Autor:

- Jose Manuel Aranda Gutiérrez
  - Correo electrónico: jomagtr@gmail.com
  - Correo electrónico ugr: jomagtr@correo.ugr.es
  - Grupo: A3

Analizador Lexico

En este fichero podremos encontrar la documentación completa del proyecto llevado a cabo para la práctica 2 de MC. Realizar un Analizador Léxico, explicar el objetivo de dicho analizador, su funcionamiento.

Para ello se ha realizado un repositorio en Github con el siguiente enlace: https://github.com/Arzenin/Analizador-Lexico

Se recomienda la visualización de la documentación desde el README.md en el github, ya que por medio de los enlaces se puede llegar más rápido a las partes en las que se esté interesado, sin embargo si se desea se puede tener la casi la misma experiencia visualizandolo tanto desde el README.md como desde el PDF que o bien se ha obtenido por medio de su envio o bien por su obtención en la carpeta `Doc/` dentro del repositorio de github

0. Índice

1. Introducción y motivación

Este proyecto corresponde a la práctica 2 de la asignatura de MC, en ella se nos pide desarrollar un analizador léxico para el reconocimiento de los patrones que nosostros estimemos oportunos. En mi caso, he dicidido que este analizador de una entrada extraiga las URL y las IP del tipo IP4, las cuales si o si deben de tener o bien el protocolo HTTP o bien el HTTPS.

Respecto a mis motivos a la hora de la elección de este tipo de analizador léxico vienen dado por varios proyectos y cursos que he realizado a lo largo del año. El proyecto en cuestión fue el montaje de un servidor casero, durante la configuración del mismo, debía de conectarme por medio de `ssh` al servidor, sin embargo perdía mucho tiempo buscando la sección de IP dentro de mi router. Esto me hizo pensar en lo práctico que habría sido tener un programa capaz de extraer del fichero todas las IP y decirme exactamente en que línea se encuentan.

Por otro lado la detección de URL viene dado por diversos cursos que he hecho los cuales me han hecho darme cuenta lo útil que sería obtener un Analizador léxico que te permitiese extraer un URL y clasificarla dependiendo del dominio al que pertenezca, esto último no es uno de los objetivos que realizaré en esta práctica, sin embargo utilizaré esto como base para expandir el programa que desarrolle.

Volver al índice

2. Objetivos

En esta sección se explicarán los objetivos de la práctica, es decir el patrón que se debe detectar exactamente dentro del texto al ejecutar el analizador léxico, para ello especificaremos que debe de encontar tanto en ***URL*** como en ***IP*** y tras esto se realizarán ***algunas aclaraciones finales*** de el por qué de algunas decisiones tomadas a la hora de la elección de objetivos.

2.1 URL

Las URL que se detectan deben de tener previas a ellas el protocolo HTTP o HTTPS, sin embargo en este caso se deberá de admitir la **posibilidad** de que tenga el **subdominio www** , además de esto debe de admitir **distintos tipos de dominio de primer nivel** y siempre deben de haber **almenos uno presente** , los TLP que admitiría son: ***.com .es .org .net .edu .gov***

2.2 IP

Al igual que con las URL debe de tener el protocolo HTTP o HTTPS, se busca tanto **el formato correcto**, como que sea una **IP válida** es decir que ninguno de sus campos puedes ser mayor a 255 ni un número negativo.

2.3 Aclaraciones Finales

Para una mayor sencillez a la hora de comprensión se aceptará que la IP o URL este entre palabras, es decir que si pusiesemos ***Holahttp://12.12.12.3Chao*** nos extraería ttp://12.12.12.3 , el motivo de esto es una mejor presentación a la hora de obtener el prompt de salida de forma más clara y concisa, ya que, como veremos más adelante, el **Analizador clasificara tanto las IP como las URL de forma ordenada y nos las devolverá posteriormente**, sin embargo a la hora de explicar el código se nos indicarán las modificaciones pertinentes en caso de querer hacer que únicamente nos detectase las IP y URL exactas, es decir o bien estando entre palabras,singnos de puntuación, comienzo de la línea y final de la linea.

Volver al índice

3. Github

En este apartado se os explicaría en profundidad todos y cada uno de los apartados relacionados con el repositorio de github, desde la estructura hasta la descarga del propio material del repositorio

3.1 Estructura del Repositorio

El repositorio esta compuesto por 3 directorios:

3.1.1 Src

En el directorio **Src** encontraremos el analizador léxico por el cual obtendremos los requisitos explicados en la sección anterior, el nombre del

archivo es AnalizadorLexico

Más adelante en el apartado explicación del código se explicará en mayor profundidad el funcionamiento del mismo

3.1.2 Test

En el directorio **Test** encontraremos podremos encontrar un fichero llamado entrada.txt por el cual podremos comprobar el correcto funcionamiento del programa.

Se recomienda encarecidamente su lectura ya que al final de este se nos indican los resultados esperados por parte del programa.

3.1.3 Results

En el directorio **Results** se almacena una muestra del resultado que deberíamos de obtener en caso de ejecutar el código utilizado el archivo de entrada proporcionado

Este directorio nos será de gran utilidad para poder comprender el formato de la salida que obtendremos previa a la ejecución del analizador léxico

3.1.4 Doc

En el directorio **Doc** se almacena el archivo documentacion.pdf el cual correspondería al fichero README.md en formato pdf para así poder obtener la documentación y poder leerla desde cualquier medio.

