Universidad Católica
"Nuestra Sra. de la Asunción"

Compiladores

Trabajo Práctico Final

Tema: Esquema de traducción dirigida por la sintaxis. Lenguaje C a PHP

Alumno: Matías Irala Aveiro

Profesores: Ernst Heinrich Goossen
Mauricio Kreitmayr

Asunción, Paraguay
2020

Índice

1. How To del Compilador

1. Al descomprimir la carpeta del proyecto se encontrará con los siguientes archivos

```
cToPhp_BisonAndFlex
|    header.h
|    Makefile
|    symboltable.c
|    translator.l
|    translator.y
|    ...
└────codigos_ejemplo
|    |    fibbonacci.c
|    |    ...
```

2. Abra una ventana de terminal, ubicándose en la carpeta `cToPhp_BisonAndFlex`
3. Compile el proyecto con el comando: `make`

```
[dyuko@dyuko-hplaptop15db0xxx cToPhp_BisonAndFlex]$ make
bison -d translator.y
translator.y: aviso: 19 conflictos desplazamiento/reducción [-Wconflicts-sr]
flex translator.l
gcc -O3 translator.tab.c lex.yy.c symboltable.c -lfl -o translator
[dyuko@dyuko-hplaptop15db0xxx cToPhp_BisonAndFlex]$
```

4. Se creará el archivo ejecutable `translator`, la ejecución se realiza pasándole dos parámetros, como muestra a continuación:
   `./translator path_to_c_code path_to_php_code`
   `path_to_c_code`: Es la ruta al archivo existente con código fuente C
   `path_to_php_code`: Es la ruta al archivo destino donde realizará la traducción
   Ejemplo:

```
[dyuko@dyuko-hplaptop15db0xxx cToPhp_BisonAndFlex]$ ls
codigos_ejemplo  lex.yy.c  symboltable.c  translator.l      translator.tab.h
header.h         Makefile  translator     translator.tab.c  translator.y
[dyuko@dyuko-hplaptop15db0xxx cToPhp_BisonAndFlex]$ ./translator codigos_ejemplo/example.c codigos_ejemplo/output.php
```

   En la carpeta codigos_ejemplo se encuentra varios códigos fuentes C que puede utilizar como prueba.

2. Funcionalidades implementadas

2.1. Declaraciones de variables

Los tipos básicos de variables en PHP son boolean, integer, float-double y string.
Teniendo en cuenta esto, el traductor solo da soporte adecuado a los tipos de variables integer, float, double y char.

Código en C:

```c
int a;
int b = 1;
float c;
float d = 0.01;
```

Código traducido a PHP:

```php
<?php
$a;
$b=1;
$c;
$d=0.01;
?>
```

2.1.1. Limitaciones en las Declaraciones de variables

* El traductor no da el soporte correcto a los tipos de variables de C que no se mencionan anteriormente, como lo son los punteros.
* En C, los strings son en realidad una matriz unidimensional de caracteres terminados por un carácter nulo '\0'.

Por lo que expresiones en C como

```c
char cadena[] = "String";
```

el traductor los estaría traduciendo a PHP como

```php
$cadena=array( "String" );
```

Mientras que lo ideal sería una traducción como:

```php
$cadena="String";
```

Causa: En la gramática, al detectar el token '[' directamente se imprime 'array('
Posible solución: Utilizar atributos sintetizados, y decidir qué imprimir recién al recibir el token "String".

2.2. Declaraciones de constantes

A partir de PHP 5.3.0 podemos declarar una constante con la palabra reservada const, muy similar a C.

Código en C:

```c
const int CONSTANTE = 1;
int x = CONSTANTE;
```

Código traducido a PHP:

```php
<?php
const CONSTANTE=1;
$x=CONSTANTE;
```

2.3. Tabla de Símbolos

Se implementó una tabla de símbolos bastante simple y reducida, el mismo se imprime en la terminal.

