

ADQUIRIENDO LAS DESTREZAS BÁSICAS PRODUCTO 2

Daniel Carrasco Luque

FP.047 – (P) Programa comandos personalizados para el sistema operativo

ÍNDICE

Introducción.....	2
Insertar fecha y hora	2
Ubicación de “producto2.txt”	3
Funcionamiento sobre la aplicación	4
Verificar IP’s.....	5
Solicitud de ruta y fichero	5
Mostrar contenido por pantalla	5
Verificar ping de ip’s	6
Seleccionar adaptador de red.....	9
Webgrafía	12

Introducción

Esta actividad del proyecto nos ha servido para adentrarnos en el uso de los ficheros como medio de comunicación entre aplicaciones de consola, tratamiento y procesamiento de la información compartida, utilización de los ficheros como medio de almacenamiento temporal, así como, medio para almacenar y exportar el resultado de nuestra aplicación.

En uno de los subproductos hemos podido trabajar con las fechas y horas, pudiendo comprobar la complejidad que supone esto y a la vez la importancia de un buen manejo del tiempo.

El programa desarrollado consta principalmente de 3 funciones, las cuales cada una desarrolla las actividades solicitadas. Posteriormente se ha creado un menú principal para acceder a las 3 funcionalidades. A continuación, se detallará el desarrollo de estas funcionalidades.

Insertar fecha y hora

Esta función pese a ser sencilla y escueta en código ha requerido de bastante investigación. Primeramente, para comprender de que forma se mide el tiempo en nuestras máquinas para posteriormente ver y comprender los métodos que se utilizan en C para poder representar y operar con el tiempo.

Las máquinas cuentan el tiempo transcurrido en segundos desde un evento llamado *epoch*. Este evento tiene diferente inicio dependiendo del sistema en el que estemos:

- UNIX: 01/01/1970
- DOS: 01/01/1980
- Windows NT: 01/01/1601

Utilizando el encabezado `<time.h>` vamos a obtener acceso a diferentes tipos de datos para el tiempo y funciones para manejarlo. En este punto tenemos 2 tipos de datos el tipo “time_t” trabaja con el tiempo de forma numérica permitiendo realizar diversas operaciones (comparación, resta y suma). El tipo “struct tm” almacena en forma de enteros cada uno de los valores que conforman el tiempo, desde segundos, minutos y horas, pasando por días, meses y años, días de la semana e incluso un indicador de horario de verano.

Después de tener un poco de comprensión del funcionamiento del tiempo y buscar diferentes ejemplos de codificación me he lanzado a desarrollar la funcionalidad que se nos solicitaba.

El primer paso ha sido obtener el instante de tiempo para después procesarlo y poderlo almacenar, esto se hace con la siguiente sentencia “time(NULL)”, y la almacenamos en una variable de tipo “time_t”.

```
time_t t = time(NULL);
```

El tiempo obtenido está en formato numérico, pero nos interesa hacer uso de “struct tm” para obtener un formato entendible por los usuarios y acorde al lugar donde se encuentra la máquina. Esto nos lleva a obtener la hora local a partir de “time_t t” utilizando la función

“localtime()” y almacenarla directamente en un “struct tm” quedando la sentencia de esta forma:

```
struct tm* tiempoLocal = localtime(&t);
```

En este punto ya tenemos la fecha y hora de forma representable, lo que necesitamos es construir de alguna forma como representar los datos. Para esto hay diferentes funciones indicadas en <time.h>. La más sencilla de usar es “char *ctime(time_t *tp)” que se usa directamente con tipo de dato “time_t” pero el formato de datos de salida está predefinido y no lo podemos cambiar. “char *asctime(const struct tm *tp)” ofrece la misma salida que “ctime()” pero haciendo uso de “struct tm”. La opción escogida es una tercera función llamada “strftime()” la cual nos permite construir un string a partir de una “struct tm” indicándole los modificadores que queremos usar para darle el formato deseado. Primeramente, requerirá de la definición de un string de salida que hemos llamado “fechaHora”.

```
strftime(fechaHora, sizeof(fechaHora), "%d-%m-%Y %H:%M:%S", tiempoLocal);
```

