Lex

- Es un utilitario que permite generar el código C de programas que manipulan y transforman texto. En particular es útil para genera el escáner de un compilador.
- Toma una especificación como entrada, que ya describiremos, y genera como salida un fuente en lenguaje C con el código del escáner
- Lex está discontinuado, la herramienta que se usa hoy dia se llama flex.
- Flex tiene algunas incompatibilidades menores con lex

Flex

- La especificación de flex tiene el siguiente formato:

```
definiciones
   %%
   reglas
   %%
   código del usuario
```

- La sección de definiciones sirve para dar nombres a ciertas ER y así simplificar y hacer más claras otras ER.
- La sección de reglas se delimita con %% y es donde indicamos mediante ER los tokens a reconocer y mediante código C las acciones a realizar cada vez que se reconoce un token.

Flex

- La sección de código de usuario es optativa. En caso de no estar, el segundo delimitador %% no es necesario. Sirve para agregar código que flex copiará tal cual al final del fuente que genere.
- Puede haber una función main que típicamente haga al menos un llamado a yylex() o bien puede haber funciones auxiliares a ser invocadas por la acciones de las reglas

Variables y Funciones del Fuente Generado

- **yylex()** es el escáner. Devuelve el token encontrado como un int.
- **yytext** variable con el lexema encontrado, en modo flex es un char * (solo accesible desde flex, se debe copiar en un registro semántico para ser usado en bison)
- **yyleng** largo de yytext
- **yyin** es el FILE * de donde el escáner lee la entrada, por defecto stdin
- **yyout** es el FILE * a donde el escáner dirige su salida, por defecto stdout

Definiciones

- La Sintaxis es:
  - nombre definición
  - Nombre debe estar al comienzo de la línea y comenzar con una letra o un guión bajo, continuando luego con más letras, guiones bajos, guiones del medio (menos) o dígitos.
  - Definición se extiende desde el primer carácter no blanco luego del nombre hasta el fin de la línea, siendo una ER que eventualmente puede usar nombres definidos previamente.
- Ejemplo:
  - `digito [0-9]`
  - `exponente [eE][+-]?{digito}+`
- Uso en las reglas
  - `{digito}+   {nro++; //esta es la accion}`

Definiciones

- Para usar un nombre lo coloco entre llaves. Se reemplaza {nombre} con (definición) siendo los paréntesis para evitar problemas de precedencia
  - Si la definición del nombre incluye `^` o `$` para indicar principio o fin de línea, la expansión es sin paréntesis para preservar el significado
    - En esto se diferencia de lex (que nunca pone paréntesis)
  - No puede usarse en definición: `<ci>` (condición inicial), `<<EOF>>`, `/` (operador de contexto)

Otros en Definiciones

- Se copia textualmente al fuente resultante:
  - Comentarios C no indentados
  - Texto indentado (código C)
  - Bloques de texto encerrados entre `%{` y `%}` estos delimitadores deben estar NO indentados y solos en su línea
  - Un bloque top, que se delimita con `%top{` y `}` , es similar al anterior pero se asegura que irá al tope del fuente generado. Si hay varios se respeta el orden entre ellos. Los includes propios suelen ir aquí

Opciones

- Opciones, pueden ir en la zona de definiciones o en la línea de comando que invoca a flex
  - `%option header-file="archivo.h"`
    - nombre del .h a generar
    - Comando: --header-file=FILE
  - `%option outfile="archivo.c"`
    - nombre del fuente a generar
    - Comando: -oFILE, --outfile=FILE
  - `%option yylineno` indica que mantega el número de línea (en la variable yylineno)
  - %option noinput %option nounput para evitar warnings

Reglas, sintaxis

- La Sintaxis es:
  - `patrón acción`
    - **Patrón** es una ER que no debe estar indentada y se separa de la acción por el primer blanco (espacio o tabulador no encerrado entre comillas o con barra invertida delante).
    - **Acción** es código C que debe comenzar en la misma línea. Si es una sentencia compuesta puede extenderse a las líneas siguientes.
      - También puede ser algunas de las macros provistas por flex

Reglas, emparejamiento

- Cada vez que ejecuta el escáner trata de emparejar el texto entrante con **todos** los patrones. Si puede lograr emparejar con más de una patrón el orden de precedencia es:
  - El patrón **más largo**, es decir el que tiene más caracteres
    - Ejemplo si tengo en patrón **`for`** (que solo encuentra la cadena for) y el patrón **`[a-z]+`** y en la entrada el escáner lee **forever** aplica la acción correspondiente al segundo patrón.
  - Si dos patrones emparejan con la misma cantidad de caracteres, aplica el que se haya **definido primero**.

