```
C grammar begins here:

Terminals:
  typedef-name integer-constant character-constant floating-constant
  enumeration-constant identifier

translation-unit: (function-definition | declaration)+

function-definition:
  declaration-specifiers? declarator declaration* block

declaration: declaration-specifiers init-declarator% ";"

declaration-specifiers:
  (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)

type-qualifier: ("const" | "volatile")

struct-or-union-specifier:
  ("struct" | "union") (
    identifier? "{" struct-declaration+ "}" |
    identifier
  )

init-declarator: declarator ("=" initializer)?

struct-declaration:
  (type-specifier | type-qualifier)+ struct-declarator%

struct-declarator: declarator | declarator? ":" constant-expression

enum-specifier: "enum" (identifier | identifier? "{" enumerator% "}")

enumerator: identifier ("=" constant-expression)?

declarator:
  pointer? (identifier | "(" declarator ")") (
    "[" constant-expression? "]" |
    "(" parameter-type-list ")" |
    "(" identifier%? ")"
  )*

pointer:
  ("*" type-qualifier*)*

parameter-type-list: parameter-declaration% ("," "...")?

parameter-declaration:
  declaration-specifiers (declarator | abstract-declarator)?

initializer: assignment-expression | "{" initializer% ","? "}"

type-name: (type-specifier | type-qualifier)+ abstract-declarator?
```

```
abstract-declarator:
  pointer ("(" abstract-declarator ")")? (
    "[" constant-expression? "]" |
    "(" parameter-type-list? ")"
  )*

statement:
  ((identifier | "case" constant-expression | "default") ":")*
  (expression? ";" |
   block |
   "if" "(" expression ")" statement |
   "if" "(" expression ")" statement "else" statement |
   "switch" "(" expression ")" statement |
   "while" "(" expression ")" statement |
   "do" statement "while" "(" expression ")" ";" |
   "for" "(" expression? ";" expression? ";" expression? ")" statement |
   "goto" identifier ";" |
   "continue" ";" |
   "break" ";" |
   "return" expression? ";"
  )

block: "{" declaration* statement* "}"

expression:
  assignment-expression%

assignment-expression: (
    unary-expression (
      "=" | "*=" | "/=" | "%=" | "+=" | "-=" | "<<=" | ">>=" | "&=" |
      "^=" | "|="
    )
  )* conditional-expression

conditional-expression:
  logical-OR-expression ( "?" expression ":" conditional-expression )?

constant-expression: conditional-expression

logical-OR-expression:
  logical-AND-expression ( "||" logical-AND-expression )*

logical-AND-expression:
  inclusive-OR-expression ( "&&" inclusive-OR-expression )*

inclusive-OR-expression:
  exclusive-OR-expression ( "|" exclusive-OR-expression )*

exclusive-OR-expression:
  AND-expression ( "^" AND-expression )*

AND-expression:
  equality-expression ( "&" equality-expression )*

equality-expression:
  relational-expression ( ("==" | "!=") relational-expression )*

relational-expression:
  shift-expression ( ("<" | ">" | "<=" | ">=") shift-expression )*
```

```
shift-expression:
  additive-expression ( ("<<" | ">>") additive-expression )*

additive-expression:
  multiplicative-expression ( ("+" | "-") multiplicative-expression )*

multiplicative-expression:
  cast-expression ( ("*" | "/" | "%") cast-expression )*

cast-expression:
  ( "(" type-name ")" )* unary-expression

unary-expression:
  ("++" | "--" | "sizeof" ) * (
    "sizeof" "(" type-name ")"                          |
    ("&" | "*" | "+" | "-" | "~" | "!" ) cast-expression |
    postfix-expression
  )

postfix-expression:
  (identifier | constant | string | "(" expression ")") (
    "[" expression "]"             |
    "(" assignment-expression% ")" |
    "." identifier                 |
    "->" identifier                |
    "++"                           |
    "--"
  )*

constant:
  integer-constant |
  character-constant |
  floating-constant |
  enumeration-constant

C grammar ends here.
```