

Generador de analizadores léxicos Flex

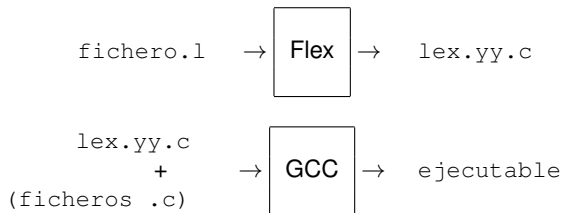
Procesadores de Lenguajes
3º Grado en Informática

Fco. José Ribadas Pena & Víctor M. Darriba Bilbao

Área de Ciencias de la Computación e Inteligencia Artificial

Marzo 2020

- Flex: traduce la especificación de un analizador léxico a un programa C que lo implementa
 - Analizador léxico descrito/especificado mediante *expresiones regulares (ER)*
 - A las ER se les pueden asociar *acciones* (código C)
 - Cada vez que el analizador encuentra una secuencia que encaja con una de las ER, ejecutará la acción asociada



- 1 Compilar la especificación del analizador y crear el fichero `lex.yy.c` con el código del autómata

```
$ flex fichero.l
```

- 2 Compilar el analizador C y generar el programa ejecutable

- 1 Enlazar con librería de Flex (proporciona implementaciones por defecto para `yywrap()` y `main()`)

```
$ gcc lex.yy.c -lfl
```

- 2 Compilar y enlazar con ficheros `.c` del usuario

```
$ gcc lex.yy.c (ficheros .c)
```

- Usuario proporciona implementaciones para `main()` e `yywrap()`
- Deberá llamar a la función `yylex()` que reconocerá un TOKEN por cada llamada.

Funcionamiento de Flex

- `lex.yy.c` contiene

	tablas del autómata generado
	función <code>int yylex(void)</code>
- `yylex()` simula el analizador especificado y sirve de interfaz con el código de usuario
 - `yylex()` debe de ser llamada desde el código de usuario

Funcionamiento:

- En cada llamada, `yylex()`:
 - 1 lee caracteres de la entrada hasta que concuerdan con una ER
 - 2 almacena el texto que ha concordado en la variable `yytext`
 - 3 ejecutan las acciones asociadas al patrón de esa ER
- Acciones:
 - Puede ser simplemente el procesamiento del texto concordante con la ER para enviarlo de nuevo a la salida.
 - En otras ocasiones podrán suponer la alteración de variables del código de usuario y la devolución a la rutina que llama a `yylex()` (mediante `return`) de un valor numérico que identifique al TOKEN reconocido

Partes de una especificación Flex

- Tres partes separadas por el símbolo % %
- Las dos primeras son obligatorias, aunque pueden estar vacías

< sección de declaraciones >

% %

< sección de reglas y acciones >

% %

< sección de rutinas de usuario >

Sección de declaraciones (I)

- Código C necesario para las acciones asociadas a los patrones
 - El código C irá entre `%{ y %}`, será copiado tal cual al principio de `lex.yy.c`
 - Generalmente serán `#include`, `#define` y estructuras o variables del código de usuario afectadas por las acciones
- Definición de macros
 - Asocia un "alias" a expr. regulares usadas en la sección de reglas (mejor legibilidad)
 - En las reglas se referencia ese "alias" poniéndolo entre llaves

Definición: `LETRA_MAYUSCULA [A-Z]`
 `DIGITO [0-9]`

Uso (en reglas): `{LETRA_MAYUSCULA} {DIGITO}`

- Definición de condiciones de arranque (*entornos de reconocimiento*)
 - Entornos dentro de los cuales se podrán reconocer subconjuntos de ERs
- Definición: `%start NOMBRE1 NOMBRE2 ... /* alternativamente %s */`
 `%x NOMBRE1 NOMBRE2 ...`

- Opciones del analizador
 - Se puede usar como alternativa a la línea de comandos de Flex

Definición: `%option [OPCION1] [OPCION2] ...`

- Formato:

```
[ER 1] [acción 1]
....
[ER n] [acción n]
```

- Cada par ([ER], [acción]) recibe el nombre de regla

- Flex espera el comienzo de la ER al principio de la línea (no indentar)
- La ER se separa de la acción con espacios o tabuladores

- Acciones:

- Si la acción involucra varias instrucciones C en más de una línea, irá entre llaves
- Si n ERs comparten la misma acción se indica con | en las $n-1$ primeras

```
[ER 1] |
```

```
...
```

```
[ER n] [acción común]
```

