PRÄPROZESSOR-SYNTAX

```
control-line ::=

        '#define' identifier token-sequence |
        '#define' identifier'(' identifier { ',' identifier } ')' token-sequence |
        '#undef' identifier |
        '#include' '<'filename'>' |
        '#include' '''filename''' |
        '#include' token-sequence |
        '#line' constant '''filename''' |
        '#line' constant |
        '#error' [ token-sequence ] |
        '#pragma' [ token-sequence ] |
        '#' |
        preprocessor-conditional

preprocessor-conditional ::=

        if-line
          text
        elif-parts
        [ else-part ]
        '#endif'

if-line ::=

        '#if' constant-expression |
        '#ifdef' identifier |
        '#ifndef' identifier

elif-parts ::=

        { '#elif' constant-expression
             text }

else-part ::=

        '#else'
           text
```

## LEXIKALISCHE ELEMENTE (TOKEN)

```
token ::= comment | identifier | keyword | constant | string_literal | operator | separator
```

KOMMETAR

```
comment ::=
  '/*' text '*/'
```

BEZEICHNER

```
identifier ::=
  letter{letter|digit}

letter ::=
  'a'|..|'z'|'A'|..|'Z'|'_'

digit ::=
  '0'|..|'9'
```

SCHLÜSSELWORT

```
keyword ::=
  'auto'    |'break'   |'case'    |'char'    |'const'   |'continue'|'default'  |'do'      |
  'double'  |'else'    |'enum'    |'extern'  |'float'   |'for'     |'goto'     |'if'      |
  'int'     |'long'    |'register'|'return'  |'short'   |'signed'  |'sizeof'   |'static'  |
  'struct'  |'switch'  |'typedef' |'union'   |'unsigned'|'void'    |'volatile' |'while'
```

KONSTANTEN LITERAL

```
constant ::=
  character_constant | integer_constant | floating_constant | enum_constant
```

ZEICHENLITERAL

```
character_constant ::=
  '''character|escape'''

escape ::=
  '\n'|'\t'|'\v'|'\b'|'\r'|'\f'|'\a'|'\\'|'\?'|'\''|'\"'|'\'oct[oct[oct]]|'\x'hex{hex}

oct ::=
  '0'|..|'7'

hex ::=
  '0'|..|'9'|'a'|..|'f'|'A'|..|'F'
```

GANZZAHLENLITERAL

```
integer_constant ::=
  octal_constant | decimal_constant | hex_constant

octal_constant ::=
  '0'oct{oct}[isuffix]

decimal_constant ::=
  dec1{dec}[isuffix]

hex_constant ::=
  '0x'hex{hex}[isuffix] | '0X'hex{hex}[isuffix]

dec1 ::=
  '1'|..|'9'

dec ::=
  '0'|..|'9'

isuffix ::=
  'u'|'U'|'l'|'L'|'ul'|'UL'
```

KOMMAZAHLENLITERAL

```
floating_constant ::=
  integer_part'.'fraction_part expchar exponent[fsuffix]|
  integer_part'.'              expchar exponent[fsuffix]|
              '.'fraction_part expchar exponent[fsuffix]|
  integer_part'.'fraction_part                 [fsuffix]|
  integer_part                 expchar exponent[fsuffix]

integer_part ::=
  dec{dec}

fraction_part ::=
  dec{dec}

expchar ::=
  'e'|'E'

exponent ::=
  ['+'|'-']dec{dec}

fsuffix ::=
  'f'|'F'|'l'|'L'
```

AUFZÄHLUNSLITERAL

```
enum_constant ::=
  identifier
```

ZEICHENKETTENLITERAL

```
string_literal ::=
  '"'{character|escape}'"'
```

OPERATOR

```
operator ::=
  '~' | '!' | '%' | '^' | '&' | '*' | '(' | ')' | '+' | '-' | '=' |
  '|' | '[' | ']' | ':' | '|' | ';' | '<' | '>' | '?' | ',' | '.' | '/' |
  '*=' | '/=' | '%=' | '+=' | '-=' | '<<=' | '>>=' | '&=' | '^=' | '|=' | '||' | '&&' | '==' |
  '!=' | '<=' | '>=' | '<<' | '>>' | '++' | '--' | '->'
```

