



Universidad Autónoma del Estado de México

Unidad académica profesional de Tianguistengo

Unidad de Aprendizaje:
Diseño de Compiladores e
Interpretes

Documentación:
Proyecto Ordinario: Compilador
para lenguaje C++

Alumno:

Jorge Darien Cabrera Bastidas

Equipo C++

Grupo: SG

Índice

Introducción ...	3
Lenguaje A Analizar ...	5
Análisis Léxico ...	8
Análisis Sintáctico ...	12
Diagrama Sintáctico ...	15
Manejo de Errores ...	17
Documentador ...	18
Conclusiones ...	20
Bibliografía ...	21

Índice imágenes

Imagen 1.1 Instalación

Imagen 1.2 Instalacion_2

Imagen 1.3 Instalacion_3

Imagen 1.4 Instalacion_4

Imagen 1.5 Instalacion_5

Imagen 1.6 Instalacion_6

Imagen 2.1 Captura_L

Imagen 3.1 Diagrama

Imagen 4.1 Captura_S

Imagen 4.2 Captura_S2

Imagen 5.1 Documentador

Imagen 5.2 Documentador_2


Imagen 5.3 Documentador_3



Introducción

Este documento presenta la documentación del proyecto para la materia de compiladores e interpretes en cual se abordó la creación de un compilador para un lenguaje en específico en cual fue dado por la profesora en el caso específico de este proyecto se llevó a cabo sobre el lenguaje de c++ , en el desarrollo de este se especificaron las partes que lo conformaría estas son: análisis léxico, análisis sintáctico, documentador así como también la interfaz donde se ejecutaría y mostraría el proyecto.

En el desarrollo de este proyecto se resalto la importancia del conocimiento básico de los conceptos desarrollados a lo largo de la duración del curso entre ellos se resalta la utilización de: Entendimiento de gramáticas, identificación de Sintaxis entre otros temas abordados en clase.



Lenguaje a Analizar

Lenguaje analizado: C++

Historia:

El lenguaje de programación C++ fue creado por Bjarne Stroustrup en 1983 como una extensión del lenguaje C. Su objetivo principal era proporcionar características adicionales de programación orientada a objetos, como clases y herencia, mientras mantenía la eficiencia y el rendimiento del lenguaje C. A lo largo de los años, C++ ha evolucionado para incluir funcionalidades como plantillas, manejo de excepciones y polimorfismo. Es ampliamente utilizado en el desarrollo de sistemas, aplicaciones de alto rendimiento, videojuegos y software de sistemas embebidos debido a su combinación de poder y control cercano al hardware.

Compilado:

C++ es un lenguaje compilado porque su diseño busca maximizar la eficiencia y el rendimiento. Al ser compilado, el código fuente de C++ se traduce directamente a código máquina específico para una arquitectura dada antes de su ejecución. Esto permite optimizar el uso de recursos del sistema y garantiza una ejecución más rápida en

comparación con los lenguajes interpretados. Además, la compilación ayuda a detectar errores en el código antes de la ejecución y mejora la seguridad del software al permitir optimizaciones y verificaciones estáticas avanzadas.

Donde Conseguirlo

Puedes obtener C++ instalando un compilador y un entorno de desarrollo integrado (IDE). Entre los compiladores más populares están GCC (GNU Compiler Collection), disponible en sistemas Unix/Linux y también en Windows a través de MinGW o Cygwin, y Microsoft Visual C++, parte de Visual Studio, disponible para Windows. IDEs como Visual Studio, Code::Blocks y CLion ofrecen un entorno completo para desarrollar en C++. Además, puedes usar plataformas en línea como repl.it y Compiler Explorer para escribir y compilar código C++ sin necesidad de instalaciones locales.