Por ejemplo, con el siguiente código en C:

```c
int main()
{
   int x;
   char y;
}
```

En la tabla de símbolos se obtiene:

```
[dyuko@dyuko-hplaptop15db0xxx cToPhp_BisonAndFlex]$ ./translator coc

                    Tabla de símbolos
Identificador       Tipo de valor           Tipo de símbolo
main                int                     Función
y                   char                    Variable
x                   int                     Variable
```

2.4. Ciclos (Loops)

Se implementó los tipos de ciclos while, for y do while.

Código en C

```
int x=0;
while(x<5)
{
    x++;
}
```

Código traducido a PHP

```
$x=0;
while($x < 5){
$x++;
}
```

Código en C

```
int x=0;
for(x=0; x<10; x++)
{
}
```

Código traducido a PHP

```
$x=0;
for( $x = 0;
$x < 10;
$x++ ){
}
```

Código en C

```
int x=0;
do
{
    x++;
} while (x<10);
```

Código traducido a PHP

```
$x=0;
do{
$x++;
}
while($x < 10);
```

2.5. Estructuras Condicionales

Se implementó las estructura condicionales if y switch case.

Código en C

```
int x=0;
if(x>0)
{
    x++;
}
```

Código traducido a PHP

```
$x=0;
if ( $x > 0){
$x++;
}
```

Código en C

```c
int x=0;
switch(x)
{
    case 0:
        x++;
        break;
    case 1:
        x++;
        break;
}
```

Código traducido a PHP

```php
$x=0;
switch ( $x ){
case 0: $x++;
break;
case 1: $x++;
break;
}
```

2.5.1. Limitaciones en las Estructuras Condicionales

El traductor no soporta la sentencia if con else.
Causa: Error en el diseño de la gramática.

2.6. Estructuras anidadas

En principio, todas las estructuras anidadas son soportadas.

Código en C

```c
int x=10;
if(x>1)
    if(x>2)
        if(x>3)
            while(x==10)
            {
                x++;
                if(x==11)
                    x=0;
            }
```

Código traducido a PHP

```php
$x=10;
if ( $x > 1)
if ( $x > 2)
if ( $x > 3)
while($x == 10){
$x++;
if ( $x == 11)
$x = 0;
}
```

2.7. Procedimientos y Funciones

Se soporta la definición y la llamada a funciones con y sin parámetros.

Código en C

```c
int suma(int x)
{
   x++;
}
int main()
{
   int y = 0;
   suma(y);
}
```

Código traducido a PHP

```php
<?php
function suma ($x ){
$x++;
}
function main (){
$y=0;
suma( $y );
}
main();
?>
```

2.7.1. Limitaciones de los Procedimientos y Funciones

No se soportan las declaraciones de las funciones (prototipo de función).
Causa: En PHP las funciones deben estar definidas antes de ser llamadas.

Además no se traduce las funciones de librería que tienen un equivalente en el lenguaje destino. Por ejemplo no se traduce printf() de C a echo() de PHP.

2.8. Comprobación de tipos

Si se encuentra una operación de suma, resta, multiplicación, división o módulo, se verifica en la tabla de símbolos que los tipos de datos de los operandos sean int, float o double, en caso contrario se imprime un mensaje de alerta en la terminal.
Ejemplo:
Código en C:

```c
int main()
{
    int x=0;
    char y= '0';
    x= x%y;

}
```

Mensaje emitido:

```
[dyuko@dyuko-hplaptop15db0xxx cToPhp_BisonAndFlex]$ ./translator codigos_ejemplo/ex
El operando y en la operación % posee un tipo (char) no permitido, (línea 5)
```

2.8.1. Limitaciones de la Comprobación de tipos

Se delimitó bastante la comprobación de tipos como se explicó anteriormente.

2.9. Vectores

Se implementó los vectores con sus usos básicos como se muestra a continuación.
En PHP la dimensión del vector es ignorada.

