<programa> -> <librerías> <espacio_nombres> <declaración_compuesta>
<declaracion_template> <función_principal> <funciones>

Arriba del main----------

<librerías> -> #include <nombre_librería> <librerías> | <espacio_nombres> | ε

<nombre_librería> -> <<identifier>.h>
| "<identifier>.h"

<espacio_nombres> -> using namespace std;

<declaración_compuesta> -> struct <identifier> <herencia> { <miembros_union> };
| class <identifier> <herencia> { <miembros_class> };
| union <identifier> { <miembros_union> };

<herencia> -> : <visibilidad> <identifier>
| ε

<visibilidad> -> public: | private: | protected:

<miembros_class> -> <miembro_class> <miembros_class>
| <visibilidad> <miembro_class> <miembros_class>
| ε

<miembro_class> -> <declaración_class>
| <funciones_class>

<miembros_union> -> <declaración> <miembros_union>
| ε

<declaracion_template> -> typename < template <identifier>> > <identifier>
<identifier>(<lista_parametros_templat>) { <enunciados> }

<lista_parametros_template> -> <parametro_template>
| <parametro_template> , <lista_parametros_template>

<parametro_template> -> <identifier> <identifier>

<función_principal> -> int main ( ) { <enunciados> return 0;}

Dentro del main----------

<declaración> -> <modificador> <tipo> <identifier> ;
| <modificador> <tipo> <identifier> = <identifier> ;
| <modificador> <tipo> <identifier> = <identifier>.<identifier>(<argumentos>);
| enum <identifier> { <lista_valores> } ;
| <declaración_tipo>

<declaración_tipo> -> <modificador> int <identifier> = <expresion_entero> ;
| <modificador> float <identifier> = <expresion_decimal> ;
| <modificador> double <identifier> = <expresion_decimal> ;
| <modificador> char <identifier> = <expresion_caracter> ;
| <modificador> bool <identifier> = <expresion_booleano> ;
| <modificador> string <identifier> = <cadena> ;
| <modificador> short <identifier> = <expresion_entero> ;
| <modificador> long <identifier> = <expresion_entero> ;
| <modificador> signed <identifier> = <expresion_entero> ;

<modificador> -> <meta_programación>
| <modificador_variable>
| ε

<expresion_entero> -> <entero>
| <entero> <operador_aritmetico> <entero>
| (<entero> <operador_aritmetico> <entero>)

<expresion_decimal> -> <decimal>
| <decimal> <operador_aritmetico> <decimal>
| (<decimal> <operador_aritmetico> <decimal>)

<expresion_caracter> -> <caracter>

<expresion_booleana> -> true
| false
| <expresion_comparacion_booleana>

<expresion_comparacion_booleana> -> <entero> <operador_relacional> <entero>
| <decimal> <operador_relacional> <decimal>

<lista_valores> -> <valor_enum> , <lista_valores>

| <valor_enum>

<valor_enum> -> <identifier>
| <identifier> = <expresión>

<enunciados> -> <enunciado> <enunciados>
| ε

<enunciado> -> <asignación> ;
| <entrada> ;
| <salida> ;
| <selección>
| <iteración>
| <llamada_función> ;
| <switch>
| <corrutina> ;
| goto <identifier> ;
| <identifier> : <enunciado>
| throw <expresión> ;
| try { <enunciados> } catch (<tipo> <identifier>) { <enunciados> }
| <declaración>
| throw <expresión> ;
| <bloque_try>
| asm("string") ;
| delete <identifier>
| sizeof(<identifier>)

<bloque_try> -> try { <enunciados> } <bloques_catch>

<bloques_catch> -> catch (<tipo> <identifier>) { <enunciados> } <bloques_catch>
| ε

<asignación> -> <asignación-int>
| <asignación-other>

<asignación-other> -> <identifier> = <expresiones>;

<asignación-int> -> <identifier>++;
| <identifier>--;
| ++<identifier> ;
| --<identifier> ;

<selección> -> if (<condiciones>) { <enunciados> } <bloques_else>

<bloques_else> -> else if (<condiciones>) { <enunciados> } <bloques_else>
| else { <enunciados> }
| ε

