Annex A (informative) Grammar summary [gram]

1 This summary of C++ syntax is intended to be an aid to comprehension. It is not an exact statement of the language. In particular, the grammar described here accepts a superset of valid C++ constructs. Disambiguation rules (6.8, 7.1, 10.2) must be applied to distinguish expressions from declarations. Further, access control, ambiguity, and type rules must be used to weed out syntactically valid but meaningless constructs.

A.1 Keywords [gram.key]

1 New context-dependent keywords are introduced into a program by `typedef` (7.1.3), `namespace` (7.3.1), `class` (clause 9), `enumeration` (7.2), and `template` (clause 14) declarations.

typedef-name:
    identifier
namespace-name:
    original-namespace-name
    namespace-alias

original-namespace-name:
    identifier

namespace-alias:
    identifier
class-name:
    identifier
    simple-template-id
enum-name:
    identifier
template-name:
    identifier

Note that a *typedef-name* naming a class is also a *class-name* (9.1).

A.2 Lexical conventions [gram.lex]

hex-quad:
    hexadecimal-digit hexadecimal-digit hexadecimal-digit hexadecimal-digit
universal-character-name:
    `\u` *hex-quad*
    `\U` *hex-quad hex-quad*
preprocessing-token:
    header-name
    identifier
    pp-number
    character-literal
    user-defined-character-literal
    string-literal
    user-defined-string-literal
    preprocessing-op-or-punc
    each non-white-space character that cannot be one of the above

token:
identifier
keyword
literal
operator
punctuator

header-name:
`<` *h-char-sequence* `>`
`"` *q-char-sequence* `"`

h-char-sequence:
h-char
h-char-sequence h-char

h-char:
any member of the source character set except new-line and `>`

q-char-sequence:
q-char
q-char-sequence q-char

q-char:
any member of the source character set except new-line and `"`

pp-number:
digit
`.` *digit*
pp-number digit
pp-number identifier-nondigit
pp-number `e` *sign*
pp-number `E` *sign*
pp-number `.`

identifier:
identifier-nondigit
identifier identifier-nondigit
identifier digit

identifier-nondigit:
nondigit
universal-character-name
other implementation-defined characters

nondigit: one of
`a b c d e f g h i j k l m`
`n o p q r s t u v w x y z`
`A B C D E F G H I J K L M`
`N O P Q R S T U V W X Y Z _`

digit: one of
`0 1 2 3 4 5 6 7 8 9`

preprocessing-op-or-punc: one of

`{`	`}`	`[`	`]`	`#`	`##`	`(`	`)`	
`<:`	`:>`	`<%`	`%>`	`%:`	`%:%:`	`;`	`:`	`...`
`new`	`delete`	`?`	`::`	`.`	`.*`			
`+`	`-`	`*`	`/`	`%`	`^`	`&`	`|`	`~`
`!`	`=`	`<`	`>`	`+=`	`-=`	`*=`	`/=`	`%=`
`^=`	`&=`	`|=`	`<<`	`>>`	`>>=`	`<<=`	`==`	`!=`
`<=`	`>=`	`&&`	`||`	`++`	`--`	`,`	`->*`	`->`
`and`	`and_eq`	`bitand`	`bitor`	`compl`	`not`	`not_eq`		
`or`	`or_eq`	`xor`	`xor_eq`					

literal:
integer-literal
character-literal
floating-literal
string-literal
boolean-literal
pointer-literal
user-defined-literal

integer-literal:
decimal-literal integer-suffix$_{opt}$
octal-literal integer-suffix$_{opt}$
hexadecimal-literal integer-suffix$_{opt}$

decimal-literal:
nonzero-digit
decimal-literal digit

octal-literal:
`0`
octal-literal octal-digit

hexadecimal-literal:
`0x` *hexadecimal-digit*
`0X` *hexadecimal-digit*
hexadecimal-literal hexadecimal-digit

