or\_expression

;

exclusive\_or\_expression

: and\_expression

| exclusive\_or\_expression '^' and\_expression

;

and\_expression

: equality\_expression

| and\_expression '&' equality\_expression

;

equality\_expression

: relational\_expression

| equality\_expression '==' relational\_expression

| equality\_expression '!=' relational\_expression

;

relational\_expression

: shift\_expression

| relational\_expression '<' shift\_expression

| relational\_expression '>' shift\_expression

| relational\_expression '<=' shift\_expression

| relational\_expression '>=' shift\_expression

;

shift\_expression

: additive\_expression

| shift\_expression '<<' additive\_expression

| shift\_expression '>>' additive\_expression

;

additive\_expression

: multiplicative\_expression

| additive\_expression '+' multiplicative\_expression

| additive\_expression '-' multiplicative\_expression

;

multiplicative\_expression

: cast\_expression

| multiplicative\_expression '\*' cast\_expression

| multiplicative\_expression '/' cast\_expression

| multiplicative\_expression '%' cast\_expression

;

cast\_expression

: unary\_expression

| '(' type\_name ')' cast\_expression

;

unary\_expression

: postfix\_expression

| '++' unary\_expression

| '--' unary\_expression

| unary\_operator cast\_expression

| SIZEOF unary\_expression

| SIZEOF '(' type\_name ')'

;

unary\_operator

: '&'

| '\*'

| '+'

| '-'

| '~'

| '!'

;

argument\_expression\_list

: assignment\_expression

| argument\_expression\_list ',' assignment\_expression

;

postfix\_expression

: primary\_expression

| postfix\_expression '[' expression ']'

| postfix\_expression '(' ')'

| postfix\_expression '(' argument\_expression\_list ')'

| postfix\_expression '.' IDENTIFIER

| postfix\_expression '->' IDENTIFIER

| postfix\_expression '++'

| postfix\_expression '--'

;

primary\_expression

: IDENTIFIER

| CONSTANT

| STRING\_LITERAL

| '(' expression ')'

;

<translation-unit> ::= {<external-declaration>}\*  
  
<external-declaration> ::= <function-definition>  
 | <declaration>  
  
<function-definition> ::= {<declaration-specifier>}\* <declarator> {<declaration>}\* <compound-statement>  
  
<declaration-specifier> ::= <storage-class-specifier>  
 | <type-specifier>  
 | <type-qualifier>  
  
<storage-class-specifier> ::= "auto"  
 | "register"  
 | "static"  
 | "extern"  
 | "typedef"  
  
<type-specifier> ::= "void"  
 | "char"  
 | "short"  
 | "int"  
 | "long"  
 | "float"  
 | "double"  
 | "signed"  
 | "unsigned"  
 | <struct-or-union-specifier>  
 | <enum-specifier>  
 | <typedef-name>  
  
<struct-or-union-specifier> ::= <struct-or-union><identifier> "{" {<struct-declaration>}+ "}"  
 | <struct-or-union> "{" {<struct-declaration>}+ "}"  
 | <struct-or-union><identifier>  
  
<struct-or-union> ::= "struct"  
 | "union"  
  
<struct-declaration> ::= {<specifier-qualifier>}\* <struct-declarator-list>  
  
<specifier-qualifier> ::= <type-specifier>  
 | <type-qualifier>  
  
<struct-declarator-list> ::= <struct-declarator>  
 | <struct-declarator-list> "," <struct-declarator>  
  
<struct-declarator> ::= <declarator>  
 | <declarator> ":" <constant-expression>  
 | ":" <constant-expression>  
  
<declarator> ::= {<pointer>}? <direct-declarator>  
  
<pointer> ::= "\*" {<type-qualifier>}\* {<pointer>}?  
  
<type-qualifier> ::= "const"  
 | "volatile"  
  
<direct-declarator> ::= <identifier>  
 | "(" <declarator> ")"  
 | <direct-declarator> "[" {<constant-expression>}? "]"  
 | <direct-declarator> "(" <parameter-type-list> ")"  
 | <direct-declarator> "(" {<identifier>}\* ")"  
  
<constant-expression> ::= <conditional-expression>  
  
<conditional-expression> ::= <logical-or-expression>  
 | <logical-or-expression> "?" <expression> ":" <conditional-expression>  
  
<logical-or-expression> ::= <logical-and-expression>  
 | <logical-or-expression "||" <logical-and-expression>  
  
<logical-and-expression> ::= <inclusive-or-expression>  
 | <logical-and-expression "&&" <inclusive-or-expression>  
  
<inclusive-or-expression> ::= <exclusive-or-expression>  
 | <inclusive-or-expression> "|" <exclusive-or-expression>  
  
<exclusive-or-expression> ::= <and-expression>  
 | <exclusive-or-expression> "^" <and-expression>  
  
<and-expression> ::= <equality-expression>  
 | <and-expression> "&" <equality-expression>  
  
<equality-expression> ::= <relational-expression>  
 | <equality-expression> "==" <relational-expression>  
 | <equality-expression> "!=" <relational-expression>  
  