Instalación Paso a Paso

Para este punto tomaremos como punto de partida lo aprendido en la investigación previa (es específico en el apartado de “Donde Obtenerlo”) 1.-Abriremos la liga de donde obtenemos



2.-Una vez en la pág. Aremos clic en el apartado Descargar



3.-Esperar a que se Descargue

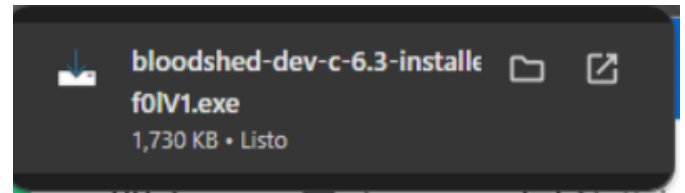


Imagen1.3

Imagen1.2

4.-Una vez descargado ejecutamos su instalador

5.-Elegimos idioma de nuestra elección y aceptamos

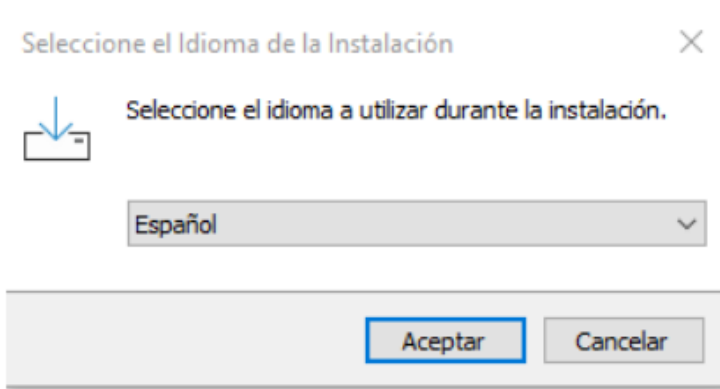


Imagen1.4



Imagen1.5

6.Listo Tenemos C++ instalado

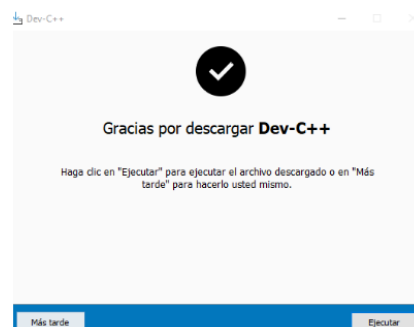


Imagen1.6



Análisis Léxico

Los componentes léxicos de C++ son las unidades básicas que conforman el código fuente. Estos incluyen palabras clave (como `int`, `class`), identificadores (nombres de variables, funciones), literales (constantes numéricas, de caracteres y cadenas), operadores (como `+`, `-`, `*`, `/`), y delimitadores (como `;`, `,`, `()`), que estructuran y dan significado al código. Estos elementos forman la base del análisis léxico en la construcción de un compilador.

Tabla de Palabras Reservadas

TOKEN	DESCRIPCIÓN	CONST	Constante (const)
INT	Entero (int)	MUTABLE	Mutable (mutable)
FLOAT	Flotante (float)	VOLATILE	Volátil (volatile)
DOUBLE	Doble precisión (double)	INLINE	En línea (inline)
CHAR	Carácter (char)	VIRTUAL	Virtual (virtual)
BOOL	Booleano (bool)	OVERRIDE	Sobrescribir (override)
VOID	Vacío (void)	FINAL	Final (final)
IF	Si (if)	CONSTEXPR	Expresión constante (constexpr)
ELSE	Sino (else)	DECLTYPE	Tipo declarado (decltype)
WHILE	Mientras (while)	EXPLICIT	Explicito (explicit)
FOR	Para (for)	EXPORT	Exportar (export)
DO	Hacer (do)	FRIEND	Amigo (friend)
SWITCH	Cambio (switch)	REGISTER	Registrar (register)
CASE	Caso (case)	TYPEDEF	Definición de tipo (typedef)
DEFAULT	Por defecto (default)	UNION	Unión (union)
RETURN	Retorno (return)	ENUM	Enumeración (enum)
BREAK	Romper (break)	EXTERN	Externo (extern)
CONTINUE	Continuar (continue)	STATIC_ASSERT	Aserción estática (static_assert)
GOTO	Ir a (goto)	REINTERPRET_CAST	Reinterpretar (reinterpret_cast)
CLASS	Clase (class)	DYNAMIC_CAST	Dinámico (dynamic_cast)
STRUCT	Estructura (struct)	CONST_CAST	Constante (const_cast)
NAMESPACE	Espacio de nombres (namespace)	STATIC_CAST	Estático (static_cast)
USING	Usar (using)		
TRY	Intentar (try)		
CATCH	Capturar (catch)		
THROW	Lanzar (throw)		
PUBLIC	Público (public)		
PRIVATE	Privado (private)		
PROTECTED	Protegido (protected)		
TEMPLATE	Plantilla (template)		
TYPENAME	Nombre de tipo (typename)		
NEW	Nuevo (new)		
DELETE	Borrar (delete)		
THIS	Este (this)		
sizeof	Tamaño de (sizeof)		
static	Estático (static)		