nonzero-digit: one of
`1 2 3 4 5 6 7 8 9`

octal-digit: one of
`0 1 2 3 4 5 6 7`

hexadecimal-digit: one of
`0 1 2 3 4 5 6 7 8 9`
`a b c d e f`
`A B C D E F`

integer-suffix:
unsigned-suffix long-suffix$_{opt}$
unsigned-suffix long-long-suffix$_{opt}$
long-suffix unsigned-suffix$_{opt}$
long-long-suffix unsigned-suffix$_{opt}$

unsigned-suffix: one of
`u U`

long-suffix: one of
`l L`

long-long-suffix: one of
`ll LL`

character-literal:
`'` *c-char-sequence* `'`
u`'` *c-char-sequence* `'`
U`'` *c-char-sequence* `'`
L`'` *c-char-sequence* `'`

c-char-sequence:
c-char
c-char-sequence c-char

c-char:
any member of the source character set except
the single-quote `'`, backslash `\`, or new-line character
escape-sequence
universal-character-name

escape-sequence:
simple-escape-sequence
octal-escape-sequence
hexadecimal-escape-sequence

simple-escape-sequence: one of
`\'` `\"` `\?` `\\`
`\a` `\b` `\f` `\n` `\r` `\t` `\v`

octal-escape-sequence:
`\` *octal-digit*
`\` *octal-digit octal-digit*
`\` *octal-digit octal-digit octal-digit*

hexadecimal-escape-sequence:
`\x` *hexadecimal-digit*
hexadecimal-escape-sequence hexadecimal-digit

floating-literal:
fractional-constant exponent-part$_{opt}$ *floating-suffix*$_{opt}$
digit-sequence exponent-part floating-suffix$_{opt}$

fractional-constant:
digit-sequence$_{opt}$`.` *digit-sequence*
digit-sequence `.`

exponent-part:
`e` *sign*$_{opt}$ *digit-sequence*
`E` *sign*$_{opt}$ *digit-sequence*

sign: one of
`+ -`

digit-sequence:
digit
digit-sequence digit

floating-suffix: one of
`f l F L`

string-literal:
encoding-prefix$_{opt}$`"` *s-char-sequence*$_{opt}$`"`
encoding-prefix$_{opt}$`R` *raw-string*

encoding-prefix:
`u8`
`u`
`U`
`L`

s-char-sequence:
s-char
s-char-sequence s-char

s-char:
any member of the source character set except
the double-quote `"`, backslash `\`, or new-line character
escape-sequence
universal-character-name

raw-string:
`"` *d-char-sequence*$_{opt}$`(` *r-char-sequence*$_{opt}$`)` *d-char-sequence*$_{opt}$`"`

r-char-sequence:
r-char
r-char-sequence r-char

r-char:
any member of the source character set, except
a right parenthesis `)` followed by the initial *d-char-sequence* (which may be empty) followed by a double quote `"`.

d-char-sequence:
d-char
d-char-sequence d-char
d-char:
any member of the basic source character set except:
space, the left parenthesis `(`, the right parenthesis `)`, the backslash `\`, and the control characters representing horizontal tab, vertical tab, form feed, and newline.

boolean-literal:
`false`
`true`
pointer-literal:
`nullptr`
user-defined-literal:
user-defined-integer-literal
user-defined-floating-literal
user-defined-string-literal
user-defined-character-literal
user-defined-integer-literal:
decimal-literal ud-suffix
octal-literal ud-suffix
hexadecimal-literal ud-suffix
user-defined-floating-literal:
fractional-constant exponent-part$_{opt}$ *ud-suffix*
digit-sequence exponent-part ud-suffix
user-defined-string-literal:
string-literal ud-suffix
user-defined-character-literal:
character-literal ud-suffix
ud-suffix:
identifier