Código C

```c
int main()
{
   int a[3];
   a[0] = 1;
   a[1] = 2;
   a[2] = 3;
   int b[3] = {1,2,3};
   int c[] = {1,2,3};
}
```

Código traducido a PHP

```php
<?php
function main (){
$a=array(  );
$a[ 0 ] = 1;
$a[ 1 ] = 2;
$a[ 2 ] = 3;
$b=array( 1 ,2 ,3 );
$c=array( 1 ,2 ,3 );
}
main();
?>
```

2.10. Detección y Recuperación de errores

Para la detección y recuperación de errores se escribió reglas utilizando el token especial 'error'. El traductor se centra en detectar y recuperarse de errores causados por la ausencia de un símbolo, como pueden ser un punto y coma, la ausencia de un cerrar llave o paréntesis, etcétera.

Al detectar un error el traductor simplemente descarta los símbolos hasta un punto donde pueda continuar operando.

A continuación se muestra un código C con errores comunes, y los mensajes emitidos por el traductor.

Código C fibonnaci_error.c:

cToPhp_BisonAndFlex > codigos_ejemplo > C fibbonacci_error.c

```c
#include <stdio.h>
int main() {
    int i, n=100, t1 = 0, t2 = 1, nextTerm;
    for i = 1; i <= n; ++i) {
        nextTerm = t1 + t2
        t1 = t2;
        t2 = nextTerm;

    return 0;
}
```

Mensajes de error emitidos:

```
[dyuko@dyuko-hplaptop15db0xxx cToPhp_BisonAndFlex]$ ./translator codigos_ejemplo/fibbonacci_error.c codigos_ejemplo/output.php
syntax error en la linea: 4    i
syntax error en la linea: 6    t1
Símbolo faltante ;
syntax error en la linea: 10
Símbolo faltante }
```

2.10.1. Limitaciones de la Detección y Recuperación de errores

No todos los posibles errores están contemplados, por lo que es posible que el programa se detenga al detectar un error, o que este genere una cadena de errores.

2.11. Arrays multidimensionales

Al igual que con los vectores, en PHP se ignora la dimensión de los arrays multidimensionales.

Código en C

```c
int main()
{
   int a[3][5];
   a[0][0] = 1;
}
```

Código traducido a PHP

```php
<?php
function main (){
$a=array(  );
$a[ 0 ][ 0 ] = 1;
}
main();
?>
```

2.11.1. Limitaciones de los Arrays multidimensionales

No se soporta la definición e inicialización de los arrays multidimensionales.

Por ejemplo, el siguiente código en C no es soportado:

```
int arreglo[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
```

Se traduciría incorrectamente a PHP como:

```
<?php
$arreglo1=array( 1 ,2 ,3 ,4 ,5 ,6 );
?>
```

Mientras que la traducción correcta sería:

```
<?php
$arreglo1=array(
   array(1 ,2 ,3),
   array(4 ,5 ,6)
);
?>
```

2.12. Reglas de Ámbito

Se limitó enormemente las reglas de ámbito, el traductor trata todo el código fuente como un solo scope.

Se realiza la verificación de que una variable no se declare más de una vez, si se detecta varias declaraciones se emite un mensaje en la terminal.

Ejemplo:

Código en C

```
int main()
{
   int x;
   int y;
   int z;
   int x;
}
```

Mensaje

```
[dyuko@dyuko-hplaptop15db0xxx cToPhp_BisonAndFlex]$ ./translator codigos_ejemplo/example.c codigos_ejemplo/output.php
x ya declarada (linea 6)
```

2.12.1. Limitaciones de las Reglas de Ámbito

Como se mencionó anteriormente, todo el código fuente en C se trata como un solo scope, la división en varios scopes hubiera conllevado a una complejidad bastante alta en código y en la estructura de la tabla de símbolos.

3. Desarrollo del traductor

3.1. Selección de los lenguajes

El lenguaje C fue escogido como fuente debido a la profundidad de conocimientos en este, adquiridos a lo largo de la carrera.
El lenguaje PHP fue escogido como destino debido a que tenía que modificar código en PHP para un proyecto de otra asignatura, por lo que aproveché este trabajo para aprender sobre su sintaxis.