Tabla de Operadores Lógicos y Aritméticos

TOKEN	DESCRIPCIÓN
SUMA	Suma (+)
RESTA	Resta (-)
MULTIPLICACION	Multiplicación (*)
DIVISION	División (/)
MODULO	Módulo (%)
INCREMENTO	Incremento (++)
DECREMENTO	Decremento (--)
IGUAL	Igualdad (==)
DIFERENTE	Desigualdad (!=)
MAYORQUE	Mayor que (>)
MENORQUE	Menor que (<)
MAYORIGUAL	Mayor o igual que (>=)
MENORIGUAL	Menor o igual que (<=)
AND	Y lógico (&&)
OR	O lógico ()
NOT	Negación lógica (!)
ASIGNACION	Asignación (=)
ASIGNACION_SUMA	Asignación suma (+=)
ASIGNACION_RESTA	Asignación resta (-=)
ASIGNACION_MULT	Asignación multiplicación (*=)
ASIGNACION_DIV	Asignación división (/=)
ASIGNACION_MOD	Asignación módulo (%=)
AND_BIT	AND bit a bit (&)
OR_BIT	OR bit a bit ()
XOR_BIT	XOR bit a bit (^)
NOT_BIT	NOT bit a bit (~)
DESPLAZA_IZQ	Desplazamiento a la izquierda (<<)
DESPLAZA_DER	Desplazamiento a la derecha (>>)

Capturas de pantalla

NO. LINEA	SIMBOLO
LINEA 1	
<Reservado numeral>	#
<Reservado include>	include
<Simbolo Menor que>	<
<Reservado iostream>	iostream
<Simbolo Mayor que>	>
LINEA 2	
LINEA 3	
<Reservada switch>	void
<Identificador>	saludar
<Dos puntos>	:
<Dos puntos>	:
<Salida por consola>	cout
<Doble menor>	<<
<Cadena>	"Hola, mundo!"
<Doble menor>	<<
<Reservado std>	std
<Dos puntos>	:
<Dos puntos>	:
<Salto de linea>	endl
<Punto y coma>	;
<Identificador>	a
<Coma>	,
<Tipo de dato int>	int
<Identificador>	b
<Parentesis de cierre>)
<Llave de apertura>	{
LINEA 8	
<Reservada return>	return
<Identificador>	a
<Operador suma>	+
<Identificador>	b

Imagen2.1



Análisis Sintáctico

El análisis sintáctico es la segunda fase de un compilador. Toma los tokens generados por el analizador léxico y verifica si forman una secuencia válida según la gramática del lenguaje. En este caso, el analizador sintáctico, utiliza reglas de producción para construir un árbol sintáctico que representa la estructura del programa.

Sintaxis del lenguaje

Declaración de Variables

En C++, las variables deben ser declaradas antes de ser usadas. La declaración incluye el tipo de la variable seguido por su nombre:

```
int x; float y; double z; char a; bool flag;
```

Asignación de Valores

Los valores se asignan a las variables usando el operador de asignación (=):

```
x = 10; y = 3.14; z = 2.71828; a = 'A'; flag = true;
```

Operaciones Aritméticas

C++ soporta las operaciones aritméticas básicas: suma (+), resta (-), multiplicación (*), división (/) y módulo (%):

```
int sum = x + 5; int diff = x - 2; int product = x * 3; int quotient = x / 2;  
int remainder = x % 2;
```

Estructuras Condicionales

Las estructuras condicionales se utilizan para ejecutar diferentes bloques de código según ciertas condiciones.

