## A13. Grammar

Below is a recapitulation of the grammar that was given throughout the earlier part of this appendix. It has exactly the same content, but is in a different order.

The grammar has undefined terminal symbols *integer-constant*, *character-constant*, *floating-constant*, *identifier*, *string*, and *enumeration-constant*; the `typewriter` style words and symbols are terminals given literally. This grammar can be transformed mechanically into input acceptable to an automatic parser-generator. Besides adding whatever syntactic marking is used to indicate alternatives in productions, it is necessary to expand the "one of" constructions, and (depending on the rules of the parser-generator) to duplicate each production with an *opt* symbol, once with the symbol and once without. With one further change, namely deleting the production *typedef-name: identifier* and making *typedef-name* a terminal symbol, this grammar is acceptable to the YACC parser-generator. It has only one conflict, generated by the `if-else` ambiguity.

*translation-unit:*
  *external-declaration*
  *translation-unit external-declaration*

*external-declaration:*
  *function-definition*
  *declaration*

*function-definition:*
  *declaration-specifiers*$_{opt}$ *declarator declaration-list*$_{opt}$ *compound-statement*

*declaration:*
  *declaration-specifiers init-declarator-list*$_{opt}$ `;`

*declaration-list:*
  *declaration*
  *declaration-list declaration*

*declaration-specifiers:*
  *storage-class-specifier declaration-specifiers*$_{opt}$
  *type-specifier declaration-specifiers*$_{opt}$
  *type-qualifier declaration-specifiers*$_{opt}$

*storage-class-specifier:* one of
  `auto register static extern typedef`

*type-specifier:* one of
  `void char short int long float double signed`
  `unsigned` *struct-or-union-specifier enum-specifier typedef-name*

*type-qualifier:* one of
  `const volatile`

*struct-or-union-specifier:*
  *struct-or-union identifier*$_{opt}$ `{` *struct-declaration-list* `}`
  *struct-or-union identifier*

*struct-or-union:* one of
  `struct union`

*struct-declaration-list:*
  *struct-declaration*
  *struct-declaration-list struct-declaration*

*init-declarator-list:*
*init-declarator*
*init-declarator-list* , *init-declarator*

*init-declarator:*
*declarator*
*declarator* = *initializer*

*struct-declaration:*
*specifier-qualifier-list struct-declarator-list* ;

*specifier-qualifier-list:*
*type-specifier specifier-qualifier-list*$_{opt}$
*type-qualifier specifier-qualifier-list*$_{opt}$

*struct-declarator-list:*
*struct-declarator*
*struct-declarator-list* , *struct-declarator*

*struct-declarator:*
*declarator*
*declarator*$_{opt}$ : *constant-expression*

*enum-specifier:*
`enum` *identifier*$_{opt}$ { *enumerator-list* }
`enum` *identifier*

*enumerator-list:*
*enumerator*
*enumerator-list* , *enumerator*

*enumerator:*
*identifier*
*identifier* = *constant-expression*

*declarator:*
*pointer*$_{opt}$ *direct-declarator*

*direct-declarator:*
*identifier*
( *declarator* )
*direct-declarator* [ *constant-expression*$_{opt}$
*direct-declarator* ( *parameter-type-list* )
*direct-declarator* ( *identifier-list*$_{opt}$ )

*pointer:*
* *type-qualifier-list*$_{opt}$
* *type-qualifier-list*$_{opt}$ *pointer*

*type-qualifier-list:*
*type-qualifier*
*type-qualifier-list type-qualifier*

*parameter-type-list:*
*parameter-list*
*parameter-list* , ...

*parameter-list:*
*parameter-declaration*
*parameter-list* , *parameter-declaration*

*parameter-declaration:*
  *declaration-specifiers declarator*
  *declaration-specifiers abstract-declarator*$_{opt}$

*identifier-list:*
  *identifier*
  *identifier-list* , *identifier*

*initializer:*
  *assignment-expression*
  { *initializer-list* }
  { *initializer-list* , }

*initializer-list:*
  *initializer*
  *initializer-list* , *initializer*

*type-name:*
  *specifier-qualifier-list abstract-declarator*$_{opt}$

*abstract-declarator:*
  *pointer*
  *pointer*$_{opt}$ *direct-abstract-declarator*

*direct-abstract-declarator:*
  ( *abstract-declarator* )
  *direct-abstract-declarator*$_{opt}$ [ *constant-expression*$_{opt}$ ]
  *direct-abstract-declarator*$_{opt}$ ( *parameter-type-list*$_{opt}$ )

*typedef-name:*
  *identifier*

*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*$_{opt}$ ;

*compound-statement:*
  { *declaration-list*$_{opt}$ *statement-list*$_{opt}$ }

*statement-list:*
  *statement*
  *statement-list statement*

*selection-statement:*
  `if` ( *expression* ) *statement*
  `if` ( *expression* ) *statement* `else` *statement*
  `switch` ( *expression* ) *statement*

*iteration-statement:*
 `while ( `*expression*` ) `*statement*
 `do `*statement*` while ( `*expression*` ) ;`
 `for ( `*expression*$_{opt}$` ; `*expression*$_{opt}$` ; `*expression*$_{opt}$` ) `*statement*

*jump-statement:*
 `goto `*identifier*` ;`
 `continue ;`
 `break ;`
 `return `*expression*$_{opt}$` ;`

*expression:*
 *assignment-expression*
 *expression* , *assignment-expression*

*assignment-expression:*
 *conditional-expression*
 *unary-expression assignment-operator assignment-expression*

*assignment-operator:* one of
 `=  *=  /=  %=  +=  -=  <<=  >>=  &=  ^=  |=`

*conditional-expression:*
 *logical-OR-expression*
 *logical-OR-expression* ? *expression* : *conditional-expression*

*constant-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:* one of
  & * + - ~ !

*postfix-expression:*
  *primary-expression*
  *postfix-expression* [ *expression* ]
  *postfix-expression* ( *argument-expression-list*$_{opt}$ )
  *postfix-expression* . *identifier*
  *postfix-expression* -> *identifier*
  *postfix-expression* ++
  *postfix-expression* --

*primary-expression:*
  *identifier*
  *constant*
  *string*
  ( *expression* )

*argument-expression-list:*
  *assignment-expression*
  *argument-expression-list* , *assignment-expression*

*constant:*
  *integer-constant*
  *character-constant*
  *floating-constant*
  *enumeration-constant*

The following grammar for the preprocessor summarizes the structure of control lines, but is not suitable for mechanized parsing. It includes the symbol *text*, which means ordinary program text, non-conditional preprocessor control lines, or complete preprocessor conditional constructions.