Modificadores usados:

%d >	Día	%H >	Horas
%m >	Més	%M >	Minutos
%Y >	Año	%S >	Segundos

El paso siguiente ha sido crear o abrir el archivo “producto2.txt que se nos solicitaba en el ejercicio para guardar la fecha y hora. El planteamiento de usar el tipo de permiso “a” o adición en lugar de “w” (borrar contenido), es por si queremos usar el archivo a modo de registro, donde de forma periódica se inserte una fecha y el proceso que venga después.

• Ubicación de “producto2.txt”

El archivo “producto2.txt” se genera en la carpeta raíz del proyecto si ejecutamos el programa desde el IDE de Visual Studio. Si el ejecutable lo usamos una vez compilado fuera del entorno de desarrollo, el archivo “producto2.txt” se generará en la carpeta raíz donde esté contenido el ejecutable

• **Funcionamiento sobre la aplicación**

Desde el menú principal seleccionamos la opción 1 “Insertar fecha y hora en documento”

```
Menu:

> 1. Insertar fecha y hora en documento
> 2. Verificar IP's desde archivo
> 3. Informacion de aptadores de red
> 0. Salir del programa

Escoja opcion: 1|
```

Una vez ejecutado se muestra por pantalla un mensaje confirmando la fecha y hora insertada en el archivo “producto2.txt”. Pulsamos una tecla y retornaremos al menú principal.

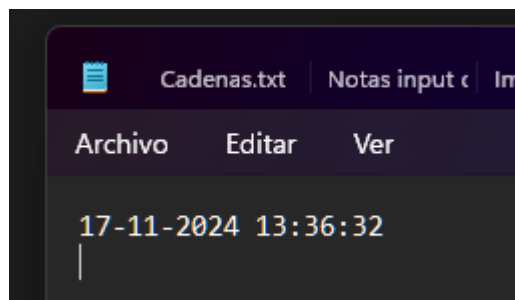
```
Menu:

> 1. Insertar fecha y hora en documento
> 2. Verificar IP's desde archivo
> 3. Informacion de aptadores de red
> 0. Salir del programa

Escoja opcion: 1

Fecha y hora actual insertadas en producto2.txt 17-11-2024 13:36:32.
Presione una tecla para continuar . . . |
```

Si revisamos la raíz del proyecto si se ha ejecutado en IDE o si vamos a la carpeta donde está contenido el ejecutable (uso del ejecutable fuera del proyecto) veremos que se ha creado el fichero con la línea de fecha y hora indicada por el programa.



Verificar IP's

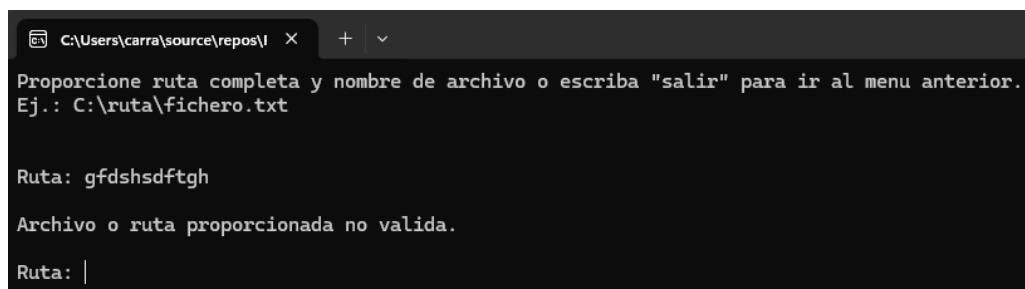
En esta funcionalidad del programa se nos ha requerido varias tareas ha desarrollar para obtener el resultado final.

- Se debe solicitar la ruta y nombre de fichero que contendrá las ip's a verificar.
- Se muestra el contenido del fichero por pantalla.
- Se realiza ping a cada una de las ip's, se muestra por pantalla si es ok o no y se almacena en el fichero "producto2.txt" aquellas ip's que son ok.