If: Ejecuta un bloque de código si una condición es verdadera.

```
if (x > 0) {  
  
}
```

If-Else: Ejecuta un bloque de código si una condición es verdadera y otro bloque si es falsa.

```
if (x > 0) {  
  
} else {  
  
}
```

Do-While: Similar a while, pero garantiza que el bloque de código se ejecute al menos una vez.

```
do {  
  
} while (x > 0);
```

While: Ejecuta un bloque de código mientras una condición sea verdadera.

```
while (x > 0) {  
  
}
```

Funciones

Las funciones permiten agrupar código reutilizable en bloques con nombre.

```
int suma(int a, int b) { return a + b; } void imprimeHola() {  
std::cout << "Hola, mundo!"; }
```

Estructuras y Clases

C++ soporta la programación orientada a objetos a través de clases y estructuras.

Estructura: Agrupa variables bajo un solo nombre.

```
struct Punto { int x; int y; };
```

Clase: Similar a las estructuras, pero con control de acceso y funciones miembro.

```
class Persona { public: std::string nombre; int edad; void saludar() {  
std::cout << "Hola, soy " << nombre; } };
```

Diagrama Sintáctico

El equipo de Tongue Twist ha elegido un tipo de GUI (Interfaz Gráfica de Usuario) para una interfaz vertical en dispositivos móviles, centrada en el formato de Preguntas y Respuestas. Esta elección se basa en varias razones clave que la hacen ideal para un software de aprendizaje y enseñanza de idiomas. La orientación vertical de la interfaz está diseñada específicamente para dispositivos móviles, aprovechando al máximo el espacio disponible en las pantallas de los teléfonos y permitiendo una interacción cómoda con una sola mano, lo que es ideal para usuarios en movimiento.

La interacción natural que ofrece el formato de preguntas y respuestas mantiene a los usuarios activos y comprometidos, ya que se requiere una respuesta constante y dinámica, reforzando así el aprendizaje continuo. Además, el sistema proporciona retroalimentación inmediata y personalizada sobre las respuestas del usuario, permitiendo correcciones instantáneas y adaptándose según el nivel de habilidad del usuario. Esto garantiza un desafío adecuado y personalizado que se ajusta al ritmo de aprendizaje de cada individuo.

