Organización de las Prácticas

Lenguajes de Programación y
Procesadores de Lenguajes
2017-18

V. 17.1

Objetivos y Organización

Objetivo:

Construir el front-end de un compilador completo para un lenguaje de programación de alto nivel sencillo pero no trivial

Partes:

- Parte I: Analizador léxico-sintáctico (entrega 29/10/17)
- Parte II: Analizador semántico (entrega 26/11/17)
- Parte III: Generador de código intermedio (16/1/18)

Fecha límite de entrega final:

Martes 16 de Enero de 2018

Evaluación del proyecto

Evaluación continua:

- Seguimiento en aula y laboratorio: 4% de la nota final
- Entregables: 3 %

Evaluación individual del proyecto:

30% de la nota final

1. PROYECTO APTO:
   - Detecta errores léxicos, sintácticos y semánticos.
   - Genera código intermedio correcto para pruebas
2. EXAMEN DE PRÁCTICAS
   - Individual en laboratorio
   - Tras examen final 16/1/2018 (recuperación 26/1/2018).
   - Ampliación y/o modificación del proyecto

Realización del Proyecto

Entorno de Desarrollo: Programación en C bajo GNU/Linux

Documentación de apoyo:

- Boletines disponibles en PoliformaT
- Manuales de Flex y Bison en PoliformaT

  Recursos/Material para Practicas/Material de Consulta

Procedimiento:

1. Probar los diferentes ejemplos de los boletines para entender los conceptos explicados
2. Realiza el ejercicio propuesto que forma parte del proyecto (Partes I, II y III)

Parte I: Analizador léxico-sintáctico

Objetivo:

Aprender a implementar analizadores léxico sintácticos usando las herramientas Flex y Bison.

Fecha límite entrega Parte I (A. Léxico-Sintáctico):

29/10/2017

Modo de entrega:

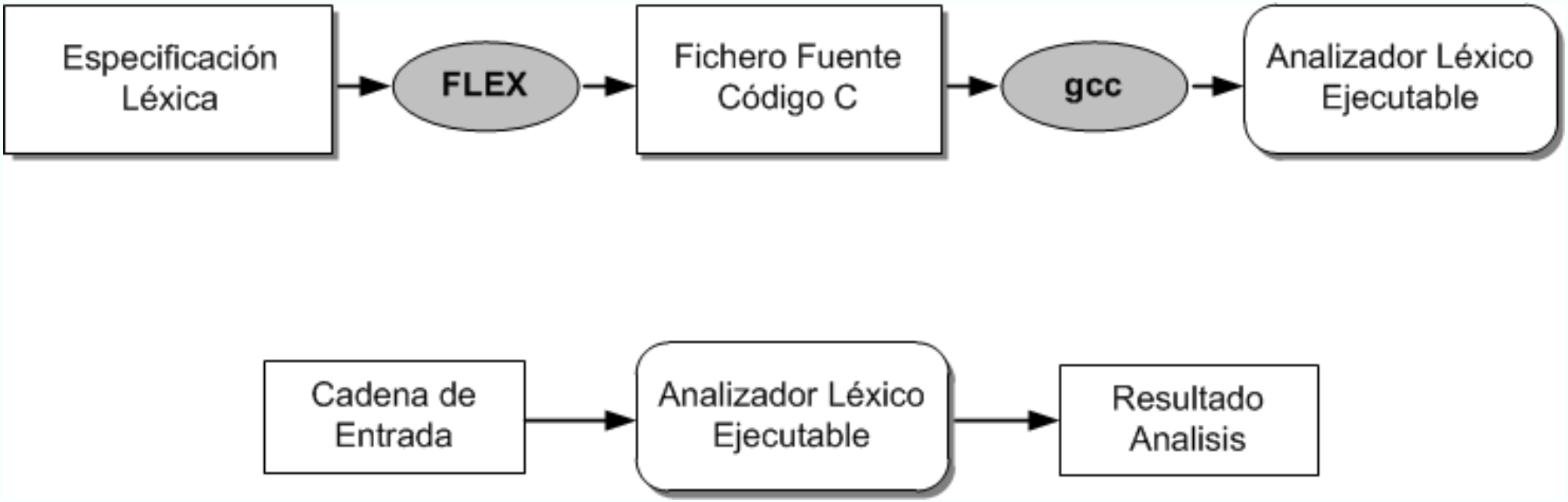
- Grupal
- A través de una tarea PoliformaT

5

Introducción a Flex

Lenguajes de Programación y
Procesadores de Lenguajes

Uso de Flex


Especificación Léxica
FLEX
Fichero Fuente Código C
gcc
Analizador Léxico Ejecutable
Cadena de Entrada
Analizador Léxico Ejecutable
Resultado Analisis

Especificación Léxica

Fichero de texto con extensión .l

Dividido en 3 partes (separadas por %%):

