Browsable C99 Grammar

Grammar extracted by **Vadim Zaytsev**, see the Grammar Zoo entry for details: c/c99/iso-9899-1999/extracted
Source used for this grammar: ISO/IEC 9899:1999(E), Programming languages — C, December 1999, Annex A, pages 402–416

Summary

- Total **69** production rules with **200** top alternatives and **788** symbols.
- Vocabulary: **156** = **73** nonterminals + **83** terminals + **0** labels + **0** markers.
- Total **73** nonterminal symbols: **69** defined (**translation-unit**, **external-declaration**, **function-definition**, **declaration-specifiers**, **storage-class-specifier**, **type-specifier**, **struct-or-union-specifier**, **struct-or-union**, **struct-declaration-list**, **struct-declaration**, **specifier-qualifier-list**, **type-qualifier**, **struct-declarator-list**, **struct-declarator**, **declarator**, **pointer**, **type-qualifier-list**, **direct-declarator**, **assignment-expression**, **conditional-expression**, **logical-OR-expression**, **logical-AND-expression**, **inclusive-OR-expression**, **exclusive-OR-expression**, **AND-expression**, **equality-expression**, **relational-expression**, **shift-expression**, **additive-expression**, **multiplicative-expression**, **cast-expression**, **unary-expression**, **postfix-expression**, **primary-expression**, **expression**, **argument-expression-list**, **type-name**, **abstract-declarator**, **direct-abstract-declarator**, **parameter-type-list**, **parameter-list**, **parameter-declaration**, **initializer-list**, **designation**, **designator-list**, **designator**, **constant-expression**, **initializer**, **unary-operator**, **assignment-operator**, **identifier-list**, **enum-specifier**, **enumerator-list**, **enumerator**, **typedef-name**, **function-specifier**, **declaration-list**, **declaration**, **init-declarator-list**, **init-declarator**, **compound-statement**, **block-item-list**, **block-item**, **statement**, **labeled-statement**, **expression-statement**, **selection-statement**, **iteration-statement**, **jump-statement**), **1** root (**translation-unit**), **0** top (—), **4** bottom (**string-literal**, **identifier**^{15}, **constant**, **enumeration-constant**^{2}).
- Total **83** terminal symbols: **37** keywords ("typedef", "extern", "static"^{3}, "auto", "register", "void", "char", "short", "int", "long", "float", "double", "signed", "unsigned", "_Bool", "_Complex", "_Imaginary", "struct", "union", "const", "restrict", "volatile", "sizeof"^{2}, "enum"^{3}, "inline", "case", "default", "if"^{2}, "else", "switch", "while"^{2}, "do", "for"^{2}, "goto", "continue", "break", "return"), **0** letters (—), **0** numerics (—), **46** signs ("{"^{8}, "}"^{8}, ";"^{11}, ","^{12}, ":"^{5}, "*"^{6}, "("^{18}, ")"^{18}, "["^{8}, "]"^{8}, "?", "||", "&&", "|", "^", "&"^{2}, "==", "!=", "<", ">", "<=", ">=", "<<", ">>", "+"^{2}, "-"^{2}, "/", "%", "++"^{2}, "--"^{2}, "."^{2}, "->", "...", "="^{4}, "~", "!", "*=", "/=", "%=", "+=", "-=", "<<=", ">>=", "&=", "^=", "|=").

Syntax

translation-unit ::=
    external-declaration
    translation-unit external-declaration

external-declaration ::=
    function-definition
    declaration

function-definition ::=
    declaration-specifiers declarator declaration-list? compound-statement

declaration-specifiers ::=
    storage-class-specifier declaration-specifiers?
    type-specifier declaration-specifiers?
    type-qualifier declaration-specifiers?
    function-specifier declaration-specifiers?

storage-class-specifier ::=
    "typedef"
    "extern"
    "static"
    "auto"
    "register"

type-specifier ::=
    "void"
    "char"
    "short"
    "int"
    "long"
    "float"
    "double"
    "signed"
    "unsigned"
    "_Bool"
    "_Complex"
    "_Imaginary"
    struct-or-union-specifier

enum-specifier
typedef-name

struct-or-union-specifier ::=
    struct-or-union identifier? "{" 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 ";"

specifier-qualifier-list ::=
    type-specifier specifier-qualifier-list?
    type-qualifier specifier-qualifier-list?