A.3 Basic concepts [gram.basic]

translation-unit:
declaration-seq$_{opt}$

A.4 Expressions [gram.expr]

primary-expression:
literal
`this`
`(` *expression* `)`
id-expression
lambda-expression
id-expression:
unqualified-id
qualified-id

unqualified-id:
identifier
operator-function-id
conversion-function-id
literal-operator-id
`~` *class-name*
`~` *decltype-specifier*
template-id

qualified-id:
nested-name-specifier `template`$_{opt}$ *unqualified-id*
`::` *identifier*
`::` *operator-function-id*
`::` *literal-operator-id*
`::` *template-id*

nested-name-specifier:
`::`$_{opt}$ *type-name* `::`
`::`$_{opt}$ *namespace-name* `::`
decltype-specifier `::`
nested-name-specifier identifier `::`
nested-name-specifier `template`$_{opt}$ *simple-template-id* `::`

lambda-expression:
lambda-introducer lambda-declarator$_{opt}$ *compound-statement*

lambda-introducer:
`[` *lambda-capture*$_{opt}$`]`

lambda-capture:
capture-default
capture-list
capture-default `,` *capture-list*

capture-default:
`&`
`=`

capture-list:
capture `...`$_{opt}$
capture-list `,` *capture* `...`$_{opt}$

capture:
identifier
`&` *identifier*
`this`

lambda-declarator:
`(` *parameter-declaration-clause* `)` `mutable`$_{opt}$
exception-specification$_{opt}$ *attribute-specifier-seq*$_{opt}$ *trailing-return-type*$_{opt}$

postfix-expression:
primary-expression
postfix-expression `[` *expression* `]`
postfix-expression `[` *braced-init-list* `]`
postfix-expression `(` *expression-list*$_{opt}$`)`
simple-type-specifier `(` *expression-list*$_{opt}$`)`
typename-specifier `(` *expression-list*$_{opt}$`)`
simple-type-specifier braced-init-list
typename-specifier braced-init-list
postfix-expression `.` `template`$_{opt}$ *id-expression*
postfix-expression `->` `template`$_{opt}$ *id-expression*
postfix-expression `.` *pseudo-destructor-name*
postfix-expression `->` *pseudo-destructor-name*
postfix-expression `++`
postfix-expression `--`
`dynamic_cast` `<` *type-id* `>` `(` *expression* `)`
`static_cast` `<` *type-id* `>` `(` *expression* `)`
`reinterpret_cast` `<` *type-id* `>` `(` *expression* `)`
`const_cast` `<` *type-id* `>` `(` *expression* `)`
`typeid` `(` *expression* `)`
`typeid` `(` *type-id* `)`

expression-list:
initializer-list

pseudo-destructor-name:
nested-name-specifier$_{opt}$ *type-name* `::` `~` *type-name*
nested-name-specifier `template` *simple-template-id* `::` `~` *type-name*
nested-name-specifier$_{opt}$ `~` *type-name*
`~` *decltype-specifier*

unary-expression:
postfix-expression
`++` *cast-expression*
`--` *cast-expression*
unary-operator cast-expression
`sizeof` *unary-expression*
`sizeof` `(` *type-id* `)`
`sizeof` `...` `(` *identifier* `)`
`alignof` `(` *type-id* `)`
noexcept-expression
new-expression
delete-expression

unary-operator: one of
`*` `&` `+` `-` `!` `~`

new-expression:
`::`$_{opt}$`new` *new-placement*$_{opt}$ *new-type-id new-initializer*$_{opt}$
`::`$_{opt}$`new` *new-placement*$_{opt}$`(` *type-id* `)` *new-initializer*$_{opt}$