3.2 Descarga del Repositorio

En este apartado se enseñara al usuario a descargar todos los recursos necesarios del repositorio por medio de la instalación de github y la clonación de un repositorio en la máquina desde la que queramos ejecutar el analizador.

3.2.1 Instalación de Git

1. Deberemos abrir el terminal
2. Nos aseguraremos de tener todas las dependencias y paquetes actualizados por medio de `sudo apt update` y `sudo apt upgrade`
3. Tras esto ejecutaremos `apt-get install git` por el cual instalaremos git junto a sus dependencias
4. Por último como en el resto de instalaciones ejecutaremos `git --version` para asegurarnos que se ha instalado todo correctamente

3.2.2 Clonar un Repositorio

Una vez instalado Git deberemos de crear un directorio personalmente recomiendo que creemos uno con el siguiente esquema `/Documents /github` sin embargo esto esto no es obligatorio.

1. Nos dirigiremos al directorio en el que queramos clonar el repositorio
2. Abrimos la terminal en ese directorio
3. Ejecutamos `git clone https://github.com/Arzenin/Analizador-Lexico.git`
4. Ejecutamos `ls` y debería de haber aparecido un nuevo directorio llamado `/Analizador-Lexico`

Volver al índice

4. Instalación de la Herramienta

En este caso explicaré la instalación de las herramientas en el entorno de programación que se ha utilizado para desarrollarla, es decir, una intalación en la distribución de **Ubuntu 22.04.3 LTS** utilizando el compilador gcc y flex, dicho esto en los dos siguientes apartados se procederá a explicar la instalación de cada una de las herramientas.

4.1 Gcc

1. Abrimos la terminal
2. Comprobamos que todas las dependencias esten actualizadas y en caso de no ser así acutalizamos con `sudo apt update` y `sudo apt upgrade`

3. Descargaremos los paquetes básicos donde van incluido gcc y todas sus librerías con el comando `sudo apt install build-essential`
4. Comprobaremos que se ha descargadado correctamente con el comando `gcc --version`

4.2 Flex

1. Abrimos la terminal
2. Comprobamos que todas las dependencias esten actualizadas y en caso de no ser así acutalizamos con `sudo apt update` y `sudo apt upgrade`
3. Ejecutamos el comando `sudo apt-get install flex`
4. Ejecuratemos `flex --version` pasa asegurarnos de que todo se ha instalado correctamente

Volver al índice

5. Explicación del Código

Una vez explicado tanto donde encontar cada uno de los materiales, pasando por la obtención de los mismos y para finalizar la instalación de programas y dependencias necesarias a la hora de ejecutar el analizador léxico, podemos comenzar con la explicación del código.+

Esta sección la dividiremos en 4 apartados estando el primero más centrado en la ***explicación del código fuente***, el segundo en la ***comprensión del fichero que recibiremos como entrada***, el tercero ***como nos devolverá los datos el programa*** , y para finalizar aprenderemos los comandos pertinentes para ***ejecutar el programa***

5.1 Explicación del Código Fuente

El código fuente del analizador léxico se ubica en el archivo ***Analizador Léxico***, este archivo se subdivide en 3 partes, la primera dedicada a declaraciones y definiciones de reglas del analizador y variables globales en C, la segunda dedicada a las acciones que se deben de realizar una vez se encuentre una ocurrencia y la última dedicada al main en el cual se inicializan algunas variables dinámicas que no se pueden inicializar en entornos globales y al formato de los datos de cara a la salida.

Estas secciones se llaman respectivamente: **Declaraciones y Definiciones Acciones Main**

5.1.1 Declaraciones y Definiciones

El código que podemos observar a continuación es el código completo de la sección, más adelante explicaremos línea por línea este código:

```
protocolo_seguro        ("https://")
protocolo_inseguro      ("http://")
dominio                 (("www"\.)?[a-zA-Z]+[\.]("com"|"es"|"org"|"net"|"edu"|"gov"))
s_ipv4                  (([0-1]?[0-9]?[0-9])|[2][0-4][0-9]|[2][5][0-5])
ipv4                    ({s_ipv4}\.{s_ipv4}\.{s_ipv4}\.{s_ipv4})
ip_seguro               {protocolo_seguro}{ipv4}
url_dominio_seguro      {protocolo_seguro}{dominio}
url_dominio_inseguro    {protocolo_inseguro}{dominio}
ip_inseguro             {protocolo_inseguro}{ipv4}


%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <stdbool.h>
    int numurl=0,numprotocolo_inseguro=0,numline=1,numword=0;
    int numprotocolo_seguro=0,numdominio=0,numipv4=0,numchar=0;
    char* ip_s = NULL;
    char* ip_u = NULL;
    char* dom_s = NULL;
    char* dom_u = NULL;

    void elimina_car_sobrantes(char* vector_car){
        int longitud = strlen(vector_car);
        if(vector_car[0]==' '||vector_car[0]=='\n'||vector_car[0]=='.'|vector_car[0]==','|vector_car[0]=='\b'){
            if(vector_car[0]=='\n'){
                numline++;
```

```
            }
            else if(vector_car[0]==' '){
                numword++;
            }
            for (int i = 0; i < longitud; i++) {
                vector_car[i] = vector_car[i + 1];
            }
            vector_car[longitud-1]='\0';
            longitud = longitud-1;
        }
        if(vector_car[longitud-1]==' '||vector_car[longitud-1]=='\n'||vector_car[longitud-1]=='.'|vector_car[longitud-1]==',
            if(vector_car[longitud-1]=='\n'){
                numline++;
            }
            else if(vector_car[longitud-1]==' '){
                numword++;
            }
            for (int i = 0; i < longitud; i++) {
                vector_car[longitud-1] = vector_car[i + 1];
            }
            vector_car[longitud-1]='\0';
        }
    }
%}
```

