**Sintaxis de C#**

**1.1. Gramática Léxica**

**1.1.1. Elementos Léxicos**

<token> ->

<palabra reservada> |

<identificador> |

<literales> |

<operadores> |

<separadores> |

<comentarios> |

<directivas del preprocesador> |

<espacios en blanco>

**1.1.2. Palabras Reservadas**

<palabra reservada> ->

abstract as base bool break

byte case catch char checked

class const continue decimal default

delegate do double else enum

event explicit extern false finally

fixed float for foreach goto

if implicit in int interface

internal is lock long namespace

new null object operator out

override params private protected public

readonly ref return sbyte sealed

short sizeof stackalloc static string

struct switch this throw true

try typeof uint ulong unchecked

unsafe ushort using virtual void

volatile while

**1.1.3. Identificadores**

<identificador> -> <letra> <letra o dígito>\*

<letra> -> uno de a-z A-Z \_

<letra o dígito> -> <letra> | <dígito>

<dígito> -> uno de 0-9

**1.1.4. Constantes**

<constante> -> <constante numérica> | <constante enumeración>

<constante numérica> -> "const" <tipo numérico> <identificador> "=" <valor numérico> ";"

<tipo numérico> -> int | double | float | decimal | long | short | byte | sbyte | uint | ulong | ushort

<valor numérico> -> <entero> | <decimal> | <hexadecimal>

<entero> -> <digito> { <digito> }

<decimal> -> <entero> . <entero>

<hexadecimal> -> 0x <dígito hexadecimal> { <dígito hexadecimal> }

<dígito> -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<digito hexadecimal> -> <digito> | A | B | C | D | E | F | a | b | c | d | e | f

<constante enumeración> -> enum <identificador> { { <miembro enumeración> } }

<miembro enumeración> -> <identificador> [ = <valor entero> ] [ , ]

<valor entero> -> <entero> | <entero literal>

<entero literal> -> 0x <dígito hexadecimal> { <dígito hexadecimal> }

| 0b <dígito binario> { <dígito binario> }

<dígito binario> -> 0 | 1

**1.1.5. Constantes Cadena**

<cadena literal> -> '"' { <carácter cadena> } '"'

<carácter cadena> -> <caracter> | "\" <caracter escape>

<caracter escape> -> "\\" | "\"" | "\'" | "\n" | "\t" | "\r" | "\0"

| "\x" con 2 <dígito hexadecimal>

| "\u" con 2 o 4 <dígito hexadecimal>

**1.1.6. Punctuators – Caracteres de Puntuación**

<punctuator> -> "(" | ")" | "[" | "]" | "{" | "}" | "." | "," | ":" | ";" | "+" | "-" | "\*" | "/" | "%" | "++" | "--" | "=>" | "?" | "!" | "&" | "|" | "&&" | "||" | "^" | "~" | "==" | "!=" | "<" | ">" | "<=" | ">=" | "=" | "+=" | "-=" | "\*=" | "/=" | "%=" | "&=" | "|=" | "^=" | "<<=" | ">>=" | "??"

**1.1.7. Nombre de Encabezados**

<atributo> -> "[" <nombre atributo> [ <argumentos atributo> ] "]"

<nombre atributo> -> <identificador>

<argumentos atributo> -> "(" [ <argumento atributo> { "," <argumento atributo> } ] ")"

<argumento atributo> -> <nombre atributo> "=" <valor atributo>

<valor atributo> -> <cadena literal> | <valor numérico> | <nombre atributo>

**1.1.8. Números de Preprocesador**

C# no tiene 'Números de Preprocesador' como en C. En lugar de ello, C# utiliza un enfoque más limitado y sofisticado de preprocesamiento. A través de directivas específicas como #if, #else, #elif, #endif, #warning, y #error, se puede controlar la inclusión de código condicional, generar advertencias o errores, y definir símbolos condicionales para mejorar la legibilidad, depuración y control de compilación del código.

**1.2. Gramática de Estructura de Frases**

**1.2.1. Expresiones**

<expresión> -> <expresión binaria>

| <expresión unaria>

| <expresión primaria>

<expresión binaria> -> <expresión> <operador binario> <expresión>

<operador binario> -> "+" | "-" | "\*" | "/" | "%" | "==" | "!=" | "<" | ">" | "<=" | ">=" | "&&" | "||" | "&" | "|" | "^" | "<<" | ">>"

<expresión unaria> -> <operador unario> <expresión>

| <expresión primaria>

<operador unario> -> "+" | "-" | "!" | "~"

<expresión primaria> -> <identificador>

| <literal>

| <expresión paréntesis>

| <llamada método>

| <acceso miembro>

| <expresión arreglo>

<expresión paréntesis> -> "(" <expresión> ")"

<llamada método> -> <identificador> "(" [ <argumentos> ] ")"

<acceso miembro> -> <identificador> "." <identificador>

<expresión arreglo> -> <identificador> "[" <expresión> "]"

<argumentos> -> <expresion> { "," <expresion> }

<literal> -> <entero literal> | <decimal literal> | <cadena literal> | <carácter literal>

<entero literal> -> <entero decimal> | <entero hexadecimal> | <entero binario>

<entero decimal> -> <digito> { <digito> }

<entero hexadecimal> -> "0x" <dígito hexadecimal> { <dígito hexadecimal> }

<entero binario> -> "0b" <dígito binario> { <dígito binario> }

<decimal literal> -> <entero decimal> "." <entero decimal>

<cadena literal> -> '"' { <carácter cadena> } '"'

<carácter literal> -> "'" <carácter> "'"

**1.2.2. Declaraciones**

<declaracion> -> <declaracion variable> ";"

| <declaración constante> ";"

| <declaración metodo>

| <declaracion clase>

<declaración variable> -> <tipo> <lista variables>

<tipo> -> <tipo primitivo>

| <tipo clase>

| <tipo arreglo>

<tipo primitivo> -> "int"

| "double"

| "float"

| "decimal"

| "long"

| "short"

| "byte"

| "bool"

| "char"

| "string"

| "void"

| <tipo numerico>

<tipo numerico> -> "uint"

| "ulong"

| "ushort"

<tipo clase> -> <identificador>

<tipo arreglo> -> <tipo> "[" "]"

<lista variables> -> <variable> { "," <variable> }

<variable> -> <identificador>

| <identificador> "=" <expresión>

<declaración constante> -> "const" <tipo> <identificador> "=" <expresión>

<declaración metodo> -> <tipo> <identificador> "(" [ <parámetros> ] ")"

[ <modificadores método> ]

<bloque código método>

<parametros> -> <parametro> { "," <parametro> }

<parámetro> -> <tipo> <identificador>

<modificadores método> -> "static" | "public" | "private" | "protected"

<bloque código método> -> "{" { <sentencia> } "}"

<declaracion clase> -> "class" <identificador> [ ":" <identificador> ]

"{" { <declaración> } "}"