Proyecto Final

C en Español

Julisa Lapa

Curso de Compiladores,
Ciencias de la Computación

December 18, 2021

Contents

1 Introducción

El estudio de compiladores es una rama fundamental en la Ciencia de la Computación, porque sin compiladores, los programadores de hoy en día tendrían que comunicarse por medio de código máquina con el computador. Los compiladores hacen este trabajo por las personas, traduciendo programas de un lenguaje escrito en alto nivel a otro de alto, medio o más bajo nivel. Sin compiladores no tendríamos ninguna aplicación informática, pues son la base de la programación en cualquier plataforma. El presente proyecto desarrolla la implementación de un compilador para un lenguaje llamado "C en Español", un lenguaje que se nos planteó en el curso de Compiladores. Para su implementación se usaron distintas herramientas como Flex, para la generación del analizador léxico, Yacc(Yet Another Compiler-Compiler) para la generación del analizador sintáctico y GoogleTest para el testeo de los analizadores implementados.

1.1 C en Español

C en Español, como su nombre lo indica, es una versión en español del lenguaje C. Mantiene su sintaxis y estructura, mas los tokens ahora son palabras en castellano. La gramática implementada fue obviamente más corta que la que se maneja en el C original, pero es muy intuitiva y posiblemente aplicable a la enseñanza de programación básica. La gramática se adjunta al final de este informe.

2 Objetivos

- Implementación de los analizadores léxico, sintáctico y semántico.
- Maneojo de errores en la implementación del compilador.
- Mejoras o agregados al compilador implementado.

3 Antecedentes

Para el desarrollo de este proyecto, se necesitó nociones previas en cursos como Arquitectura de Computadores, Teoría de la Computación y Algoritmos y Estructuras de Datos. Cursos que se llevaron previamente, y fueron recordados a lo largo del curso en las clases dictadas. Temas como máquinas de estados o el análisis de los algoritmos presentados a lo largo del curso, asimismo determinar que estructuras serían las más convenientes a la hora de la implementación semántica fueron muy necesarios en este proyecto.

4 Fundamento teórico

En todo compilador se pueden determinar dos partes, llamadas frontend y backend. Por un lado, el frontend se divide en otras cuatro partes, estas son: escáner(análisis léxico),

parser(análisis sintáctico), analizador semántico y generador de código intermedio. Por otro lado, en el backend están el optimizador del código y el generador de código objetivo. Estos componentes procesan lo que el compilador recibe, código fuente y se traduce a instrucciones entendibles para la máquina. Cabe mencionar el uso también de una tabla de símbolos definidos por el lenguaje que también alamcena las variables que se crean en el código fuente. Este proyecto se centró en la parte del frontend de un compilador.

5 Métodos y Desarrollo

Ahora, se mostrará el proceso de implementación de los componentes del frontend en nuestro compilador

5.1 Implementación del Scanner (Flex)

Este se realizó acorde la información recibida sobre los símbolos admitidos por nuestro lenguaje, se tradujo estos a tokens gracias a la herramienta flex, y se trabajó con ellos durante el resto de la implementación del compilador.

SÍMBOLOS	
C en Español	**Flex**
entero	ENTERO
retorno	RETORNO
sin_tipo	SIN_TIPO
mientras	MIENTRAS
si	SI
sino	SINO
main	MAIN
OPERADORES	
'+', '-', '*', '/', '<', '<=', '>', '>=', '==', '!=', '=', ';', ',', '(', ')', '[', ']', '{', '}'	SUM_OP, RES_OP, MUL_OP, DIV_OP, LT, LEQ, GT, GEQ, EQ, NEQ, ASSIGN, D_COM, COM, PAR_BEGIN, PAR_END, COR_BEGIN, COR_END, BRA_BEGIN, BRA_END

TOKENS DEFINIDOS	
Token label	Token Definition
ID	{LETRA}+ , dónde LETRA está definido por [a-z—A-Z—_]
NUM	{DIGIT}+, dónde DIGIT está definido por [0-9]
ws	{DELIM}+, dónde DELIM está definido por [\t]
Los comentarios no son reconocidos como tokens, sólo les ignoraba.	

