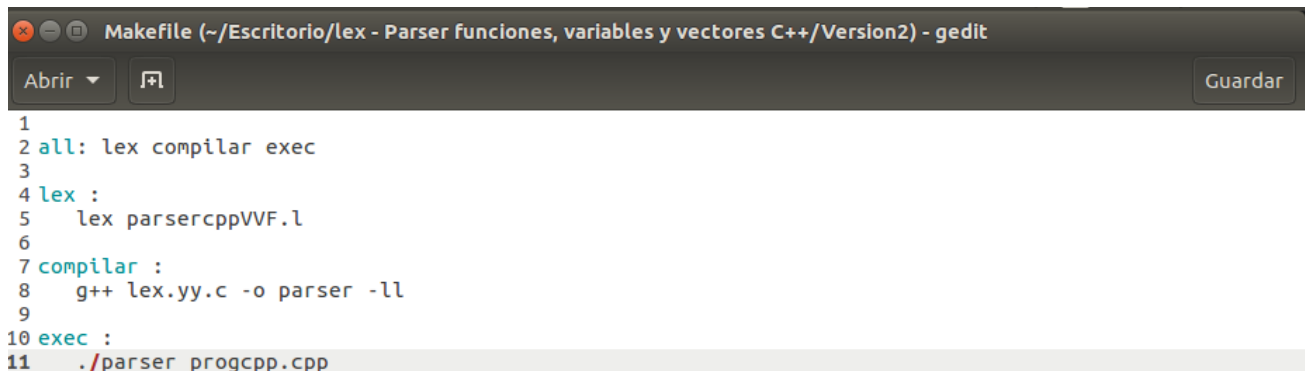


Práctica 2: lex

En esta práctica de lex como localizador de expresiones regulares con acciones asociadas hemos decidido abordar el siguiente ejercicio:

Procesador de un código C++ para identificar los operadores, las bibliotecas, las variables, los vectores y las funciones (detecta si el uso de los corchetes y paréntesis es correcto) e informará de cuánta memoria se reservará para las variables y los vectores que aparecen en el fichero.

La entrada será un programa simple C++ al que hemos llamado “progcpp.cpp” y a nuestro procesador lex lo hemos llamado “parsercppVVF.l”. Las ordenes de compilación que hemos usado son las siguientes, las hemos incluido en este simple makefile:



```
1
2 all: lex compilar exec
3
4 lex :
5     lex parsercppVVF.l
6
7 compilar :
8     g++ lex.yy.c -o parser -ll
9
10 exec :
11     ./parser progcpp.cpp
```

Estructura del fichero lex “parsercppVVF.l”:

→ Sección de Declaraciones:

- Subsección donde definimos todo lo que irá tal cual en el lex.yy.c generado tras ejecutar la orden lex parsercppVVF.l. Tenemos la declaración de todos los contadores que utilizaremos para saber el numero de operadores, variables, funciones, memoria ocupada, etc. del programa C++ a analizar.

Definimos una estructura para cada cosa que queremos analizar:

-includes: vector simple

-variables: Hemos decidido usar una estructura de datos formada por un conjunto de pares, los cuales tendrán como .first otro par, el cual almacenará en su .first el tipo de la variable (string) y en el .second el nombre de la variable (string), y en el .second del primer par se almacenará la memoria reservada por la variable (int).

-vectores: En este caso la estructura de datos será igual que la anterior con la diferencia que el .second del par principal contendrá un “string” que almacenará el numero de bloques de dicho vector. Este dato tendrá que ser multiplicado por el tamaño del tipo de variable que almacena el vector.

-funciones: La estructura de datos es un conjunto de pares, en los que en el .first se almacenará el tipo de la función (string) y en el .second el nombre de la misma (string).

-Subsección de definición de Alias donde nombramos las expresiones regulares que necesitamos.

→ Sección de Reglas:

Donde tenemos 5 partes bien diferenciadas: la primera referente a contar los operadores; la segunda dedicada a las bibliotecas, las cuales incluirá directamente en el vector de includes; La tercera, la cuarta y la quinta parte o subsección la explicaremos a continuación:

-Para las funciones hemos tenido que definir una regla para cada tipo básico de función que sigue la forma:

`(tipo)(“ ”){funcion}`

La acción asociada a esta regla es llamar a la función “analizarFuncion” e incrementar el contador de funciones.

