Practica 1.- Analizador Léxico de C--

TALF 2019/20

16 de junio de 2020

Índice

1. Descripción de la práctica

Objetivo: El alumno deberá implementar un analizador lexico en Flex para una versión reducida (y con algunos cambios) de C, a la que llamaremos C--. El analizador recibirá como argumento el path del fichero de entrada conteniendo el programa que se quiera analizar, y escribirá en la consola (o en un fichero) la lista de tokens encontrados en el fichero de entrada, saltando los comentarios.

Documentación a presentar: El código fuente se enviará a través de Faitic. El nombre del fichero estará formado por los apellidos de los autores en orden alfabético, separados por un guión, y sin acentos ni eñes.

Ej.- DarribaBilbao-VilaresFerro.l

Grupos: Se podrá realizar individualmente o en grupos de dos personas.

Fecha de entrega: El código se subirá a Faitic como muy tarde el 22 de junio de 2020, a las 23:59.

Nota máxima: ~~1'5~~ 2'25 ptos. Se evaluará al alumno por las partes del analizador que se hayan hecho satisfactoriamente:

- ~~0'2~~ 0'3 por las palabras reservadas e identificadores
- ~~0'2~~ 0'3 por los delimitadores y operadores
- ~~0'3~~ 0'45 puntos por las constantes numéricas (enteras y reales)
- ~~0'5~~ 0'75 puntos por constantes carácter, cadenas y archivos de cabecera.
- ~~0'3~~ 0'45 puntos por los comentarios.

Ejemplo: Para un programa como el siguiente:

```
#include </local/usr/lib/math.h>
#include <stdio.h>

#define PI 31.41592e-1 // Definicion de una constante

int main
{
   /* Variables */
   float area, radio;

   printf("\nRadio de la /*circunferencia*/\
          \151\144\x69\157\x74\141: ");
   scanf("%f", &radio); /* Entrada de dato */

   printf("hola22");
   /* Calculo del area */
   area = PI * pow(radio, 2);

   /* El resultado del área se saca por la "consola":
      se trata de un número real */
   printf("\nArea de la \"circunferencia\": %f", area); printf("\n");

   return 0;
}
```

la salida debe parecerse a:

```
linea 1, operador #
linea 1, palabra reservada: include
linea 1, path: </local/usr/lib/math.h>
linea 2, operador #
linea 2, palabra reservada: include
linea 2, path: <stdio.h>
linea 4, operador #
linea 4, palabra reservada: define
linea 4, identificador: PI
linea 4, ctc real: 31.41592e-1
linea 6, palabra reservada: int
linea 6, identificador: main
linea 7, delimitador: {
linea 9, palabra reservada: float
linea 9, identificador: area
linea 9, delimitador: ,
linea 9, identificador: radio
linea 9, delimitador: ;
linea 11, identificador: printf
linea 11, delimitador: (
linea 11, cadena: "\nRadio de la /*circunferencia*/\
          \151\144\x69\157\x74\141: "
linea 12, delimitador: )
linea 12, delimitador: ;
linea 13, identificador: scanf
linea 13, delimitador: (
linea 13, cadena: "%f"
linea 13, delimitador: ,
linea 13, operador &
linea 13, identificador: radio
```

```
linea 13, delimitador: )
linea 13, delimitador: ;
linea 15, identificador: printf
linea 15, delimitador: (
linea 15, cadena: "hola22"
linea 15, delimitador: )
linea 15, delimitador: ;
linea 17, identificador: area
linea 17, operador =
linea 17, identificador: PI
linea 17, operador *
linea 17, identificador: pow
linea 17, delimitador: (
linea 17, identificador: radio
linea 17, delimitador: ,
linea 17, ctc entera: 2
linea 17, delimitador: )
linea 17, delimitador: ;
linea 21, identificador: printf
linea 21, delimitador: (
linea 21, cadena: "\nArea de la \"circunferencia\": %f"
linea 21, delimitador: ,
linea 21, identificador: area
linea 21, delimitador: )
linea 21, delimitador: ;
linea 21, identificador: printf
linea 21, delimitador: (
linea 21, cadena: "\n"
linea 21, delimitador: )
linea 21, delimitador: ;
linea 23, palabra reservada: return
linea 23, ctc entera: 0
linea 23, delimitador: ;
linea 24, delimitador: }
```

2. Especificación léxica de C--

Para que podais escribir el analizador léxico, vamos a especificar a continuación cada una de los constituyentes léxicos de los programas C--.

