Universidad Católica "Nuestra Señora de la Asunción"

Facultad de Ciencias y Tecnología

Ingeniería Informática

Compiladores

Implementación de un esquema de traducción de un código en lenguaje C a uno equivalente en JavaScript

Christian Arce
Camila Do Rego Barros

2024

Introducción

El proyecto tiene como objetivo desarrollar un esquema de traducción que permita convertir el código fuente escrito en un lenguaje L1 a su equivalente en un lenguaje L2. En este caso, se ha elegido C como L1 y JavaScript como L2.

Para realizar esta traducción, se emplea el lenguaje C junto con Flex, un generador de analizadores léxicos, y Bison, un generador de analizadores sintácticos. El archivo con la extensión .l (lex.l) contiene la implementación del analizador léxico, donde se describen las expresiones regulares y las acciones asociadas. Por otro lado, el archivo con la extensión .y (yacc.y) implementa el analizador sintáctico, que define la gramática y las acciones sintácticas correspondientes.

Al utilizar Bison, procesamos una gramática LALR(1), que es una versión simplificada de la gramática canónica LR(1). En esta notación, la primera "L" indica que la entrada se lee de izquierda a derecha, mientras que la "R" significa que las derivaciones se realizan por la derecha.

Aspectos Implementados

Tipos de datos y estructuras	Int, Float, Char, Strings, arreglos unidimensionales y bidimensionales(tabla de símbolos)
Operadores	**Aritméticos:** Suma + , resta - , multiplicación *, división /, módulo % **De comparación:** Igualdad ==, no igualdad !=, mayor que >, mayor o igual que >=, menor <, menor o igual que <=, **Lógicos:** Conjunción lógica && , disyunción lógica ||, negación lógica ! **De asignación: "=**" **De Incremento/Decremento:** ++, – –
Instrucciones condicionales	if, if-else
Instrucciones de bucle	while
Funciones	Definición de funciones (con y sin parámetros), llamado de funciones
Instrucciones de salida	printf, console.log
Otros	Comentarios de una línea y multilinea

Utilización del traductor

1. Clonar el repositorio de GitHub

```
git clone https://github.com/camidorego/c2js_transpiler.git
```

2. Instalación de Flex y Bison(para Linux):

```
sudo apt install flex
```

```
sudo apt install bison
```

3. Ubicarse en el directorio del repositorio

```
cd c2js_transpiler
```

4. Compilar y ejecutar el traductor

a) Ejecutar un archivo específico

```
make FILE=prueba/<archivo.c>
```

b) Si no se especifica ningún archivo, por defecto se ejecuta prueba.c

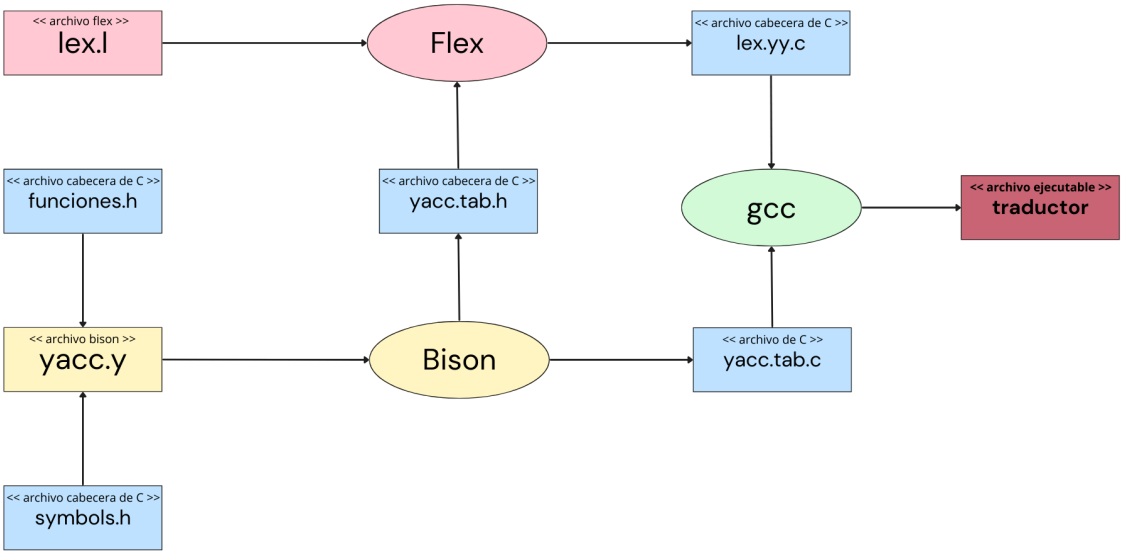
```
make
```

5. Se genera el archivo resultante de Javascript output_file.js

Proceso de Construcción del Traductor

Primero, el archivo `lex.l` es procesado por Flex para generar `lex.yy.c`, que contiene el analizador léxico. Paralelamente, Bison toma el archivo `yacc.y` y lo convierte en `yacc.tab.c` y `yacc.tab.h`, generando el analizador sintáctico. Estos archivos `.h` y `.c` también incluyen referencias a los archivos de cabecera `funciones.h` y `symbols.h`, que definen las funciones y la tabla de símbolos necesarias para la detección de errores y otras tareas. Finalmente, gcc compila `lex.yy.c`, `yacc.tab.c` y las cabeceras asociadas para crear el archivo ejecutable `traductor`, que es capaz de convertir el código fuente de C a JavaScript según las reglas definidas en los archivos de Flex y Bison.

