FACULTAD DE INFORMATICA

Seminario de Lenguaje C

Informe Trabajo Final

GRUPO 40
Curccio, Elian
Giambruni, Cristian
Giraldo, José

Informe Trabajo Final
e-mail: cgcrack86@gmail.com

Índice de contenido

Introducción

- Trabajo en equipo
- Implementación

Funcionamiento del programa

- Funcionamiento en modo interactivo
    - Cargar texto base desde el programa.
    - El archivo de menú
    - Creación manual del conjunto de datos

- Enviando un parámetro
- Enviando dos parámetros
- Enviando tres parámetros
- Enviando cuatro parámetros

Detalle de algunas funciones

- Combinación de Textos
- Función “Retorna Orden”
- Archivos de recursos
- ¿Memoria o archivo?
- Contenido de archivos fuente
- Problemas con makefile

Introducción

El trabajo en equipo

Desde el comienzo del desarrollo de la aplicación para el trabajo final del Seminario de Lenguaje C, vimos el problema en forma general, sin tener noción de los casos puntuales aún. No fue sencillo el trabajo en equipo, tuvimos al principio algunos problemas para separar correctamente las partes del problema haciendo del problema total (el enunciado) sub problemas totalmente individuales, probarlos y una vez que estábamos seguros que funcionaba para los casos críticos que pensamos que podrían ser una complicación para el programa, venía la parte de mezclar esos módulos y hacer que funcione.

Implementación

La aplicación fue desarrollada en el compilador gcc de DEV-C++ 4.9.9.2, ejecutándose bajo Windows XP.

Esperando que la aplicación no pierda méritos por no ser desarrollada en Linux, a continuación explico el porque de esta decisión.

Los tres integrantes no poseemos de máquinas que soporten Linux, o mejor dicho, tuvimos complicaciones en el momento de crear varias particiones en un mismo disco para poder instalar Linux. Al principio teníamos la idea de hacer una aplicación compatible para los dos sistemas, o mejor dicho, tener dos versiones.

Cabe aclarar que la aplicación no fue probada en gcc de Linux, por eso rogamos que se nos corrija bajo Windows.

Funcionamiento del programa

Funcionamiento del modo interactivo

Como pide el enunciado, debe ser un programa capaz de realizar una combinación de textos con campos variables, asociando a cada campo con un dato de un conjunto de datos.

El programa recibe o no, argumentos, como pueden ser el nombre del archivo base, el nombre del archivo donde se encuentra el conjunto de datos, y la forma de mostrar el resultado de la combinación (por pantalla o en un archivo que el usuario especifica el nombre).

De la siguiente manera se llamará al programa sin argumentos.

```
combinar
```

De esta manera se inicia el "modo interactivo" del programa. Como el programa también es capaz de cargar un archivo de texto, se pedirá por teclado que se ingrese el texto, los "<<>>". Este proceso de escribir en el programa finaliza cuando llega el carácter '&'.

Este archivo se guarda en un archivo temporal, pero luego es subido a memoria.

Cargar texto base desde el programa (solo modo interactivo)

También el programa dispone de un archivo como si fuera un menú, donde estarán algunos campos posibles que se pueden reemplazar en diferentes textos. Por ejemplo, el archivo de menú dispone desde su programación con los campos fecha y hora, y es indispensable para que el programa funcione que existan estos campos. Además de fecha y hora el programa también incluye (aunque no sean necesarios) los campos nombre, apellido, edad. Son datos personales básicos que es probable que un texto quiera reemplazar esos campos por otro nombre, apellido, etc. A su vez, el programa dispone de una función que puede ir agregando posibles campos variables al menú.