2.1. Palabras reservadas

Son las siguientes:

```
auto  break  case  char  continue  default  define  do double  else  enum  extern
float  for  goto  if  include  int  long  register  return  short  signed  static
struct  switch  typedef  union  unsigned  void  while
```

2.2. Identificadores

Un identificador es una secuencia de caracteres, que pueden pertencer a las siguientes categorías:

- letras mayúsculas o minúsculas pertenecientes al juego de caracteres ASCII.
- el subrayado: '_'

- dígitos entre '0' y '9'.

Importante: El primer carácter del identificador sólo puede ser una letra o '_'.

Ejemplo:

```
identificadores     NO identificadores
---------------     ------------------
uno                 úno
_25diciembre        25diciembre
tabla_123
TABLA
Array_Modificado
```

2.3. Constantes

Vamos a considerar cinco tipos de constantes: números enteros, números reales, caracteres, cadenas y nombres de archivos de cabecera.

Constantes enteras: vamos a considerar tres notaciones: decimal, octal y hexadecimal. En los tres casos las constantes están formadas por uno o más caracteres en los siguientes rangos:

- En notación decimal, los dígitos del '0' al '9'.
- En notación octal, dígitos de '0' a '7'. Además, la secuencia de dígitos estará precedida por un cero.
- En hexadecimal, los dígitos de '0' a '9' y las letras de la 'a' a la 'f', tanto en mayúcula como en minúscula, con la secuencia '0x' (o '0X') al principio de la constante entera.

Ejemplos:

```
0x23   // 35 en hexadecimal
057    // 47 en octal
0XFfF  // 4095 en hexadecimal
023    // 18 en octal
25
38
```

Constantes reales: consideraremos dos tipos de números decimales:

- Los formados por una parte entera (que es opcional), el punto decimal '.', y una parte fraccionaria. Los dígitos de la parte entera y fraccionaria pueden ser decimales o hexadecimales, usando ambas partes la **misma** codificación. En el caso de usar codificación hexadecimal, la parte entera (o el punto '.', si no hay parte entera) está precedida por la secuencia '0x' (o '0X').

  Ejemplos:

  ```
  0x27.5  .45  0X.72  38.25  .258
  ```

- Los formados por una mantisa y un exponente. La mantisa puede ser un número entero o fraccionario en notación decimal o hexadecimal. El exponente está formado por el carácter 'e' (o 'E') seguido por uno o más dígitos decimales (entre '0' y '9'), que pueden, opcionalmente, estar precedidos de un signo ('+' o '-').

  Ejemplos:

  ```
  0x27.5e-7  45E10  0X.72e+1  38E-4  .258e2
  ```

Caracteres: están formadas por dos comillas simples ('), que irán antes y después de:

- un único carácter, excepto el salto de línea, la comilla simple o el backslash '\'.
- los siguientes caracteres escapados:

  ```
  \'  \"  \?  \\  \n  \r  \t  \v
  ```

- el número del carácter expresado en octal, formado por '\', seguido de entre uno y tres dígitos octales.

  Para tocar las narices, vamos a considerar que trabajamos con el juego de caracteres ASCII (numerados entre 0 y 127), de modo que el mayor valor de un carácter en octal es '\177' (=127). Cualquier octal de tres dígitos mayor que '\177' **no** es un carácter válido.