• Solicitud de ruta y fichero

Esta funcionalidad no requiere de nada excepcional, se solicita al usuario que introduzca la ruta completa y nombre de fichero dando un ejemplo de como se debe escribir. Ej: "C:\ruta\fichero.txt". Se han implementado 3 posibles soluciones para esta entrada:

- **Nos equivocamos con la ruta:** nos la solicita de nuevo hasta ser ok. Para ello verifica la apertura del fichero indicado con una función auxiliar llamada "verificarRuta()" y nos deriva a seguir intentando la entrada correcta.



```

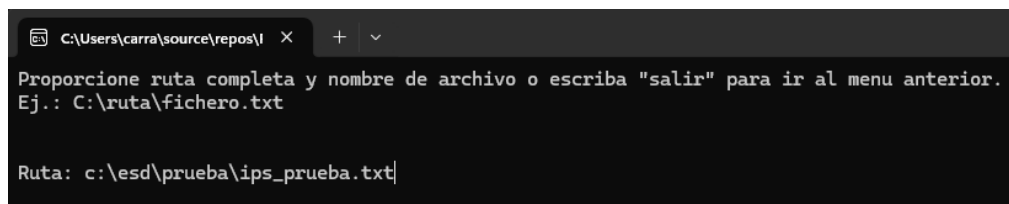
C:\Users\carra\source\repos\I  X  +  v
Proporcione ruta completa y nombre de archivo o escriba "salir" para ir al menu anterior.
Ej.: C:\ruta\fichero.txt

Ruta: gfdshsdftgh

Archivo o ruta proporcionada no valida.

Ruta: |
  
```

- **Ponemos una ruta y fichero existente:** procede con el siguiente paso de mostrar el contenido del fichero por pantalla.
- **Opción de escapar al menú principal:** Si por cualquier motivo no queremos introducir la ruta o no acertamos tenemos la opción de escribir "salir" o "SALIR" y retornaremos al menú principal.



```

C:\Users\carra\source\repos\I  X  +  v
Proporcione ruta completa y nombre de archivo o escriba "salir" para ir al menu anterior.
Ej.: C:\ruta\fichero.txt

Ruta: c:\esd\prueba\ips_prueba.txt|
  
```

• Mostrar contenido por pantalla

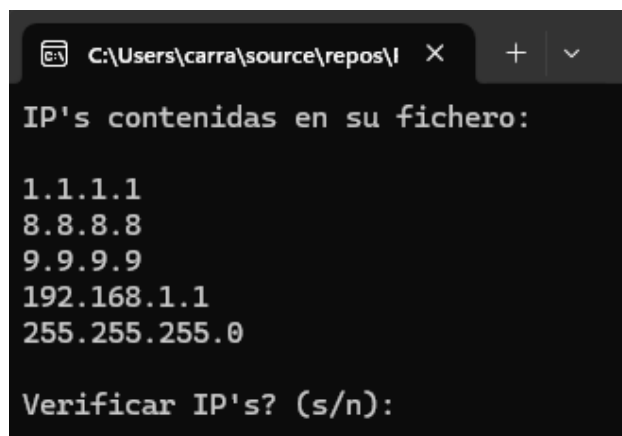
Para realizar esta parte creamos un puntero de tipo FILE al archivo introducido por el usuario en modo lectura "r", damos algo de formato en pantalla limpiándola e indicando el tipo de información que vamos a mostrar, para posteriormente, recorrer el fichero línea a línea mediante la combinación de un bucle "while" con un "fgets" de esta forma almacenamos la línea leída por el puntero del fichero en un string, creado previamente, que después podremos tratar para lo que necesitemos, en este caso mostrarlo por pantalla.

Este punto ha sido básico para posteriormente implementar todas las funcionalidades de lectura y búsqueda que se desarrollarán más adelante.

```
char lineaOrigen[100];
while (fgets(lineaOrigen, sizeof(lineaOrigen), fileListIp)) {
    printf("%s", lineaOrigen);
}
```

Llegados a este punto se muestra por pantalla las ip's contenidas en el fichero y una pregunta de tipo si o no donde se solicita si queremos verificar las ip's.

