Practica No 3

Introducción

Debido a que lex solo retorna valores enteros, es necesario tener una manera de almacenar los valores de los tokens reconocidos en un archivo fuente para este fin se utiliza la variable yylval. La variable yylval guarda el valor del token actual, entonces esta variable se actualizara cada vez que se reconozca un nuevo token.

Otra de las características importantes de lex es el uso de los estados léxicos, estos se usan para definir que expresiones regulares están activas en diferentes momentos de la ejecución del programa; dentro de las acciones léxicas a ejecutar se definen las transiciones de un estado a otro, lo cual da mayor flexibilidad al programador para manipular el autómata generado por lex.

Desarrollo

1. Copiar el código proporcionado en un archivo llamado valor.h

Debido a que no se esta usando yacc se debe declarar en un archivo de cabecera la definición de la variable yylval para poder utilizarla.

2. Crear un archivo llamado ejemmplo2.l con el código siguiente

```
%{
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include "valor.h"
%}
letra [a-zA-Z]
digito [0-9]
%option yylineno%{

        #include <stdio.h>
%}
%x comment

%%


<INITIAL>"/*"      {BEGIN(comment);}
```

```
<comment>[^*\n]*
<comment>"*"+[^*/\n]*
<comment>\n
<comment>"*"+"/"      {BEGIN(INITIAL); printf("comment\n");}

%%

int main () {
yylex () ;
return 0;
}
%%
{letra}+    {yylval.sval=( char *) realloc ( yylval.sval , yyleng * sizeof ( char )) ; strcpy ( yylval.sval , yytext ); return ID;}
{digito}+   { yylval.ival = atoi ( yytext );  return ENTERO;}
{digito}*"."{digito}+   { yylval.dval = atof ( yytext ) ;  return DOUBLE;}
{digito}*"."{digito}+[Ff]  { yylval.dval = atof ( yytext );   return REAL;}
%%
int yywrap(){
return 1;
}
int main ( int argc , char ** argv ){
   if( argc > 1) {
      FILE * file ;
      file = fopen ( argv [1] , "r") ;
      if (! file ) {
         fprintf (stderr , "No se puede abrir el archivo %s \n", argv[1]);
         exit (1) ;
      }
      yyin = file ;
   }
   int res=yylex ();
   while(res){            //Revisa si se llegó al final del archivo
      printf("<%d>",res); //Imprime el valor de la clase del token encontrado
      res=yylex ();       //Guarda el valor de yylex()
   }
   fclose ( yyin ) ;
   return 0;
}
```

Compilar y ejecutar el programa.

(a) ¿Para qué y cómo se utiliza la función realloc?
Es una función que recibe un apuntador y un entero y se encarga de reservar la cantidad de bytes especificada por el entero y que dicho apuntador tenga esa memoria disponible

(b) Modifique las líneas donde se declaran los flotantes y los doubles de tal manera que la línea donde aparecen los flotantes esté un renglón más arriba que la de los doubles. Compilar y ejecutar. ¿Qué ocurre con el analizador?
El analizador léxico sigue funcionando correctamente, sin embargo es importante tener en cuenta la jerarquía de las reglas y el orden en el que se escriben, ya que existen reglas que pueden 'absorber' a otras y ocasionar que algunos casos sean 'imposibles' de alcanzar.

(c) ¿Para que sirve %option yyline?
Al escribir esa instrucción se le indica a lex que debe lleva una guía del número de línea que va leyendo de forma que al llamar a yylex en formas consecuentes continúe correctamente leyendo el archivo

(d) Modifique el programa y quite la declaración de la función yywrap y en su lugar utilice la opción noyywrap, la cual sustituye a dicha función.

3. Investigar para que sirven las funciones yyless y yymore y poner un ejemplo de como usar cada una de ellas en el código anterior

- yyless(n) es una función que al llamarla guarda el resultante de yytext a excepción de los primeros n caracteres de la cadena, y la concatena con el yytext actual.
- yymore() es una función que guarda el yytext y concatena con el siguiente.

4. En ocasiones es útil el uso de estados para facilitar la tarea de reconocimiento del analizador léxico. En lex los estados se definen en la sección de definiciones de la manera siguiente:
%x nombre_estado
%s nombre_estado
y se utiliza la instrucción BEGIN para hacer el cambio de un estado a otro.

BEGIN nombre_estado ;

Esta instrucción se debe colocar dentro de las acciones léxicas correspondientes a cada expresión regular donde se requiera una transición.

5. El AFD de la figura 1 reconoce las cadenas sobre Σ = {a b} donde nunca se repite más de dos veces una letra, generar una expresión regular para este AFD ser´a muy compleja por ello se hace uso de los estados.

El código para el AFD de la figura 1 es el siguiente

Figure 1: AFD para cadenas que no admiten dos letras iguales consecutivas

(a) ¿Qué es INITIAL?
Es el estado de inicio del autómata

(b) ¿Con qué instrucciones se declara un estado?
%x y %s , dependiendo del tipo de estado que se quiera usar.

(c) ¿Cómo se cambia de un estado a otro?

Con la función BEGIN(), poniendo de parámetro la etiqueta con la cual el estado se identificó

6. Programe en lex el AFD

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

letra [a-z]
digito [0-9]

%x uno
%x dos
%x tres
%x cuatro
%s cinco
%s seis
%s siete

%%

<INITIAL>{letra} {BEGIN uno; printf("cero a uno");}
<INITIAL>{digito} {BEGIN cuatro; printf("cero a cuatro");}
<INITIAL>- {BEGIN tres; printf("cero a tres");}
<INITIAL>"+" {BEGIN dos; printf("cero a dos");}
<uno>{letra} {BEGIN uno; printf("uno a uno");}
<uno>{digito} {BEGIN uno; printf("uno a uno");}
<cuatro>{digito} {BEGIN cuatro; printf("cuatro a cuatro");}
<cuatro>e {BEGIN cinco; printf("cuatro a cinco");}
<cinco>{digito} {BEGIN siete; printf("cinco a siete");}
<cinco>"-"|"+" {BEGIN seis; printf("cinco a seis");}
<seis>{digito} {BEGIN siete; printf("seis a siete");}
<siete>{digito} {BEGIN siete; printf("siete a siete");}

.|\n {}
%%
int main () {
yylex () ;
return 0;
}
```

7. Programe un scanner que reconozca los comentarios del lenguaje C (/* ..... */) sin el uso de estados.

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

%%

"/*"([^*]|\*[^/])*"*/" {printf("Comentario\n");}
[\n\t ] {}
.+ {printf("No fue un comentario\n");}
%%

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

8. Programe un scanner que reconozca los comentarios del lenguaje C (/* .... */) usando estados.

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

%%
<INITIAL>"/*"       {BEGIN(comment);}
<comment>[^*\n]*
<comment>"*"+[^*/\n]*
<comment>\n
<comment>"*"+"/"       {BEGIN(INITIAL); printf("comment\n");}

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

9. Escriba sus conclusiones

La elaboración de las actividades de la práctica, incentivó una investigación más amplia de lex y cómo funcionan sus operadores, las funciones que ocupa y las reglas que deben seguir.

A raíz de esto se notó cuestiones en el código proporcionado, como la inconveniencia de que en las reglas se encuentren valores de retorno con valor 0, ya que ese valor es el que devuelve yylex cuando llega al final del archivo que esté leyendo.