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

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

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

(6.5.15) *conditional-expression:*
*logical-OR-expression*
*logical-OR-expression* **?** *expression* **:** *conditional-expression*

(6.5.16) *assignment-expression:*
*conditional-expression*
*unary-expression assignment-operator assignment-expression*

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

(6.5.17) *expression:*
*assignment-expression*
*expression* **,** *assignment-expression*

(6.6) *constant-expression:*
*conditional-expression*

## A.2.2 Declarations

(6.7) *declaration:*
*declaration-specifiers init-declarator-list*$_{opt}$ **;**
*static_assert-declaration*

(6.7) *declaration-specifiers:*
*storage-class-specifier declaration-specifiers*$_{opt}$
*type-specifier declaration-specifiers*$_{opt}$
*type-qualifier declaration-specifiers*$_{opt}$
*function-specifier declaration-specifiers*$_{opt}$
*alignment-specifier declaration-specifiers*$_{opt}$

(6.7) *init-declarator-list:*
*init-declarator*
*init-declarator-list* **,** *init-declarator*

(6.7) *init-declarator:*
*declarator*
*declarator* **=** *initializer*

(6.7.1) *storage-class-specifier:*
**typedef**
**extern**
**static**
**_Thread_local**
**auto**
**register**

(6.7.2) *type-specifier:*
**void**
**char**
**short**
**int**
**long**
**float**
**double**
**signed**
**unsigned**
**_Bool**
**_Complex**
*atomic-type-specifier*
*struct-or-union-specifier*
*enum-specifier*
*typedef-name*

(6.7.2.1) *struct-or-union-specifier:*
*struct-or-union* *identifier*$_{opt}$ **{** *struct-declaration-list* **}**
*struct-or-union* *identifier*

(6.7.2.1) *struct-or-union:*
**struct**
**union**

(6.7.2.1) *struct-declaration-list:*
*struct-declaration*
*struct-declaration-list* *struct-declaration*

(6.7.2.1) *struct-declaration:*
*specifier-qualifier-list* *struct-declarator-list*$_{opt}$ **;**
*static_assert-declaration*

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

(6.7.2.1) *struct-declarator-list:*
*struct-declarator*
*struct-declarator-list* **,** *struct-declarator*

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

(6.7.2.2) *enum-specifier:*
**enum** *identifier*$_{opt}$ **{** *enumerator-list* **}**
**enum** *identifier*$_{opt}$ **{** *enumerator-list* **, }**
**enum** *identifier*

(6.7.2.2) *enumerator-list:*
*enumerator*
*enumerator-list* **,** *enumerator*

(6.7.2.2) *enumerator:*
*enumeration-constant*
*enumeration-constant* **=** *constant-expression*

(6.7.2.4) *atomic-type-specifier:*
**_Atomic (** *type-name* **)**

(6.7.3) *type-qualifier:*
**const**
**restrict**
**volatile**
**_Atomic**

(6.7.4) *function-specifier:*
**inline**
**_Noreturn**

(6.7.5) *alignment-specifier:*
**_Alignas (** *type-name* **)**
**_Alignas (** *constant-expression* **)**

(6.7.6) *declarator:*
*pointer*$_{opt}$ *direct-declarator*

(6.7.6) *direct-declarator:*
*identifier*
`(` *declarator* `)`
*direct-declarator* `[` *type-qualifier-list*$_{opt}$ *assignment-expression*$_{opt}$ `]`
*direct-declarator* `[` `static` *type-qualifier-list*$_{opt}$ *assignment-expression* `]`
*direct-declarator* `[` *type-qualifier-list* `static` *assignment-expression* `]`
*direct-declarator* `[` *type-qualifier-list*$_{opt}$ `*` `]`
*direct-declarator* `(` *parameter-type-list* `)`
*direct-declarator* `(` *identifier-list*$_{opt}$ `)`

(6.7.6) *pointer:*
`*` *type-qualifier-list*$_{opt}$
`*` *type-qualifier-list*$_{opt}$ *pointer*

(6.7.6) *type-qualifier-list:*
*type-qualifier*
*type-qualifier-list type-qualifier*

(6.7.6) *parameter-type-list:*
*parameter-list*
*parameter-list* `,` `...`

(6.7.6) *parameter-list:*
*parameter-declaration*
*parameter-list* `,` *parameter-declaration*

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

(6.7.6) *identifier-list:*
*identifier*
*identifier-list* `,` *identifier*

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

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

(6.7.7) *direct-abstract-declarator:*
**(** *abstract-declarator* **)**
*direct-abstract-declarator*$_{opt}$ **[** *type-qualifier-list*$_{opt}$ *assignment-expression*$_{opt}$ **]**
*direct-abstract-declarator*$_{opt}$ **[ static** *type-qualifier-list*$_{opt}$ *assignment-expression* **]**
*direct-abstract-declarator*$_{opt}$ **[** *type-qualifier-list* **static** *assignment-expression* **]**
*direct-abstract-declarator*$_{opt}$ **[ * ]**
*direct-abstract-declarator*$_{opt}$ **(** *parameter-type-list*$_{opt}$ **)**

(6.7.8) *typedef-name:*
*identifier*

(6.7.9) *initializer:*
*assignment-expression*
**{** *initializer-list* **}**
**{** *initializer-list* **, }**

(6.7.9) *initializer-list:*
*designation*$_{opt}$ *initializer*
*initializer-list* **,** *designation*$_{opt}$ *initializer*

(6.7.9) *designation:*
*designator-list* **=**

(6.7.9) *designator-list:*
*designator*
*designator-list designator*

(6.7.9) *designator:*
**[** *constant-expression* **]**
**.** *identifier*

(6.7.10) *static_assert-declaration:*
**_Static_assert (** *constant-expression* **,** *string-literal* **) ;**

### A.2.3 Statements

(6.8) *statement:*
*labeled-statement*
*compound-statement*
*expression-statement*
*selection-statement*
*iteration-statement*
*jump-statement*

(6.8.1) *labeled-statement:*
*identifier* **:** *statement*
**case** *constant-expression* **:** *statement*
**default :** *statement*

(6.8.2) *compound-statement:*
**{** *block-item-list*$_{opt}$ **}**

(6.8.2) *block-item-list:*
*block-item*
*block-item-list block-item*

(6.8.2) *block-item:*
*declaration*
*statement*

(6.8.3) *expression-statement:*
*expression*$_{opt}$ **;**

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

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

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

### A.2.4 External definitions

(6.9) *translation-unit:*
*external-declaration*
*translation-unit external-declaration*

(6.9) *external-declaration:*
*function-definition*
*declaration*

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

(6.9.1) *declaration-list:*
*declaration*
*declaration-list declaration*

## A.3 Preprocessing directives

(6.10) *preprocessing-file:*
*group*$_{opt}$

(6.10) *group:*
*group-part*
*group group-part*

(6.10) *group-part:*
*if-section*
*control-line*
*text-line*
**#** *non-directive*

(6.10) *if-section:*
*if-group elif-groups*$_{opt}$ *else-group*$_{opt}$ *endif-line*

(6.10) *if-group:*
**# if** *constant-expression new-line group*$_{opt}$
**# ifdef** *identifier new-line group*$_{opt}$
**# ifndef** *identifier new-line group*$_{opt}$

(6.10) *elif-groups:*
*elif-group*
*elif-groups elif-group*

(6.10) *elif-group:*
**# elif** *constant-expression new-line group*$_{opt}$