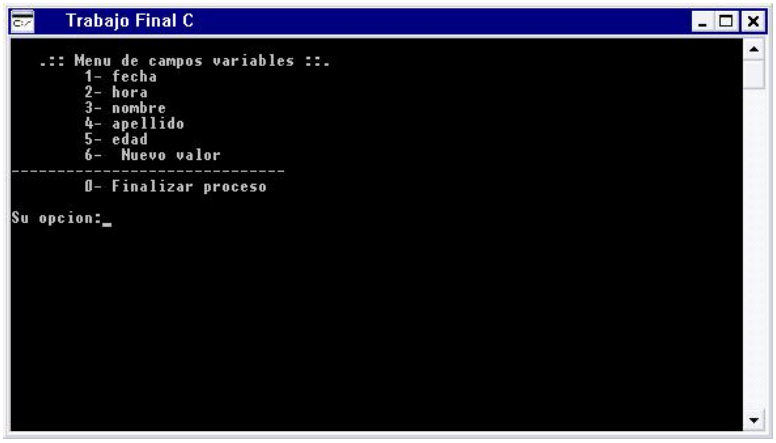


Fig 1- Función para llenar los campos variables de un texto base al cual todavía no se especificaron.

En la figura 1 se puede ver con claridad que además de los campos predefinidos en el menú, se cuenta con dos funciones mas; la de terminar con el proceso y la de agregar un nuevo valor. Al agregarse el nuevo valor, se agrega en la lista de campos variables.

Por ejemplo, si se ingresa un texto base como el siguiente.

```
<<>>.
Al alumno <<>> <<>>.
Se le comunica mediante esta carta que el examen de <<>> ha sido corregido.
Su nota es de <<>>.
Atte. La facultad.
```

Una vez ingresado el texto se le preguntará al usuario si desea cambiar alguna línea del texto. Se debe especificar la línea e ingresar el nuevo texto.
Después de esto aparecerá en pantalla con un listado similar al de la figura 1.

El orden que se ingresarán los campos en el texto base, será de acuerdo con el orden del ingreso. Ej.: se ingresan: fecha, nombre, apellido, materia (es agregado con la opción “Nuevo valor”, y nota (idem anterior).
El texto base quedará de la siguiente manera:

```
<<f>>.
Al alumno <<nombre>> <<apellido>>.
Se le comunica mediante esta carta que el examen de <<materia>> ha sido corregido.
Su nota es de <<nota>>.
Atte. La facultad.
```

La fecha y hora en esta instancia no se especifican, eso se hará en la *definición de conjuntos*.

Se debe recordar que esta parte del programa no tiene en cuenta la cantidad de campos variantes que tiene el texto base, por lo cual pedirá datos hasta que el usuario ingrese la función 0. Los datos que están de más, no serán tomados en cuenta. No hubo tiempo para implementar una mejora a esto, pero si se pensó. La idea sería pedir n veces (n= cantidad de “<<>>”) que se ingresen campos, que no se pueda cancelar el proceso hasta que no se hayan ingresado todos. De esa manera evitaríamos campos vacíos y el no ingresar campos de más.

El archivo de menú

El archivo de menú, es del que se habló anteriormente. El formato físico del archivo es: fecha|hora|opcion_3|… |opcion_n|.
Como se explicó antes, fecha y hora deben estar, después los campos se pueden ir agregando, pero no borrarlos desde el programa, se pueden borrar manualmente abriendo el archivo de menú. No es deseable borrar un conjunto así, pero en esta versión no incluye el borrado de una opción del menú desde el programa.
El archivo se guarda en: `directorio_ del_ programa\source\menu.dat`

Creación manual de conjuntos de datos

Se hará referencia a esta función cuando al programa no le vengan argumentos y también cuando se especifique solo el archivo de texto base.

La función recorre el texto base ya con los campos definidos, extrae su información y pide el ingreso de datos. Por ejemplo, sigamos con el texto base anterior el primer campo a pedir será la fecha. Por lo que la función imprimirá:

```
Ingrese dato para ‘fecha’ o 0 para finalizar:
```

En el segundo campo (nombre) pedirá de la siguiente manera:

```
Ingrese dato para ‘nombre’:
```

Solo en el primer campo de un conjunto se podrá cancelar el ingreso de datos, esto es así para que existan todos los datos.

¿Y que hay con fecha y hora? , en la especificación se desplegará un menú con los formatos de fecha u hora (según lo que se haya especificado). Las fechas irán desde el formato 0 al formato 3, y hora desde el 0 al 1.

