INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIO DE CÓMPUTO

ANALIZADOR LÉXICO

FLEX

Introducción

- Lex
  - Generador de programas (C)
  - Procesos léxico de cadenas de caracteres
  - No es lenguaje sino un generador para C
- Convierte expresiones y acciones del usuario en un programa yylex en c
- Internamente va a actuar como un autómata
- Una vez reconocida la cadena representada por el autómata ejecutara el código asociado a la regla
- 1975 surge el concepto de generador automático de analizadores léxicos

FLEX

- Herramienta para generar escáneres (programa que reconoce patrones léxicos)
- Lee ficheros de entrada (dados o estándar)
- Descripción se encuentra en forma de parejas de expresiones regulares y código en C (REGLAS)
- Salida ‘lex.yy.c’ con rutina yylex()
- En la ejecución, busca las expresiones regulares. Ya que la encuentra ejecuta el correspondiente código C

Paso de Compilación

- 1. Compilar la especificación del analizador y crear el fichero yy.lex.c con el código del autómata
  - $ flex fichero.1
- 2. Compilar el analizador C y generar el programa ejecutable
  - Enlazar con librería de FLEX (implementaciones por defecto para yywrap() y main())

    $gcc lex.yy.c –lfl
  - Compilar y enlazar los fichero .c

    $ gcc lex.yy.c (ficheros.c)
  - Se debe implementar 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 int yylex(void)
    - Deberá ser llamada por el código del usuario
    - Simula el analizador especificado y sirve como interfaz con el código de usuario
    - En cada llama tomara caracteres de la entrada hasta que mache una de las expresiones regulares de la especificación. Entonces, se almacenará el texto que ha macheado la expresión regular en la variable **`yytext`** y se ejecutarán las acciones asociadas al patrón.

Funcionamiento de FLEX

- Las acciones podrán ser simplemente el procesamiento del texto macheado y 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()` de algún tipo de dato por medio de RETURN (generalmente será un valor numérico que identifique la TOKEN encontrado).

Partes de Especificación de 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

- Código C necesario para las acciones asociadas a los patrones
  - El código C irá entre `%` y `%,` será copiado tal cual a `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 expresiones regulares usadas en la sección de reglas (mejoran la 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}
```

Sección de declaraciones

- Definición de entornos de reconocimiento *(start conditions)*
  - Entornos dentro de los cuales se podrán reconocer sub-expresiones dependiendo del contexto
  - Permite generar ”mini-analizadores” dentro del analizador

Definición: `%start NOMBRE`

Uso:

```
[patron] {BEGIN(NOMBRE)} /*inicio
                    entorno*/
```

(en reglas)

```
<NOMBRE>[sub-patron] {[accion]}
<NOMBRE>[sub-patron] {BEGIN(INITIAL)}
    /*fin entorno*/
```

Sección de reglas

- Esta sección tiene el siguiente formato:

```
(exp reg 1 ) (acción 1 )
....
....
(exp reg n ) (acción n )
```

  Cada par (exp. reg., acción) recibe el nombre de regla

- NOTA: El primer carácter de la expresión regular debe de comenzar en la primera columna de texto. (de lo contrario se consideraría un trozo de código C)

- Cuando la acción involucra varias sentencias C es necesario encerrarlas entre llaves

Sección de reglas

- Cuando varias expresiones regulares comparten la misma acción se utilizará el símbolo `|` . Indica que se debe tomar como acción para esta regla la misma que se especifique para la siguiente expresión regular.

  Ej.:

```
exp reg1 |
exp reg2 |
. . .
. . .
exp regN acción común
```

- Si no se especifica ninguna acción se terminará la regla con punto y coma ";" En este caso se considera como una acción vacía y por defecto se copiará directamente el `string` macheado en la salida.

Sección de reglas

- **Manejo de ambigüedades**
  - Si la entrada machea más de 1 expresión regular se elegirá la expresión que machee mayor número de caracteres
  - Si el número caracteres macheados es el mismo, se elige a la que aparezca primero en el fichero FLEX

Sección de rutinas de usuario

- Esta sección zona se puede escribir código C adicional, bien funciones llamadas desde las acciones de las reglas o, en el caso de programas pequeños, suelen incluirse las funciones `main()` y `yywrap()` propias del usuario
- **Nota:** La función `int yywrap()` se ejecuta cada vez que se alcanza el final del fichero de 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
  - Implementación por defecto

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

Expresiones Regulares en FLEX

- Expresiones simples

`c`	reconoce el carácter `'c'`
`.`	cualquier carácter excepto salto de linea (`'nn'`)
`[abc]`	cualquier carácter del conjunto (`'a','b'` ó `'c'`)
`[^abc]`	cualquier carácter excepto los del conjunto
`[a-z]`	cualquier carácter del rango indicado
`[^a-z]`	cualquier carácter excepto los del rango
`"xxxx"`	reconoce la cadena indicada de forma literal
`{ALIAS}`	expande la expr. reg. asociada al alias indicado (definido en sec. declaraciones)

Expresiones Regulares en FLEX

- Operadores (de mayor a menor precedencia)

`R*`	reconoce 0 ó más repeticiones de `R`
`R+`	reconoce 1 ó más repeticiones de `R`
`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 linea
`R$`	reconoce la expr. `R` si está al final de linea

Siendo `R` y `S` expresiones regulares

Expresiones Regulares en FLEX

`R/S` reconoce `R` sólo si lo que sigue a continuación encaja con `S` (`S` define el contexto donde reconocer `R`)

`<AAA>` reconoce la condición de arranque `AAA` (normalmente la acción asociada será `{ BEGIN(AAA) }` para indicar a FLEX que entre en el contexto de esa condicion de arranque)

`<AAA>R` reconoce `R` si se está dentro del contexto de la condición de arranque `AAA`

- Algunos símbolos de escape:

`\n` salto de línea

`\t` tabulador

`\\` barra invertida

`\+; \*; \^; \$`

`\[; \]; \(; \); \{; \}`

Variables y funciones predefinidas

- 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 expresión regular. Está declarada como `char`
  - **`yytext:`** (puede modificarse pero no es aconsejable) Su contenido sólo es estable dentro de las reglas y entre llamadas consecutivas a la función `yylex()`
  - **`yyleng`**: Contiene la longitud de la cadena `yytext`. Tampoco se debe modificar

Variables y funciones predefinidas

- **`yyin`**: Variable declarada de tipo `FILE *`. Contiene un puntero al fichero del que lee caracteres el reconocedor. Por defecto se corresponde con la entrada estándar. Será necesario cambiarla para que el analizador lea de un fichero distinto.

- **`yyout`**: Variable declarada del tipo `FILE *`. Contiene un puntero al fichero estándar en el que escribe el analizador léxico al utilizar la acción `ECHO`. Se puede cambiar libremente.