Para comenzar nos centraremos en las 5 primeras líneas:

```
protocolo_inseguro      ("http://")
protocolo_seguro        ("https://")
dominio                 (("www"\.)?[a-zA-Z]+[\.]("com"|"es"|"org"|"net"|"edu"|"gov"))
s_ipv4                  (([0-1]?[0-9]?[0-9])|[2][0-4][0-9]|[2][5][0-5])
ipv4                    ({s_ipv4}\.{s_ipv4}\.{s_ipv4}\.{s_ipv4})
```

Estas dos líneas establecen las reglas para la correcta detección de los protocolos HTTP y HTTPS para así poder llevar a cabo su correcta detección, tras esto estableceremos en la tercera línea lo que es un dominio, de aquí podemos destacar la sección `(("www"\.)?)` la cual nos permite la **opcionalidad del subdominio**, tras el cual será necesario como mínimo proporcionar **una palabra de al menos 1 letra** tras esto deberá de seguir el dominio de primer nivel de las opciones que hemos proporcionado.

De la línea 4 nos sirve para definir que los números correctos de una IP son del número 0 al 255 y en la línea 5 indicamos el formato de IPV4, es decir `***.***.***.***` Siendo `***` un número de no más de 3 dígitos inferior a

255.

Tras esto deberemos de definir el formato de protocolo e IP o URL unidos, y dependiendo del tipo de protocolo clasificarlo de una forma o de otra, esto se hace en las líneas 6,7,8 y 9:

```
ip_seguro               {protocolo_seguro}{ipv4}
url_dominio_seguro      {protocolo_seguro}{dominio}
url_dominio_inseguro    {protocolo_inseguro}{dominio}
ip_inseguro             {protocolo_inseguro}{ipv4}
```

En las líneas 6 y 9 definimos la clasificación de las IP dependiendo del protocolo que lleven previamente y en las líneas 7 y 8 lo mismo pero con las URL.

Hay algo de importancia que se debe de mencionar sobre estas líneas, en el analzador léxico que se ha realizado para esta práctica lo que se busca es la correcta detección del patrón y la correcta clasificación, siendo esta segunda característica la de mayor importancia, sin embargo si lo que se desease es que únicamente se acptase el patrón siendo este una palabra propia individual se deberían realizar las siguientes modificaciones en estas líneas:

```
ip_seguro               (^|\b|\n|[ \.\,]){protocolo_seguro}{ipv4}($|\b|\n|[ \.\,])
url_dominio_seguro      (^|\b|\n|[ \.\,]){protocolo_seguro}{dominio}($|\b|\n|[ \.\,])
url_dominio_inseguro    (^|\b|\n|[ \.\,]){protocolo_inseguro}{dominio}($|\b|\n|[ \.\,])
ip_inseguro             (^|\b|\n|[ \.\,]){protocolo_inseguro}{ipv4}($|\b|\n|[ \.\,])
```

Con este comando se nos permitiría aceptar estos patrones únicamente si o bien van a principio `^` o a final de línea `$` , van seguidos de un salto de línea `\n` , un espacio en blanco al final de la línea, un punto, una coma `[ \.\,]` y para finalizar un hueco entre palabras gracias a `\b`

Antes de proseguir a las declaraciones y definiciones en C me gustaría hacer una aclaración, en caso de querer probar el supuesto anterior haría falta realizar un reordenado de reglas, tanto en la sección de acciones, como en la de declaraciones y definiciones, esto a raíz de que cuando Flex puede entrar en conflcto en caso de que 2 patrones se cumplan simultáneamente, esto haría que se ejecutase únicamente la acción del patrón más general, por esto mismo, habría que plantearse tanto el reordenado, como la refinición y borrado de algunas acciones evitando así conflictos.

