(Con respecto a las varias faltas ortograficas que se veran a continuacion (sobre todo la falta de tildes), debo aclarar que estoy escribiendo en un teclado ingles, y todavia no he encontrado en Linux, la forma de ponerle las tildes a las vocales).

Cuestionario

1. Un *debugger* es un programa diseñado para probar, identificar y eliminar los errores de otros programas. La **importancia** de un buen depurador (dicho en español) no puede ser exagerada. De hecho, la existencia y la **calidad** de tal herramienta para un lenguaje y una plataforma dadas a menudo puede ser el **factor de decisión** en su uso, incluso si otro lenguaje/plataforma es más adecuado para la tarea. Mas alla de todo, esta **sigue siendo en buena medida una actividad manual** (*debugging*), que desafía la paciencia, la imaginación y la intuición de programadores. Muchas veces se requiere incluir en el código fuente instrucciones auxiliares que permitan el seguimiento de la ejecución del programa.

   Algunos depuradores operan en un simple lenguaje específico mientras que otros pueden manejar múltiples lenguajes transparentemente.

2. En programación, especialmente en depuración del programa, un punto de parada o breakpoint, es una **pausa intencional y controlada durante la ejecución de un programa**. Al detener el proceso podemos inspeccionar el valor de las variables, los argumentos de la pila, y las direcciones de memoria sin que el proceso modifique estos valores hasta que no se lo indicas de esa forma al depurador.

   Mientras programamos en **Eclipse**, este IDE nos permite añadir *breakpoints*, e incluso **nos lo sugiere** siempre que encuentra algun error (de cualquier tipo) mientras nosotros (los programadores) codeamos.

3. Una vez puesto un punto de interrupción, podemos **ejecutar el programa línea a línea**, para comprobar su correcto funcionamiento.

   Para ejecutar paso a paso el programa, podemos utilizar las funciones *Step Into, Step Over, Step Out y Run to Cursor*, que nos permiten respectivamente:

Step Into*:* ejecuta la siguiente línea de código. Si esa línea es una llamada a otra función, el programa entrará en esa función.

Step Over: igual que la anterior, pero si la siguiente línea es una función, la ejecuta sin entrar en ella.

Step Out*:* sale de la función actual.

Informe

Para realizar este breve informe, me gustaria ir paso a paso mencionando como fue que realice el TP.

La primera parte se basó en aprender (y luego aplicar) como leer el archivo base (el csv, aquel con todos los datos con los cuales trabajaría luego) y donde guardar los datos de éste. En un primer momento se me ocurrió trabajar con memoria dinamica y punteros, es decir, guardar la informacion en arrays manejados mediante punteros. Sin embargo, luego de consutarlo con el profesor, me di cuenta que estaba rebuscando mucho la consigna, y decidi finalmente que arrancaría de cero: guardaria la informacion de manera que cada línea del archivo fuera un struct, y que cada uno de los datos contenidos en cada una de las lineas fuera un atributo del struct.

A partir de este cambio, pude finalmente leer el archivo (tambien con ayuda del profesor), y asi poder verificar que estaba guardando todos los datos de manera correcta. Por lo tanto, y finalizada la primera parte del TP, la informacion quedaba guardada en un array (no en memoria dinamica como habia considerado en un principio) de 230 elementos, cada uno de los cuales es un struct.

Luego, la segunda parte (tal vez la que menos me costo tanto en tiempo como en dificultad), era la de codear la “salida”, es decir la parte del programa que interactua con el usuario. Una vez habiendo leido el archivo y guardado todo, esta parte me resulto relativamente facil, y mas alla de algunos errores que me marcaba Eclipse, los cuales no sabia de donde surgian, finalmente pude solucionar todo y terminar de darle forma al codigo del TP. Tal vez la parte mas clave de todo estuvo en aprender que los arrays son punteros, y que por lo tanto podia modificarlos dentro de una funcion y, sin devolverlos (ya que no es posible que una funcion devuelva un array), usarlos en otra ya modificados.

Finalmente, quedaba la parte “estetica”, si se puede decir asi, del TP. Me refiero a corregir aquellas cosas necesarias para cumplir las convenciones de programacion (por ejemplo, que los nombres de las clases deben comenzar con mayuscula). Ademas, separe el TP en mas archivos (cree 4 archivos nuevos entre los ‘.h’ y ‘.cpp’) y escribi las pre y poscondiciones de cada una de las funciones que asi lo requerian. De esta manera, con estas modificaciones el TP quedaria mas legible para todo aquel programador (o no) que quisiera leerlo posteriormente.

Manual de usuario

Este manual tiene como objetivo facilitar el uso del programa que ha adquirido. Este debe facilitarle a usted la tarea de obtener los datos de cierta estacion de metrobus.

Usted tiene tres opciones en el menu principal: filtrar por longitud, por latitud (en ambos casos debera indicar un valor minimo o un valor maximo) y filtrar por ID exacto (en este caso debera ingresar un ID para obtener todos los datos correspondientes a el).

El programa esta pensado para que usted utilice alguna de las dos primeras funciones del menu, y a partir de todas las estaciones obtenidas (donde se le indicara el ID), filtre por ID exacto para obtener mayor informacion de una estacion en particular (nombre, interseccion de calles, lineas de colectivo, etc.).

Sin embargo, le debemos dar un conjunto de advertencias para el correcto uso del programa:

- A pesar de que en la gran mayoria de los casos, hemos validado para usted los ingresos, para evitar ingresos invalidos (por ejemplo, si ingresa un ID que no esta dentro de nuestra base de datos), no podemos garantizarle que en estos casos el programa vaya a funcionar correctamente, por lo tanto, no intente realizar algo fuera de lo indicado en el menu (por ejemplo, ingresar una palabra cuando se le pida un numero, y viceversa).
- En caso de que el programa deje de funcionar correctamente por algun ingreso invalido no anticipado, le recomendamos presionar el boton de stop, y reiniciar el programa.
- Si observa que alguna casilla tiene este formato: “Calle 2: .”, debe saber que simplemente ese dato no esta en nuestra base de datos o no aplica para la estacion solicitada.