<iteración> -> while (<condiciones>) { <enunciados> }
| for (<declaración> ; <condiciones> ; <asignación-int>) { <enunciados> }
| do { <enunciados> } while (<condiciones>) ;

<condiciones> -> <condición> <operador_relacional> <condición>
| <condición> <operador_relacional> <condición> <operador_lógico>

<condición> -> <identifier>
| <entero>
| <decimal>
| <identifier>.<identifier>()
| <condición> <operador_relacional> <condición>
| <condición> <operador_relacional> <condición> <operador_lógico>

<expresiones> -> <expresión> <operador_aritmético> <expresión>
| (<expresión> <operador_aritmético> <expresión>)
| <conversión> <tipo> (<expresión>)
| <expresion_ternaria>
| <expresión>

<expresión> -> <identifier>
| <entero>
| <decimal>
| nullptr

<expresion_ternaria> -> <expresión> ? <expresión> : <expresión>

<switch> -> switch (<identifier>) { <casos> }

<casos> -> case <tipo_casos> : <enunciados_switch> <casos>
| default : <enunciados_switch>
| ε

<tipo_casos> -> <cadena>
| <entero>
| <decimal>

<enunciados_switch> -> <enunciado_switch> <enunciados_switch>
| ε

<enunciado_switch> -> <asignación> ;
| <entrada> ;
| <salida> ;
| <selección>
| <switch>
| <iteración>
| <llamada_función> ;
| break ;
| continue ;

LLamar a función----------

<llamada_función> -> <identifier> (<argumentos>)
| <identifier> :: <identifier> (<argumentos>)

<argumentos> -> <expresión> , <argumentos>
| <expresión>
| ε

Terminales----------

<corrutina> -> co_await <expresión>
| co_yield <expresión>
| co_return <expresión>

<conversión> -> const_cast
| dynamic_cast
| static_cast
| reinterpret_cast

<meta_programación> -> decltype
| constexpr
| consteval
| constinit

<declarador> -> extern
| export
| explicit
| default
| enum
| concept
| compl

<tipo> -> int
| float
| double
| char
| string
| bool
| wchar_t
| char8_t
| char16_t
| char32_t
| auto
| decltype (<expresión>)
| short
| short int
| long
| long int
| signed
| signed int
| unsigned int

<modificador_variable> -> inline
| mutable
| register
| alignas
| thread_local
| volatile
| unsigned
| const
| typedef

<modificador_función> -> noexcept
| requires
| template
| static_assert
| virtual

<identifier> -> identificador

<entero> -> 0...9

<decimal> -> 0..9 . 0...9

<cadena> -> " cualquier texto "

<booleano> = true
| false

<operador_relacional> -> ==
| !=
| <
| >
| <=
| >=
| not_eq

<operador_aritmético> -> +
| -
| *
| /
| %

<operador_aritmético_compuesto> -> +=
| -=
| *=
| /=
| %=

<operador_binario> -> &
| bitor
| xor
| ^
| bitand

<operador_binario_unario> -> ~
| compl

<operador_binario_compuesto> -> &=
| |=
| ^=
| <<=
| >>=

<not> -> not
| !

<operador_lógico> -> &&
| and
| ||
| or
| or_eq
| =`

<operador_logico_compuesto> -> and_eq
| or_eq
| xor_eq

Entrada----------

<entrada> -> std :: cin >> <identifier> ;
| cin >> <identifier> ;

Imprimir en pantalla----------

<salida> -> std :: cout << <elementos_salida> ;
| cout << <elementos_salida> ;

<elementos_salida> -> <elemento_salida> << <elementos_salida>
| <elemento_salida>

<elemento_salida> -> <cadena>
| <identifier>
| <identifier>.<identifier>()
| endl
| std :: endl
| <llamada_función>

<cadena_o_variable> -> <cadena>
| <identifier>
| <cadena_o_variable> << <cadena_o_variable>

Despues del
main----------

<funciones> -> <función> <funciones>
| ε

<función> -> <tipo_función> <<identifier>> (<parámetros>) { <enunciados> <retorno> }
| <modificador_función> <tipo_función> <<identifier>> (<parámetros>) {
<enunciados> <retorno> }
| <declarador> <tipo_función> <<identifier>> (<parámetros>) { <enunciados>
<retorno> }