Prosiguiendo con la explicación esta sería la sección de declaraciones y definiciones en C:

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <stdbool.h>
    int numurl=0,numprotocolo_inseguro=0,numline=1,numword=0;
    int numprotocolo_seguro=0,numdominio=0,numipv4=0,numchar=0;
    char* ip_s = NULL;
    char* ip_u = NULL;
    char* dom_s = NULL;
    char* dom_u = NULL;

    void elimina_car_sobrantes(char* vector_car){
        int longitud = strlen(vector_car);
        if(vector_car[0]==' '||vector_car[0]=='\n'||vector_car[0]=='.'|vector_car[0]==','|vector_car[0]=='\b'){
            if(vector_car[0]=='\n'){
                numline++;
            }
            else if(vector_car[0]==' '){
                numword++;
            }
            for (int i = 0; i < longitud; i++) {
```

```
                vector_car[i] = vector_car[i + 1];
            }
            vector_car[longitud-1]='\0';
            longitud = longitud-1;
        }
        if(vector_car[longitud-1]==' '||vector_car[longitud-1]=='\n'||vector_car[longitud-1]=='.'|vector_car[longitud-1]==',
            if(vector_car[longitud-1]=='\n'){
                numline++;
            }
            else if(vector_car[longitud-1]==' '){
                numword++;
            }
            for (int i = 0; i < longitud; i++) {
                vector_car[longitud-1] = vector_car[i + 1];
            }
            vector_car[longitud-1]='\0';
        }
    }
%}
```

Podemos observar la sección dedicada a importación de librerías, en el cual se incluyen librerías como string,h.Esta nos aportará determinadas funciones que nos facilitarán el almacenado de ocurrencias dentro de vectores dinámicos por medio de punteros:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Tras esto encontaremos la declaración y definición de todos los contadores que nos permitirán llevar el número de ocurrencias de cada clase para así, al finalizar el programa, mostrar el conteo de ocurrencias.

```
int numurl=0,numprotocolo_inseguro=0,numline=1,numword=0;
int numprotocolo_seguro=0,numdominio=0,numipv4=0,numchar=0;
```

De la línea 18 a la 21 podremos observar la declaración de estos futuros vectores dinámicos, todos ellos nos permitirán el poder almacenar las ocurrencias conforme se vayan detectando para así al finalizar el programa mostrarlas por pantalla y en orden correctamente clasificadas:

```
char* ip_s = NULL;
char* ip_u = NULL;
char* dom_s = NULL;
char* dom_u = NULL;
```

La razón de que los vectores estén inicalizados a 0 es por que este bloque de código equivale a uno global en C. A raíz de esto no podremos inicializarlos hasta definir y declarar el `main()` ya que si no podríamos llegar a tener problemas en lo que respecta a la gestión de la memoria durante la ejecución del programa.

Para acabar esta sección completamente se debe de explicar la función `elimina_car_sobrantes` , esta no es necesaria para la ejecución del analizador léxico que he diseñado para esta práctica, sin embargo si que lo es en caso de querer hacer el caso de querer realizar las modificaciones para que únicamente se detecte la IP y la URL como una *"Palabra individual"*

La necesidad de eliminar caracteres es que a la hora de la salida de los resultados obtenidos, en caso de no eliminar caracteres como los saltos de línea, espacios en blanco, puntos, comas... etc nos aparecerían dentro del fichero salida.txt.

```
void elimina_car_sobrantes(char* vector_car){
    int longitud = strlen(vector_car);
    if(vector_car[0]==' '||vector_car[0]=='\n'||vector_car[0]=='.'|vector_car[0]==','|vector_car[0]=='\b'){
        if(vector_car[0]=='\n'){
            numline++;
        }
        else if(vector_car[0]==' '){
            numword++;
```

```
            }
            for (int i = 0; i < longitud; i++) {
                vector_car[i] = vector_car[i + 1];
            }
            vector_car[longitud-1]='\0';
            longitud = longitud-1;
        }
        if(vector_car[longitud-1]==' '||vector_car[longitud-1]=='\n'||vector_car[longitud-1]=='.'|vector_car[longitud-1]==','|ve
            if(vector_car[longitud-1]=='\n'){
                numline++;
            }
            else if(vector_car[longitud-1]==' '){
                numword++;
            }
            for (int i = 0; i < longitud; i++) {
                vector_car[longitud-1] = vector_car[i + 1];
            }
            vector_car[longitud-1]='\0';
        }
    }
```

Esta función recibirá como parámetro un vector, comprobará que el primer y el último caracter no sean ninguno de los mencionados anterior mente, en caso de ser así, borrara los caracteres correspondientes y redimensionará el vector reasignando el '\0' a su nuevo lugar.

Por último la razón de que se aumenten los contadores de salto de línea y de espacios en blanco, es por que por los problemas generados por el choque de reglas puede que el conteo de algunos espacios en blanco o saltos de línea se pierdan

Volver al índice

5.1.2 Acciones

En la siguiente sección del código se definen las **acciones que se realizarán al momento de encontrar un patrón complejo** como puede ser el de **ip_seguro** como el de patrones más **sencillos** como los **saltos de línea**

A continuación podemos observar el código de esta sección:

```
%%
{ip_inseguro}           {numurl++;numipv4++;numprotocolo_inseguro++;
                        char numer[50];
                        sprintf(numer, "Num line: %d \tIP:  ",numline);
                        ip_u = realloc(ip_u,strlen(ip_u) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(ip_u,"\n");
                        strcat(ip_u,"\t");
                        strcat(ip_u,numer);
                        strcat(ip_u,yytext);
                        }

