
Organización de las Prácticas

Lenguajes de Programación y
Procesadores de Lenguajes

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 (Individual): Analizador léxico-sintáctico.
Entrega 31/10/21
- Parte II (Grupos de 3 ó 4): Analizador semántico.
Entrega 28/11/21
- Parte III (Grupos de 3 ó 4): Generador de código intermedio.
Entrega 19/1/22

Fecha límite de entrega final (recuperación):

Martes 31 de Enero de 2022

Evaluación del proyecto

Evaluación continua (105):

- **Seguimiento en laboratorios (entregables): 6%**

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 (y tras examen de recuperación).
- Ampliación y/o modificación del proyecto

Realización del Proyecto

Entorno de Desarrollo: Programación en C bajo GNU/Linux

Documentación de apoyo:

- Boletines y manuales de Flex y Bison
- Disponibles en carpeta de red del DSIC y PoliformaT

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):

31/10/2021

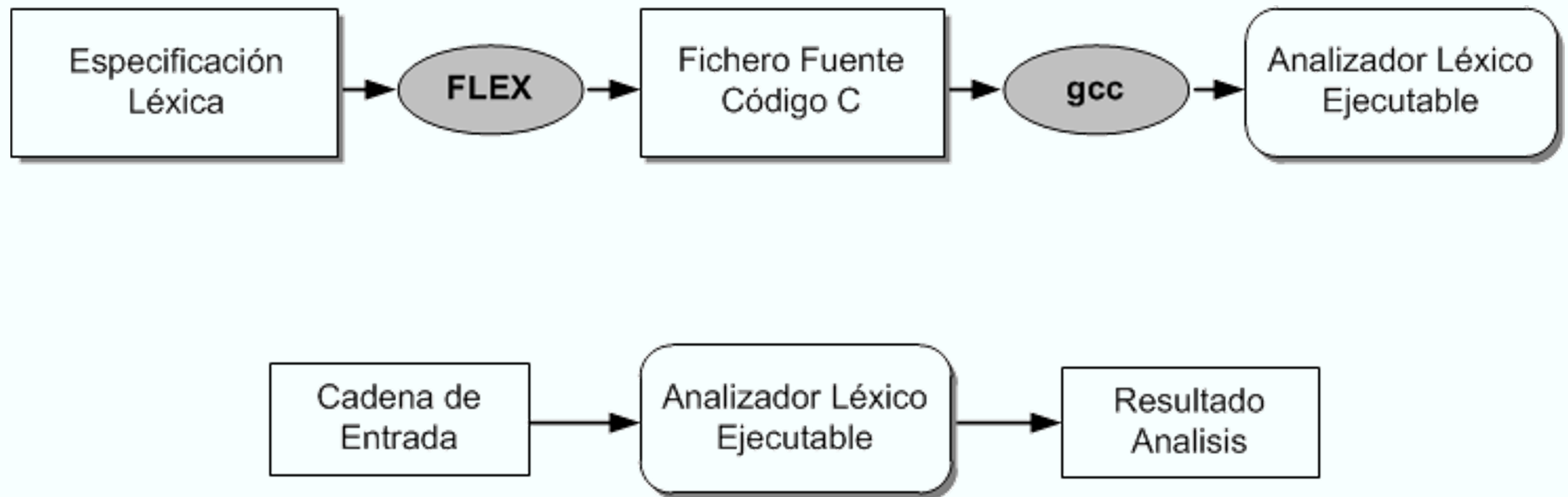
Modo de entrega:

- Individual (solo parte I)
- A través de una tarea PoliformaT

Introducción a Flex

Lenguajes de Programación y
Procesadores de Lenguajes

Uso de Flex



Especificación Léxica

Fichero de texto con extensión .l

Dividido en 3 partes (separadas por %%):

1. Cabeceras y definiciones: Declaraciones C y definiciones regulares

2. Reglas: Reglas del tipo “patrón – acción”

3. Funciones de usuario: Funciones C opcionales

Sección de cabeceras y definiciones

Preámbulo C:

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

Definiciones Flex

<nombre> <definición>

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

Ejemplo:

```
digito    [0-9]  
entero    {digito}+
```

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(" operador \n"); }  
{entero} { printf(" constante \n"); }
```

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 por el analizador léxico.

```
int main () {  
    yylex () ;  
    return 0 ;  
}
```

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 Flex

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")`

Fichero generado por Flex: `int yylex()`

`char* yytext:`

Cadena que contiene el lexema analizado.

`int yyleng:`

Longitud de yytext

Fichero de Flex

`%option yylineno`

Mantiene en la variable yylineno el número de línea.

`%option caseless`

El analizador no distingue mayúsculas.

ECHO:

Copia yytext en el fichero de salida

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 (1/3): alex.l

```
%{
#include <stdio.h>
#include "header.h"
#define retornar(x) {if (verbosidad) ECHO;}
}%

%option yylineno
delimitador    [ \t\n]
digito          [0-9]
entero          {digito}+
%%

{delimitador}    {if (verbosidad) ECHO ; }
"+"              { retornar (MAS_) ; }
"-"              { retornar (MENOS_) ; }
"*"              { retornar (POR_) ; }
"/"              { retornar (DIV_) ; }
"("              { retornar (OPAR_) ; }
")"              { retornar (CPAR_) ; }
{entero}          { retornar (CTE_) ; }
.                 { yyerror("Caracter desconocido"); }
%%
```

Ejemplo (2/3): alex.l

```
int verbosidad = FALSE;

void yyerror(const char *msg){
    fprintf(stderr, "\nError en la linea %d: %s\n", yylineno, msg);
}

int main(int argc, char **argv) {
    int i, n=1 ;

    for (i=1; i<argc; ++i)
        if (strcmp(argv[i], "-v")==0) { verbosidad = TRUE; n++; }
    if (argc == n+1)
        if ((yyin = fopen (argv[n], "r")) == NULL)
            fprintf (stderr, "El fichero '%s' no es valido\n", argv[n]) ;
        else yylex ();
    else fprintf (stderr, "Uso: cmc [-v] fichero\n");
    return (0);
}
```

Ejemplo (3/3): header.h

```
#define TRUE 1
#define FALSE 0

extern int yylex() ;

extern FILE *yyin;          /* Fichero de entrada */
extern int yylineno;        /* Contador del numero de linea */

extern void yyerror(const char * msg) ;
    /* Tratamiento de errores */

extern int verbosidad ;
```

Compilación

Generación de código C

`flex -o<fichero de salida> <fichero de entrada>`

Ejemplo:

`flex -oalex.c alex.l → alex.c`

Generación de código objeto

Compilar todos los fuentes con la biblioteca de Flex: `-lfl`

Ejemplo:

`gcc -o cmc alex.c -lfl → cmc`

Automatizado con *makefile* (opción recomendada)

`make → cmc`