Un archivo de conjuntos para el texto base definido puede ser el siguiente (se mostrará con su formato)

(lo que está entre comentarios no forman parte del formato del conjunto)

```
/*conjunto 1*/
f|0
nombre|Cristian
apellido|Giambruni
materia|Seminario de Lenguaje C
nota|9
*     /*el ‘*’ indica el fin del conjunto*/
/*conjunto 2*/
f|0
nombre|Elian
apellido|Curccio
materia|Inglés
nota|8
*
…
/*conjunto n*/
f|0
nombre|Jose
apellido|Giraldo
materia|IBD
nota|4
*     /*fin de conjunto n*/
/*fin del archivo de conjunto*/
```

Como se puede observar en todos los campos coinciden el conjunto de fecha (f|0), eso es porque cuando se está definiendo el conjunto de datos, la fecha solo se especifica solo una vez al igual que la hora.

El proceso de creación del conjunto, se va haciendo sobre un archivo temporal en:

`directorio_ del_ programa\source\crt.dat`

Casi olvido decir que los nombres de los archivos que se guardan temporalmente, están definidos como constantes en el programa. Los nombres que se dieron y los que se van a dar, son los predefinidos por nosotros, que pueden tomar otro nombre si se cambia.

Lo que resta del funcionamiento del programa en el modo interactivo es genérico a los demás:

- Pregunta si se quiere modificar algún conjunto (modificar línea o eliminación del conjunto entero).
- Pregunta si se desean imprimir los resultados por algún orden en especial (luego se explicará como funciona).
- Se pregunta si solo se quiere imprimir un rango del conjunto de datos.
- Se hace la combinación con el texto base y el conjunto especificados.
- Se imprime por pantalla.
- Pregunta si se desea guardar el archivo en el disco y el nombre con que se guardará.
- Fin del programa.

Enviando un parámetro al programa

Nuestro programa asume que si entra un parámetro al programa es porque:

1) `combinar [nombre_archivo]`
2) `combinar [/h]`

Las comprobaciones de la cantidad de argumentos que son enviados al programa son tratados por `main.c` que solo hace un `switch` con la cantidad de argumentos que llegaron, en caso que sean demasiados argumentos informa "`[error] Demasiados parámetros`" y cierra el programa.

Cuando el `argc` indica 2 quiere decir que se ingresó un parámetro. Y `main.c` lo deriva a la función `modo2` que está en `programa.c`.

Mas adelante explico los archivos fuentes del proyecto y sus funciones.

En el caso de 1) se toma "nombre_archivo" como un texto base existente. Y el programa avanza como en el modo interactivo, solo que no pide el ingreso del texto base por teclado y asume que el archivo base es correcto y que no se modificarán alguna línea.

En el caso de 2) imprimirá la una muy breve explicación del programa con sus argumentos. El archivo de ayuda está en `directorio_ del_ programa\source\ayuda.dat`

En caso de no encontrarse el archivo de ayuda, devuelve un mensaje de error.

Enviando dos parámetros al programa

```
combinar [nombre_archivo] [nombre_archivo_conjunto]
```

La aplicación, cuando recibe dos parámetros, asume que el primero de ellos es el texto base, y el otro parámetro es un archivo de conjunto existente. El programa dispone de una función que verifica si el archivo de conjunto es válido. Porque puede pasar que el usuario confunda el orden de ingreso de los archivos y ponga que el primer archivo corresponde al de conjunto y el segundo al texto base. En caso que el archivo de conjunto sea inválido se cerrará la aplicación informando que el conjunto es inválido.

Las instrucciones que sigue el programa son:

- Pregunta si se quiere modificar algún conjunto (modificar línea o eliminación del conjunto entero).
- Pregunta si se desean imprimir los resultados por algún orden en especial (luego se explicará como funciona).
- Se pregunta si solo se quiere imprimir un rango del conjunto de datos.
- Se hace la combinación con el texto base y el conjunto especificados.
- Se imprime por pantalla.
- Pregunta si se desea guardar el archivo en el disco y el nombre con que se guardará.
- Fin del programa.