Flujo de compilación del traductor



¿Por qué escogimos estos lenguajes?

Ambos lenguajes son extremadamente populares y ampliamente utilizados en la industria. C es un lenguaje fundamental que ha influido en muchos otros lenguajes de programación, mientras que JavaScript es el lenguaje principal del desarrollo web.

Una de las principales razones para elegir C es nuestra familiaridad con él. Fue uno de los primeros lenguajes que nos enseñaron en la facultad, lo que nos permite comprender y manejar su sintaxis y semántica con facilidad. Esta experiencia educativa nos proporciona una base sólida para trabajar con C.

JavaScript, por otro lado, es conocido por su versatilidad y uso generalizado tanto en el frontend como en el backend del desarrollo web. Su popularidad y flexibilidad lo convierten en una excelente opción para la traducción desde un lenguaje más estructurado como C. Además, la sintaxis de JavaScript no es tan estricta como la de C. Por ejemplo, en JavaScript, el punto y coma al final de las declaraciones es opcional, lo que hace que el código sea más flexible. También realiza un casting implícito cuando se efectúan operaciones lógicas y aritméticas entre variables de diferentes tipos de datos, facilitando el manejo de estas operaciones sin errores de tipo.

Otra ventaja de JavaScript es su manejo de estructuras de datos y objetos de manera poderosa y fácil de entender, lo que facilita la traducción de estructuras desde C.

Limitaciones

- No se traducen correctamente las estructuras anidadas, debido a la falta de indentación. Para ninguno de los lenguajes esto es un impedimento y pueden ejecutarse sin problema. Una forma de solucionar esto puede ser definiendo funciones específicas para el formateo.
- Se verifica el scope de las variables al declararse y también se controla si se intenta declarar una variable con el mismo nombre dentro del mismo scope y en ese caso, se imprime un mensaje de error y se especifica la línea donde se intentó redefinir la variable. Sin embargo, igual se traduce el intento de redefinición debido a la forma en la que implementamos la traducción, mediante appends al archivo js. Una forma de solucionar esto sería encarar de otra forma la traducción, mediante funciones más específicas. Tampoco se verifica llamadas a variables en scopes incorrectos.

Técnicas de detección de errores

La tabla de símbolos (symbols.h) se utiliza para la detección de errores. Se implementaron las siguientes comprobaciones

- Verificación si falta un punto y coma al final de un statement en el archivo .c
- Cuando se intenta acceder a un elemento de un array, de la forma arr[0]; se verifica que la variable sea realmente un array y también si es de la dimensión correcta. Por ejemplo, si se intenta acceder al elemento matrix[0] pero matrix es un array bidimensional, se imprimirá un error. Lo mismo, si matrix resulta que no es un array
- Cuando se declara una variable, se verifica que no exista otra variable con el mismo nombre dentro del mismo scope
- Para las operaciones aritméticas, se verifica que los operandos sean del mismo tipo

Para todos los casos, se imprime un mensaje de error indicando el problema y la linea del código donde ocurre

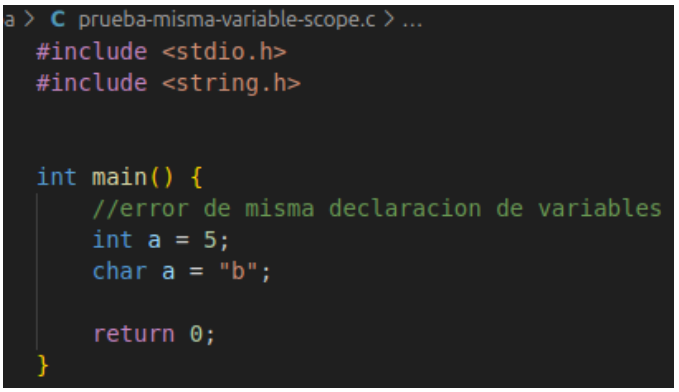
Ejecutando archivo: prueba-arrays.c

```c
a > C prueba-arrays.c > ...
#include <stdio.h>
#include <string.h>

int main() {
    //en caso de no poner punto y coma, nofica y lo traduce igualmer
    int flag = 1
    // error al llamar una variable int como array
    printf("%d", flag[1]);
    int arr[3];
    // error al llamar un array de dimension 1 como dimension 2
    printf("%d", arr[2][1]);

    return 0;
}
```

```
./traductor < prueba/prueba-arrays.c
Comenzando a traducir a JavaScript
Error de punto y coma en la linea 7: falta punto y coma
Error de dimension en la linea 8: flag no es un array
Error de dimension en la linea 11: arr array pero se le llama con dimension incorrecta
Tabla de Simbolos:
-----------------------------------------------------------------------------------------------------------
| Tipo       | Nombre     | Constancia    | Tipo de dato | Dimension    | Args | Scope      | Linea de declaracion
-----------------------------------------------------------------------------------------------------------
| VAR        | arr        | no constante | int          | 1            | 0    | 0          | 9
| VAR        | flag       | no constante | int          | 0            | 0    | 0          | 7
| FUNC       | main       | no constante | int          | 0            | 0    | No aplica | 4
-----------------------------------------------------------------------------------------------------------
```

