

Annex A (informative)

Grammar summary

[gram]

A.1 General

[gram.general]

- ¹ This summary of C++ grammar 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 (8.9, 9.2, 6.5.2) are applied to distinguish expressions from declarations. Further, access control, ambiguity, and type rules are used to weed out syntactically valid but meaningless constructs.

A.2 Keywords

[gram.key]

- ¹ New context-dependent keywords are introduced into a program by `typedef` (9.2.4), `namespace` (9.8.2), `class` (Clause 11), `enumeration` (9.7.1), and `template` (Clause 13) declarations.

typedef-name:
identifier
simple-template-id

namespace-name:
identifier
namespace-alias

namespace-alias:
identifier

class-name:
identifier
simple-template-id

enum-name:
identifier

template-name:
identifier

A.3 Lexical conventions

[gram.lex]

n-char: one of
 any member of the translation character set except the U+007D RIGHT CURLY BRACKET or new-line character

n-char-sequence:
n-char
n-char-sequence n-char

named-universal-character:
 \N{ *n-char-sequence* }

hex-quad:
hexadecimal-digit hexadecimal-digit hexadecimal-digit hexadecimal-digit

simple-hexadecimal-digit-sequence:
hexadecimal-digit
simple-hexadecimal-digit-sequence hexadecimal-digit

universal-character-name:
 \u *hex-quad*
 \U *hex-quad hex-quad*
 \u{ *simple-hexadecimal-digit-sequence* }
named-universal-character

preprocessing-token:

header-name
import-keyword
module-keyword
export-keyword
identifier
pp-number
character-literal
user-defined-character-literal
string-literal
user-defined-string-literal
preprocessing-op-or-punc
 each non-whitespace character that cannot be one of the above

token:

identifier
keyword
literal
operator-or-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 translation character set except new-line and U+003E GREATER-THAN SIGN

q-char-sequence:

q-char
q-char-sequence q-char

q-char:

any member of the translation character set except new-line and U+0022 QUOTATION MARK

pp-number:

digit
 . *digit*
pp-number identifier-continue
pp-number ' *digit*
pp-number ' *nondigit*
pp-number e *sign*
pp-number E *sign*
pp-number p *sign*
pp-number P *sign*
pp-number .

identifier:

identifier-start
identifier identifier-continue

identifier-start:

nondigit
 an element of the translation character set with the Unicode property `XID_Start`

identifier-continue:

digit
nondigit
 an element of the translation character set with the Unicode property `XID_Continue`

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

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}
unsigned-suffix size-suffix_{opt}
long-suffix unsigned-suffix_{opt}
long-long-suffix unsigned-suffix_{opt}
size-suffix unsigned-suffix_{opt}

unsigned-suffix: one of

u U

long-suffix: one of

l L

long-long-suffix: one of

ll LL

size-suffix: one of

z Z

character-literal:

encoding-prefix_{opt} ' *c-char-sequence* '

encoding-prefix: one of

u8 u U L

c-char-sequence:

c-char
c-char-sequence c-char

c-char:

basic-c-char
escape-sequence
universal-character-name

basic-c-char:

any member of the translation character set except the U+0027 APOSTROPHE,
U+005C REVERSE SOLIDUS, or new-line character

escape-sequence:

simple-escape-sequence
numeric-escape-sequence
conditional-escape-sequence

simple-escape-sequence:

\ *simple-escape-sequence-char*

simple-escape-sequence-char: one of

' " ? \ a b f n r t v

numeric-escape-sequence:

octal-escape-sequence
hexadecimal-escape-sequence

simple-octal-digit-sequence:

octal-digit
simple-octal-digit-sequence octal-digit

octal-escape-sequence:

\ *octal-digit*
\ *octal-digit octal-digit*
\ *octal-digit octal-digit octal-digit*
\ o{ *simple-octal-digit-sequence* }

hexadecimal-escape-sequence:

\x *simple-hexadecimal-digit-sequence*
\x{ *simple-hexadecimal-digit-sequence* }

conditional-escape-sequence:

\ *conditional-escape-sequence-char*

conditional-escape-sequence-char:

any member of the basic character set that is not an *octal-digit*, a *simple-escape-sequence-char*, or the characters N, o, u, U, or x

floating-point-literal:

decimal-floating-point-literal

hexadecimal-floating-point-literal

decimal-floating-point-literal:

fractional-constant *exponent-part*_{opt} *floating-point-suffix*_{opt}

digit-sequence *exponent-part* *floating-point-suffix*_{opt}

hexadecimal-floating-point-literal:

hexadecimal-prefix *hexadecimal-fractional-constant* *binary-exponent-part* *floating-point-suffix*_{opt}

hexadecimal-prefix *hexadecimal-digit-sequence* *binary-exponent-part* *floating-point-suffix*_{opt}

fractional-constant:

*digit-sequence*_{opt} . *digit-sequence*

digit-sequence .

hexadecimal-fractional-constant:

*hexadecimal-digit-sequence*_{opt} . *hexadecimal-digit-sequence*

hexadecimal-digit-sequence .

exponent-part:

e *sign*_{opt} *digit-sequence*

E *sign*_{opt} *digit-sequence*

binary-exponent-part:

p *sign*_{opt} *digit-sequence*

P *sign*_{opt} *digit-sequence*

sign: one of

+ -

digit-sequence:

digit

digit-sequence ' _{opt} *digit*

floating-point-suffix: one of

f l f16 f32 f64 f128 bf16 F L F16 F32 F64 F128 BF16

string-literal:

*encoding-prefix*_{opt} " *s-char-sequence*_{opt} "

*encoding-prefix*_{opt} R *raw-string*

s-char-sequence:

s-char

s-char-sequence *s-char*

s-char:

basic-s-char

escape-sequence

universal-character-name

basic-s-char:

any member of the translation character set except the U+0022 QUOTATION MARK,
U+005C REVERSE SOLIDUS, or new-line character

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 translation character set, except a U+0029 RIGHT PARENTHESIS followed by
the initial *d-char-sequence* (which may be empty) followed by a U+0022 QUOTATION MARK

d-char-sequence:

d-char

d-char-sequence d-char

d-char:

any member of the basic character set except:

U+0020 SPACE, U+0028 LEFT PARENTHESIS, U+0029 RIGHT PARENTHESIS, U+005C REVERSE SOLIDUS,
U+0009 CHARACTER TABULATION, U+000B LINE TABULATION, U+000C FORM FEED, and new-line

boolean-literal:

false

true

pointer-literal:

nullptr

user-defined-literal:

user-defined-integer-literal

user-defined-floating-point-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

binary-literal ud-suffix

user-defined-floating-point-literal:

fractional-constant exponent-part_{opt} ud-suffix

digit-sequence exponent-part ud-suffix

hexadecimal-prefix hexadecimal-fractional-constant binary-exponent-part ud-suffix

hexadecimal-prefix hexadecimal-digit-sequence binary-exponent-part ud-suffix

user-defined-string-literal:

string-literal ud-suffix

user-defined-character-literal:

character-literal ud-suffix

ud-suffix:

identifier

A.4 Basics

[gram.basic]

translation-unit:

declaration-seq_{opt}

global-module-fragment_{opt} module-declaration declaration-seq_{opt} private-module-fragment_{opt}

A.5 Expressions

[gram.expr]

primary-expression:

literal

this

(expression)

id-expression

lambda-expression

fold-expression

requires-expression

id-expression:

unqualified-id

qualified-id

unqualified-id:

identifier

operator-function-id

conversion-function-id

literal-operator-id

~ type-name

~ decltype-specifier

template-id

qualified-id:
nested-name-specifier *template*_{opt} *unqualified-id*

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

lambda-expression:
lambda-introducer *attribute-specifier-seq*_{opt} *lambda-declarator* *compound-statement*
lambda-introducer < *template-parameter-list* > *requires-clause*_{opt} *attribute-specifier-seq*_{opt}
lambda-declarator *compound-statement*

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

lambda-declarator:
lambda-specifier-seq *noexcept-specifier*_{opt} *attribute-specifier-seq*_{opt} *trailing-return-type*_{opt}
noexcept-specifier *attribute-specifier-seq*_{opt} *trailing-return-type*_{opt}
*trailing-return-type*_{opt}
 (*parameter-declaration-clause*) *lambda-specifier-seq*_{opt} *noexcept-specifier*_{opt} *attribute-specifier-seq*_{opt}
*trailing-return-type*_{opt} *requires-clause*_{opt}