new-placement:
`(` *expression-list* `)`

new-type-id:
type-specifier-seq new-declarator$_{opt}$

new-declarator:
ptr-operator new-declarator$_{opt}$
noptr-new-declarator

noptr-new-declarator:
`[` *expression* `]` *attribute-specifier-seq*$_{opt}$
noptr-new-declarator `[` *constant-expression* `]` *attribute-specifier-seq*$_{opt}$
new-initializer:
`(` *expression-list*$_{opt}$`)`
braced-init-list
delete-expression:
`::`$_{opt}$`delete` *cast-expression*
`::`$_{opt}$`delete [ ]` *cast-expression*
noexcept-expression:
`noexcept (` *expression* `)`
cast-expression:
unary-expression
`(` *type-id* `)` *cast-expression*
pm-expression:
cast-expression
pm-expression `.*` *cast-expression*
pm-expression `->*` *cast-expression*
multiplicative-expression:
pm-expression
multiplicative-expression `*` *pm-expression*
multiplicative-expression `/` *pm-expression*
multiplicative-expression `%` *pm-expression*
additive-expression:
multiplicative-expression
additive-expression `+` *multiplicative-expression*
additive-expression `-` *multiplicative-expression*
shift-expression:
additive-expression
shift-expression `<<` *additive-expression*
shift-expression `>>` *additive-expression*
relational-expression:
shift-expression
relational-expression `<` *shift-expression*
relational-expression `>` *shift-expression*
relational-expression `<=` *shift-expression*
relational-expression `>=` *shift-expression*
equality-expression:
relational-expression
equality-expression `==` *relational-expression*
equality-expression `!=` *relational-expression*
and-expression:
equality-expression
and-expression `&` *equality-expression*
exclusive-or-expression:
and-expression
exclusive-or-expression `^` *and-expression*
inclusive-or-expression:
exclusive-or-expression
inclusive-or-expression `|` *exclusive-or-expression*
logical-and-expression:
inclusive-or-expression
logical-and-expression `&&` *inclusive-or-expression*

logical-or-expression:
logical-and-expression
logical-or-expression || *logical-and-expression*

conditional-expression:
logical-or-expression
logical-or-expression `?` *expression* `:` *assignment-expression*

assignment-expression:
conditional-expression
logical-or-expression assignment-operator initializer-clause
throw-expression

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

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

constant-expression:
conditional-expression

A.5 Statements [gram.stmt]

statement:
labeled-statement
attribute-specifier-seq$_{opt}$ *expression-statement*
attribute-specifier-seq$_{opt}$ *compound-statement*
attribute-specifier-seq$_{opt}$ *selection-statement*
attribute-specifier-seq$_{opt}$ *iteration-statement*
attribute-specifier-seq$_{opt}$ *jump-statement*
declaration-statement
attribute-specifier-seq$_{opt}$ *try-block*

labeled-statement:
attribute-specifier-seq$_{opt}$ *identifier* `:` *statement*
attribute-specifier-seq$_{opt}$`case` *constant-expression* `:` *statement*
attribute-specifier-seq$_{opt}$`default :` *statement*

expression-statement:
expression$_{opt}$`;`

compound-statement:
`{` *statement-seq*$_{opt}$`}`

statement-seq:
statement
statement-seq statement

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

condition:
expression
attribute-specifier-seq$_{opt}$ *decl-specifier-seq declarator* `=` *initializer-clause*
attribute-specifier-seq$_{opt}$ *decl-specifier-seq declarator braced-init-list*

iteration-statement:
`while (` *condition* `)` *statement*
`do` *statement* `while (` *expression* `) ;`
`for (` *for-init-statement condition*$_{opt}$`;` *expression*$_{opt}$`)` *statement*
`for (` *for-range-declaration* `:` *for-range-initializer* `)` *statement*

for-init-statement:
expression-statement
simple-declaration
for-range-declaration:
attribute-specifier-seq$_{opt}$ *decl-specifier-seq declarator*
for-range-initializer:
expression
braced-init-list
jump-statement:
`break` `;`
`continue` `;`
`return` *expression*$_{opt}$`;`
`return` *braced-init-list* `;`
`goto` *identifier* `;`
declaration-statement:
block-declaration

A.6 Declarations [gram.dcl]

declaration-seq:
declaration
declaration-seq declaration
declaration:
block-declaration
function-definition
template-declaration
explicit-instantiation
explicit-specialization
linkage-specification
namespace-definition
empty-declaration
attribute-declaration
block-declaration:
simple-declaration
asm-definition
namespace-alias-definition
using-declaration
using-directive
static_assert-declaration
alias-declaration
opaque-enum-declaration
alias-declaration:
`using` *identifier attribute-specifier-seq*$_{opt}$ `=` *type-id* `;`
simple-declaration:
decl-specifier-seq$_{opt}$ *init-declarator-list*$_{opt}$`;`
attribute-specifier-seq decl-specifier-seq$_{opt}$ *init-declarator-list* `;`
static_assert-declaration:
`static_assert` `(` *constant-expression* `,` *string-literal* `)` `;`
empty-declaration:
`;`
attribute-declaration:
attribute-specifier-seq `;`