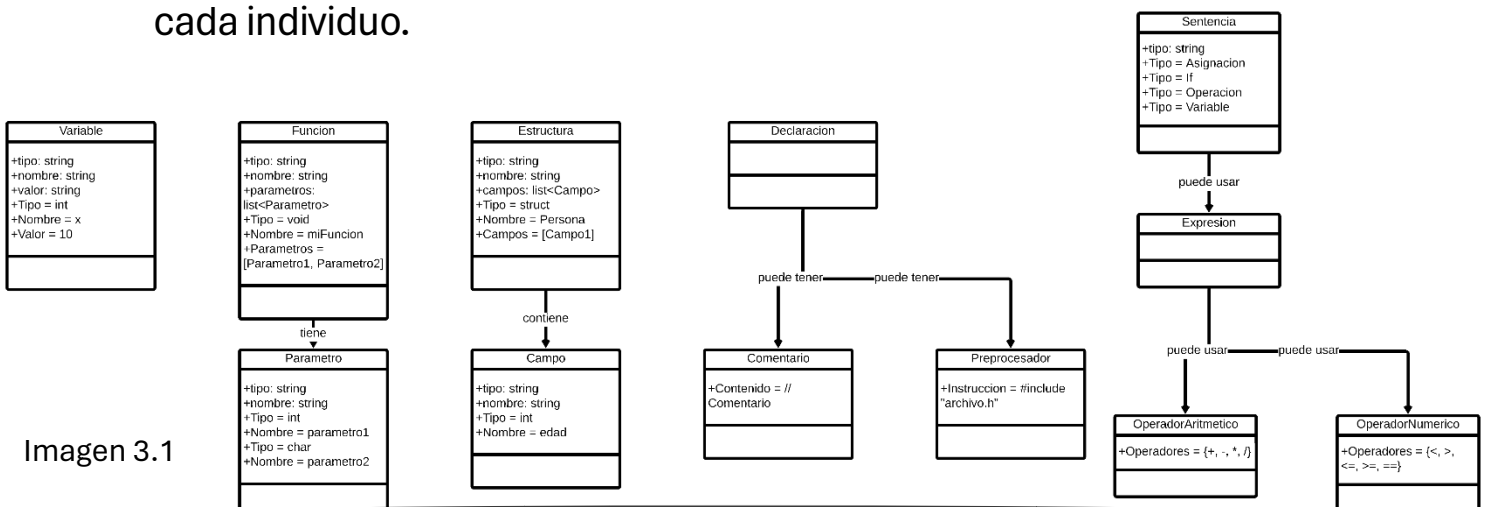


Imagen 3.1



Gramáticas

SENTENCIA ::=

SENTENCIA DECLARACION |

DECLARACION |

SENTENCIA IF |

IF |

SENTENCIA FOR|

FOR |

SENTENCIA WHILE |

WHILE |

SENTENCIA DO_WHILE |

DO_WHILE |

SENTENCIA

OPERACIONES_COMPLEMENTARIAS P_coma |

OPERACIONES_COMPLEMENTARIAS P_coma |

SENTENCIA COUT |

COUT |

SENTENCIA CIN |

CIN |

SENTENCIA SWITCH|

SWITCH |

SENTENCIA PRINTF |

PRINTF |

SENTENCIA SCANF |

SCANF

;

Explicación:

SENTENCIA::=: Esto indica que estamos definiendo una regla de producción llamada SENTENCIA.

SENTENCIA DECLARACION | DECLARACION | ... |

SCANF: Esta es la definición de la regla de producción SENTENCIA. Aquí hay varias partes que se pueden entender así:

Cada línea separada por "|" representa una opción o alternativa para la sentencia. Por ejemplo, **"SENTENCIA DECLARACION"** significa que una sentencia puede ser seguida por una declaración.

"DECLARACION" representa una declaración de variables o funciones.

"IF" representa una sentencia condicional "if".

"FOR" representa un bucle "for".

"WHILE" representa un bucle "while".

"DO_WHILE" representa un bucle "do-while".

"OPERACIONES_COMPLEMENTARIAS P_coma" representa otras operaciones o sentencias que pueden seguirse de un punto y coma.

"COUT" representa una sentencia para mostrar salida en la consola.

"CIN" representa una sentencia para tomar entrada desde la consola.

"SWITCH" representa una sentencia de selección múltiple "switch".

"PRINTF" representa una sentencia para imprimir en pantalla con formato (similar a printf en C).

"SCANF" representa una sentencia para escanear o leer desde la entrada estándar con formato (similar a scanf en C).

Manejo de los Errores

Este código maneja los errores de sintaxis al analizar una entrada de texto utilizando un analizador sintáctico generado por JavaCup. Primero, se obtiene el texto de un componente de interfaz de usuario y se crea una instancia del analizador sintáctico. Luego, se intenta realizar el análisis sintáctico utilizando el método `parse()` de la clase `Sintaxis`. Si el análisis se completa sin errores, se muestra un mensaje indicando que el análisis se realizó correctamente. Sin embargo, si se produce un error durante el análisis, se captura la excepción y se obtiene información sobre el error (línea, columna y texto) del símbolo de error (`Symbol sym`) para mostrar un mensaje de error específico en la interfaz de usuario.