{ip_seguro}             {numurl++;numipv4++;numprotocolo_seguro++;
                        char numer[50];
                        sprintf(numer, "Num line: %d \tIP:  ", numline);
                        ip_s = realloc(ip_s,strlen(ip_s) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(ip_s,"\n");
                        strcat(ip_s,"\t");
                        strcat(ip_s,numer);
                        strcat(ip_s,yytext);
                        }

{url_dominio_inseguro}  {numurl++;numdominio++;numprotocolo_inseguro++;
                        char numer[50];
                        sprintf(numer, "Num line: %d \tURL: ", numline);
                        dom_u = realloc(dom_u,strlen(dom_u) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(dom_u,"\n");
                        strcat(dom_u,"\t");
                        strcat(dom_u,numer);
                        strcat(dom_u,yytext);
                        }

{url_dominio_seguro}    {numurl++;numdominio++;numprotocolo_seguro++;
```

```
                        char numer[50];
                        sprintf(numer, "Num line: %d \tURL: ", numline);
                        dom_s = realloc(dom_s,strlen(dom_s) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(dom_s,"\n");
                        strcat(dom_s,"\t");
                        strcat(dom_s,numer);
                        strcat(dom_s,yytext);
                        }

\n                      {numline++;}

[ ]                     {numword++;}

.                       {numchar++;}

%%
```

Al igual que en el anterior apartado, se explicará sección por sección el código para su mejor comprensión.

De la línea 54 a la 92 se definen las acciones de mayor complejitud, siendo estas las que se realizarán cuando se encuentre un patrón complejo dentro de la entrada recibida por el analizador.En este caso únicamente será necesario explicar una de las cuatro acciones ya que en estructura y funciones son prácticamente idénticas, dado que, la función de estas acciones se centra más en clasificar que en realizar acciones dentro del código.

Para este caso explicaremos la acción de `ip_inseguro` siendo esta la encargada de detectar IP con protocolo HTTP:

```
{ip_inseguro}           {numurl++;numipv4++;numprotocolo_inseguro++;
                        char numer[50];
                        sprintf(numer, "Num line: %d \tIP:  ",numline);
                        ip_u = realloc(ip_u,strlen(ip_u) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(ip_u,"\n");
                        strcat(ip_u,"\t");
                        strcat(ip_u,numer);
```

```
strcat(ip_u,yytext);
}
```

Estas líneas corresponden a las líneas de la 54 a la 62, las acciones que se realizan son las siguientes:

1. Se aumentan los contadores pertinentes dependiendo de los requisitos que cumpla la regla
2. Generamos un vector estático auxiliar para almacenar una cabecera que se mostrará previa a la IP/URL
3. Insertamos la cabezera que nos indicará el número de línea donde encontrar la IP/URL en el vector estático
4. Aumentamos el tamaño del vector donde se almacenan todas las ocurrecias de esa regla tanto como sea neceario
5. En las dos siguientes línea insertamos un `\n` y `\t` al vector dinámico para fonmatear la salida
6. Insertamos la cabecera que nos indica el número de línea en el vector dinámico
7. Por último añadimos al vector dinámico la ocurrencia que hemos encontrado correspondiente a la regla

Por último me gustaría destacar que en caso de querere llevar a cabo el suspuesto que hemos mencionado ya varias veces se debería de insertar previo al comando `ip_u = realloc(Parámetros);` una llamada a la función `elimina_car_sobrantes` a la cual le pasemos como parámetro el vector `yytext` el cual en ese caso contendría la ocurrencia en forma bruta de la regla sobre la que se esta llevando a cabo la acción.

Tras esto vamos a explicar las últimas líneas de sección de este código dedicadas al conteo de palabras, líneas y caractéres:

```
\n                      {numline++;}

[ ]                     {numword++;}

.                       {numchar++;}
```

1. En la primera línea al detectar el salto de línea se aumentaría el número de línea *(El cual esta incializado a 1)*
2. En la segunda línea se contarían los espacios en blanco contando así las palabras que hay en el texto.
    i. Quiero destacar que esto sería dando por sentado que el usuario va a ser honesto y no va a poner más de 2 espacios seguidos, en

caso de que no se honesto en vez de `[ ]` deberemos poner `\b`

3. Por último contaremos cada uno de los caracteres para así devolver el número total de los mismos.

Volver al índice

5.1.3 Main

Esta sería la última sección dentro de la explicación del código fuente, como menciona el título de la sección es la que explicaría el repertorio de acciones que realizará el main:

```
int main(){
    ip_u = realloc(ip_u,1);
    ip_s = realloc(ip_s,1);
    dom_u = realloc(dom_u,1);
    dom_s = realloc(dom_s,1);

    yylex();
    printf("Analisis del Prompt Introducido:\nNum lineas:%d\tNum Palabras:%d\tNum Caracteres:%d\n",
    numline,numword,numchar);
    printf("Num URL/IP: %d\nNum Protocolo HTTP: %d\tNum Protocolo HTTPS: %d\tNum IPV4: %d\tNum URL: %d\n",
    numurl,numprotocolo_inseguro,numprotocolo_seguro,numipv4,numdominio);
    if(dom_u!=" "||ip_u!=" "){
        printf("\nURL e IP con protocolo HTTP:\n%s%s\n",dom_u,ip_u);
    }
    if(dom_s!=" "||ip_s!=" "){
        printf("\nURL e IP con protocolo HTTPS:\n%s%s\n",dom_s,ip_s);