- el número del carácter expresado en hexadecimal, formado por '\x' y uno o dos dígitos hexadecimales, hasta un máximo de 7F (=127).

Ejemplos:

```
'\0'  'a'  '9'  '\n'  '\xD'  '\x1f'  '\''
```

Cadenas: están formadas por dos comillas dobles ("), que irán antes y después de una secuencia de 0 o más:

- caracteres, exceptuando el salto de línea, la comilla doble y el backslash '\'.
- los caracteres escapados definidos en el apartado anterior.
- caracteres expresados en octal y hexadecimal, tal y como se definió en el apartado anterior.
- '\', seguido de un salto de línea.

Ejemplos:

```
"\nRadio de la /*circunferencia*/: "  // es una cadena
"linea1 \                             // es una cadena
 linea2 \                             // que abarca más de una línea
 linea3"
"eres un \151\144\x69\157\x74\141\n"  // es una cadena
```

Paths de archivos de cabecera: están formados por '<' y '>', delimitando una secuencia de 0 o más nombres de directorios seguidos por un nombre de fichero, separados por el forward slash '/'. Dichos nombre (tanto de directorio como de fichero) están formados por cualquier carácter excepto el salto de línea, las comillas dobles, '>', y el forward slash '/'. El primer directorio puede ser el raíz (o no).

Ejemplo:

```
#include <stdio.h>
#include <normal/src/libnull.h>
#include </usr/local/lib/math.h>
```

2.4. Delimitadores

Consideramos los siguientes (no los he entrecomillado para facilitar la lectura):

```
(   )   {   }   ?   :   ;   ,
```

2.5. Operadores

Consideramos los siguientes clases:

- Operadores aritméticos:

  ```
  +   -   *   **   /   %   --   ++   *=   /=   %=   +=   -=
  ```

- Operadores de acceso a memoria (estructuras y punteros):

  ```
  .   ->   *   &   [   ]
  ```

- Operadores de bits:

  ```
  ~    &   |   ^   <<   >>   <<=   >>=   &=   ^=   |=
  ```

- Operadores relacionales:

  ```
  <   >   <=   >=   ==   <>
  ```

- Operadores lógicos:

  ```
  !   &&   ||
  ```

- Otros:

  ```
  =   #   sizeof
  ```

Algunos de esos operadores tienen varios significados. Por ejemplo, '*' puede representar el producto o usarse para acceder al contenido de un puntero, '-' puede representar la resta o el menos unario,... Estas ambiguedades son resueltas estudiando el contexto en el que aparecen, es decir, son resueltas durante las siguientes fases de análisis. Para el analizador léxico es suficiente con especificar que se ha encontrado dicho operador.

2.6. Comentarios

En C-- podemos encontrar dos tipos de comentarios:

- los que comienzan con la secuencia '//' y abarcan hasta el final de la línea.
- los comentarios multilínea, delimitados por '/*' y '*/'.

Importante: Cuando las secuencias '//', '/*' o '*/' aparecen dentro de un cadena, el nombre de archivo de cabecera o un comentario **no** pueden ser interpretados como comentarios.

Otra Cosa Importante: Se ignorará la posibilidad de anidamiento de comentarios multilínea. Se considera que si, en el futuro, algún programador siente la tentación de anidar comentarios multilínea en su código, será identificado y arrestado por la sección Q del SOE antes de que pueda hacerlo.

Ejemplos:

```
"a//b"                    // una cadena
#include "<//e>"          // un nombre de archivo de cabecera
// */                     // un comentario de una sola línea
f = g/**//h;              // f = g / h;
m = n//**/o
    + p;                  // m = n + p;
```

2.7. Errores

Cuando el analizador encuentre una porción de la cadena de entrada que no se corresponda con ninguno de los tokens anteriores (o con espacios, tabuladores o saltos de línea), devolverá un mensaje de error, indicando la línea en el que ha encontrado el error, y seguirá con el análisis.