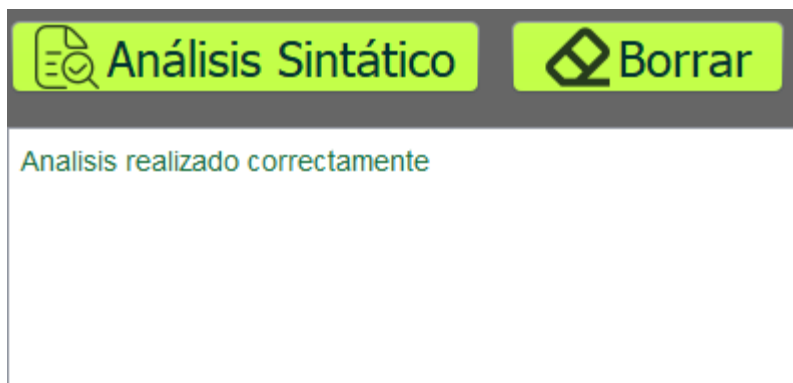


Imagen 4.1

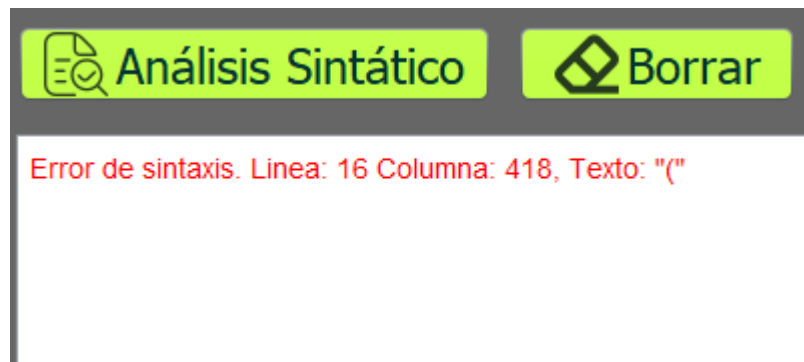


Imagen 4.2



Documentador

Tener una explicación del código en lenguaje natural junto con el código propuesto tiene varias ventajas significativas:

Facilita la comprensión: Muchas veces, el código por sí solo puede ser difícil de entender, especialmente para personas que no están familiarizadas con el lenguaje de programación o el contexto específico del código. La explicación en lenguaje natural proporciona una descripción clara y comprensible de lo que hace el código y cómo funciona.

Promueve la colaboración: Al incluir una explicación en lenguaje natural, se fomenta la colaboración entre diferentes miembros del equipo de desarrollo. Esto es especialmente útil cuando varios desarrolladores están trabajando en el mismo proyecto y necesitan entender rápidamente cómo funciona una parte específica del código.

Ayuda en la documentación: La explicación en lenguaje natural sirve como una parte importante de la documentación del código. Esto es útil para futuras referencias, mantenimiento del código y para ayudar a nuevos desarrolladores a comprender rápidamente el propósito y la funcionalidad del código existente.