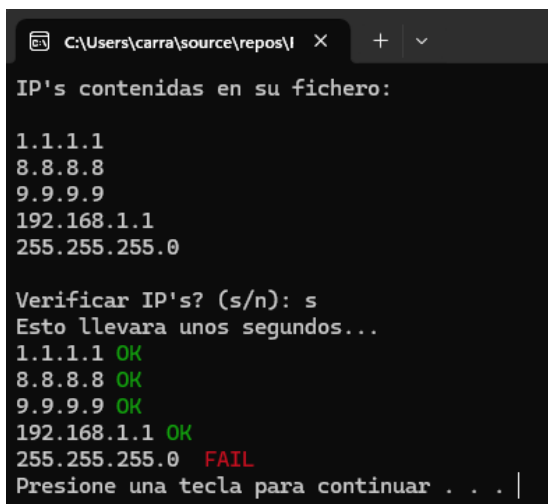
- Indicando "n" nos llevará al menú principal
- Indicando "s" comenzará el proceso de verificación de las ip's.



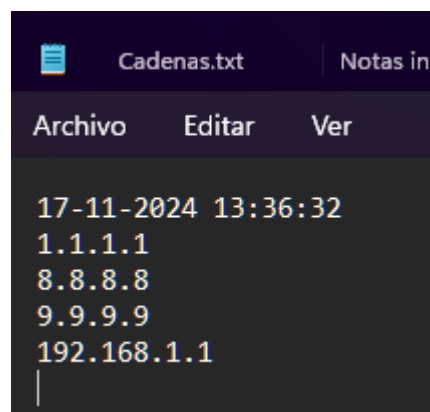
```
C:\Users\carra\source\repos\I X + v
IP's contenidas en su fichero:
1.1.1.1
8.8.8.8
9.9.9.9
192.168.1.1
255.255.255.0
Verificar IP's? (s/n):
```

• Verificar ping de ip's

Al indicar "s" comienza la verificación de las ip y podremos ver como el programa realiza las comprobaciones de las ip's en tiempo real, mostrando la ip que está tratando en el momento y si tiene éxito o no indicándolo con un OK en verde y un FAIL en rojo. Las ip's que han ido marcando ok por pantalla se han ido almacenando en el fichero de salida "producto2.txt.



```
C:\Users\carra\source\repos\I X + v
IP's contenidas en su fichero:
1.1.1.1
8.8.8.8
9.9.9.9
192.168.1.1
255.255.255.0
Verificar IP's? (s/n): s
Esto llevara unos segundos...
1.1.1.1 OK
8.8.8.8 OK
9.9.9.9 OK
192.168.1.1 OK
255.255.255.0 FAIL
Presione una tecla para continuar . . . |
```



```
Cadenas.txt | Notas in
Archivo Editar Ver
17-11-2024 13:36:32
1.1.1.1
8.8.8.8
9.9.9.9
192.168.1.1
|
```

Nota: Es importante indicar que no se ha implementado ningún proceso que verifique si las ip's a procesar son realmente ip's con formato valido, es una tarea que ha quedado pendiente. Requiere aprender desde cero el tratamiento de expresiones regulares en C usando <regex.h> y no contaba con el tiempo suficiente para embarcarme en ese proceso. Queda pendiente de acabar de averiguar para el siguiente producto donde es muy posible que sea necesario.