    }
    if(ip_u!=" "||ip_s!=" "){
        printf("\nDIRECCION IP:\n%s%s\n",ip_u,ip_s);
    }
    if(dom_u!=" "||dom_s!=" "){
        printf("\nDIRECCION URL:\n%s%s\n",dom_u,dom_s);
```

```
    }

    free(ip_u);
    free(ip_s);
    free(dom_u);
    free(dom_s);

    return 1;
    }
```

Dentro de este código podemos identificar ***3 secciones***, __la primera*__ de la línea 103 a la 109 que serían las encargadas de la **definición inicial de los vectores dinámicos**, ya que como se mencionó anteriormente los vectores dinámicos no pueden incializarse en un contexto global. Además de esto la línea 109 sería la encargada de la ejecución del analizador léxico.

```
  ip_u = realloc(ip_u,1);
  ip_s = realloc(ip_s,1);
  dom_u = realloc(dom_u,1);
  dom_s = realloc(dom_s,1);

  yylex();
```

La segunda sección sería la encargada de **devolver la salida** o bien al fichero de salida o bien a la terminal desde la que estemos ejecutando el código.

```
printf("Analisis del Prompt Introducido:\nNum lineas:%d\tNum Palabras:%d\tNum Caracteres:%d\n",
numline,numword,numchar);
printf("Num URL/IP: %d\nNum Protocolo HTTP: %d\tNum Protocolo HTTPS: %d\tNum IPV4: %d\tNum URL: %d\n",
numurl,numprotocolo_inseguro,numprotocolo_seguro,numipv4,numdominio);
if(dom_u!=" "||ip_u!=" "){
    printf("\nURL e IP con protocolo HTTP:\n%s%s\n",dom_u,ip_u);
}
```

```
if(dom_s!=" "||ip_s!=" "){
    printf("\nURL e IP con protocolo HTTPS:\n%s%s\n",dom_s,ip_s);

}
if(ip_u!=" "||ip_s!=" "){
    printf("\nDIRECCION IP:\n%s%s\n",ip_u,ip_s);
}
if(dom_u!=" "||dom_s!=" "){
    printf("\nDIRECCION URL:\n%s%s\n",dom_u,dom_s);
}
```

Los dos primeros `printf()` sirven para mostrar el conteo de datos obtenido siendo estas de la línea 110 a la 113, tras esto se comenzaría a imprimir los vectores los cuales contengan como mínimo una ocurrencia, por eso las líneas `if(VECTOR != VACÍO)` que corresponden de la línea 114 a la 126.

Por **último la tercera** y última sección corresponden a la **liberación de los recursos** de los vectores dinámicos y `return 1`

```
free(ip_u);
free(ip_s);
free(dom_u);
free(dom_s);

return 1;
```

Volver al índice

5.1.4 Código completo

En esta sección se mostrará el código fuente del fichero AnalizadorLexico que podremos obtener en el directorio Src

```
protocolo_seguro        ("https://")
```

```
protocolo_inseguro      ("http://")
dominio                 (("www"\.)?[a-zA-Z]+[\.]("com"|"es"|"org"|"net"|"edu"|"gov"))
s_ipv4                  (([0-1]?[0-9]?[0-9])|[2][0-4][0-9]|[2][5][0-5])
ipv4                    ({s_ipv4}\.{s_ipv4}\.{s_ipv4}\.{s_ipv4})
ip_seguro               {protocolo_seguro}{ipv4}
url_dominio_seguro      {protocolo_seguro}{dominio}
url_dominio_inseguro    {protocolo_inseguro}{dominio}
ip_inseguro             {protocolo_inseguro}{ipv4}


%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    int numurl=0,numprotocolo_inseguro=0,numline=1,numword=0;
    int numprotocolo_seguro=0,numdominio=0,numipv4=0,numchar=0;
    char* ip_s = NULL;
    char* ip_u = NULL;
    char* dom_s = NULL;
    char* dom_u = NULL;

    void elimina_car_sobrantes(char* vector_car){
        int longitud = strlen(vector_car);
        if(vector_car[0]==' '||vector_car[0]=='\n'||vector_car[0]=='.'|vector_car[0]==','|vector_car[0]=='\b'){
            if(vector_car[0]=='\n'){
                numline++;
            }
            else if(vector_car[0]==' '){
                numword++;
            }
            for (int i = 0; i < longitud; i++) {
                vector_car[i] = vector_car[i + 1];
            }
            vector_car[longitud-1]='\0';
            longitud = longitud-1;
```

```
        }
        if(vector_car[longitud-1]==' '||vector_car[longitud-1]=='\n'||vector_car[longitud-1]=='.'|vector_car[longitud-1]==',
            if(vector_car[longitud-1]=='\n'){
                numline++;
            }
            else if(vector_car[longitud-1]==' '){
                numword++;
            }
            for (int i = 0; i < longitud; i++) {
                vector_car[longitud-1] = vector_car[i + 1];
            }
            vector_car[longitud-1]='\0';
        }
    }
%}

