

Applications of Context-Free Grammars to Programming Language Specification

Context-Free Grammars are used to specify programming language syntaxes. From these, parsers and compilers can be generated using tools such as *Antlr* (www.antlr.org)

For example:

ANSI C specification as Context-Free Grammar (in *Backus-Naur Form*) (from Kernigan & Ritchie - adapted from www.cs.man.ac.uk/~pjj/bnf/c_syntax.bnf)

```
%token int_const char_const float_const id string
enumeration_const
%%

translation_unit      : external_decl
                       | translation_unit external_decl
                       ;
external_decl         : function_definition
                       | decl
                       ;
function_definition   : decl_specs declarator decl_list
compound_stat         :
                       |         declarator decl_list compound_stat
                       | decl_specs declarator         compound_stat
                       |         declarator         compound_stat
                       ;
decl                  : decl_specs init_declarator_list ';'
                       | decl_specs                      ';'
                       ;
decl_list             : decl
                       | decl_list decl
                       ;
decl_specs            : storage_class_spec decl_specs
                       | storage_class_spec
                       | type_spec decl_specs
                       | type_spec
                       | type_qualifier decl_specs
                       | type_qualifier
                       ;
storage_class_spec    : 'auto' | 'register' | 'static' | 'extern' |
                       'typedef'
                       ;
type_spec             : 'void' | 'char' | 'short' | 'int' | 'long' |
                       'float'
                       | 'double' | 'signed' | 'unsigned'
                       | struct_or_union_spec
                       | enum_spec
                       | typedef_name
                       ;
type_qualifier        : 'const' | 'volatile'
                       ;
```

```

struct_or_union_spec      : struct_or_union id '{' struct_decl_list
                           '{'
                           | struct_or_union      '{' struct_decl_list '}'
                           | struct_or_union id
                           ;
struct_or_union           : 'struct' | 'union'
                           ;
struct_decl_list         : struct_decl
                           | struct_decl_list struct_decl
                           ;
init_declarator_list     : init_declarator
                           | init_declarator_list ',' init_declarator
                           ;
init_declarator          : declarator
                           | declarator '=' initializer
                           ;
struct_decl              : spec_qualifier_list struct_declarator_list
                           ';'
                           ;
spec_qualifier_list      : type_spec spec_qualifier_list
                           | type_spec
                           | type_qualifier spec_qualifier_list
                           | type_qualifier
                           ;
struct_declarator_list   : struct_declarator
                           | struct_declarator_list ',' struct_declarator
                           ;
struct_declarator        : declarator
                           | declarator ':' const_exp
                           | declarator ':' const_exp
                           ;
enum_spec                : 'enum' id '{' enumerator_list '}'
                           | 'enum'      '{' enumerator_list '}'
                           | 'enum' id
                           ;
enumerator_list          : enumerator
                           | enumerator_list ',' enumerator
                           ;
enumerator               : id
                           | id '=' const_exp
                           ;
declarator               : pointer direct_declarator
                           | direct_declarator
                           ;
direct_declarator        : id
                           | '(' declarator ')'
                           | direct_declarator '[' const_exp ']'
                           | direct_declarator '['          ']'
                           | direct_declarator '(' param_type_list ')'
                           | direct_declarator '(' id_list ')'
                           | direct_declarator '('          ')'
                           ;

```

```

pointer          : '*' type_qualifier_list
                  | '*'
                  | '*' type_qualifier_list pointer
                  | '*'
                    pointer
                  ;
type_qualifier_list : type_qualifier
                    | type_qualifier_list type_qualifier
                    ;
param_type_list    : param_list
                    | param_list ',' '...'
                    ;
param_list         : param_decl
                    | param_list ',' param_decl
                    ;
param_decl         : decl_specs declarator
                    | decl_specs abstract_declarator
                    | decl_specs
                    ;
id_list           : id
                  | id_list ',' id
                  ;
initializer       : assignment_exp
                  | '{' initializer_list '}'
                  | '{' initializer_list ',' '}'
                  ;
initializer_list  : initializer
                  | initializer_list ',' initializer
                  ;
type_name        : spec_qualifier_list abstract_declarator
                  | spec_qualifier_list
                  ;
abstract_declarator : pointer
                    | pointer direct_abstract_declarator
                    | direct_abstract_declarator
                    ;
direct_abstract_declarator: '(' abstract_declarator ')'
                          | direct_abstract_declarator '[' const_exp ']'
                          | direct_abstract_declarator '[' const_exp ']'
                          | direct_abstract_declarator '[' ']'
                          | direct_abstract_declarator '[' ']'
                          | direct_abstract_declarator '(' param_type_list
                          | direct_abstract_declarator '(' param_type_list ')'
                          | direct_abstract_declarator '(' ' ' ')'
                          | direct_abstract_declarator '(' ' ' ')'
                          ;
typedef_name      : id
                  ;
stat             : labeled_stat
                  | exp_stat
                  | compound_stat
                  | selection_stat