type-qualifier ::=
    "const"
    "restrict"
    "volatile"

struct-declarator-list ::=
    struct-declarator
    struct-declarator-list "," struct-declarator

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

declarator ::=
    pointer? direct-declarator

```
pointer ::=
    "*" type-qualifier-list?
    "*" type-qualifier-list? pointer
```

```
type-qualifier-list ::=
    type-qualifier
    type-qualifier-list type-qualifier
```

```
direct-declarator ::=
    identifier
    "(" declarator ")"
    direct-declarator "[" type-qualifier-list? assignment-expression? "]"
    direct-declarator "[" "static" type-qualifier-list? assignment-expression "]"
    direct-declarator "[" type-qualifier-list "static" assignment-expression "]"
    direct-declarator "[" type-qualifier-list? "*" "]"
    direct-declarator "(" parameter-type-list ")"
    direct-declarator "(" identifier-list? ")"
```

```
assignment-expression ::=
    conditional-expression
    unary-expression assignment-operator assignment-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 "(" argument-expression-list? ")"
    postfix-expression "." identifier
    postfix-expression "->" identifier
    postfix-expression "++"
    postfix-expression "--"
    "(" type-name ")" "{" initializer-list "}"
    "(" type-name ")" "{" initializer-list "," "}"

primary-expression ::=

    identifier
    constant
    string-literal
    "(" expression ")"

expression ::=

    assignment-expression
    expression "," assignment-expression

argument-expression-list ::=

    assignment-expression
    argument-expression-list "," assignment-expression

type-name ::=

    specifier-qualifier-list abstract-declarator?

abstract-declarator ::=
  pointer
  pointer? direct-abstract-declarator

direct-abstract-declarator ::=
  "(" abstract-declarator ")"
  direct-abstract-declarator? "[" assignment-expression? "]"
  direct-abstract-declarator? "[" "*" "]"
  direct-abstract-declarator? "(" parameter-type-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?

initializer-list ::=
  designation? initializer
  initializer-list "," designation? initializer

designation ::=
  designator-list "="

designator-list ::=
  designator
  designator-list designator

designator ::=
  "[" constant-expression "]"
  "." identifier

constant-expression ::=
    conditional-expression

initializer ::=
    assignment-expression
    "{" initializer-list "}"
    "{" initializer-list "," "}"

unary-operator ::=
    "&"
    "*"
    "+"
    "-"
    "~"
    "!"

assignment-operator ::=
    "="
    "*="
    "/="
    "%="
    "+="
    "-="
    "<<="
    ">>="
    "&="
    "^="
    "|="

identifier-list ::=
    identifier
    identifier-list "," identifier

enum-specifier ::=
    "enum" identifier? "{" enumerator-list "}"
    "enum" identifier? "{" enumerator-list "," "}"
    "enum" identifier

enumerator-list ::=
  enumerator
  enumerator-list "," enumerator

enumerator ::=
  enumeration-constant
  enumeration-constant "=" constant-expression

typedef-name ::=
  identifier

function-specifier ::=
  "inline"

declaration-list ::=
  declaration
  declaration-list declaration

declaration ::=
  declaration-specifiers init-declarator-list? ";"

init-declarator-list ::=
  init-declarator
  init-declarator-list "," init-declarator

init-declarator ::=
  declarator
  declarator "=" initializer

compound-statement ::=
  "{" block-item-list? "}"

block-item-list ::=
  block-item
  block-item-list block-item

block-item ::=
  declaration

statement

statement ::=
 labeled-statement
 compound-statement
 expression-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
 "for" "(" declaration expression? ";" expression? ")" statement

jump-statement ::=
 "goto" identifier ";"
 "continue" ";"
 "break" ";"
 "return" expression? ";"

POWERED BY GrammarLab

Maintained by Dr. Vadim Zaytsev a.k.a. @grammarware. Last updated in September 2015. [↑]