Punto y coma faltante en la línea 7
Se intenta acceder al elemento flag[1] en la línea 8 pero flag no es un array
Se intenta acceder al elemento arr[2][1] en la línea 11 pero arr es un array de una dimensión

Ejecutando archivo: prueba-misma-variable-scope.c

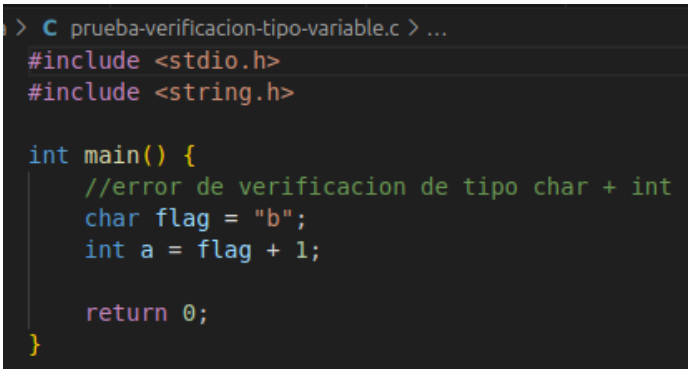


```c
#include <stdio.h>
#include <string.h>


int main() {
    //error de misma declaracion de variables
    int a = 5;
    char a = "b";

    return 0;
}
```

```
gcc -o traductor lex.yy.c yacc.tab.c -lfl
./traductor < prueba/prueba-misma-variable-scope.c
Comenzando a traducir a JavaScript
Scope error linea 8: ya existe var a en este scope
Tabla de Simbolos:
-----------------------------------------------------------------------------------------------------------------
| Tipo       | Nombre     | Constancia   | Tipo de dato | Dimension   | Args | Scope      | Linea de declaracion
-----------------------------------------------------------------------------------------------------------------
| VAR        | a          | no constante | char         | 0           | 0    | 0          | 8
| VAR        | a          | no constante | int          | 0           | 0    | 0          | 7
| FUNC       | main       | no constante | int          | 0           | 0    | No aplica | 5
-----------------------------------------------------------------------------------------------------------------
```

En la línea 8 se intenta declarar una variable a pero ya existe otra variable con ese mismo nombre en el mismo scope, se imprime el error

Ejecutando archivo: prueba-verificacion-tipo-variable.c



```c
#include <stdio.h>
#include <string.h>

int main() {
    //error de verificacion de tipo char + int
    char flag = "b";
    int a = flag + 1;

    return 0;
}
```

```
./traductor < prueba/prueba-verificacion-tipo-variable.c
Comenzando a traducir a JavaScript
Error de tipo en la linea 7: flag (char) no puede operar con 1 (int)
Tabla de Simbolos:
-----------------------------------------------------------------------------------------------------------------
| Tipo       | Nombre     | Constancia   | Tipo de dato | Dimension   | Args | Scope      | Linea de declaracion
-----------------------------------------------------------------------------------------------------------------
| VAR        | a          | no constante | int          | 0           | 0    | 0          | 7
| VAR        | flag       | no constante | char         | 0           | 0    | 0          | 6
| FUNC       | main       | no constante | int          | 0           | 0    | No aplica | 4
-----------------------------------------------------------------------------------------------------------------
```

En la línea 7 se intenta realizar una operación de suma entre un elemento de tipo char y otro de tipo int

Conclusión

Este proyecto de traducción de C a JavaScript ha sido una experiencia enriquecedora tanto a nivel técnico como educativo. Utilizando Flex para el análisis léxico y Bison para el análisis sintáctico, hemos logrado crear una herramienta efectiva que convierte código C en su equivalente en JavaScript, combinando lo mejor de ambos lenguajes.

Hemos implementado varias comprobaciones en la tabla de símbolos para detectar errores comunes, como la falta de un punto y coma o el uso incorrecto de arrays y tipos de datos. Estas verificaciones aseguran que el código traducido sea preciso y funcional, manteniendo la integridad del programa original.

Además, el proyecto nos ha permitido profundizar en nuestra comprensión de la traducción de lenguajes de programación y en el uso de herramientas poderosas como Flex y Bison.