lambda-specifier:
constexpr
constexpr
mutable
static

lambda-specifier-seq:
lambda-specifier
lambda-specifier *lambda-specifier-seq*

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

capture-default:
 &
 =

capture-list:
capture
capture-list , *capture*

capture:
simple-capture
init-capture

simple-capture:
identifier ..._{opt}
 & *identifier* ..._{opt}
 this
 * this

init-capture:
 ..._{opt} *identifier* *initializer*
 & ..._{opt} *identifier* *initializer*

fold-expression:
 (*cast-expression* *fold-operator* ...)
 (... *fold-operator* *cast-expression*)
 (*cast-expression* *fold-operator* ... *fold-operator* *cast-expression*)

fold-operator: one of
 + - * / % ^ & | << >>
 += -= *= /= %= ^= &= |= <<= >>= =
 == != < > <= >= && || , .* ->*

```

requires-expression:
    requires requirement-parameter-listopt requirement-body

requirement-parameter-list:
    ( parameter-declaration-clause )

requirement-body:
    { requirement-seq }

requirement-seq:
    requirement
    requirement requirement-seq

requirement:
    simple-requirement
    type-requirement
    compound-requirement
    nested-requirement

simple-requirement:
    expression ;

type-requirement:
    typename nested-name-specifieropt type-name ;

compound-requirement:
    { expression } noexceptopt return-type-requirementopt ;

return-type-requirement:
    -> type-constraint

nested-requirement:
    requires constraint-expression ;

postfix-expression:
    primary-expression
    postfix-expression [ expression-listopt ]
    postfix-expression ( expression-listopt )
    simple-type-specifier ( expression-listopt )
    typename-specifier ( expression-listopt )
    simple-type-specifier braced-init-list
    typename-specifier braced-init-list
    postfix-expression . templateopt id-expression
    postfix-expression -> templateopt id-expression
    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

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

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