3.2. Proceso de desarrollo

Para comenzar a trabajar necesitaba la gramática Yacc junto a la especificación Lex compatible para el lenguaje fuente C.
Buscando en internet encontré varias opciones, tomé como base las siguientes:
ANSI C grammar, Lex specification: http://www.quut.com/c/ANSI-C-grammar-l-2011.html
ANSI C, Yacc grammar: http://www.quut.com/c/ANSI-C-grammar-y.html

La especificación Lex es sencilla de entender, en cambio la gramática de Yacc es complicada debido a su larga extensión.

Estas dos especificaciones no soportan las directivas de preprocesador como #include, para no tener que estar borrando manualmente dichas directivas del código fuente, puse para que sean ignorados en el archivo Lex una vez detectadas. Para ello utilicé la expresión regular encontrada en:
https://stackoverflow.com/questions/1420017/regular-expression-to-match-c-include-file

Luego me di cuenta que necesitaría utilizar atributos sintetizados para la traducción, y que lo mejor sería utilizar los valores de los tokens como strings. Para ello seguí la siguiente guía: https://stackoverflow.com/questions/43125576/using-only-string-with-flex-bison

En este punto, lentamente fui estableciendo en yacc las reglas semánticas para la traducción a PHP.

Para la detección y recuperación de errores seguí las directivas de este tutorial: http://dinosaur.compilertools.net/bison/bison_9.html

La implementación de la tabla de símbolos lo obtuve de la siguiente página: https://www.gnu.org/software/bison/manual/html_node/Mfcalc-Symbol-Table.html

Las dudas conceptuales que tenía los resolví consultando el libro: Compiladores principios técnicas y herramientas, segunda edición.

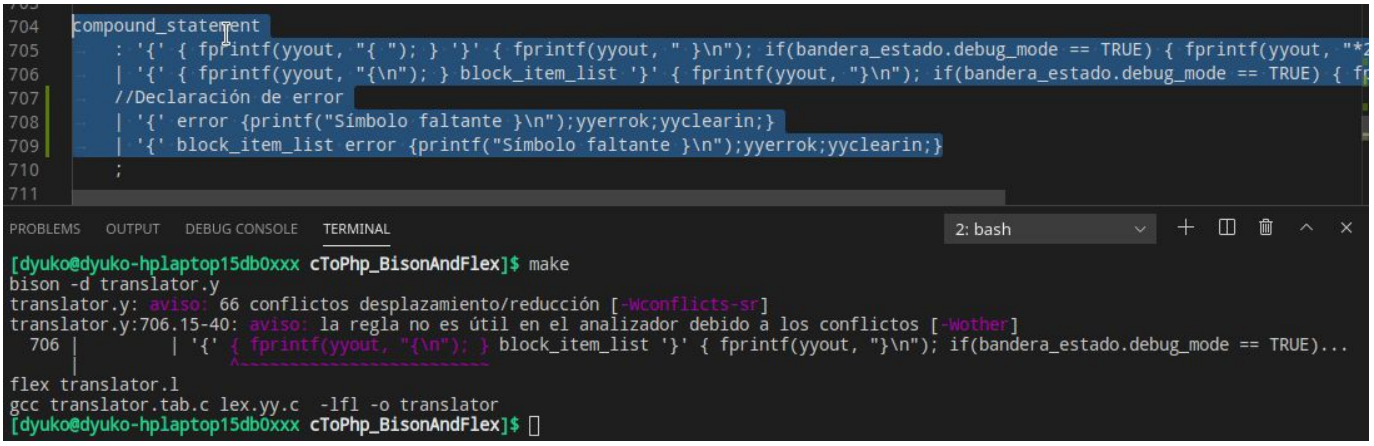
3.3. Problemas superados durante el desarrollo

La cantidad de errores que se experimentó durante el desarrollo del traductor son innumerables, la mayoría causados debido a que trabajar con la gramática en Yacc es bastante complicado ya que no se encuentra disponible un IDE decente que lo soporte, es un trabajo con texto plano.

