MINISTERIO DE
EDUCACIÓN
Y CULTURA

Universidad
Católica
"Nuestra Señora de la Asunción"

Universidad Católica

Facultad de Ciencias y Tecnología

Ingeniería Informática

Compiladores - Trabajo Práctico 2017.

Traductor C → PHP

- Gabriel Carballude.
- Orlando Martinez.

Loops

Entre los tres tipos de Loops que ofrece el Lenguaje C se implementó la sentencias de

1) While

2) For

Ya que es posible representar con cualquiera de estos loops la sentencia **do while,** no se implementó el mismo.
Como restricción adicional tenemos que toda bloque de sentencias dentro de un while o for, por más que tenga una sola sentencia, debe ir envuelto entre llaves { }.

```
while (c <= n){
  c = c+1;

}
 for(i=0; i<10; i++){
  continue;

}
```

```
while( $c <= $n ){
$c = $c + 1;
}
for( $i = 0;
$i < 10;
$i++ ){
continue;
}
```

Como observación, podemos ver que el **for** pierde su formato, pero eso es solo estetica, al ejecutar el .php no creará conflicto por ese motivo.

Estructuras Condicionales

Las estructuras implementadas fueron

1) If.

2) Case.

Como restricción tenemos que toda bloque de sentencias dentro de un
If o Case, por más que tenga una sola sentencia, debe ir envuelto entre llaves { }.

Como observación, esta implementado el **if else**, **default** pero por razones extrañas, no puede traducir, creando conflicto como la función **printf.**

```
if(a == 5){

    switch(grade) {
          case 'A' :
             a = a + 1;
             break;
          case 'B' :
             a = a - 1;
          case 'C' :
             a = a * 1;
             break;
          case 'D' :
             a = a + 2;
             break;

       }
    }
```

```
if( $a == 5 ){
switch( $grade ){
case 'A': $a = $a + 1;
break;
case 'B': $a = $a - 1;
case 'C': $a = $a * 1;
break;
case 'D': $a = $a + 2;
break;
}
}
```

Estructuras Anidada

Como se había mencionado en las secciones anteriores, existe la limitación de que todos los bloques de las sentencias if, case, while y for, deben estar envueltos entre llaves { }.

```
if(a){
  while (c <= n){
    result = result * c;
    c = c+1;

  }
}
```

```
if($a){
while( $c <= $n ){
$result = $result * $c;
$c = $c + 1;
}
}
```

Procedimientos y funciones

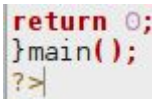
Se permite la definición de funciones globales (en C todas las funciones son globales y no se pueden definir funciones dentro de otras funciones) con valores de retorno según los tipos de datos permitidos, y con cualquier cantidad de argumentos de los tipos de datos permitidos.
No se permite el uso de prototipos de funciones debido a que su sintaxis demasiada flexible complica el guardado de datos en la tabla de símbolos y además podrían llevar a la necesidad de tener que actualizar dichos datos lo cual complica las verificaciones de tipos.
Se permiten definiciones del tipo:
tipo nombre ***tipo función(tipo param1 nombre param1, ...) { bloque de sentencias }***

```
int factorial(int n)
{
  int c;
  int result;
  result = 1;
  c = 1;

      while (c <= n){
        result = result * c;
        c = c+1;
   }

  return result;
}
int a(int de){
}

char b(int efe){
}
```

```
function factorial( $n ){
$c;
$result;
$result = 1;
$c = 1;
while( $c <= $n ){
$result = $result * $c;
$c = $c + 1;
}
return $result;
}
function a(  $de ){  }
function b(  $efe ){  }
```

Función Main

Si se encuentra dentro del código fuente la función main y si esta no recibe ningún parámetro y es la última función definida dentro del código en C entonces se agregara una llamada a main() al final del archivo de salida. Con el fin de que pueda ejecutarse dicho script posteriormente.



Comprobación de tipo

Se realizan dos clases de comprobaciones de tipo:

- Comprobación de tipo de expresiones: En una asignación se var = expresión se comprueba que los tipos de var y expresión sean compatibles.
  En una expresión del tipo expr1 operador expr2 ocurre lo mismo: Los tipos de expr1 y de expr2 deben ser compatibles con los tipos soportados por el operador.