Para llevar a cabo el proceso de verificar las ip's del fichero, debemos de volver al inicio de este, ya que está abierto como lectura y nos sigue interesando acceder de este modo, pero desde el inicio. Para ello utilizamos la siguiente función aplicada al puntero que enlaza con el fichero proporcionado por el usuario:

```
rewind(fileListIp);
```

Una vez en el inicio volvemos a usar un while combinado con un fgets para leer de nuevo línea a línea y extraer de nuevo las ip's. La ip leída directamente del fichero tiene el inconveniente de contener una “\n” al final y se debe quitar para no tener problemas al ejecutar el ping. Para este cometido se ha creado una función auxiliar llamada “limpiarBarraN()” a la cual se le pasa la cadena origen y devuelve la cadena de salida sin “\n”. En este punto estamos listos para crear la sentencia personalizada con la ip objetivo que lanzaremos mediante system(). Para ello hemos usado un seguido de concatenaciones hechas con la función “strcat()”:

```
strcat(instruccion, "ping ");
strcat(instruccion, ip);
strcat(instruccion, " > ");
strcat(instruccion, FILE_PING);
// llamada a ping
system(instruccion);
```

Una vez hecho esto solamente debemos abrir el fichero y analizar los datos que nos ha devuelto. Se ha hecho un estudio sobre qué datos retorna el comando ip cuando se ejecuta sobre una ip correcta. Posteriormente se ha buscado lo que pasa cuando la ip no conecta.

Una vez vistas las igualdades y diferencias de estas 2 casuísticas se ha definido la estrategia de buscar el string “Paquetes: enviados”. De esta forma obtenemos la línea del fichero que nos facilitará si el ping es “OK” o “FAIL”. Esto se realiza mediante el uso de la función “strstr()” dándole por parámetro la línea en la cual se debe buscar el fragmento a encontrar, en el primer uso “Paquetes: enviados”. En la segunda búsqueda, usando la misma metodología buscamos mediante “strstr()” “recibidos = 4”. Si localizamos esta frase sabremos que ping ha sido exitoso al 100% sobre la ip objetivo, si por el contrario strstr() retorna NULL sabremos que no se han recibido el 100% de los paquetes y el ping lo daremos como “FAIL”.

Cuando una ip da ok procedemos a guardarla en el fichero “producto2.txt” abierto previamente en modo adición “a”.

En gris se marca lo analizado en los diferentes ping realizados en el estudio:

```

Windows PowerShell
PS C:\Users\carra> ping 1.1.1.1

Haciendo ping a 1.1.1.1 con 32 bytes de datos:
Respuesta desde 1.1.1.1: bytes=32 tiempo=7ms TTL=55
Respuesta desde 1.1.1.1: bytes=32 tiempo=16ms TTL=55
Respuesta desde 1.1.1.1: bytes=32 tiempo=6ms TTL=55
Respuesta desde 1.1.1.1: bytes=32 tiempo=7ms TTL=55

Estadísticas de ping para 1.1.1.1:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
        (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 6ms, Máximo = 16ms, Media = 9ms
PS C:\Users\carra>
  
```

```

Windows PowerShell
PS C:\Users\carra> ping 255.255.255.0

Haciendo ping a 255.255.255.0 con 32 bytes de datos:
PING: error en la transmisión. Error general.
PING: error en la transmisión. Error general.
PING: error en la transmisión. Error general.
PING: error en la transmisión. Error general.

Estadísticas de ping para 255.255.255.0:
    Paquetes: enviados = 4, recibidos = 0, perdidos = 4
        (100% perdidos),
PS C:\Users\carra>
  
```

```

Windows PowerShell
PS C:\Users\carra> ping 1.1.1.1

Haciendo ping a 1.1.1.1 con 32 bytes de datos:
Respuesta desde 1.1.1.1: bytes=32 tiempo=7ms TTL=55
Respuesta desde 1.1.1.1: bytes=32 tiempo=16ms TTL=55
Respuesta desde 1.1.1.1: bytes=32 tiempo=6ms TTL=55
Respuesta desde 1.1.1.1: bytes=32 tiempo=7ms TTL=55

Estadísticas de ping para 1.1.1.1:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
        (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 6ms, Máximo = 16ms, Media = 9ms
PS C:\Users\carra>
  