Imagen 5.1



Imagen 5.2

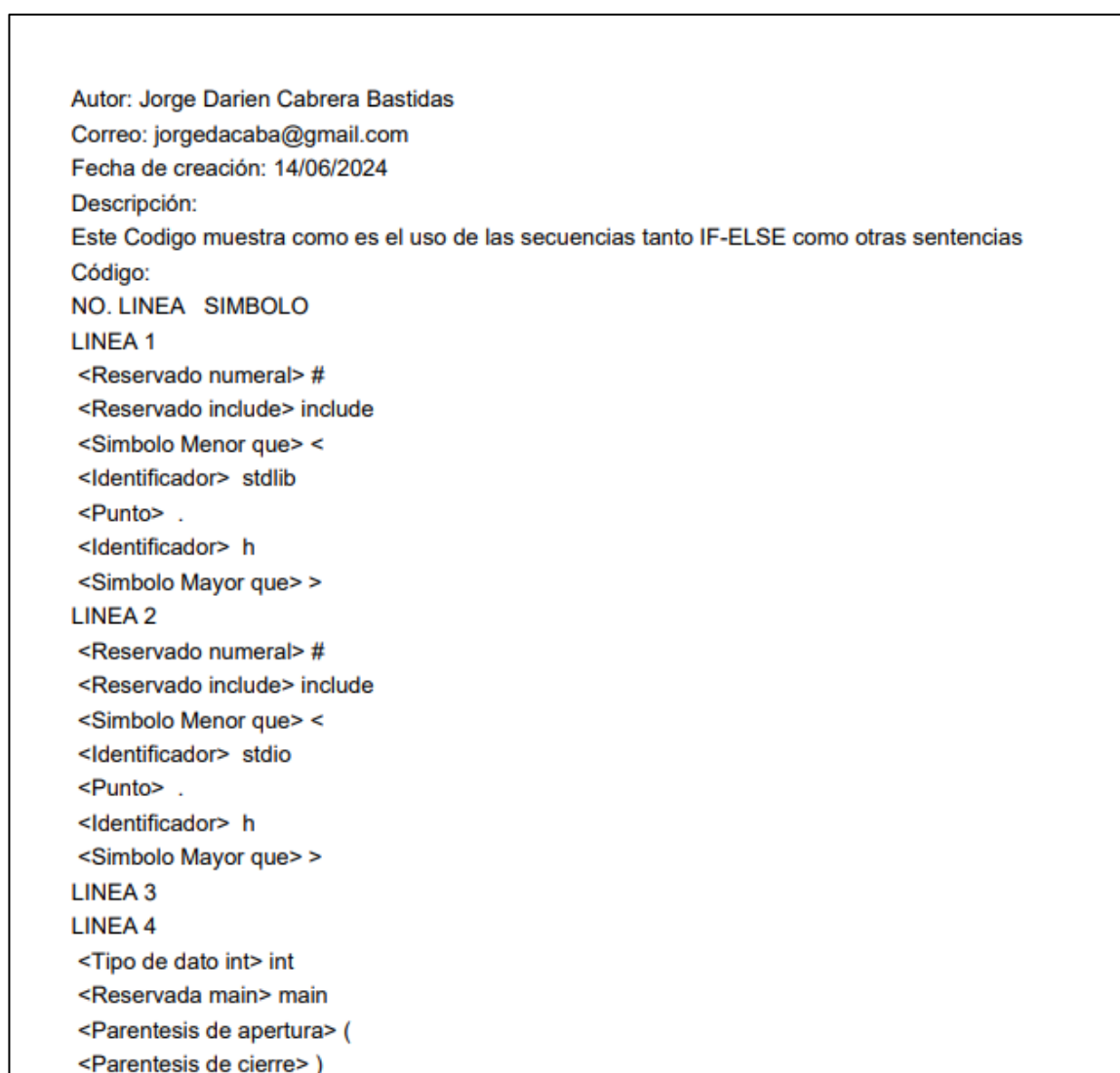


Imagen 5.3



Conclusiones

Jorge Darien Cabrera Bastidas

En el desarrollo de este proyecto se desarrollan habilidades sumamente importantes para nuestra formación como ingenieros en software, ya que con ellas podemos identificar tanto problemas sintácticos como léxicos de cualquier lenguaje de programación, aparte de desarrollar habilidades como las antes mencionadas, podemos identificar errores siquiera antes de escribirlos mejorando de manera radical habilidades de programación, gracias a ello podemos desarrollar aplicaciones como la presente de este programas que nos van a ayudar para la lógica que tanto deberíamos desarrollar y trabajar por el lugar donde nos desarrollamos para este tipo de carreras en las que estamos estudiando;

Los proyectos como estos son de alto valor educacional.



Bibliografía

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Pearson/Addison Wesley. Enlace a Amazon

Cooper, K. D., & Torczon, L. (2011). *Engineering a Compiler*. Morgan Kaufmann. Enlace a Amazon

Appel, A. W. (2004). *Modern Compiler Implementation in C*. Cambridge University Press. Enlace a Amazon

Parr, T. (2010). *Language Implementation Patterns*. The Pragmatic Bookshelf. Enlace a Amazon

Nystrom, R. (2015). *Crafting Interpreters*. Sitio web. Crafting Interpreters