<parámetros> -> <tipo_parametro> <identifier> , <parámetros>
| <tipo_parametro> <identifier>
| ε

<tipo_parametro> -> <modificador_parametro> <tipo> <RefoPuntero>
| <tipo> & // referencia
| <tipo> * // puntero
| <identifier> // nombre de clase
| <identifier> &
| <identifier> *

<modificador_parámetro> -> const
| constexpr

<tipo_funcion> -> void
| int
| float
| double
| char
| string
| bool

<retorno> -> return <expresión> ;
| ε

Dentro de clases y
struct----------

<funciones_class> -> <función_class> <funciones_class>
| ε

<función_class> -> <tipo_función> <nombre_class_función> (<parámetros>) {
<enunciados_class> <retorno> }
| <modificador_función> <tipo_función> <nombre_class_función> (<parámetros>) {
<enunciados_class> <retorno> }
| <declarador> <tipo_función> <nombre_class_función> (<parámetros>) {
<enunciados_class> <retorno> }

| <identificador> () { enunciados_class}
| <identifier>();

<nombre_class_función> -> <identifier>
| operator <operator_valido>

<operador_valido> -> +
| -
| *
| /
| ==
| !=
| []
| ()
| <
| >

<declaración_class> -> <modificador> <tipo> <identifier> ;
| <modificador> <tipo> <identifier> = <opciones_class>;
| <modificador> <tipo> <identifier> = <identifier>.<identifier>(<argumentos>);
| enum <identifier> { <lista_valores> } ;
| <constructor> -> <identifier> (<parámetros>) { <enunciados_class> }
| <destructor> -> ~ <identifier> ( ) { <enunciados_class> }

<opciones_class> -> this
| * this
| this -> <identifier>
| (* this) . <identifier>
| <identifier>

<lista_valores_class> -> <valor_enum_class> , <lista_valores_class>
| <valor_enum_class>

<valor_enum_class> -> <identifier>
| <identifier> = <expresión_class>

<enunciados_class> -> <enunciado_class> <enunciados_class>
| ε

<enunciado_class> -> <asignación_class> ;

| <entrada> ;
| <salida> ;
| <selección>
| <iteración>
| <llamada_función> ;
| <switch_class>
| <corrutina> ;
| goto <identifier> ;
| <identifier> : <enunciado_class
| throw <expresión_class> ;
| try { <enunciados_class> } catch (<tipo> <identifier>) { <enunciados_class> }
| <declaración_class>
| <declaracion_tipo>
| throw <expresión_class> ;
| <bloque_try_class>

<bloque_try_class> -> try { <enunciados_class> } <bloques_catch>

<bloques_catch> -> catch (<tipo> <identifier>) { <enunciados_class> } <bloques_catch>
| ε

<asignación_class> -> <asignación-int>
<asignación-other_class>

<asignación-other_class> -> <identifier> = <expresiones_class>;

<retornoclass> -> return <expresión_class> ;
| ε

<expresiones_class> -> <expresión_class> <operador_aritmético> <expresión_class>
| (<expresión_class> <operador_aritmético> <expresión_class>)

<expresión_class> -> <identifier>
| <entero>
| <decimal>
| nullptr
| <conversión> <tipo> (<expresión_class>)
| <conversión> <tipo> (<expresión_class>)
| <expresión_class> <operador_aritmético> <expresión_class>
| (<expresión_class> <operador_aritmético> <expresión_class>)
| this
| * this
| this -> <identifier>
| (* this) . <identifier>

<switch_class> -> switch (<identifier>) { <casos_classn> }

<casos_class> -> case <tipo_casos> : <enunciados_switch_class> <casos>
| default : <enunciados_switch_class>
| ε

<enunciados_switch_class> -> <enunciado_switch_class> <enunciados_switch_class>
| ε

<enunciado_switch_class> -> <asignación_class ;
| <entrada> ;
| <salida> ;
| <selección>
| <switch_class>
| <iteración>
| <llamada_funcion> ;
| break ;
| continue ;

<modificador_función_class> -> noexcept
| requires
| template
| static_assert
| virtual
| friend