decl-specifier:
storage-class-specifier
type-specifier
function-specifier
`friend`
`typedef`
`constexpr`

decl-specifier-seq:
decl-specifier attribute-specifier-seq$_{opt}$
decl-specifier decl-specifier-seq

storage-class-specifier:
`register`
`static`
`thread_local`
`extern`
`mutable`

function-specifier:
`inline`
`virtual`
`explicit`

typedef-name:
identifier

type-specifier:
trailing-type-specifier
class-specifier
enum-specifier

trailing-type-specifier:
simple-type-specifier
elaborated-type-specifier
typename-specifier
cv-qualifier

type-specifier-seq:
type-specifier attribute-specifier-seq$_{opt}$
type-specifier type-specifier-seq

trailing-type-specifier-seq:
trailing-type-specifier attribute-specifier-seq$_{opt}$
trailing-type-specifier trailing-type-specifier-seq

simple-type-specifier:
nested-name-specifier$_{opt}$ *type-name*
nested-name-specifier `template` *simple-template-id*
`char`
`char16_t`
`char32_t`
`wchar_t`
`bool`
`short`
`int`
`long`
`signed`
`unsigned`
`float`
`double`
`void`
`auto`
decltype-specifier

type-name:
class-name
enum-name
typedef-name
simple-template-id

decltype-specifier:
`decltype (` *expression* `)`

elaborated-type-specifier:
class-key attribute-specifier-seq$_{opt}$ *nested-name-specifier*$_{opt}$ *identifier*
class-key nested-name-specifier$_{opt}$`template`$_{opt}$ *simple-template-id*
`enum` *nested-name-specifier*$_{opt}$ *identifier*

enum-name:
identifier

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

enum-head:
enum-key attribute-specifier-seq$_{opt}$ *identifier*$_{opt}$ *enum-base*$_{opt}$
enum-key attribute-specifier-seq$_{opt}$ *nested-name-specifier identifier*
enum-base$_{opt}$

opaque-enum-declaration:
enum-key attribute-specifier-seq$_{opt}$ *identifier enum-base*$_{opt}$ `;`

enum-key:
`enum`
`enum class`
`enum struct`

enum-base:
`:` *type-specifier-seq*

enumerator-list:
enumerator-definition
enumerator-list `,` *enumerator-definition*

enumerator-definition:
enumerator
enumerator `=` *constant-expression*

enumerator:
identifier

namespace-name:
original-namespace-name
namespace-alias

original-namespace-name:
identifier

namespace-definition:
named-namespace-definition
unnamed-namespace-definition

named-namespace-definition:
original-namespace-definition
extension-namespace-definition

original-namespace-definition:
`inline`$_{opt}$ `namespace` *identifier* `{` *namespace-body* `}`

extension-namespace-definition:
`inline`$_{opt}$ `namespace` *original-namespace-name* `{` *namespace-body* `}`

unnamed-namespace-definition:
`inline`$_{opt}$ `namespace` `{` *namespace-body* `}`

namespace-body:
declaration-seq$_{opt}$

namespace-alias:
identifier

namespace-alias-definition:
`namespace` *identifier* `=` *qualified-namespace-specifier* `;`

qualified-namespace-specifier:
nested-name-specifier$_{opt}$ *namespace-name*

using-declaration:
`using typename`$_{opt}$ *nested-name-specifier unqualified-id* `;`
`using ::` *unqualified-id* `;`

using-directive:
attribute-specifier-seq$_{opt}$`using namespace` *nested-name-specifier*$_{opt}$ *namespace-name* `;`

asm-definition:
`asm ( ` *string-literal* ` ) ;`

linkage-specification:
`extern` *string-literal* `{` *declaration-seq*$_{opt}$`}`
`extern` *string-literal declaration*