5.2 Implementación del Parser (Bison, Yacc)

Para la implementación de este se utilizó la herramienta Bison, y lo que se hizo, fue traer las reglas gramaticales que se nos proporcionaron sobre el lenguaje C en Español, y se les adaptó a la sintaxis aceptada por Bison, usando los tokens previamente procesados por el analizador léxico. A continuación unos ejemplos ejemplos de cómo se realizó.

```
%union{
    int     int_val;
    string* id_val;
    string* op_val;
}

%start  programa


%token <op_val> ENTERO
%token <op_val> SIN_TIPO
%token <op_val> RETORNO
%token <op_val> MIENTRAS
%token <op_val> MAIN
%token <op_val> SINO
%token <op_val> SI

%token <int_val> NUM
%token <id_val> ID

%token <op_val> SUM_OP
%token <op_val> RES_OP
%token <op_val> MUL_OP
%token <op_val> DIV_OP
```

Figure 1: Definicion de los tokens

```
programa: lista_declaracion
    ;

lista_declaracion:
    lista_declaracion declaracion {}
    | declaracion {}
    ;

declaracion:
    var_declaracion {$$ = $1;}
    | fun_declaracion {$$ = $1;}
    ;

var_declaracion:
    ENTERO ID more {…
    | ENTERO ID COR_BEGIN NUM COR_END D_COM {…
;

fun_declaracion:
    ENTERO MAIN PAR_BEGIN params PAR_END sent_compuesta {…
    | SIN_TIPO MAIN PAR_BEGIN params PAR_END sent_compuesta {…
    | ENTERO ID PAR_BEGIN params PAR_END sent_compuesta {…
    | SIN_TIPO ID PAR_BEGIN params PAR_END sent_compuesta{…
;
```

Figure 2: Ejemplo de reglas gramaticales adaptadas

5.3 Analizador Semántico (C++)

No se llegó a completar en su totalidad, mas sí se creo la tabla de símbolos respectiva, que guardara todas las características de los tokens que recíbía. Para esto nos apoyamos de dos estructuras definidas como attr, que corresponde a un token con sus características y SymbolTable que se comportará como nuestra tabla de símbolos con los respectivos métodos de inserción, búsqueda y borrado.

```
struct attr{
    ID_type id;
    F_type f;
    V_type v;
    int asize;
    int val;
};

struct SymbolTable {
    vector<map<string, attr> > tables;

    SymbolTable() {…

    map<string, attr>& current() {…

    void insert_symbol(string id, attr val) {…

    bool search_symbol_local(string id) {…

    bool search_symbol_global(string id) {…

    void update_symbol(string id, attr new_val) {…

    void add_scope() {…

    void remove_scope() {…
};
```

Figure 3: Estructuras

6 Resultados y conclusiones

6.1 Resultados

Se probó el compilador con varios archivos txt de prueba, cabe mencionar que en el proceso de implementar las reglas gramaticales, se imprimieron muchas sentencias a modo de logs de que el código fue procesado correctamente. Igualmente se dispuso de dos funciones para tratar los errores, no la corrección de estos, pero sí la ubicación para ayudar al programador. A continuación un ejemplo de programa de prueba y output respectivo.

```
1   entero m;
2
3   entero fun_o(entero b, entero c[]){
4       entero c[8];
5       retorno b;
6   }
7
8   sin_tipo fun_a(entero b){
9       entero c[8];
10      retorno b;
11  }
12
13  /* comentario owo */
14
15  entero main(){
16      entero b[5];
17      entero a;
18      fun_o(a);
19      si (a == 2){
20          a = 3;
21      }
22      mientras (a != 1){
23          a = a +1;
24      }
25
26      retorno a;
27  }
```

Figure 4: Prueba

```
# jlr @ jlr-ROG-Zephyrus-G14-GA401IV-GA401IV in ~/Desktop/CICLO6
$ ./ces test3.txt
Declarando variable "m" de tipo entero en linea 1.
Función con parametros en linea 3.
Declarando arreglo "c" de tipo entero en linea 4.
Sentencia de retorno procesada correctamente.
Función tipo entero declarada correctamente.
Función con parametros en linea 8.
Declarando arreglo "c" de tipo entero en linea 9.
Sentencia de retorno procesada correctamente.
Función sin_tipo declarada correctamente.
Función sin parametros en linea 15.
Declarando arreglo "b" de tipo entero en linea 16.
Declarando variable "a" de tipo entero en linea 17.
Llamada a función "fun_o" procesada correctamente en linea 18.
Sentencia condicional declarada correctamente.
Sentencia condicional declarada correctamente.
Sentencia de retorno procesada correctamente.
Función main tipo entero declarada correctamente.
```

Figure 5: Output

6.2 Conclusiones

Disfruté mucho del desarrollo de este proyecto, aprendí muchas cosas que no sabía antes sobre los compiladores y cómo trabajaban, C en Español es una forma muy entretenida

para aprender sobre compiladores a un nivel de Frontend. Si bien no se llegó a la implementación completa de todas las reglas semánticas, se puso mucho empeño en la implementación de un analizador sintáctico y léxico correctos. También se tuvieron varias dificultades en el proceso, sin embargo se busco agregar pequeñas mejoras al analizador léxico, sintáctico en compensación por falta de generación de código intermedio.

7 Apéndice

El proyecto puede encontrarse en el siguiente repositorio. GitHub

8 Referencias

- Material proporcionado por el profesor del curso. (Enunciado del proyecto con reglas gramáticales)
- Flex, Official Website
- Bison, Official Website