TRENNZEICHEN

```
separators ::=
  White Space (CR,NL,HT,VT,SP,FF)
```

# C-SYNTAX

## PROGRAMMSTRUKTUR

```
translation-unit ::=
  external-declaration { external-declaration }

external-declaration ::=
  function-definition | declaration

function-definition ::=
  [ declaration-specifiers ] declarator compound-statement

declaration-specifiers ::=
  [ storage-class-specifier ] [ type-qualifier ] type-specifier

storage-class-specifier ::=
  'auto' | 'register' | 'static' | 'extern' | 'typedef'

type-qualifier ::=
  'const' | 'volatile'

type-specifier ::=
  'void' |
  'char' | 'signed char' | 'unsigned char' |
  'short' | 'short int' | 'signed short' | 'signed short int' |
  'unsigned short' |'unsigned short int' |
  'int' | 'signed' | 'signed int' | 'unsigned' | 'unsigned int' |
  'long' | 'long int' | 'signed long' | 'signed long int' |
  'unsigned long' | 'unsigned long int' |
  'float' | 'double' | 'long double' |
  struct-or-union-specifier | enum-specifier | identifier

struct-or-union-specifier ::=
  struct-or-union [ identifier ] '{' struct-declaration { struct-declaration } '}' |
  struct-or-union identifier

struct-or-union ::=
  'struct' | 'union'

struct-declaration ::=
  [ type-qualifier ] type-specifier struct-declarator { ',' struct-declarator } ';'

struct-declarator ::=
  declarator |
  [ declarator ] ':' constant-expression

declarator ::=
  [ pointer ] direct-declarator

pointer ::=
  '*' [ type-qualifier ] { '*' [ type-qualifier ] }

direct-declarator ::=
  identifier |
  '(' declarator ')' |
  direct-declarator '[' [ constant-expression ] ']' |
  direct-declarator '(' parameter-declaration { ',' parameter-declaration } [ ',' '...' ] ')' |
  direct-declarator '(' [ identifier { ',' identifier } ] ')'
```

AUSDRÜCKE

```
constant-expression ::=
  conditional-expression

conditional-expression ::=
  logical-OR-expression [ '?' 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 { equop relational-expression }

relational-expression ::=
  shift-expression { relop shift-expression }

shift-expression ::=
  additive-expression { shiftop additive-expression }

additive-expression ::=
  multiplicative-expression { addop multiplicative-expression }

multiplicative-expression ::=
  cast-expression { multop cast-expression }

cast-expression ::=
  [ '(' type-name ')' ] unary-expression

unary-expression ::=
  postfix-expression |
  '++' unary-expression |
  '--' unary-expression |
  unop cast-expression |
  'sizeof' unary-expression |
  'sizeof' '(' type-name ')'

postfix-expression ::=
  primary-expression |
  postfix-expression '[' expression ']' |
  postfix-expression '(' [ assignment-expression { ',' assignment-expression } ] ')' |
  postfix-expression '.' identifier |
  postfix-expression '->' identifier |
  postfix-expression '++' |
  postfix-expression '--'

primary-expression ::=
  identifier |
  string |
  constant |
  '(' expression ')'

expression ::=
  assignment-expression { ',' assignment-expression }

assignment-expression ::=
  conditional-expression |
  unary-expression assop assignment-expression

assop ::=
  '=' | '*=' | '/=' | '%=' | '+=' | '-=' | '<<=' | '>>=' | '&=' | '^=' | '|='

unop ::=
  '&' | '*' | '+' | '-' | '~' | '!'
```

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

type-name ::=
  [ type-qualifier ] type-specifier  [ abstract-declarator ]

abstract-declarator ::=
  pointer | [ pointer ] direct-abstract-declarator

direct-abstract-declarator ::=
  '(' abstract-declarator ')' |
  [ direct-abstract-declarator ] '[' [ constant-expression ] ']' |
  [ direct-abstract-declarator ] '(' [ parameter-declaration { ',' parameter-declaration }
                                                                                  [ ',' '...' ] ] ')'
parameter-declaration ::=
  declaration-specifiers declarator |
  declaration-specifiers [ abstract-declarator ]

multop ::=
  '*' | '/' | '%'

addop ::=
  '+' | '-'

shiftop ::=
  '<<' | '>>'

relop ::=
  '<' | '>' | '<=' | '>='

equop ::=
  '==' | '!='

enum-specifier ::=
  'enum' [ identifier ] '{' enumerator { ',' enumerator } '}' |
  'enum' identifier

enumerator ::=
  identifier [ '=' constant-expression ]

declaration ::=
  declaration-specifiers [ init-declarator { ',' init-declarator } ] ';'

init-declarator ::=
  declarator [ '=' initializer ]

initializer ::=
  assignment-expression |
  '{' initializer { ',' initializer } [ ',' ] '}'
```

ANWEISUNGEN

```
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 [ '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 ] ';'
```