attribute-specifier-seq:
attribute-specifier-seq$_{opt}$ *attribute-specifier*

attribute-specifier:
`[ [` *attribute-list* `] ]`
alignment-specifier

alignment-specifier:
`alignas ( ` *type-id* `...`$_{opt}$`)`
`alignas ( ` *alignment-expression* `...`$_{opt}$`)`

attribute-list:
attribute$_{opt}$
attribute-list `,` *attribute*$_{opt}$
attribute `...`
attribute-list `,` *attribute* `...`

attribute:
attribute-token attribute-argument-clause$_{opt}$

attribute-token:
identifier
attribute-scoped-token
attribute-scoped-token:
attribute-namespace `::` *identifier*
attribute-namespace:
identifier
attribute-argument-clause:
`(` *balanced-token-seq* `)`
balanced-token-seq:
balanced-token$_{opt}$
balanced-token-seq balanced-token
balanced-token:
`(` *balanced-token-seq* `)`
`[` *balanced-token-seq* `]`
`{` *balanced-token-seq* `}`
any *token* other than a parenthesis, a bracket, or a brace

A.7 Declarators [gram.decl]

init-declarator-list:
init-declarator
init-declarator-list `,` *init-declarator*
init-declarator:
declarator initializer$_{opt}$
declarator:
ptr-declarator
noptr-declarator parameters-and-qualifiers trailing-return-type
ptr-declarator:
noptr-declarator
ptr-operator ptr-declarator
noptr-declarator:
declarator-id attribute-specifier-seq$_{opt}$
noptr-declarator parameters-and-qualifiers
noptr-declarator `[` *constant-expression*$_{opt}$`]` *attribute-specifier-seq*$_{opt}$
`(` *ptr-declarator* `)`
parameters-and-qualifiers:
`(` *parameter-declaration-clause* `)` *attribute-specifier-seq*$_{opt}$ *cv-qualifier-seq*$_{opt}$
ref-qualifier$_{opt}$ *exception-specification*$_{opt}$
trailing-return-type:
`->` *trailing-type-specifier-seq abstract-declarator*$_{opt}$
ptr-operator:
`*` *attribute-specifier-seq*$_{opt}$ *cv-qualifier-seq*$_{opt}$
`&` *attribute-specifier-seq*$_{opt}$
`&&` *attribute-specifier-seq*$_{opt}$
nested-name-specifier `*` *attribute-specifier-seq*$_{opt}$ *cv-qualifier-seq*$_{opt}$
cv-qualifier-seq:
cv-qualifier cv-qualifier-seq$_{opt}$
cv-qualifier:
`const`
`volatile`
ref-qualifier:
`&`
`&&`

declarator-id:
...$_{opt}$ *id-expression*
nested-name-specifier$_{opt}$ *class-name*

type-id:
type-specifier-seq abstract-declarator$_{opt}$

abstract-declarator:
ptr-abstract-declarator
noptr-abstract-declarator$_{opt}$ *parameters-and-qualifiers trailing-return-type*
abstract-pack-declarator

ptr-abstract-declarator:
noptr-abstract-declarator
ptr-operator ptr-abstract-declarator$_{opt}$

noptr-abstract-declarator:
noptr-abstract-declarator$_{opt}$ *parameters-and-qualifiers*
noptr-abstract-declarator$_{opt}$ `[` *constant-expression*$_{opt}$ `]` *attribute-specifier-seq*$_{opt}$
`(` *ptr-abstract-declarator* `)`

abstract-pack-declarator:
noptr-abstract-pack-declarator
ptr-operator abstract-pack-declarator

noptr-abstract-pack-declarator:
noptr-abstract-pack-declarator parameters-and-qualifiers
noptr-abstract-pack-declarator `[` *constant-expression*$_{opt}$ `]` *attribute-specifier-seq*$_{opt}$
...

parameter-declaration-clause:
parameter-declaration-list$_{opt}$*...*$_{opt}$
parameter-declaration-list `,` *...*