%%
{ip_inseguro}           {numurl++;numipv4++;numprotocolo_inseguro++;
                        char numer[50];
                        sprintf(numer, "Num line: %d \tIP:  ",numline);
                        ip_u = realloc(ip_u,strlen(ip_u) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(ip_u,"\n");
                        strcat(ip_u,"\t");
                        strcat(ip_u,numer);
                        strcat(ip_u,yytext);
                        }

{ip_seguro}             {numurl++;numipv4++;numprotocolo_seguro++;
                        char numer[50];
                        sprintf(numer, "Num line: %d \tIP:  ", numline);
                        ip_s = realloc(ip_s,strlen(ip_s) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(ip_s,"\n");
                        strcat(ip_s,"\t");
                        strcat(ip_s,numer);
                        strcat(ip_s,yytext);
```

```
                        }

{url_dominio_inseguro}  {numurl++;numdominio++;numprotocolo_inseguro++;
                        char numer[50];
                        sprintf(numer, "Num line: %d \tURL: ", numline);
                        dom_u = realloc(dom_u,strlen(dom_u) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(dom_u,"\n");
                        strcat(dom_u,"\t");
                        strcat(dom_u,numer);
                        strcat(dom_u,yytext);
                        }

{url_dominio_seguro}    {numurl++;numdominio++;numprotocolo_seguro++;
                        char numer[50];
                        sprintf(numer, "Num line: %d \tURL: ", numline);
                        dom_s = realloc(dom_s,strlen(dom_s) + strlen(yytext) + strlen((char*)numer) + 3);
                        strcat(dom_s,"\n");
                        strcat(dom_s,"\t");
                        strcat(dom_s,numer);
                        strcat(dom_s,yytext);
                        }

\n                      {numline++;}

[ ]                     {numword++;}

.                       {numchar++;}

%%


int main(){
    ip_u = realloc(ip_u,1);
    ip_s = realloc(ip_s,1);
    dom_u = realloc(dom_u,1);
```

```
dom_s = realloc(dom_s,1);

yylex();
printf("Analisis del Prompt Introducido:\nNum lineas:%d\tNum Palabras:%d\tNum Caracteres:%d\n",
numline,numword,numchar);
printf("Num URL/IP: %d\nNum Protocolo HTTP: %d\tNum Protocolo HTTPS: %d\tNum IPV4: %d\tNum URL: %d\n",
numurl,numprotocolo_inseguro,numprotocolo_seguro,numipv4,numdominio);
if(dom_u!=" "||ip_u!=" "){
    printf("\nURL e IP con protocolo HTTP:\n%s%s\n",dom_u,ip_u);
}
if(dom_s!=" "||ip_s!=" "){
    printf("\nURL e IP con protocolo HTTPS:\n%s%s\n",dom_s,ip_s);

}
if(ip_u!=" "||ip_s!=" "){
    printf("\nDIRECCION IP:\n%s%s\n",ip_u,ip_s);
}
if(dom_u!=" "||dom_s!=" "){
    printf("\nDIRECCION URL:\n%s%s\n",dom_u,dom_s);
}

free(ip_u);
free(ip_s);
free(dom_u);
free(dom_s);

return 1;
}
```

Volver al índice

5.2 Explicación del Test

En este apartado explicaremos brevemente el fichero entrada.txt el cual podremos encontrar en el directorio Test.

Las primeras 6 líneas siven para comprobar que ignora todo aquello que no sea del formato especificado, incluyendo líneas en blanco.

```
Hola a todos

Este texto es un Texto de prueba para comprobar que funciona el Analizador


//URL con protocolo http:
```

Tras esto se van especificando por clasificación todas las URL e IP, para explicar este fragmento nos centraremos únicamente en una de ellas, la que va de la línea 16 a la 24

```
//URL con protocolo https
https://midominio.es                https://midominio.es    //Dos en una misma linea
https://www.midominio.es
https://estenoesmidominio.com
https://www.estenoesmidominio.com
https://estaesundominioincorrecto                           //Este dominio no sería válido
https://www.youtube.org
https://pipo.ugr.es                                         //Este dominio no sería válido
https://12345.com                                           //Este dominio no sería válido
```

El analizador deberá de aceptar tódas las líneas que no tengan ningun comentario comenzando por `//Comentario` a excepción de la línea `//Dos en una misma línea` ya que esta deberá de reconocer 2 URL en una misma línea sin ningún problema.

Lo último que quedaría por comentar serían de la línea 42 a la 50 que serían los resultados que nos debería de proporcionar el conteo de las diferentes clasificaciones obtenidas

```
En total este documento debería de tener:

URL e IP validas: 17
URL validas: 10
URL HTTP: 6
URL HTTPS: 5
IP validas: 6
IP HTTP: 3
IP HTTPS: 3
HTTP:9
HTTPS:8
```

Volver al índice

5.2.1 Archivo de Test Completo

En esta sección se muestra el fichero completo entrada.txt el cual podremos encontrar en el directorio Test.