```



```

await-expression:
    co_await cast-expression

noexcept-expression:
    noexcept ( expression )

new-expression:
    ::opt new new-placementopt new-type-id new-initializeropt
    ::opt new new-placementopt ( type-id ) new-initializeropt

new-placement:
    ( expression-list )

new-type-id:
    type-specifier-seq new-declaratoropt

new-declarator:
    ptr-operator new-declaratoropt
    noptr-new-declarator

noptr-new-declarator:
    [ expressionopt ] attribute-specifier-seqopt
    noptr-new-declarator [ constant-expression ] attribute-specifier-seqopt

new-initializer:
    ( expression-listopt )
    braced-init-list

delete-expression:
    ::opt delete cast-expression
    ::opt delete [ ] cast-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

compare-expression:
    shift-expression
    compare-expression <=> shift-expression

relational-expression:
    compare-expression
    relational-expression < compare-expression
    relational-expression > compare-expression
    relational-expression <= compare-expression
    relational-expression >= compare-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*

yield-expression:
 co_yield *assignment-expression*
 co_yield *braced-init-list*

throw-expression:
 throw *assignment-expression*_{opt}

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

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

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

constant-expression:
 conditional-expression

A.6 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*

init-statement:
 expression-statement
 simple-declaration
 alias-declaration

condition:
 expression
 *attribute-specifier-seq*_{opt} *decl-specifier-seq* *declarator* *brace-or-equal-initializer*

label:
 *attribute-specifier-seq*_{opt} *identifier* :
 *attribute-specifier-seq*_{opt} *case* *constant-expression* :
 *attribute-specifier-seq*_{opt} *default* :

labeled-statement:
label statement

expression-statement:
expression_{opt} ;

compound-statement:
{ statement-seq_{opt} label-seq_{opt} }

statement-seq:
statement
statement-seq statement

label-seq:
label
label-seq label

selection-statement:
if constexpr_{opt} (init-statement_{opt} condition) statement
if constexpr_{opt} (init-statement_{opt} condition) statement else statement
if !_{opt} consteval compound-statement
if !_{opt} consteval compound-statement else statement
switch (init-statement_{opt} condition) statement

iteration-statement:
while (condition) statement
do statement while (expression) ;
for (init-statement condition_{opt} ; expression_{opt}) statement
for (init-statement_{opt} for-range-declaration : for-range-initializer) statement

for-range-declaration:
attribute-specifier-seq_{opt} decl-specifier-seq declarator
attribute-specifier-seq_{opt} decl-specifier-seq ref-qualifier_{opt} [identifier-list]

for-range-initializer:
expr-or-braced-init-list

jump-statement:
break ;
continue ;
return expr-or-braced-init-list_{opt} ;
coroutine-return-statement
goto identifier ;

coroutine-return-statement:
co_return expr-or-braced-init-list_{opt} ;

declaration-statement:
block-declaration

A.7 Declarations

[gram.dcl]

declaration-seq:
declaration
declaration-seq declaration

declaration:
name-declaration
special-declaration

name-declaration:
block-declaration
nodeclspec-function-declaration
function-definition
template-declaration
deduction-guide
linkage-specification
namespace-definition
empty-declaration
attribute-declaration
module-import-declaration

special-declaration:

- explicit-instantiation*
- explicit-specialization*
- export-declaration*

block-declaration:

- simple-declaration*
- asm-declaration*
- namespace-alias-definition*
- using-declaration*
- using-enum-declaration*
- using-directive*
- static_assert-declaration*
- alias-declaration*
- opaque-enum-declaration*

nodeclspec-function-declaration:

- attribute-specifier-seq_{opt}* *declarator* ;

alias-declaration:

- using* *identifier* *attribute-specifier-seq_{opt}* = *defining-type-id* ;

simple-declaration:

- decl-specifier-seq* *init-declarator-list_{opt}* ;
- attribute-specifier-seq* *decl-specifier-seq* *init-declarator-list* ;
- attribute-specifier-seq_{opt}* *decl-specifier-seq* *ref-qualifier_{opt}* [*identifier-list*] *initializer* ;

static_assert-declaration:

- static_assert* (*constant-expression*) ;
- static_assert* (*constant-expression* , *string-literal*) ;

empty-declaration:

- ;

attribute-declaration:

- attribute-specifier-seq* ;

decl-specifier:

- storage-class-specifier*
- defining-type-specifier*
- function-specifier*
- friend*
- typedef*
- constexpr*
- constexpr*
- constexpr*
- inline*

decl-specifier-seq:

- decl-specifier* *attribute-specifier-seq_{opt}*
- decl-specifier* *decl-specifier-seq*

storage-class-specifier:

- static*
- thread_local*
- extern*
- mutable*

function-specifier:

- virtual*
- explicit-specifier*

explicit-specifier:

- explicit* (*constant-expression*)
- explicit*

typedef-name:

- identifier*
- simple-template-id*

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*

defining-type-specifier:
 type-specifier
 class-specifier
 enum-specifier

defining-type-specifier-seq:
 defining-type-specifier *attribute-specifier-seq*_{opt}
 defining-type-specifier *defining-type-specifier-seq*

simple-type-specifier:
 *nested-name-specifier*_{opt} *type-name*
 nested-name-specifier *template* *simple-template-id*
 decltype-specifier
 placeholder-type-specifier
 *nested-name-specifier*_{opt} *template-name*
 char
 char8_t
 char16_t
 char32_t
 wchar_t
 bool
 short
 int
 long
 signed
 unsigned
 float
 double
 void

type-name:
 class-name
 enum-name
 typedef-name

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

decltype-specifier:
 decltype (*expression*)

placeholder-type-specifier:
 *type-constraint*_{opt} *auto*
 *type-constraint*_{opt} *decltype* (*auto*)

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

init-declarator:
 declarator *initializer*_{opt}
 declarator *requires-clause*

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*) *cv-qualifier-seq_{opt}*
 ref-qualifier_{opt} noexcept-specifier_{opt} attribute-specifier-seq_{opt}

trailing-return-type:
 -> *type-id*

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*

type-id:
 type-specifier-seq abstract-declarator_{opt}

defining-type-id:
 defining-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} *this*_{opt} *decl-specifier-seq* *declarator*
- attribute-specifier-seq*_{opt} *decl-specifier-seq* *declarator* = *initializer-clause*
- attribute-specifier-seq*_{opt} *this*_{opt} *decl-specifier-seq* *abstract-declarator*_{opt}
- attribute-specifier-seq*_{opt} *decl-specifier-seq* *abstract-declarator*_{opt} = *initializer-clause*

initializer:

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

brace-or-equal-initializer:

- = *initializer-clause*
- braced-init-list*

initializer-clause:

- assignment-expression*
- braced-init-list*

braced-init-list:

- { *initializer-list* ,_{opt} }
- { *designated-initializer-list* ,_{opt} }
- { }

initializer-list:

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

designated-initializer-list:

- designated-initializer-clause*
- designated-initializer-list* , *designated-initializer-clause*

designated-initializer-clause:

- designator* *brace-or-equal-initializer*

designator:

- . *identifier*

expr-or-braced-init-list:

- expression*
- braced-init-list*

function-definition:

- attribute-specifier-seq*_{opt} *decl-specifier-seq*_{opt} *declarator* *virt-specifier-seq*_{opt} *function-body*
- attribute-specifier-seq*_{opt} *decl-specifier-seq*_{opt} *declarator* *requires-clause* *function-body*

function-body:

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

enum-name:

- identifier*

enum-specifier:

- enum-head* { *enumerator-list*_{opt} }
- enum-head* { *enumerator-list* , }

enum-head:

- enum-key* *attribute-specifier-seq*_{opt} *enum-head-name*_{opt} *enum-base*_{opt}

enum-head-name:

- nested-name-specifier*_{opt} *identifier*

opaque-enum-declaration:

- enum-key* *attribute-specifier-seq*_{opt} *enum-head-name* *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 attribute-specifier-seqopt

using-enum-declaration:
    using enum using-enum-declarator ;

using-enum-declarator:
    nested-name-specifieropt identifier
    nested-name-specifieropt simple-template-id

namespace-name:
    identifier
    namespace-alias

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

named-namespace-definition:
    inlineopt namespace attribute-specifier-seqopt identifier { namespace-body }

unnamed-namespace-definition:
    inlineopt namespace attribute-specifier-seqopt { namespace-body }

nested-namespace-definition:
    namespace enclosing-namespace-specifier :: inlineopt identifier { namespace-body }

enclosing-namespace-specifier:
    identifier
    enclosing-namespace-specifier :: inlineopt identifier

namespace-body:
    declaration-seqopt

namespace-alias:
    identifier

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

qualified-namespace-specifier:
    nested-name-specifieropt namespace-name

using-directive:
    attribute-specifier-seqopt using namespace nested-name-specifieropt namespace-name ;

using-declaration:
    using using-declarator-list ;

using-declarator-list:
    using-declarator . . . opt
    using-declarator-list , using-declarator . . . opt

using-declarator:
    typenameopt nested-name-specifier unqualified-id

asm-declaration:
    attribute-specifier-seqopt asm ( string-literal ) ;

linkage-specification:
    extern string-literal { declaration-seqopt }
    extern string-literal name-declaration

attribute-specifier-seq:
    attribute-specifier-seqopt attribute-specifier