Enviando tres parámetros al programa

```
combinar [nombre_archivo] [nombre_archivo_conjunto] [/p]

combinar [nombre_archivo] [nombre_archivo_conjunto] [/P]
```

Se verifica si el tercer argumento es ‘/p’ ó ‘/P’. En ese caso es igual al procedimiento anterior, solo que no pregunta si se quiere guardar el resultado de la combinación en disco, ya que se ha especificado que solo quiere verse el resultado por pantalla.

En caso que los argumentos pasados no sean ‘/p’ ó ‘/P’ hará el proceso del programa exactamente como si se hubieran ingresado dos parámetros, pero informa que se ingresó un parámetro desconocido.

Enviando cuatro parámetros al programa

```
combinar [nombre_archivo] [nombre_archivo_conjunto] [/a] [arch_destino]

combinar [nombre_archivo] [nombre_archivo_conjunto] [/A] [arch_destino]
```

Se verifica si el tercer argumento es ‘/a’ ó ‘/A’. En este modo el programa solo imprime los resultados en el archivo indicado por “arch_destino”. Si dicho archivo existe se lo reemplaza sin preguntar.

En caso que el tercer argumento no sea '/a' ó '/A', el programa no hará nada y saldrá indicando un error al ingreso de parámetros.

Detalle de algunas funciones

Combinación de textos

Cabe mencionar que la función combinar texto, fue una de la que mas problema nos trajo desde un principio. En un primer momento el algoritmo se torno algo engorroso y difícil de comprender, por lo que decidimos utilizar la recurcion; transformando así el antiguo código, en otro mucho mas legible y obviamente menos extenso.

Básicamente lo que se intento, fue trabajar con cada línea del *"texto base*" en particular, analizando el caso de que posea algún *campo variable,* de ser así, se extraía el contenido de este, ej. **nombre**, y luego de ser buscado en el *conjunto de datos*, ingresado previamente por el usuario, se cargaba con el dato real, ej. ***Cristian***, para enviar así nuevamente la línea con la reciente modificación de forma recursiva. Este caso se va a repetir hasta que en la línea no se encuentre ningún campo variable *"caso base"*, en donde se procede a escribirla directamente en el archivo.

Ejemplo:

Línea 1:

Al alumno <<**nombre**>> <<**apellido**>>

La línea que se reenvía a la función por recursión, luego del análisis es la siguiente:

Al alumno **Cristian** <<**apellido**>>

Como aun posee *campo variable* se efectúa el correspondiente cambio, quedando del siguiente modo:

Al alumno **Cristian Giambruni**

Y ahora si, al ser enviada, entrara por el *caso base*, escribiéndose de este modo en el archivo.

Cabe aclarar, que de no existir el contenido del *campo variable*, se escribirá como *dato real*, la frase ::NO EXISTE DATO::

Como restricción al momento de ingresar los *campos variables*, el único carácter prohibido es el '>', de obviarse esta aclaración, el programa no se colgaría, pero a pesar de que dicho campo exista en el *conjunto de datos*, el *dato real* será ::NO EXISTE DATO::, como se muestra en el siguiente ejemplo. :

Ejemplo:

Texto base

Al alumno <<**nonbre**>> <<**ape>llido**>>

Conjunto de Datos

Nombre|Cristian

```
Ape>llido|Giambruni
```

El resultado final será:

```
Al alumno ::NO EXISTE DATO:: ::NO EXISTE DATO::
```