```
Hola a todos

Este texto es un Texto de prueba para comprobar que funciona el Analizador


//URL con protocolo http:
http://midominio.es
http://www.midominio.es
http://estenoesmidominio.com
Adios http://www.estenoesmidominio.com hola
http://estaesundominioincorrecto                    //Este diminio no sería valido
http://www.youtube.org
http://pipo.ugr.es                                  //Este dominio no sería válido
http://12345.com                                    //Este dominio no sería válido
```

```
//URL con protocolo https
https://midominio.es                https://midominio.es    //Dos en una misma linea
https://www.midominio.es
https://estenoesmidominio.com
https://www.estenoesmidominio.com
https://estaesundominioincorrecto                           //Este dominio no sería válido
https://www.youtube.org
https://pipo.ugr.es                                         //Este dominio no sería válido
https://12345.com                                           //Este dominio no sería válido

//IP con protocolo http
http://1.1.1.1
http://233.22.1.22
http://123.123.123.123
http://333.333.333.333                                      //Esta IP no sería válida
http://12.12.2                                              //Esta IP no sería válida

//IP con protocolo http
https://1.1.1.1
https://233.22.1.22
https://123.123.123.123
https://333.333.333.333                                     //Esta IP no sería válida
https://12.12.2                                             //Esta IP no sería válida

En total este documento debería de tener:

URL e IP validas: 17
URL validas: 10
URL HTTP: 6
URL HTTPS: 5
IP validas: 6
IP HTTP: 3
IP HTTPS: 3
HTTP:9
```

```
HTTPS:8
```

5.3 Estructura del Resultado

Respecto a explicaciones, lo único que quedaría por aclarar sería el formato de la salída al ejecutar el programa, para ello primero mostraré el código completo y explicaré las dos secciones en las que se divide.

Este ejemplo de salida podemos encontrarlo en el fichero salida.txt ubicado en el directorio Results

```
Analisis del Prompt Introducido:
Num lineas:50   Num Palabras:321    Num Caracteres:1758
Num URL/IP: 17
Num Protocolo HTTP: 8   Num Protocolo HTTPS: 9  Num IPV4: 6 Num URL: 11

URL e IP con protocolo HTTP:

    Num line: 7     URL: http://midominio.es
    Num line: 8     URL: http://www.midominio.es
    Num line: 9     URL: http://estenoesmidominio.com
    Num line: 10    URL: http://www.estenoesmidominio.com
    Num line: 12    URL: http://www.youtube.org
    Num line: 27    IP:  http://1.1.1.1
    Num line: 28    IP:  http://233.22.1.22
    Num line: 29    IP:  http://123.123.123.123

URL e IP con protocolo HTTPS:

    Num line: 17    URL: https://midominio.es
    Num line: 17    URL: https://midominio.es
    Num line: 18    URL: https://www.midominio.es
    Num line: 19    URL: https://estenoesmidominio.com
    Num line: 20    URL: https://www.estenoesmidominio.com
```

```
    Num line: 22    URL: https://www.youtube.org
    Num line: 34    IP:  https://1.1.1.1
    Num line: 35    IP:  https://233.22.1.22
    Num line: 36    IP:  https://123.123.123.123

DIRECCION IP:

    Num line: 27    IP:  http://1.1.1.1
    Num line: 28    IP:  http://233.22.1.22
    Num line: 29    IP:  http://123.123.123.123
    Num line: 34    IP:  https://1.1.1.1
    Num line: 35    IP:  https://233.22.1.22
    Num line: 36    IP:  https://123.123.123.123

DIRECCION URL:

    Num line: 7     URL: http://midominio.es
    Num line: 8     URL: http://www.midominio.es
    Num line: 9     URL: http://estenoesmidominio.com
    Num line: 10    URL: http://www.estenoesmidominio.com
    Num line: 12    URL: http://www.youtube.org
    Num line: 17    URL: https://midominio.es
    Num line: 17    URL: https://midominio.es
    Num line: 18    URL: https://www.midominio.es
    Num line: 19    URL: https://estenoesmidominio.com
    Num line: 20    URL: https://www.estenoesmidominio.com
    Num line: 22    URL: https://www.youtube.org
```

De la línea 1 a la 4 sirven para mostrar el conteo realizado durante la ejecución del analizador, desde el número de líneas hasta el numero de IPV4 encontradas.

De la línea 5 en adelante se nos mostrarán de forma clasificada las IP y URL encontradas primero por protocolo HTTP o HTTPS y tras esto se nos mostrarán por separado las IP y las URL obtenidas.

5.4 Ejeccución del Código

Lo único que quedaría por explicar serían los comandos para poder ejecutar el analizador,en esta sección se dará por sentado que se ejecutará desde una copia del repositorio bajada en una máquina con la disto **Ubuntu**.

1. `cd Src/`
2. `flex AnalizadorLexico`
3. Se nos habrá generado un fichero llamado `lex.yy.c`
4. Compilamos el fichero con el comando `gcc lex.yy.c -lfl`
5. Tras esto ejecutaremos a.out
6. En caso de querer hacerlo sin un fichero de entrada ejecutaremos `./a.out`
    i. Tras esto escribiremos todas las IP o URL en el formato especificado o no
    ii. Pulsamos `ctrl+d` y deberíamos de obtener la salida correspondiente
7. En caso de querer hacerlo sin un fichero de entrada ejecutaremos `./a.out < ../Test/entrada.txt > salida.txt`
    i. Debemos de obtener el mismo resultado que `Results/salida.txt`