-Para las variables hemos tenido que definir una regla para cada tipo básico de variable que sigue la forma:

`(tipo)(“ ”){variable}`

La acción asociada a esta regla es llamar a la función “analizarVariable” e incrementar el contador de variables.

-Para los vectores hemos hecho lo mismo que en las dos subsecciones anteriores, definir una regla para cada tipo básico de dato que sigue la forma:

`(tipo)(“ ”){vector}{caracter}”]`

De este modo tenemos incluido en el alias de vector el corchete que abre, seguidamente un valor numérico de la expresión regular “caracter” y un corchete de cerramiento especificado entre comillas (sin incluir en ninguno de los alias).

La acción asociada a esta regla es llamar a la función “analizarFuncion” e incrementar el contador de funciones.

Estas funciones asociadas a la acción de la regla las explicamos en la siguiente sección.

→ Sección de Procedimientos de Usuario:

En esta sección primeramente analizamos el fichero pasado como argumento y comprobamos si las expresiones son correctas a través de los contadores de paréntesis, corchetes y funciones. Luego mostramos los contadores y las bibliotecas, variables, vectores y funciones a través de las funciones definidas para mostrarlos que pasamos a explicar a continuación.

-Funciones de Usuario:

-analizarVariable→ Función void que recibe un puntero a la variable a analizar. El análisis consiste en detectar el tipo, el nombre y el tamaño que reserva en memoria cuando se declara una variable de ese tipo. Para ello nos

declaramos unos punteros donde los almacenaremos. Sabemos que antes del espacio tendremos el tipo y después el nombre, por lo que jugamos apuntando al espacio los punteros.

El tipo lo almacenamos en el .first del par anidado y el nombre en el .second. El tamaño requerido en memoria por la variable lo guardaremos en el .second del par principal (dependiendo del tipo de dato, calculará con “sizeof” el tamaño reservado para la variable).

Finalmente insertará la variable procesada en el conjunto de variables.

-mostrarVariables: Función que recorrerá el conjunto de variables (conjunto formado por un par de pares) e irá mostrando la información almacenada referente a las variables procesadas. Para ello nos declaramos el iterador para nuestra estructura y la recorremos mostrando y calculando la memoria total reservada para las variables.

-analizarVector: Función void que recibe un puntero al vector. El análisis consiste en detectar el tipo, el nombre y el tamaño que reserva en memoria cuando se declara un vector de ese tipo. Para ello nos declaramos unos punteros donde los almacenaremos. Sabemos que antes del espacio tendremos el tipo, después y hasta “[“ tendremos el nombre y después y hasta “]” tendremos los bloques reservados por el vector.

El tipo lo almacenamos en el .first del par anidado y el nombre en el .second. La memoria reservada por el vector lo guardaremos en el .second del par principal (dependiendo del tipo de variable, guardaremos en .second su tamaño reservado en memoria multiplicado por el tamaño del vector). Finalmente incrementará el contador de memoria total reservada para los vectores y añadirá el vector procesado al conjunto de vectores.

-mostrarVectores: Función que recorrerá el conjunto de vectores (conjunto formado por un par de pares) e irá mostrando la información referente a los vectores procesados. Para ellos nos declaramos el iterador para nuestra estructura y la recorremos mostrando y calculando la memoria total reservada para los vectores.

-analizarFuncion: Función void que recibe un puntero a la función. El análisis consiste en detectar el tipo y el nombre. Para ello nos declaramos unos punteros donde los almacenaremos. Sabemos que antes del espacio tendremos el tipo y después y hasta el “(“ el nombre. El tipo lo almacenamos en el .first del par y en el .second el nombre.

Finalmente añadimos la función al conjunto de funciones.

-mostrarFunciones: Función que recorrerá el conjunto de funciones (conjunto formado por un par) e irá mostrando la información referente a las funciones procesadas. Para ello nos declaramos el iterador para nuestra estructura y la recorremos mostrando la memoria total reservada para las funciones.

-mostrarLibreria: Recorrerá el vector de “includes” mostrando su contenido.