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

parameter-declaration:
attribute-specifier-seq$_{opt}$ *decl-specifier-seq declarator*
attribute-specifier-seq$_{opt}$ *decl-specifier-seq declarator* `=` *initializer-clause*
attribute-specifier-seq$_{opt}$ *decl-specifier-seq abstract-declarator*$_{opt}$
attribute-specifier-seq$_{opt}$ *decl-specifier-seq abstract-declarator*$_{opt}$`=` *initializer-clause*

function-definition:
attribute-specifier-seq$_{opt}$ *decl-specifier-seq*$_{opt}$ *declarator virt-specifier-seq*$_{opt}$ *function-body*

function-body:
ctor-initializer$_{opt}$ *compound-statement*
function-try-block
`= default ;`
`= delete ;`

initializer:
brace-or-equal-initializer
`(` *expression-list* `)`

brace-or-equal-initializer:
`=` *initializer-clause*
braced-init-list

initializer-clause:
assignment-expression
braced-init-list

initializer-list:
initializer-clause ...$_{opt}$
initializer-list , *initializer-clause* ...$_{opt}$

braced-init-list:
{ *initializer-list* ,$_{opt}$ }
{ }

A.8 Classes [gram.class]

class-name:
identifier
simple-template-id

class-specifier:
class-head { *member-specification*$_{opt}$}

class-head:
class-key attribute-specifier-seq$_{opt}$ *class-head-name class-virt-specifier*$_{opt}$ *base-clause*$_{opt}$
class-key attribute-specifier-seq$_{opt}$ *base-clause*$_{opt}$

class-head-name:
nested-name-specifier$_{opt}$ *class-name*

class-virt-specifier:
final

class-key:
class
struct
union

member-specification:
member-declaration member-specification$_{opt}$
access-specifier : *member-specification*$_{opt}$

member-declaration:
attribute-specifier-seq$_{opt}$ *decl-specifier-seq*$_{opt}$ *member-declarator-list*$_{opt}$;
function-definition ;$_{opt}$
using-declaration
static_assert-declaration
template-declaration
alias-declaration

member-declarator-list:
member-declarator
member-declarator-list , *member-declarator*

member-declarator:
declarator virt-specifier-seq$_{opt}$ *pure-specifier*$_{opt}$
declarator brace-or-equal-initializer$_{opt}$
identifier$_{opt}$ *attribute-specifier-seq*$_{opt}$: *constant-expression*

virt-specifier-seq:
virt-specifier
virt-specifier-seq virt-specifier

virt-specifier:
override
final

pure-specifier:
= 0

A.9 Derived classes [gram.derived]

base-clause:
: *base-specifier-list*

base-specifier-list:
base-specifier ...$_{opt}$
base-specifier-list , *base-specifier* ...$_{opt}$

base-specifier:
attribute-specifier-seq$_{opt}$ *base-type-specifier*
attribute-specifier-seq$_{opt}$ `virtual` *access-specifier*$_{opt}$ *base-type-specifier*
attribute-specifier-seq$_{opt}$ *access-specifier* `virtual`$_{opt}$ *base-type-specifier*

class-or-decltype:
nested-name-specifier$_{opt}$ *class-name*
decltype-specifier

base-type-specifier:
class-or-decltype

access-specifier:
`private`
`protected`
`public`

A.10 Special member functions [gram.special]

conversion-function-id:
`operator` *conversion-type-id*

conversion-type-id:
type-specifier-seq conversion-declarator$_{opt}$

conversion-declarator:
ptr-operator conversion-declarator$_{opt}$

ctor-initializer:
`:` *mem-initializer-list*

mem-initializer-list:
mem-initializer ...$_{opt}$
mem-initializer , *mem-initializer-list* ...$_{opt}$

mem-initializer:
mem-initializer-id `(` *expression-list*$_{opt}$`)`
mem-initializer-id braced-init-list

mem-initializer-id:
class-or-decltype
identifier

A.11 Overloading [gram.over]