- Comprobación de tipo de argumentos en llamada a función: Cuando tenemos un llamada a función de la forma func(expr1, expr2, expr3) se controla que la cantidad de argumentos que recibe func concuerda con la de su definición y que el tipo de cada expr concuerda con el tipo del parámetro.

- Se utilizan atributos sintetizados para ir calculando los valores de los no terminales a partir de los terminales.

- Si una expresión primaria deriva en una constante. El tipo de esa expresión se corresponde al tipo de dicha constante.

- Si una expresión primaria deriva en un identificador. Se busca dicho identificador dentro de la tabla de símbolos. El tipo del resultado de la búsqueda determina el tipo de la expresión. Si la respuesta es nula en cambio, significa que se está utilizando un identificador que no ha sido declarado. Es un error semántico.

Vectores

En Php, los vectores son los casi equivalentes de los vectores unidimensionales en C, solo que en php, no se necesita decir el tipo de dato, y declarar con =array antes de cualquier asignación del mismo.

```c
int vector_1[] = {5,1,0};
```

```php
$vector_1= array( 5 ,1 ,0 );
```

Reglas de ámbito

Se permite la declaración de variables globales como locales y los tipos permitidos son INT, FLOAT y CHAR.

Variables Locales

- Sólo pueden declararse al inicio de la definición de la función.
- Se puede declarar var1,var2; .

```c
int factorial(int n)
{
  int c,d;
```

```php
function factorial( $n ){
$c;
$d;
```

- Los parámetros con los que se definen una función se consideran variables locales de dicha función.
- Php no requiere que se declaren las variables (opcional).

Variables Globales

- Las variables Globales deben ser declaradas arriba del main, por problema de ambigüedad al realizar el parseo en bison.

```c
int aa, bb;

int main()
{
```

```php
$aa;
$bb;
 main(  ){
```

Detección y Recuperación de errores

Detección:

Mediante reglas gramaticales, podemos detectar errores como

```
int c[4:
```

Y especificando la linea problemática

```
syntax error: ERROR en la Linea 6 cerca de:  4
```

En el ejemplo, faltaría cerrar el corchete además del punto y coma al final.

Correction:

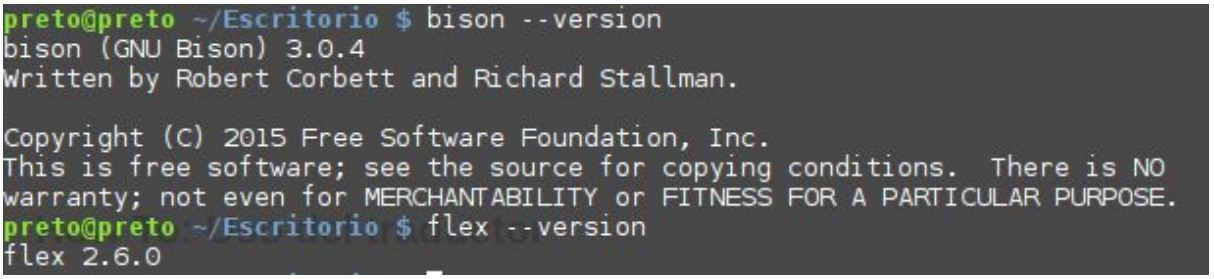
Mediante el no terminal reservado del bison error pudimos ignorar la entrada en caso de errores hasta ciertos caracteres de sincronización como el punto y coma .

```
int vector_1[] = {5,1,0}
```

```
$vector_1=array( 5 ,1 ,0 );
```

How To: Uso del traductor

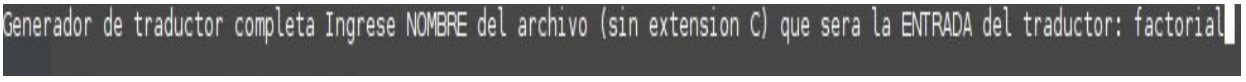
Para compilar y ejecutar el traductor es necesario tener un sistema operativo Unix o Linux con Flex y Bison instalados y usar una interfaz de línea de comandos.