```

```

        | iteration_stat
        | jump_stat
        ;
labeled_stat      : id ':' stat
                  | 'case' const_exp ':' stat
                  | 'default' ':' stat
                  ;
exp_stat          : exp ';'
                  | ';'
                  ;
compound_stat     : '{' decl_list stat_list '}'
                  | '{'          stat_list '}'
                  | '{' decl_list          '}'
                  | '{'          '}'
                  ;
stat_list         : stat
                  | stat_list stat
                  ;
selection_stat    : 'if' '(' exp ')' stat
                  | 'if' '(' exp ')' stat 'else' stat
                  | 'switch' '(' exp ')' stat
                  ;
iteration_stat     : 'while' '(' exp ')' stat
                  | 'do' stat 'while' '(' exp ')' ';'
                  | 'for' '(' exp ';' exp ';' exp ')' stat
                  | 'for' '(' exp ';' exp ';'      ')' stat
                  | 'for' '(' exp ';' ';' exp ')' stat
                  | 'for' '(' exp ';' ';'      ')' stat
                  | 'for' '('      ';' exp ';' exp ')' stat
                  | 'for' '('      ';' exp ';'      ')' stat
                  | 'for' '('      ';' ';' exp ')' stat
                  | 'for' '('      ';' ';'      ')' stat
                  ;
jump_stat         : 'goto' id ';'
                  | 'continue' ';'
                  | 'break' ';'
                  | 'return' exp ';'
                  | 'return'      ';'
                  ;
exp              : assignment_exp
                  | exp ',' assignment_exp
                  ;
assignment_exp    : conditional_exp
                  | unary_exp assignment_operator assignment_exp
                  ;
assignment_operator : '=' | '*=' | '/=' | '%=' | '+=' | '-=' |
'<<='
                  | '>>=' | '&=' | '^=' | '|='
                  ;
conditional_exp   : logical_or_exp
                  | logical_or_exp '?' exp ':' conditional_exp
                  ;

```

```

const_exp      : conditional_exp
                ;
logical_or_exp  : logical_and_exp
                | logical_or_exp '||' logical_and_exp
                ;
logical_and_exp : inclusive_or_exp
                | logical_and_exp '&&' inclusive_or_exp
                ;
inclusive_or_exp : exclusive_or_exp
                | inclusive_or_exp '|' exclusive_or_exp
                ;
exclusive_or_exp : and_exp
                | exclusive_or_exp '^' and_exp
                ;
and_exp         : equality_exp
                | and_exp '&' equality_exp
                ;
equality_exp    : relational_exp
                | equality_exp '==' relational_exp
                | equality_exp '!=' relational_exp
                ;
relational_exp  : shift_expression
                | relational_exp '<' shift_expression
                | relational_exp '>' shift_expression
                | relational_exp '<=' shift_expression
                | relational_exp '>=' shift_expression
                ;
shift_expression : additive_exp
                | shift_expression '<<' additive_exp
                | shift_expression '>>' additive_exp
                ;
additive_exp    : mult_exp
                | additive_exp '+' mult_exp
                | additive_exp '-' mult_exp
                ;
mult_exp        : cast_exp
                | mult_exp '*' cast_exp
                | mult_exp '/' cast_exp
                | mult_exp '%' cast_exp
                ;
cast_exp        : unary_exp
                | '(' type_name ')' cast_exp
                ;
unary_exp       : postfix_exp
                | '++' unary_exp
                | '--' unary_exp
                | unary_operator cast_exp
                | 'sizeof' unary_exp
                | 'sizeof' '(' type_name ')'
                ;
unary_operator  : '&' | '*' | '+' | '-' | '~' | '!'
                ;

```

```

postfix_exp      : primary_exp
                  | postfix_exp '[' exp ']'
                  | postfix_exp '(' argument_exp_list ')'
                  | postfix_exp '(' ' ' ')'
                  | postfix_exp '.' id
                  | postfix_exp '->' id
                  | postfix_exp '++'
                  | postfix_exp '--'
;

primary_exp      : id
                  | const
                  | string
                  | '(' exp ')'
;

argument_exp_list : assignment_exp
                  | argument_exp_list ',' assignment_exp
;

const            : int_const
                  | char_const
                  | float_const
                  | enumeration_const
;

```