```


attribute-specifier:
 [[*attribute-using-prefix*_{opt} *attribute-list*]]
alignment-specifier

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

attribute-using-prefix:
using *attribute-namespace* :

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*_{opt})

balanced-token-seq:
balanced-token
balanced-token-seq *balanced-token*

balanced-token:
 (*balanced-token-seq*_{opt})
 [*balanced-token-seq*_{opt}]
 { *balanced-token-seq*_{opt} }
 any *token* other than a parenthesis, a bracket, or a brace

A.8 Modules

[gram.module]

module-declaration:
*export-keyword*_{opt} *module-keyword* *module-name* *module-partition*_{opt} *attribute-specifier-seq*_{opt} ;

module-name:
*module-name-qualifier*_{opt} *identifier*

module-partition:
 : *module-name-qualifier*_{opt} *identifier*

module-name-qualifier:
identifier .
module-name-qualifier *identifier* .

export-declaration:
export *name-declaration*
export { *declaration-seq*_{opt} }
export-keyword *module-import-declaration*

module-import-declaration:
import-keyword *module-name* *attribute-specifier-seq*_{opt} ;
import-keyword *module-partition* *attribute-specifier-seq*_{opt} ;
import-keyword *header-name* *attribute-specifier-seq*_{opt} ;

global-module-fragment:
module-keyword ; *declaration-seq*_{opt}

private-module-fragment:
module-keyword : *private* ; *declaration-seq*_{opt}

A.9 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
 using-declaration
 using-enum-declaration
 static_assert-declaration
 template-declaration
 explicit-specialization
 deduction-guide
 alias-declaration
 opaque-enum-declaration
 empty-declaration