Ejecución:

```
josele@josele-VB: ~/Escritorio/lex - Parser funciones, variables y vectores C++/Version2
josele@josele-VB: ~/Escritorio/lex - Parser funciones, variables y vectores C++/Version2 115x67
josele@josele-VB:~/Escritorio/lex - Parser funciones, variables y vectores C++/Version2$ make
lex parsercppVVF.l
g++ lex.yy.c -o parser -ll
./parser progcpp.cpp
Libreria: #include<iostream>

Libreria: #include<cmath>

Libreria: #include<stdio>

;

# = 500;

Funcion: int multiplicar(
Variable: int num1
, Variable: int num2

    Variable: int res

;

    = 1 2;
    ;

Funcion: bool multiplicado(
3, 1, 2
    Variable: bool multi
= ;
    3 == 12
        = ;
    ;

Funcion: int main(

    Vector: string vectors[4]
;
    Vector: int vector[100]
;
    Vector: double vector2[50]
;
    Vector: float vector3[10]
;
    Vector: char vector4[200]
;
    Vector: unsigned vector5[7]
;
    Vector: bool vector8[8]
;

    Variable: double var2
;
    Variable: float var3
;
    Variable: char var4
;
    Variable: unsigned var6
;
    Variable: unsigned var7
;
    Variable: double var8
;
    Variable: int num1
;
    Variable: int num2
;
```

```
josele@josele-VB: ~/Escritorio/lex - Parser funciones, variables y vectores C++/Version2
josele@josele-VB: ~/Escritorio/lex - Parser funciones, variables y vectores C++/Version2 115x67
Variable: int i
= 0; < 5;
3;
4 = 3 1;

4 = 4;
4 <= 25
4;
3>0
3;
3=1;
3=1;

*****PARSER C++*****
La expresion es correcta
El operador:
+ aparece 1 veces
- aparece 1 veces
* aparece 3 veces
/ aparece 1 veces
++ aparece 3 veces
-- aparece 1 veces

Usa las librerias:
#include<iostream>
#include<cmath>
#include<stdio>

Existen 14 variables:
Tipo: bool Nombre: multi Tamaño reservado: 1 Bytes
Tipo: char Nombre: var4 Tamaño reservado: 1 Bytes
Tipo: double Nombre: var2 Tamaño reservado: 8 Bytes
Tipo: double Nombre: var8 Tamaño reservado: 8 Bytes
Tipo: float Nombre: var3 Tamaño reservado: 4 Bytes
Tipo: int Nombre: i Tamaño reservado: 4 Bytes
Tipo: int Nombre: num1 Tamaño reservado: 4 Bytes
Tipo: int Nombre: num2 Tamaño reservado: 4 Bytes
Tipo: int Nombre: num3 Tamaño reservado: 4 Bytes
Tipo: int Nombre: num4 Tamaño reservado: 4 Bytes
Tipo: int Nombre: res Tamaño reservado: 4 Bytes
Tipo: string Nombre: var9 Tamaño reservado: 32 Bytes
Tipo: unsigned Nombre: var6 Tamaño reservado: 4 Bytes
Tipo: unsigned Nombre: var7 Tamaño reservado: 4 Bytes
Total de memoria reservada por las variables: 86 Bytes

Existen 7 vectores:
Tipo: bool Nombre: vector8 Tamaño reservado: 8 Bytes
Tipo: char Nombre: vector4 Tamaño reservado: 200 Bytes
Tipo: double Nombre: vector2 Tamaño reservado: 400 Bytes
Tipo: float Nombre: vector3 Tamaño reservado: 40 Bytes
Tipo: int Nombre: vector Tamaño reservado: 400 Bytes
Tipo: string Nombre: vectors Tamaño reservado: 128 Bytes
Tipo: unsigned Nombre: vector5 Tamaño reservado: 28 Bytes
Total de memoria reservada por los vectores: 1204 Bytes

Existen 3 funciones:
Tipo: bool Nombre: multiplicado
Tipo: int Nombre: main
Tipo: int Nombre: multiplicar
josele@josele-VB:~/Escritorio/lex - Parser funciones, variables y vectores C++/Version2$
```