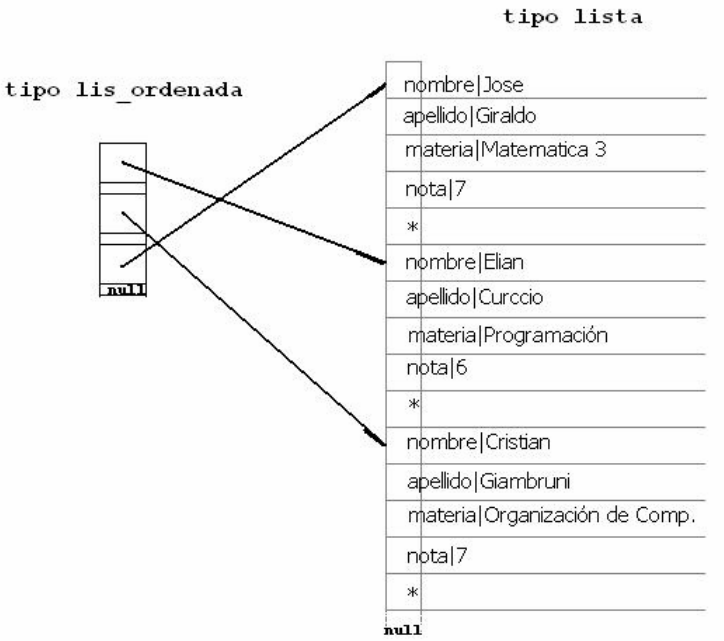
Función “Retorna orden”

Con la función `retorna_orden`, tratamos de que quede avalado, no solo la posibilidad de que el usuario desee imprimir el/los textos bases utilizando el/los conjuntos de datos en algún orden especifico, sino que también permite hacerlo considerando algún rango en especial, es decir, si existieran 5 conjuntos de datos y el usuario solo desea trabajar con los datos desde del conjunto 2 hasta el 4, quedarían de esta forma, descartados los conjuntos 1 y 5.

La función recibe el criterio por el cual se desea ordenar los datos en forma de puntero a `char`; la lista con el total de los conjuntos de datos; y dos variables de tipo entero que indican el rango “comienzo y fin” de conjuntos con el cual se desea trabajar; y `retorna` un puntero del tipo “lis_ordenada”.

En caso de que se desee procesar los datos sin ningún orden en especial `criterio == “0”`.

A continuación se ejemplifica el caso de la devolución de una lista de tipo “lis_ordenada”, ordenada por el *criterio* **apellido.**



De esa manera se ordenan los datos en un conjunto por apellido.

Archivos de recursos

El programa usa como directorio auxiliar `directorio_ del_ programa\source\`. Esto se hizo para que los archivos que se crean por necesidad del programa no estorben el directorio del programa, la cual debe ser usada por el usuario.

En este directorio se encuentran los archivos que se definen en “defines.h”.

¿Memoria o archivo?

La idea general del programa era subir todo a memoria principal y trabajar de esa manera, pero esta idea no se cumple en todos los casos.

Decidimos separar estas funciones. Por un lado tenemos al “gran archivo” listas.c que en el se hacen la mayoría de funciones que tengan que ver con la memoria; cargar a memoria un archivo, enlistar cadenas, etc.

De igual manera quisimos de ser auto-explicativos con los nombres de funciones y variables.

Contenido de los archivos fuente

listas.c como se explicó antes, contienen la mayoría de las referencias a memoria.

archivos.c contienen referencias a archivos, como la función de guardar una lista en disco, etc.

programa.c tiene todas las funciones de los modos según los parámetros ingresados.

defines.h es el archivo de constantes simbólicas y de macros. Es muy recomendable analizar este archivo para la correción.

Luego están los correspondientes .h de cada archivo mencionado y obviamente el main.c.

Problemas con makefile

En la “recta final” del proceso de programación, surgió el problema en cuanto a la compilación de los archivos desde la consola.

Después de probar métodos que páginas webs publicaban y de verificar que para Dev C++ no funcionaba, indagamos por otro lado.

El compilador que usa Dev C++ es el gcc (obviamente no es el de Linux). Y en el archivo compilar.bat está el resultado de la compilación que ofrece gcc. Pero desde la consola no nos funciona.

Por esto se vio demorada nuestra entrega.

Pretendemos que aún de esto, se analice el programa, ya que nuestro empeño en desarrollar una aplicación que resuelva el problema de “combinación de correspondencia” se ha resuelto, y en nuestras pruebas ha salido exitoso.

Para que lo puedan probar por si se puede compilar, dejamos el proyecto .dev disponible para que lo puedan analizar en su entorno.