

Un nuevo analizador

Práctica LEX

-Modelos de Computación-

Por:

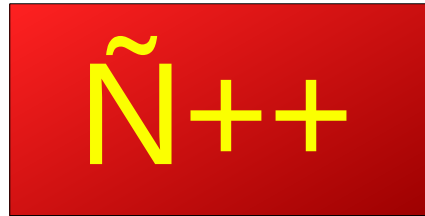
Miguel Robles Urquiza

Carlijn Rolinde Viviane Bönner

Jesús Sánchez de Lechina Tejada

Siguiendo el ejemplo de las compañías multimillonarias, hemos creado un problema que no existía para vender una solución.

Os presentamos:



Para hacernos a la idea... Ñ++ es un lenguaje que está inspirado en el lenguaje de programación C++.

Pero este tiene algunas peculiaridades.

Las palabras reservadas:

- En español
- Y en MAYÚSCULAS

Por ejemplo: MIENTRAS, DEVOLVER, SI...

(semejantes a while, return, if... en c++)

Los identificadores para declarar variables:

- En minúsculas
- Que comiencen con minúscula o '_' y puedan contener guiones bajos o números

Nosotros proponemos un analizador que, dado un programa en este lenguaje, sea capaz de determinar si está correctamente escrito y notificar de los errores que detecte

Las expresiones

Para cumplir con la descripción introducimos estas expresiones

letra [a-z]

alfanumerico [a-z_0-9]

entero [0-9]+

real {entero} "." {entero}

identificador ({letra} | "_") (alfanumerico)*

cadena (\" [^\"]* \")

caracter (\' [^\'] \')

mayus [A-Z]+

Inicialización

Para poder llevar a cabo tareas de depuración hemos incluido unas variables a nuestro analizador

```
int linea = 0; // Contador para indicar la línea donde se produjo el error
```

```
int openBrackets = 0; // Nº de paréntesis/llaves abiertas
```

```
int openBraces = 0;
```

```
int firstOpenBracket; // Índice del primer paréntesis que se quedó abierto
```

```
int firstOpenBrace; // Índice de la primera llave que se quedó abierta
```


Reglas

Para admitir las palabras reservadas tenemos un conjunto de reglas que, si se detecta esta cadena, se acepta sin dar error

"#INCLUIR"

"ENTERO"

"REAL"

...

Reglas

Existen reglas para contar la línea en la que estamos y cuando se detecte una palabra en mayúsculas que no esté entre las reservadas dará un error. O también para cuando se detecte algo desconocido.

```
[r\n] ++linea;
```

```
{mayus} printf("\n(Linea %d) Error lexico: Palabra desconocida %s\n",linea,yytext);
```

```
. printf("\n(Linea %d) Error lexico: Token desconocido %s\n",linea,yytext);
```

Reglas

Hemos añadido unas reglas algo más complejas para comprobar que cuando se abra una llave esta se cierre en algún momento.

```
// Incrementamos o decrementamos la cuenta de paréntesis abiertos
```

```
"(" {openBrackets++; if(openBrackets == 1) firstOpenBracket = linea;}
```

```
")" {openBrackets--; if(openBrackets < 0){ printf("\n(Linea %d) Cierre de paréntesis inesperado.\n",linea); openBrackets = 0;}}
```

```
// Incrementamos o decrementamos la cuenta de llaves abiertas
```

```
"{" {openBraces++; if(openBraces == 1) firstOpenBrace = linea;}
```

```
"}" {openBraces--; if(openBraces < 0) { printf("\n(Linea %d) Cierre de llave inesperado.\n",linea); openBraces = 0;}}
```

Reglas

<<EOF>>: Al terminar llegar al final del fichero hacemos las cuentas

```
<<EOF>> {  
    if(openBrackets != 0) {  
        printf("\nQueda algún paréntesis por cerrar. Abierto en línea %d\n",lastOpenBracket);  
    }  
  
    if(openBraces != 0) {  
        printf("\nQueda alguna llave por cerrar. Abierta en línea %d\n",lastOpenBrace);  
    }  
    yyterminate();  
}
```

yyterminate() es necesario para detener el flujo de lectura cuando usamos <<EOF>>

Nuestro Ejemplo

```
#INCLUIR "libreria.h"
```

```
ENTERO _ve, ve_  
REAL vreal_1, vreal2;
```

```
ENTEO funcion1 (ENTERO entero_1, REAL _real22) {  
    ENTERO _x1 = 0;  
    REAL _real13 = 0.0;  
    DEVOLVER _x1;  
}
```

```
}  
CARACTER funcion2 (CARACTER _e1, CARACTER _e2){  
    SALIDA "introduzca dos caracteres: ";  
    ENTRADA e1, e2;  
    SI (e1 == 'a')  
    DEVOLVER e1;  
    SINO SI (e1 != 'b')  
    DEVOLVER e2;  
    SINO  
    DEVOLVER ' ';  
}
```



```
BOOLEANO funcion3 (ENTERO _ent){  
    OPCION(_ent):  
    CASO 1: DEVOLVER VERDADERO;  
    CASO 2: _ent++;  
    ROMPER;  
    POR-DEFECTO: DEVOLVER FALSO;  
(  
}  
{
```



Palabra mal escrita



Paréntesis y llave cerrados prematuramente



Paréntesis y llave que no se cierran

Compilación

De acorde a lo establecido:

```
cuack@cuack:~/Escritorio/MC-LEX/Codigo$ make  
lex -o ficheroLex.cpp expresiones.lex  
g++ -o analizar ficheroLex.cpp -ll
```

Resultado

```
cuack@cuack:~/Escritorio/MC-LEX/Codigo$ make ejecuta  
./analizar ejemplo.txt
```

```
(Linea 11) Error lexico: Palabra desconocida ENTEO
```

```
(Linea 21) Cierre de llave inesperado.
```

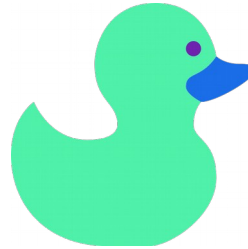
```
(Linea 26) Cierre de paréntesis inesperado.
```

```
Queda algún paréntesis por cerrar. Abierto en línea 28
```

```
Queda alguna llave por cerrar. Abierta en línea 47
```

FIN

Gracias por su atención



Vuelve a ver esta presentación cuando quieras en:



www.github.com/jojelupipa/MC-LEX