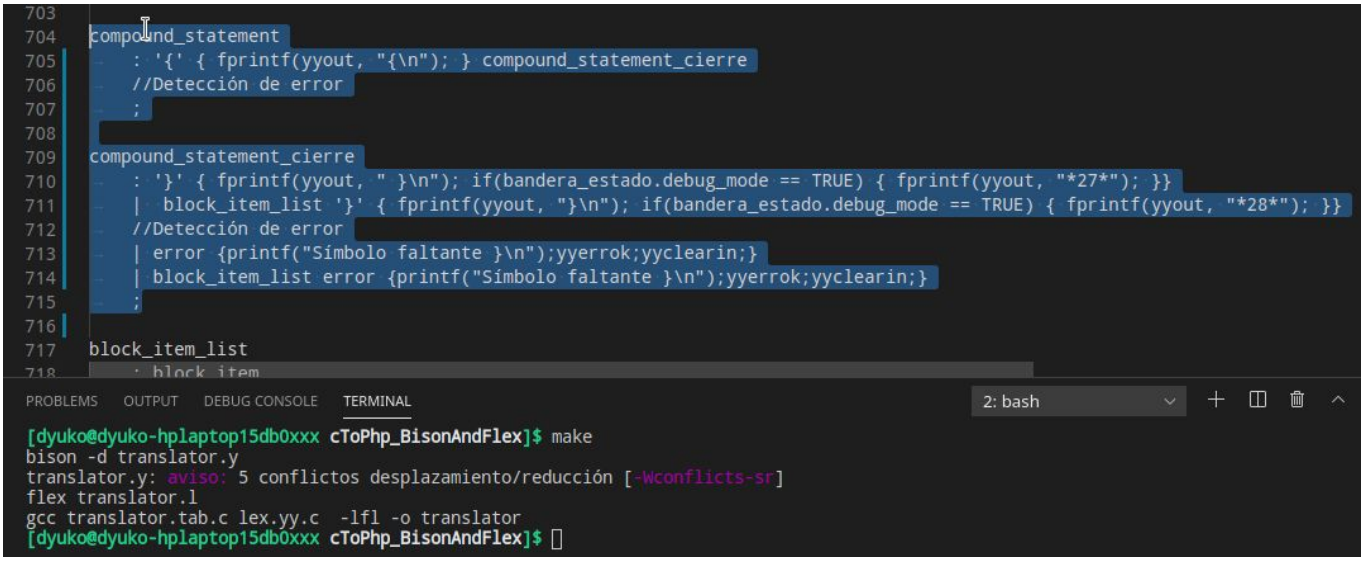
Algunas problemas superados fueron:

- **Problema al introducir las reglas**

Al introducir las reglas se encontró con errores debido a los conflictos de desplazamiento/reducción.