- Si no se especifica ninguna acción, se aplica la acción por defecto `ECHO` (copia el contenido de `yytext` a la salida)
 - La acción vacía (no hacer nada) se escribe como `' ; '`
- Por defecto, los caracteres de la entrada que no concuerdan con ninguna ER también se copian a la salida

Sección de rutinas de usuario

En esta sección se puede escribir código C adicional

- Funciones llamadas desde las acciones de las reglas
- En programas sencillos se suele incluir en esta sección las funciones `main()` y `yywrap()` definidas por el usuario

Función `int yywrap()`: se ejecuta cada vez que se alcanza el final de la entrada

- Permite manejar múltiples ficheros de entrada
- Devuelve 1 para indicar que no quedan ficheros de entrada por procesar y que el procesamiento ha terminado (0 en caso contrario)
- Implementación por defecto (opción `-lfl` de `gcc`)

```
int yywrap() {  
    return(1);  
}
```


Expresiones regulares en Flex (I)

Expresiones simples:

<code>c</code>	el carácter 'c'
<code>\a, \b, \f, \n, \r, \t, \v</code>	símbolos especiales de ANSI-C
<code><<EOF>></code>	fin de fichero
<code>.</code>	cualquier carácter excepto salto de línea (' <code>\n</code> ')
<code>[abc]</code>	cualquier carácter del conjunto ('a', 'b' ó 'c')
<code>[^abc]</code>	cualquier carácter excepto los del conjunto
<code>[a-z]</code>	cualquier carácter del rango indicado
<code>[^a-z]</code>	cualquier carácter excepto los del rango
<code>[a-z]{-}[aeiou]</code>	diferencia de clases de caracteres (consonantes en minúsculas)
<code>"xxxx"</code>	la cadena <code>xxxx</code> reconocida de forma literal
<code>". "</code>	el carácter '.'
<code>{ALIAS}</code>	expande la ER asociada la macro <code>ALIAS</code>

Expresiones regulares en Flex (II)

Operadores (de mayor a menor precedencia):

R^*	reconoce 1 ó más repeticiones de R (cierre reflexivo-transitivo)
R^+	reconoce 0 ó más repeticiones de R (cierre transitivo)
$R?$	reconoce 0 ó 1 ocurrencia de R (opcional)
$R\{n\}$	reconoce n repeticiones exactas de R
$R\{n, m\}$	reconoce de n a m repeticiones de R
(R)	agrupa expresiones regulares
RS	reconoce la concatenación de R y S
$R S$	reconoce o R o S
R	reconoce la expr. R si está al inicio de línea
$R\$$	reconoce la expr. R si está al final de línea

Siendo R y S expr. regulares

Expresiones regulares en Flex (III)

R/S	reconoce R sólo si lo que sigue a continuación encaja con S (S define el contexto donde reconocer R)
$\langle \text{UNO} \rangle R$	reconoce R si la condición de arranque UNO está activa
$\langle \text{UNO}, \text{DOS} \rangle R$	reconoce R si una de las condiciones de arranque UNO o DOS están activas
$\langle * \rangle R$	reconoce R si alguna de las condiciones de arranque definidas en el analizador están activas

Modificadores:

$(?a:R)$	$i = R$ insensible a mayúsculas/minúsculas,
$(?-a:R)$	$a, b \in \{i, s, x\}$ $s = '.'$ incluye el salto de línea (' $\backslash n$ ') $x =$ ignora espacios en blanco y comentarios en R

Para leer los operadores como caracteres:

(comillas)	"."	"*"	"+"	"\$"	"["	"]"	"("	")"	"{"	"}"
(escapados)	$\backslash .$	$\backslash *$	$\backslash +$	$\backslash \$$	$\backslash [$	$\backslash]$	$\backslash ($	$\backslash)$	$\backslash \{$	$\backslash \}$

Uno de los mayores problemas en el análisis de lenguajes formales

- En Flex:
 - Existencia de varias concordancias simultáneas entre el texto y ERs
- Solución (heredada de Lex):
 - Seleccionar una de las ERs posibles

Mecanismo de resolución de ambigüedades en Flex

- 1 Si la entrada concuerda con más de una ER se elige la que abarca el mayor núm. posible de caracteres
 - Flex es voraz (greedy) y ese comportamiento **no se puede cambiar**
- 2 Si más de una ER concuerda con el mayor número de caracteres, se elige la que aparezca primero en la especificación Flex
 - El orden de las reglas puede tener importancia
 - Heurística: escribir primero las reglas con las ERs más específicas

Condiciones de arranque (entornos de reconocimiento) (I)