```

Seleccionar adaptador de red

Para obtener los adaptadores de red de una máquina usamos el comando “ipconfig” esto nos mostrará según la máquina un listado más grande o pequeño según la máquina en cuestión. En el caso de mi ordenador me ha mostrado un listado con 5 pero de los cuales solamente 2 contienen direcciones IPv4 con máscara de red y puerta de enlace, esto me ha hecho decidir que no quiero mostrar los adaptadores que no tengan IPv4. Por lo cual, las palabras claves que debo buscar son “Adaptador”, para encontrar la línea en el fichero de texto del nombre del adaptador, y posteriormente filtrar los adaptadores por palabra clave “IPv4”, descartando los adaptadores que no tengan la segunda búsqueda. De esta forma ya tenemos la estrategia definida para obtener la información necesaria a mostrar por pantalla.

```
PS C:\Users\carra> ipconfig

Configuración IP de Windows

Adaptador de Ethernet Ethernet:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de Ethernet Ethernet 2:

    Sufijo DNS específico para la conexión. . :
    Vínculo: dirección IPv6 local. . . : fe80::deb:8e40:930f:453e%18
    Dirección IPv4. . . . . : 192.168.56.1
    Máscara de subred . . . . . : 255.255.255.0
    Puerta de enlace predeterminada . . . . . :

Adaptador de LAN inalámbrica Conexión de área local* 1:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Conexión de área local* 10:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Wi-Fi:

    Sufijo DNS específico para la conexión. . : home
    Vínculo: dirección IPv6 local. . . : fe80::c0ee:7fb6:dcd3:8ea8%15
    Dirección IPv4. . . . . : 192.168.1.13
    Máscara de subred . . . . . : 255.255.255.0
    Puerta de enlace predeterminada . . . . . : 192.168.1.1
PS C:\Users\carra> |
```

Lanzamos mediante `system()` la sentencia construida mediante `strcat()` que nos generará el archivo “ipconfig.txt”

```
strcat(instruccion, "ipconfig > ");
strcat(instruccion, IPCONFIG_FILE);
system(instruccion);
```

Abriremos el fichero generado en modo lectura “r” y un segundo fichero “temp.txt” en modo “w” para evitar problemas en caso de que el archivo ya existiera. En el fichero temporal recogeremos los adaptadores susceptibles de ser escogidos para guardar en el documento definitivo.

Comenzamos la lectura del fichero “ipconfig.txt” de nuevo usando `while` combinado con `fgets()` para realizar primeramente la búsqueda de “Adaptador”, si encontramos una línea que contenga el nombre de un adaptador la guardamos en un string auxiliar para recuperarlo más tarde, seguidamente buscamos líneas que contengan “IPv4”. Según avancemos por las líneas del fichero se irá guardando el ultimo adaptador encontrado y solamente se procederá a guardar el adaptador y sus datos si encontramos una línea con “IPv4”.

En este punto se me pasaron por la cabeza varias formas de almacenar esta información para después recuperarla, pero finalmente opté por guardar todos los adaptadores elegibles en el fichero “temp.txt” de forma indexada, de esta forma obtengo 2 ventajas, mostrar la información por pantalla de forma ordenada y una forma sencilla de seleccionar el adaptador, por parte del usuario, para almacenarlo en el fichero de salida.

Para desarrollar el sistema de indexación simplemente utilizo un contador que se incrementa en cada interacción donde se guarda un adaptador y se le da un formato específico. Para ello lo más, destacable es que se debe convertir el valor de contador de tipo “int” a un string.

```
// Convertir contador de int a string
sprintf(contadorStr, "%d", contador);

// Asignar indice al adaptador
strcat(lineaConcat, "> ");
strcat(lineaConcat, contadorStr);
strcat(lineaConcat, ". ");
strcat(lineaConcat, lineaAux);
```

De esta forma hemos obtenido un índice con formato “> 1. “ hasta “> 99. “ lo cual nos permite almacenar en un fichero hasta 99 Adaptadores de red, más que suficiente para nuestro propósito.