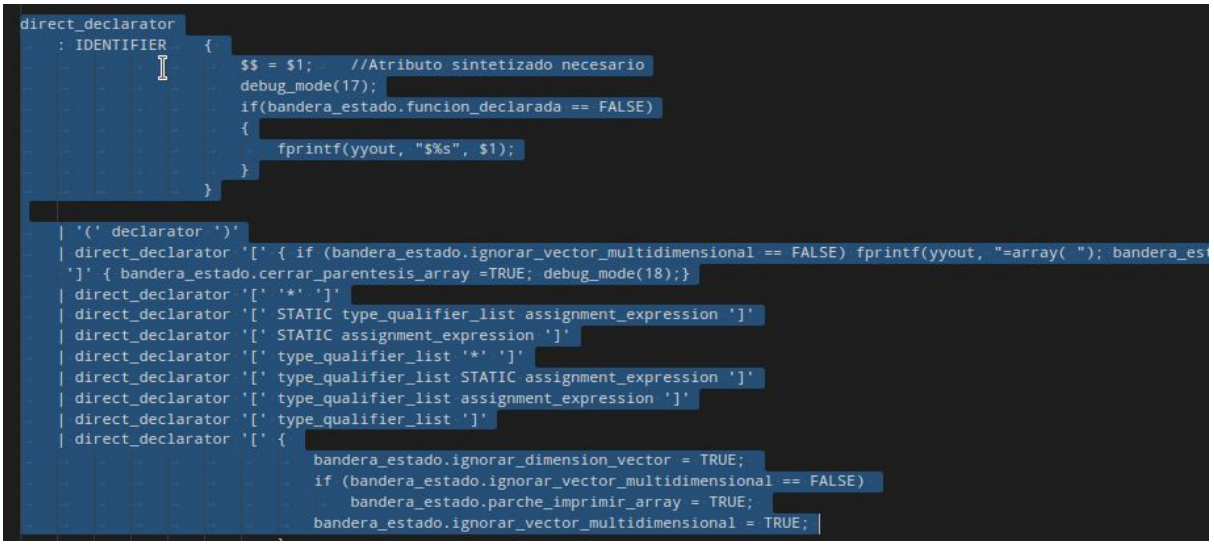


Entonces para poder introducir las reglas se modificó la gramática como muestra a continuación.



- **Problema con la recursividad en la gramática**

Tratar la recursividad de la gramática fue bastante complejo, tenía reglas asociadas al caso base y reglas asociadas a los casos recursivos; si entraba al caso recursivo debía ignorar la regla del caso base, lo que no era posible.

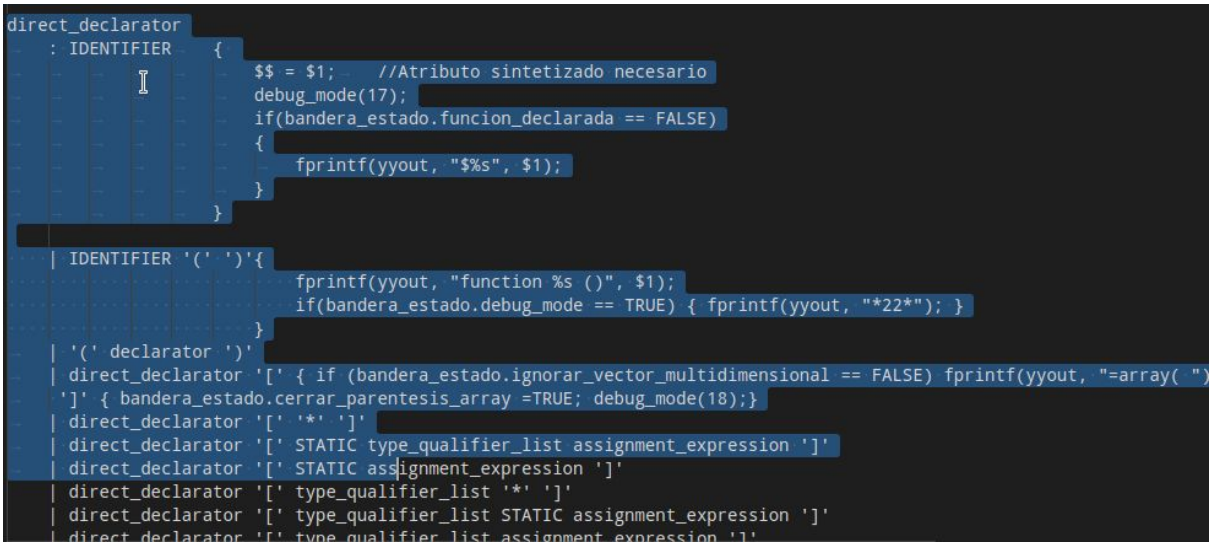


```
direct_declarator
    : IDENTIFIER    {
                        $$ = $1;    //Atributo sintetizado necesario
                        debug_mode(17);
                        if(bandera_estado.funcion_declarada == FALSE)
                        {
                            fprintf(yyout, "$%s", $1);
                        }
                    }

    | '(' declarator ')'
    | direct_declarator '[' { if (bandera_estado.ignorar_vector_multidimensional == FALSE) fprintf(yyout, "=array( "); bandera_est
     ']' { bandera_estado.cerrar_parentesis_array =TRUE; debug_mode(18);}
    | direct_declarator '[' '*' ']'
    | direct_declarator '[' STATIC type_qualifier_list assignment_expression ']'
    | direct_declarator '[' STATIC assignment_expression ']'
    | direct_declarator '[' type_qualifier_list '*' ']'
    | direct_declarator '[' type_qualifier_list STATIC assignment_expression ']'
    | direct_declarator '[' type_qualifier_list assignment_expression ']'
    | direct_declarator '[' type_qualifier_list ']'
    | direct_declarator '[' {
                                bandera_estado.ignorar_dimension_vector = TRUE;
                                if (bandera_estado.ignorar_vector_multidimensional == FALSE)
                                    bandera_estado.parche_imprimir_array = TRUE;
                                bandera_estado.ignorar_vector_multidimensional = TRUE;
```