Usadas para definir mini-analizadores:

- Reglas que se usan sólo cuando se cumplen determinadas condiciones

Se definen en la zona de declaraciones:

- Condiciones de arranque inclusivas (`%start, %s`)
 - También permiten usar las reglas no asociadas a ninguna condición de arranque
- Condiciones de arranque exclusivas (`%x`)
 - Sólo permiten usar las reglas asociadas a la condición de arranque

Se cambian en las acciones de las reglas:

- Durante la ejecución del analizador siempre hay una condición activa
- `INITIAL`: condición de arranque por defecto
- `BEGIN`: macro usada para cambiar a otra condición de arranque
 - `BEGIN (INITIAL)` o `BEGIN (0)` sirve para volver al entorno por defecto

Condiciones de arranque (entornos de reconocimiento) (II)

Ejemplo: eliminación de comentarios multilínea en C

- Solución obvia: una expresión regular con la acción vacía

```
"/*" (.|\n) "*"*/" ;
```

Debido a la voracidad de Flex, funciona “demasiado” bien

- Elimina todo el texto desde el primer "/" al último "*"
- Una solución mejor: usar una condición de arranque exclusiva

```
%x COMENTARIO /* exclusiva: no queremos usar otras reglas */  
...  
%%  
...  
"/*" BEGIN(COMENTARIO); /* entramos en modo comentario */  
<COMENTARIO> .|\n ; /* nos comemos el comentario */  
<COMENTARIO> "*" BEGIN(INITIAL); /* volvemos al analizador */  
...
```

Se regula a través de dos variables de tipo puntero a fichero (`FILE *`)

- `yyin`: entrada desde la que el analizador lee caracteres
 - Por defecto la entrada estándar `STDIN` (terminal)
- `yyout`: salida a la que escribe el analizador
 - Por defecto la salida estándar `STDOUT` (terminal)
- Se pueden redireccionar a archivos usando la función de C `fopen()` en el código de usuario (generalmente en el `main()`):

```
yyin = fopen("entrada.txt", "r");  
yyout = fopen("salida.txt", "w");
```

Las funciones de C que podemos usar para escribir en la salida del analizador cambian según a donde apunte `yyout`:

- Si apunta a `STDOUT`, podemos usar `putchar()` o `printf()`
- Si redireccionamos a un archivo, usaremos `fputc()` o `fprintf()`

Variables Flex accesibles desde el código de usuario

- `yytext`: Puntero a una cadena de caracteres que contiene la última cadena de texto que encajó con una ER
 - Por defecto, está declarada como `char *yytext`
 - Su contenido sólo es estable dentro de las reglas y entre llamadas consecutivas a la función `yylex()`
- `yylen`: Entero con la longitud de la cadena `yytext`

Otros elementos predefinidos (II)

Opciones del analizador

- `case-insensitive`: hace que el analizador ignore mayúsculas y minúsculas
- `header-file=" [nombre] "`: hace que flex genere un archivo de cabecera en C (`.h`) para el analizador con el nombre indicado
- `output-file=" [nombre] "`: el archivo fuente del analizador se llamara `[nombre]`, en lugar de `lex.yy.c`
- `yylineno`: le dice al analizador que guarde el número de línea de la posición actual en la entrada, en la variable global entera `yylineno`
- `yywrap`: hace que `yylex()` omita la llamada a `yywrap()` al llegar al final de la entrada, terminando la ejecución del analizador
- `main`: hace que Flex proporcione una versión por defecto de `main()`, con una llamada a `yylex()`. Esta opción incluye a la opción `yywrap()`
- `c++`: ordena generar el analizador en C++, en lugar de en C.
- `array`: hace que `yytext` sea definida como array, en lugar de como `char *`

Otros elementos predefinidos (III)

Funciones y macros predefinidas

- `REJECT`: Rechaza la concordancia entre la ER actual y el texto almacenado en `yytext` y pasa a la siguiente regla que encaje (si la hay)
- `YY_START`: Devuelve el valor entero correspondiente a la condición de arranque actual
- `yymore()`: Le dice al analizador que en la próxima concordancia con una ER, en lugar de reemplazar el contenido actual de `yytext`, concatene la nueva cadena a la actual
- `yyless(n)`: Devuelve todos los caracteres en `yytext` a la entrada, excepto los n primeros
- `unput(c)`: mete el carácter `c` en la posición actual de la entrada
- `input()`: lee el siguiente carácter de la entrada, avanzando una posición en la misma
- `yyterminate()`: termina la ejecución del analizador, devolviendo 0