La información del archivo temporal quedará de la siguiente forma almacenada

```

Cadenas.txt  Notas input output  Implementacion k  Permisos FILE.txt  ipco

Archivo  Editar  Ver

> 1. Adaptador de Ethernet Ethernet 2:
Dirección IPv4. . . . . : 192.168.56.1
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . :
> 2. Adaptador de LAN inalámbrica Wi-Fi:
Dirección IPv4. . . . . : 192.168.1.13
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . : 192.168.1.1
  
```

Nota: En este punto se puede observar que tengo problemas con ciertos caracteres especiales y no he logrado evitar que esto pase. Los ficheros obtenidos están en formato ANSI en lugar de UTF-8, me tocará seguir investigando sobre esto.

Ahora solamente nos toca mostrar esta información por pantalla para que el usuario pueda escoger el adaptador que le interesa escoger. Para ello hemos cerrado el archivo temporal que estaba en escritura y lo abrimos para lectura “r”. Posteriormente damos formato a la pantalla indicando lo que vamos a mostrar y utilizamos un ya clásico while combinado con fgets() para mostrar todas las líneas del fichero por pantalla, junto a una línea de petición para escoger la opción a modo del menú principal.

```

C:\Users\carra\source\repos\I  X  +  v

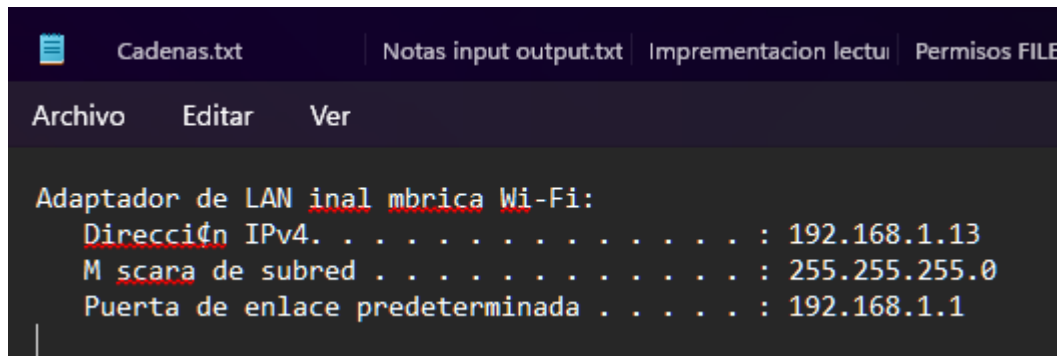
Listado de adaptadores de red escogibles:

> 1. Adaptador de Ethernet Ethernet 2:
Dirección IPv4. . . . . : 192.168.56.1
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . :
> 2. Adaptador de LAN inalámbrica Wi-Fi:
Dirección IPv4. . . . . : 192.168.1.13
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . : 192.168.1.1

Escoja opcion: |
  
```

Escogemos el adaptador y comenzará el proceso de búsqueda en “temp.txt” y posterior guardado en el fichero “adaptador.txt”. Para guardar la información de forma correcta en el archivo de destino se ha creado una función auxiliar llamada “eliminarIndexación()” la cual se le pasan 2 strings por argumento, la cadena de entrada y la de salida, recorre el string de entrada hasta la “A” de adaptador y copia el string restante en el string de salida, obteniendo así un string sin índice y copiándolo en el fichero de salida “adaptador.txt” junto a la ip, máscara y puerta.

Fichero “adaptador.txt” resultante:



The screenshot shows a text editor window with a dark theme. The title bar at the top lists several files: 'Cadenas.txt', 'Notas input output.txt', 'Implementacion lectu...', and 'Permisos FILE...'. Below the title bar is a menu bar with 'Archivo', 'Editar', and 'Ver'. The main text area contains the following content:

```
Adaptador de LAN inal mbrica Wi-Fi:
Dirección IPv4. . . . . : 192.168.1.13
M scara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . : 192.168.1.1
```

Webgrafía

- [¿Cómo usar colores en C?](#)
- [Varios COLORES en tus PRINTF](#)
- [Manejo de hora y fecha en C](#)
- [c++11 - Mostrar fecha y hora actual en C++ - Stack Overflow en español](#)
- [escribir fecha en un fichero - Foros del Web](#)