Código adicional en Reglas

- Si después del inicio de la sección de reglas, marcada por %% y antes de la primer regla se coloca código indentado o entre %{ y %} (estos delimitadores NO identados) dicho código se coloca al inicio de la rutina del escáner.
- Así es posible declarar variables locales a la rutina del escáner y/o código que se ejecutará siempre al inicio de la rutina dicha rutina
- Si agrego código después de la primer regla no está definido donde será colocado y suele terminar dando errores de compilación (evitar, está no más porque posix lo pide)

Patrones

- Las ER que usa flex son muy similares a las del programa egrep. Coincidencias:

Operador	Comentario
.	Cualquier carácter menos \n y EOF
\t \n \0123 \x2a … etc	Secuencias de escape propias de C
\+ * \. \(… etc	Secuencias de escape para los operadores de ER
|	Pipe para unión
* +	Clausura de Kleene, clausura positiva
?	0 o 1 repetición
[ ] [^] [-]	1 carácter del conjunto
{n} {n,m} {n,}	Potencia
( )	Paréntesis para modificar precedencias
^ $	Al inicio o al fin de la línea
[:digit:] [:^digit:] … etc	Como en isdigit de ctype.h y su complemento. Vale para otros isXXX (debe volver a encerrarse entre [])

Patrones

- Algunas particularidades de flex:

Operador	Comentario
<<EOF>>	Empareja con EOF (si no está al encontrar EOF devuelve cero)
r/s	ER r siempre y cuando esté seguida inmediatamente por la ER s. Se empareja y devuelve r pero se toma el largo de rs para seleccionar que patrón seleccionar si más de uno empareja.
{-}	Para restar conjuntos: [a-z]{-}[xf] encuentra cualquier minúscula salvo x o f
{+}	Similar al anterior pero hace unión de conjuntos
<ci>r	Encuentra r solo si se está en Condición Inicial ci
<ci1,ci2,ci3>r	r si la condición inicial es ci1 o ci2 o ci3
<*>r	r en cualquier condición inicial
"." "a+" … etc	Lo que encierro entre comillas dobles es literal, por ejemplo una a seguida de un +, y no la clasura positiva de a

Acciones

- Indican que hacer cuando se encuentra el patrón asociado a las mismas
  - Ejemplo: `\n caracteres++; lineas++;`
  - Al encontrar un \n incrementa dos contadores
- Puede ser la sentencia nula o un comentario para indicar que no se hace nada, simplemente se descarta el patrón encontrado
- Si no se puede emparejar ningún patrón se aplica la acción por defecto, esto es tomar el primer carácter de entrada y copiarlo tal cual en la salida
- Para copiar un patrón emparejado, tal cual en la salida se usa la directiva ECHO (seguido de un ;)
- Si se usa un | (pipe) como acción lo que se está indicando es que debe realizar la misma acción que el patrón siguiente.

Código de Usuario

- Es código que se agregará al final de fuente generado por flex
- Hay dos casos típicos
  - Agregar una rutina main que llama al escáner. Es el caso cuando se quiere hacer un programa que procese un texto, para modificarlo y/o hacer estadísticas sobre el mismo.
  - Agregar rutinas auxiliares que puedan ser invocadas desde las acciones, por supuesto habrá que declararlas antes, por ejemplo, agregando código en la sección de definiciones

Condiciones Iniciales

- Permiten definir “sub-escáneres”, es decir cambiar el modo de escaneo. Es útil cuando las reglas cambian, por ejemplo en comentarios o en literales cadena.
- Se los declara en la sección de definiciones con %s si es una condición inclusiva o %x si es una condición exclusiva
  - Ejemplos:
    - `%s CODIGO DATOS`
    - `%x CADENA`
- Hay una condición predefinida llamada INITIAL cuyo valor es cero, si una regla no tiene condición inicial, tiene implícitamente la condición INITIAL

Condiciones Iniciales

- Para activar una condición inicial se usa en la acción la directiva BEGIN(condición)
- La condición queda activa hasta que se activa otra en su reemplazo, por ejemplo BEGIN(INITIAL)
- Cuando una condición exclusiva está activa solo las reglas con esa condición se toman en cuenta para emparejar patrones.
- Cuando una condición inclusiva está activa se consideran la acciones con esa condición y las acciones sin condición.

Uso Elemental

- Si mi archivo de configuración se llama prueba.l armo el fuente C con
  - `flex prueba.l`
- Compilo teniendo en cuenta de agregar la biblioteca de lex
  - `gcc -o prueba prueba.c -lfl`
- Ejecuto usando el achivo prueba.txt como entrada
  - `./prueba <prueba.txt`

Licencia