1. Definiciones: Declaraciones C y expresiones regulares

2. Reglas: Reglas del tipo "patrón – acción"

3. Funciones de usuario: Funciones C opcionales

Sección de definiciones

Preámbulo C:

```
%{
  #include <stdio.h>
%}
```

Definiciones Flex

<nombre> <definición>

Usadas posteriormente en reglas. Se expandirán por su valor. Referenciadas entre llaves.

```
digito    [0-9]
```


9

Sección de reglas

Contiene reglas de la forma:

<patrón> <acción>

- Las acciones son código C.
- Cada patrón acaba al final del primer carácter de espacio en blanco.
- El resto de la línea es la acción.

```
"+"        { printf(" más "); }
{digito}   { printf(" digito "); }
```

Sección código usuario

- Es opcional

- El código se copiará en el fichero `lex.yy.c‘ (nombre por defecto) .

- Se usa para incorporar funciones que llaman (o son llamadas) al (por el) analizador léxico.

```
main ()
{
  yylex ();
}
```

Lenguaje de expresiones regulares

x Casa con el carácter `x'

. Cualquier carácter excepto nueva línea

[xyz] Una clase de caracteres. En este caso casará con una `x', una `y', o una `z'

[abj-oZ] Una clase de caracteres que incluye un rango. En este caso casará con una `a', una `b', o cualquier letra entre la `j' y la `o', o una `Z'

[^A-Z] Una clase de caracteres negados. En este caso casará con cualquier carácter excepto una letra mayúscula.

[^A-Z\n] Cualquier carácter excepto una letra mayúscula o un carácter de nueva línea.

r* Cero o más ocurrencia de la expresión regular r

r+ Una o más ocurrencias de r

Lenguaje de expresiones regulares

r? Cero o una ocurrencia de r

r{2,5} De 2 a 5 ocurrencias de r.

{nombre} La expansión de la definición “nombre”

"[xyz]\"foo” La cadena literal : '[xyz]"foo'

\x Si x es una 'a', 'b', 'f ', 'n', 'r', 't', o 'v', se toma la interpretación típica de ANSI-C. Si no, se toma 'x' (se usa por ejemplo para indicar mediante '*' el carácter '*' que tiene su propio significado en Fle

rs La expresión regular r seguida de la expresión regular s (concatenación).

r| s r ó s (unión)

<<EOF>> Fin de fichero

int yylex()

- La función principal del analizador léxico es int yylex()
- Fichero de entrada: FILE *yyin
  Por defecto apunta a la entrada estándar: stdin.
- Fichero de salida: FILE *yyout
  Por defecto apunta a la salida estándar: stdout.
- Se pueden modificar asignando a yyin o yyout otro fichero.

  yyin = fopen(<fichero>,"r")

int yylex()

`char* yytext:`

Cadena que contiene el lexema analizado.

`int yyleng:`

Longitud de yytext

`ECHO:`

Copia yytext en el fichero de salida

Directivas:

`%option yylineno`

Mantiene en la variable yylineno el número de línea.

`%option caseless`

El analizador no distingue mayúsculas.

Uso de Flex y conflictos

- ¿ Qué ocurre si una secuencia de entrada casa con más de un patrón?

  Se escogerá la secuencia de caracteres más larga.

- ¿Y si tienen la misma longitud?

  Se seleccionará la regla que aparece antes en el fichero Flex.

Ejemplo (1/2)

```
%{
#include <stdio.h>
void visualizar(char* texto);
%}
%option yylineno
delimitador   [ \t\n]
letra         [A-Za-z]
digito        [0-9]
%%
{delimitador}                   { }
"print"                         { visualizar("Pal. reservada: "); }
"="                             { visualizar("operador: "); }
{digito}+                       { visualizar("constante: "); }
{letra}({letra}|{digito})*      { visualizar("id: "); }
.                               {visualizar("caracter invalido: "); }
%%
void visualizar(char* texto){
  printf("%3d-%s %s\n", yylineno, texto, yytext);}

int main (){
    printf("%2d.- ", yylineno);
     yylex (); }
```

Ejemplo (2/2)

Entrada:

```
print a
b = 53
@
print b
```

Salida:

```
1-Pal. reservada:  print
1-id:  a
2-id:  b
2-operador:  =
2-constante :  53
3-caracter invalido:  @
4-Pal. reservada:  print
4-id:  b
```

Compilación

Generación de código C

```
flex <fichero de entrada>   →   lex.yy.c
flex -o<fichero de salida> <fichero de entrada>

flex -oejemplo.c ejemplo.l   →   ejemplo.c
```

Generación de código objeto

Compilar con la biblioteca de Flex, por ejemplo en gcc : **-lfl**

Si **ejemplo.l** contiene el main()

```
gcc -o ejemplo lex.yy.c -lfl   →   ejemplo
```

En caso contrario:

```
gcc -c lex.yy.c                          →  lex.yy.o
gcc -o ejemplo main.c lex.yy.o -lfl      →  ejemplo
```