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

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

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

virt-specifier:
 override
 final

pure-specifier:
 = 0

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

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

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

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

base-specifier-list:
base-specifier ... *opt*
base-specifier-list , *base-specifier* ... *opt*

base-specifier:
*attribute-specifier-seq*_{opt} *class-or-decltype*
*attribute-specifier-seq*_{opt} *virtual* *access-specifier*_{opt} *class-or-decltype*
*attribute-specifier-seq*_{opt} *access-specifier* *virtual*_{opt} *class-or-decltype*

class-or-decltype:
*nested-name-specifier*_{opt} *type-name*
nested-name-specifier *template* *simple-template-id*
decltype-specifier

access-specifier:
private
protected
public

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

mem-initializer-list:
mem-initializer ... *opt*
mem-initializer-list , *mem-initializer* ... *opt*

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

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

A.10 Overloading

[gram.over]

operator-function-id:
operator *operator*

operator: one of

<i>new</i>	<i>delete</i>	<i>new</i> []	<i>delete</i> []	<i>co_await</i> ()	[]	->	->*
~	!	+	-	*	/	%	&
	=	+=	--	*=	/=	%=	&=
=	==	!=	<	>	<=	>=	<=>
	<<	>>	<<=	>>=	++	--	,

literal-operator-id:
operator *string-literal* *identifier*
operator *user-defined-string-literal*

A.11 Templates

[gram.temp]

template-declaration:
template-head *declaration*
template-head *concept-definition*

template-head:
template < *template-parameter-list* > *requires-clause*_{opt}

template-parameter-list:
template-parameter
template-parameter-list , *template-parameter*

requires-clause:
requires *constraint-logical-or-expression*

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

constraint-logical-and-expression:
primary-expression
constraint-logical-and-expression && *primary-expression*

template-parameter:
type-parameter
parameter-declaration

type-parameter:
type-parameter-key ... *opt* *identifier*_{opt}
*type-parameter-key identifier*_{opt} = *type-id*
type-constraint ... *opt* *identifier*_{opt}
*type-constraint identifier*_{opt} = *type-id*
template-head type-parameter-key ... *opt* *identifier*_{opt}
*template-head type-parameter-key identifier*_{opt} = *id-expression*

type-parameter-key:
class
typename

type-constraint:
*nested-name-specifier*_{opt} *concept-name*
*nested-name-specifier*_{opt} *concept-name* < *template-argument-list*_{opt} >

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

constraint-expression:
logical-or-expression

deduction-guide:
*explicit-specifier*_{opt} *template-name* (*parameter-declaration-clause*) -> *simple-template-id* ;

concept-definition:
concept *concept-name* *attribute-specifier-seq*_{opt} = *constraint-expression* ;

concept-name:
identifier

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.12 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}
 ...
noexcept-specifier:
noexcept (*constant-expression*)
noexcept

A.13 Preprocessing directives

[gram.cpp]

preprocessing-file:
*group*_{opt}
module-file

module-file:
*pp-global-module-fragment*_{opt} *pp-module* *group*_{opt} *pp-private-module-fragment*_{opt}

pp-global-module-fragment:
module ; *new-line* *group*_{opt}

pp-private-module-fragment:
module : *private* ; *new-line* *group*_{opt}

group:
group-part
group *group-part*

group-part:
control-line
if-section
text-line
 # *conditionally-supported-directive*

control-line:
 # *include* *pp-tokens* *new-line*
pp-import
 # *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*
 # *warning* *pp-tokens*_{opt} *new-line*
 # *pragma* *pp-tokens*_{opt} *new-line*
 # *new-line*

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}
 # *elifdef* *identifier* *new-line* *group*_{opt}
 # *elifndef* *identifier* *new-line* *group*_{opt}

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

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

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

conditionally-supported-directive:
 pp-tokens *new-line*

lparen:
 a (character not immediately preceded by whitespace

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

defined-macro-expression:
 defined identifier
 defined (*identifier*)

h-preprocessing-token:
 any *preprocessing-token* other than >

h-pp-tokens:
 h-preprocessing-token
 h-pp-tokens *h-preprocessing-token*

header-name-tokens:
 string-literal
 < *h-pp-tokens* >

has-include-expression:
 __has_include (*header-name*)
 __has_include (*header-name-tokens*)

has-attribute-expression:
 __has_cpp_attribute (*pp-tokens*)

pp-module:
 export_{opt} module *pp-tokens*_{opt} ; *new-line*

pp-import:
 export_{opt} import *header-name* *pp-tokens*_{opt} ; *new-line*
 export_{opt} import *header-name-tokens* *pp-tokens*_{opt} ; *new-line*
 export_{opt} import *pp-tokens* ; *new-line*

va-opt-replacement:
 __VA_OPT__ (*pp-tokens*_{opt})