Compilación y Ejecución

La compilación se realiza con un script de bash. Una vez ubicado dentro de la carpeta que contiene a los archivos y ejecute (dentro del directorio):
> bash compilar.sh

Luego aparecerá el siguiente mensaje

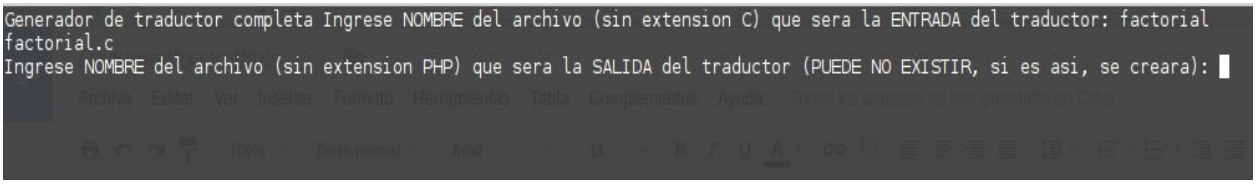


“Generador de traductor completa Ingrese NOMBRE del archivo (sin extension C) que sera la ENTRADA del traductor: ”

donde ingresamos el nombre del archivo .C para ser traducido. Si no encuentra el archivo, le aparecerá *“archivo inexistente, Favor INGRESE EL ARCHIVO C sin su extensión (.c) ”*

Y da la opción para volver a ingresar.

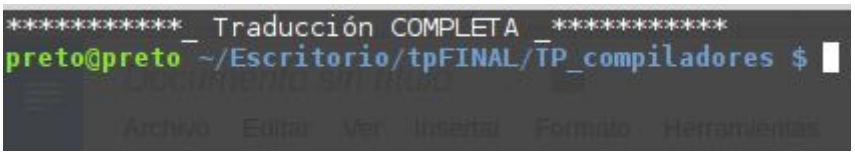
Una vez ingresado el nombre aparecera el siguiente mensaje.



“Ingrese NOMBRE del archivo (sin extension PHP) que será la SALIDA del traductor (PUEDE NO EXISTIR, si es asi, se creara): ”

Donde se debe introducir el nombre del archivo de salida, si existe, se sobrescribe y si no existe, se crea.

Si todo finalizo bien, se obtiene el siguiente mensaje.



Observación: Al ingresar los nombre **no introducir la extensión de los archivos.**

Limitaciones del Programa

Las secciones anterior se detalló implícitamente cuáles fueron las construcciones implementadas (y cómo fueron implementadas).
Las funcionalidades que tuvimos que dejar de lado por falta directa de equivalente con el lenguaje destino fueron:

- **Uso de punteros:** Todas aquellas declaraciones y operadores relacionados con los mismos. En este apartado se notan las diferencias en cuanto a nivel de abstracción entre C y Php.
- **Problema a la traducción del IF ELSE:** Esta implementadas las reglas, pero no determinó el problema del porque no se aplica los mismo.

(archivo Bison)

```
else_statement
                    : ELSE  { fprintf(yyoutput, "else"); } statement
                    | %prec IFX
                    ;
```

(archivo FLEX)

```
"if"            { return IF; }
"else"          { return ELSE; }
```

- Cualificadores como UNSIGNED, SHORT, LONG, CONSTANT, etc. Ya que en ese sentido la flexibilidad de Php nos permite tomar como representantes a los tipos de datos esenciales INT, FLOAT, CHAR.

- Structs y definición de nuevos tipos.

- Problema a la hora de la declaración de Matrices o array multidimensionales.

- **Directivas relacionadas con el preprocesamiento: include, define, enum etc. Nos mantuvimos al margen de traducir el código en C propiamente.**

*El archivo **factorial.c** (ejemplo) contienen el caso favorable, donde el traductor traduce correctamente.*

En la carpeta de entrega, existe archivos con prefijos error_ .c, el cual tienen sintaxis que nuestro traductor no procesa.