

Practica 1.- Analizador Léxico de C-Dull

TALF 2017/18

6 de marzo de 2019

Índice

1. Descripción de la práctica	1
2. Especificación léxica de C-Dull	4
2.1. Palabras reservadas	4
2.2. Identificadores	4
2.3. Constantes	4
2.4. Delimitadores	6
2.5. Operadores	6
2.6. Comentarios	6
2.7. Errores	7

1. Descripción de la práctica

Objetivo: El alumno deberá implementar un analizador lexico en Flex para un lenguaje basado en la sintaxis y las características más enervantes de los diferentes dialectos de C, al que llamaremos C-Dull. 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 Fatic. 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.

Defensa: Consistirá en una demo al profesor, que calificará tanto los resultados como las respuestas a las preguntas que realice acerca de la implementación de la práctica.

Fecha de entrega y defensa: El código se subirá a Fatic como muy tarde el 20 de marzo de 2019 a las 08:00. La defensa tendrá lugar en las clases de aula pequeña los días 20, 21 y 25 de marzo.

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

- 0'25 por las palabras reservadas e identificadores
- 0'25 por los delimitadores y operadores
- 0'25 puntos por las constantes numéricas (enteras y reales)
- 0'5 puntos por constantes caracter y cadenas
- 0'25 puntos por los comentarios.

Ejemplo: Para un programa como el siguiente:

```
namespace inutil {
    using standard.math;
    using standard.io;

    double PI 31.41592e-1 // Definicion de una constante

    int main(int argc, char *argv[]) {
        /* Variables */
        float area, radio;

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

        printf("\nhola22");

        // 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, palabra reservada: namespace
linea 1, identificador: inutil
linea 1, delimitador: {
linea 2, palabra reservada: using
linea 2, identificador: standard
linea 2, operador: .
linea 2, identificador: math
linea 2, delimitador: ;
linea 3, palabra reservada: using
linea 3, identificador: standard
linea 3, operador: .
linea 3, identificador: io
linea 3, delimitador: ;
linea 5, palabra reservada: double
linea 5, identificador: PI
linea 5, ctc real: 31.41592e-1
linea 5, delimitador: ;
linea 7, palabra reservada: int
linea 7, identificador: main
linea 7, delimitador: (
linea 7, palabra reservada: int
linea 7, identificador: argc
linea 7, delimitador: ,
linea 7, palabra reservada: char
linea 7, operador: *
linea 7, identificador: argv
```

```

linea 7, operador: [
linea 7, operador: ]
linea 7, delimitador: )
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*/\n
\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 18, identificador: area
linea 18, operador: =
linea 18, identificador: PI
linea 18, operador: *
linea 18, identificador: pow
linea 18, delimitador: (
linea 18, identificador: radio
linea 18, delimitador: ,
linea 18, ctc entera: 2
linea 18, delimitador: )
linea 18, delimitador: ;
linea 22, identificador: printf
linea 22, delimitador: (
linea 22, cadena: "\nArea de la \"circunferencia\": %f"
linea 22, delimitador: ,
linea 22, identificador: area
linea 22, delimitador: )
linea 22, delimitador: ;
linea 22, identificador: printf
linea 22, delimitador: (
linea 22, cadena: "\n"
linea 22, delimitador: )
linea 22, delimitador: ;
linea 24, palabra reservada: return
linea 24, ctc entera: 0
linea 24, delimitador: ;
linea 25, delimitador: }
linea 26, delimitador: }

```

2. Especificación léxica de C-Dull

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

2.1. Palabras reservadas

Son las siguientes:

```
auto base boolean break case catch class char continue default do double else
enum extern false finally float for goto if include int interface long namespace
new override private protected public return sealed short signed sizeof static
struct switch this throw true typedef union unsigned using virtual void while
```

2.2. Identificadores

Un identificador es una secuencia de caracteres, que pueden pertenecer 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 cuatro tipos de constantes: números enteros, números reales, caracteres y cadenas.

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 mayúscula 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 un juego de caracteres ASCII extendido (numerados entre 0 y 255), de modo que el mayor valor de un carácter en octal es '\377' (=255). Cualquier octal de tres dígitos mayor que '\377' **no** es un carácter válido.

- el número del carácter expresado en hexadecimal, formado por '\x' y uno o dos dígitos hexadecimales.

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 \  
linea2 \  
linea3" // que abarca más de una línea  
"eres un \151\144\x69\157\x74\141\n" // es una cadena
```

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 ambigüedades 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-Dull 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 una cadena o 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. Cuando se encuentre un error, podeis optar por seguir el análisis o interrumpirlo.