Una opción para solucionar esto fue utilizar atributos sintetizados, pero con la recursividad se tornó bastante complejo ya que el valor de los tokens eran strings, por lo que finalmente se renunció a esta idea.

Luego se intentó modificar la gramática, como muestra a continuación

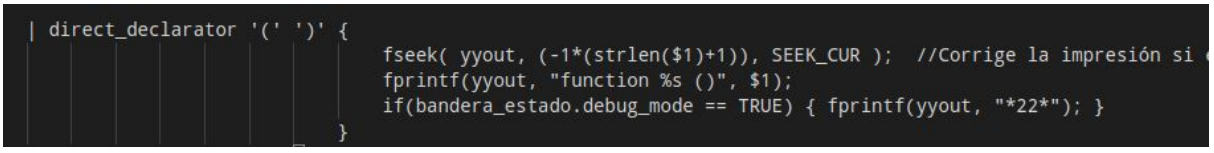


```
direct_declarator
    : IDENTIFIER    {
                        $$ = $1;    //Atributo sintetizado necesario
                        debug_mode(17);
                        if(bandera_estado.funcion_declarada == FALSE)
                        {
                            fprintf(yyout, "$%s", $1);
                        }
                    }

    | IDENTIFIER '(' ')'{
                            fprintf(yyout, "function %s ()", $1);
                            if(bandera_estado.debug_mode == TRUE) { fprintf(yyout, "*22*"); }
                        }
    | '(' declarator ')'
    | direct_declarator '[' { if (bandera_estado.ignorar_vector_multidimensional == FALSE) fprintf(yyout, "=array( ")
     ']' { bandera_estado.cerrar_parentesis_array =TRUE; debug_mode(18);}
    | direct_declarator '[' '*' ']'
    | direct_declarator '[' STATIC type_qualifier_list assignment_expression ']'
    | direct_declarator '[' STATIC assignment_expression ']'
    | direct_declarator '[' type_qualifier_list '*' ']'
    | direct_declarator '[' type_qualifier_list STATIC assignment_expression ']'
    | direct_declarator '[' type_qualifier_list assignment_expression ']'
```

Pero al modificar la gramática nos traía problemas con el reconocimiento de otras sentencias.

Finalmente se optó por parchear este problema retrocediendo en el archivo de escritura para borrar lo que escribió el caso base.



```
| direct_declarator '(' ')' {
                                fseek( yyout, (-1*(strlen($1)+1)), SEEK_CUR );  //Corrige la impresión si
                                fprintf(yyout, "function %s ()", $1);
                                if(bandera_estado.debug_mode == TRUE) { fprintf(yyout, "*22*"); }
                            }
```

Y así se encontró con innumerables problemas que se solucionaron de una u otra manera, a veces con parches no muy elegantes.