operator-function-id:
`operator` *operator*

operator: one of

`new`	`delete`	`new[]`	`delete[]`					
`+`	`-`	`*`	`/`	`%`	`^`	`&`	`|`	$\sim$
`!`	`=`	`<`	`>`	`+=`	`-=`	`*=`	`/=`	`%=`
`^=`	`&=`	`|=`	`<<`	`>>`	`>>=`	`<<=`	`==`	`!=`
`<=`	`>=`	`&&`	`||`	`++`	`--`	`,`	`->*`	`->`
`()`	`[]`							

literal-operator-id:
`operator ""` *identifier*

A.12 Templates [gram.temp]

template-declaration:
`template <` *template-parameter-list* `>` *declaration*

template-parameter-list:
template-parameter
template-parameter-list , *template-parameter*
template-parameter:
type-parameter
parameter-declaration
type-parameter:
`class` ...$_{opt}$ *identifier*$_{opt}$
`class` *identifier*$_{opt}$`=` *type-id*
`typename` ...$_{opt}$ *identifier*$_{opt}$
`typename` *identifier*$_{opt}$`=` *type-id*
`template <` *template-parameter-list* `> class` ...$_{opt}$ *identifier*$_{opt}$
`template <` *template-parameter-list* `> class` *identifier*$_{opt}$`=` *id-expression*
simple-template-id:
template-name `<` *template-argument-list*$_{opt}$`>`
template-id:
simple-template-id
operator-function-id `<` *template-argument-list*$_{opt}$`>`
literal-operator-id `<` *template-argument-list*$_{opt}$`>`
template-name:
identifier
template-argument-list:
template-argument ...$_{opt}$
template-argument-list , *template-argument* ...$_{opt}$
template-argument:
constant-expression
type-id
id-expression
typename-specifier:
`typename` *nested-name-specifier identifier*
`typename` *nested-name-specifier* `template`$_{opt}$ *simple-template-id*
explicit-instantiation:
`extern`$_{opt}$ `template` *declaration*
explicit-specialization:
`template < >` *declaration*

A.13 Exception handling [gram.except]

try-block:
`try` *compound-statement handler-seq*
function-try-block:
`try` *ctor-initializer*$_{opt}$ *compound-statement handler-seq*
handler-seq:
handler handler-seq$_{opt}$
handler:
`catch (` *exception-declaration* `)` *compound-statement*
exception-declaration:
attribute-specifier-seq$_{opt}$ *type-specifier-seq declarator*
attribute-specifier-seq$_{opt}$ *type-specifier-seq abstract-declarator*$_{opt}$
...
throw-expression:
`throw` *assignment-expression*$_{opt}$

exception-specification:
dynamic-exception-specification
noexcept-specification

dynamic-exception-specification:
`throw (` *type-id-list*$_{opt}$`)`

type-id-list:
type-id `...`$_{opt}$
type-id-list `,` *type-id* `...`$_{opt}$

noexcept-specification:
`noexcept (` *constant-expression* `)`
`noexcept`

A.14 Preprocessing directives [gram.cpp]

preprocessing-file:
group$_{opt}$

group:
group-part
group group-part

group-part:
if-section
control-line
text-line
 # non-directive

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

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

elif-groups:
elif-group
elif-groups elif-group

elif-group:
`# elif` *constant-expression new-line group*$_{opt}$

else-group:
`# else` *new-line group*$_{opt}$

endif-line:
`# endif` *new-line*

control-line:
`# include` *pp-tokens new-line*
`# define` *identifier replacement-list new-line*
`# define` *identifier lparen identifier-list*$_{opt}$`)` *replacement-list new-line*
`# define` *identifier lparen* `...` `)` *replacement-list new-line*
`# define` *identifier lparen identifier-list,* `...` `)` *replacement-list new-line*
`# undef` *identifier new-line*
`# line` *pp-tokens new-line*
`# error` *pp-tokens*$_{opt}$ *new-line*
`# pragma` *pp-tokens*$_{opt}$ *new-line*
`#` *new-line*

text-line:
pp-tokens$_{opt}$ *new-line*

non-directive:
pp-tokens new-line

lparen:
a `(` character not immediately preceded by white-space

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

replacement-list:
pp-tokens$_{opt}$

pp-tokens:
preprocessing-token
pp-tokens preprocessing-token

new-line:
the new-line character