<relational-expression> ::= <shift-expression>  
 | <relational-expression> "<" <shift-expression>  
 | <relational-expression> ">" <shift-expression>  
 | <relational-expression> "<=" <shift-expression>  
 | <relational-expression> ">=" <shift-expression>  
  
<shift-expression> ::= <additive-expression>  
 | <shift-expression> "<<" <additive-expression>  
 | <shift-expression> ">>" <additive-expression>  
  
<additive-expression> ::= <multiplicative-expression>  
 | <additive-expression> "+" <multiplicative-expression>  
 | <additive-expression> "-" <multiplicative-expression>  
  
<multiplicative-expression> ::= <cast-expression>  
 | <multiplicative-expression> "\*" <cast-expression>  
 | <multiplicative-expression> "/" <cast-expression>  
 | <multiplicative-expression> "%" <cast-expression>  
  
<cast-expression> ::= <unary-expression>  
 | "(" <type-name> ")" <cast-expression>  
  
<unary-expression> ::= <postfix-expression>  
 | "++" <unary-expression>  
 | "--" <unary-expression>  
 | <unary-operator><cast-expression>  
 | "sizeof" <unary-expression>  
 | "sizeof" <type-name>  
  
<postfix-expression> ::= <primary-expression>  
 | <postfix-expression> "[" <expression> "]"  
 | <postfix-expression> "(" {<assignment-expression>}\* ")"  
 | <postfix-expression> "." <identifier>  
 | <postfix-expression> "->" <identifier>  
 | <postfix-expression> "++"  
 | <postfix-expression> "--"  
  
<primary-expression> ::= <identifier>  
 | <constant>  
 | <string>  
 | "(" <expression> ")"  
  
<constant> ::= <integer-constant>  
 | <character-constant>  
 | <floating-constant>  
 | <enumeration-constant>  
  
<expression> ::= <assignment-expression>  
 | <expression> "," <assignment-expression>  
  
<assignment-expression> ::= <conditional-expression>  
 | <unary-expression><assignment-operator><assignment-expression>  
  
<assignment-operator> ::= "="  
 | "\*="  
 | "/="  
 | "%="  
 | "+="  
 | "-="  
 | "<<="  
 | ">>="  
 | "&="  
 | "^="  
 | "|="  
  
<unary-operator> ::= "&"  
 | "\*"  
 | "+"  
 | "-"  
 | "~"  
 | "!"  
  
<type-name> ::= {<specifier-qualifier>}+ {<abstract-declarator>}?  
  
<parameter-type-list> ::= <parameter-list>  
 | <parameter-list> "," ...  
  
<parameter-list> ::= <parameter-declaration>  
 | <parameter-list> "," <parameter-declaration>  
  
<parameter-declaration> ::= {<declaration-specifier>}+ <declarator>  
 | {<declaration-specifier>}+ <abstract-declarator>  
 | {<declaration-specifier>}+  
  
<abstract-declarator> ::= <pointer>  
 | <pointer><direct-abstract-declarator>  
 | <direct-abstract-declarator>  
  
<direct-abstract-declarator> ::= ( <abstract-declarator> )  
 | {<direct-abstract-declarator>}? "[" {<constant-expression>}? "]"  
 | {<direct-abstract-declarator>}? "(" {<parameter-type-list>|? ")"  
  
<enum-specifier> ::= "enum" <identifier> "{" <enumerator-list> "}"  
 | "enum" "{" <enumerator-list> "}"  
 | "enum" <identifier>  
  
<enumerator-list> ::= <enumerator>  
 | <enumerator-list> "," <enumerator>  
  
<enumerator> ::= <identifier>  
 | <identifier> "=" <constant-expression>  
  
<typedef-name> ::= <identifier>  
  
<declaration> ::= {<declaration-specifier>}+ {<init-declarator>}\*  
  
<init-declarator> ::= <declarator>  
 | <declarator> "=" <initializer>  
  
<initializer> ::= <assignment-expression>  
 | "{" <initializer-list> "}"  
 | "{" <initializer-list> "," "}"  
  
<initializer-list> ::= <initializer>  
 | <initializer-list> "," <initializer>  
  
<compound-statement> ::= "{" {<declaration>}\* {<statement>}\* "}"  
  
<statement> ::= <labeled-statement>  
 | <expression-statement>  
 | <compound-statement>  
 | <selection-statement>  
 | <iteration-statement>  
 | <jump-statement>  
  
<labeled-statement> ::= <identifier> ":" < statement>  
 | "case" <constant-expression> ":" <statement>  
 | "default" ":" <statement>  
  
<expression-statement> ::= {<expression>}? ";"  
  
<selection-statement> ::= "if" "(" <expression> ")" <statement>  
 | "if" "(" <expression> ")" <statement> "else" <statement>  
 | "switch" "(" <expression> ")" <statement>  
  
<iteration-statement> ::= "while" "(" <expression> ")" <statement>  
 | "do" <statement> "while" "(" <expression> ")" ";"  
 | "for" "(" {<expression>}? ";" {<expression>}? ";" {<expression>}? ")" <statement>  
  
<jump-statement> ::= "goto" <identifier> ";"  
 | "continue" ";"  
 | "break" ";"  
 | "return" {<expression>}? ";"