Finalmente, para salir del menu principal, podra presionar el boton ‘*’, seguido de Enter.

Manual de programador

Para realizar este apartado, voy a comenzar leyendo el main, y a partir de este, vere a que archivos y funciones me va llevando el codigo.

Comienzo leyendo:

#include <string>

#include “lectura_y_guardado.h”

#include “lineaArchivo.h”

#include “salida.h”

using namesapace std;

El primer include incluye ‘string’, de manera que pueda utilizar el tipo nombrado de esa manera.

Del segundo al cuarto, estoy incluyendo los archivos ‘.h’ que usare en el main (el segundo y el cuarto son archivos vinculados a ‘.cpp’s y el tercero es un ‘.h’ con una definicion del struct utilizado en el TP.

El quinto renglon me hace evitar tener que utilizar ‘std::’ cada vez que quiera usar un ‘cin’, ‘cout’, etc.

Luego ingreso a la funcion main, y en la primer linea me encuentro con la definicion de ‘archivo’, de tipo string, donde guardare el archivo a leer.

En la segunda linea, se llama a la primera funcion, por lo tanto nos dirigimos al archivo ‘lectura_y_guardado.cpp’ para continuar.

En el archivo recien mencionado, nos debemos dirigir a su funcion principal (leerArchivoyGuardarInformacion); una vez alli podemos ver que se comienza abriendo el archivo y leyendo la primera linea.

Luego arranco dos contadores, uno para el numero de dato y otro para el numero de linea (del archivo).

En la siguiente linea, inicio un ciclo while que finalizara cuando llegue al final del archivo (ya que esa es la funcion de ‘.eof()’). Alli dentro leo un dato con la funcion ‘getline’ cuyo delimitador cambiara según este leyendo un dato cualquiera o, por otro lado, si este es el ultimo de esa linea (ya que el delimitador en este ultimo caso sera ‘/n’). Luego entro a la funcion ‘procesarUnaLinea’ (previamente utilizando un condicional *if* para verificar nuevamente el final del archivo, ya que sino me lanzaba un error).

En la funcion mencionada anteriormente, lo que se realiza es, ante todo identificar el tipo de dato que almacena la variable dato (no el tipo como ‘int’, ‘double’, etc., sino tipo refiriendome a atributo del struct, es decir, dato de la estacion de metrobus, por ejemplo: longitud, nombre, interseccion, etc.). Luego la funcion, mediante muchos *if*s, guarda el dato en el atributo correcto del struct correspondiente. Finalmente aumenta el contador del numero de dato en 1, y devuelve este valor para ser utilizado en la siguiente iteracion del ciclo while.

Finalizada la funcion y la iteracion del ciclo while, se cierra el archivo, y a continuacion finaliza la funcion principal de este ‘.cpp’. Por lo tanto, nos toca volver al archivo ‘main.cpp’.

La siguiente linea de la funcion main llama a la funcion ‘ejecucionDelArchivo’, por lo tanto nos toca dirigirnos al archivo correspondiente a ella (‘salida.cpp’).

En el archivo mencionado, debemos dirigirnos a la funcion principal (la mencionada en el main). Ante todo, nos encontramos con la definicion e inicializacion del ingreso del usuario (como ‘not *’, para remarcar que ingresara en el ciclo while a continuacion).

Una vez dentro del ciclo while, el programa imprime el menu de opciones:

Si el usuario ingresa 1 o 2 (pidiendo filtrar por longitud o latitud, respectivamente), se le pedira ingresar, primero la longitud (o latitud) minima, y luego la maxima. La siguiente linea, le pregunta al ususario si desea imprimir los datos asociados a las lineas del archivo que cumplan con el filtro (debera ingresar S/s o N/n, caso contrario, se podra ver a continuacion en el codigo un ciclo while en el que ingresara el usuario hasta que tipee una de estas opciones).

Luego de esto, el programa derivara en una funcion secundaria del archivo (filtrarDatosPorLongitud o filtrarDatosPorLatitud, según corresponda), y alli mediante un contador, y un ciclo *for* que ira de 0 a 230 (cantidad de lineas), aumentara el contador cada vez que entre en el *if* (es decir, cuando cumpla con el filtro designado por el usuario), e imprimira, en caso de que el usuario asi lo haya indicado, los datos asociados a las estaciones que cumplen con el filtro. Finalmente, imprimira tambien (siempre) la cantidad de estaciones totales que cumplen con el filtro.

Por otro lado, si el usuario ingresa 3, se le pedira ingresar (como se puede ver en las primeras lineas dentro del condicional *if*) un ID que quedara guardado en la variable IDIngresado. Luego, la funcion ingresa en una funcion secundaria llamada filtrarPorIDExacto. Si nos dirigimos al codigo de esa funcion, podemos observar que sus primeras lineas son un ciclo while en el que, mientras no coincide el ID ingresado por el usuario con el de la linea (definida por el contador i) del archivo analizada, aumenta i en 1.

Luego, si i es menor a 231 al salir del ciclo while, quiere decir que se encontro el ID dentro del archivo, y por lo tanto, como vemos en las lineas posteriores, se imprimira cada uno de los datos asociados a ese ID, obteniendolos del struct correspondiente del array.

Si por otro lado, en cambio, no se cumpliera el condicional *if*, esto quiere decir que se leyeron todos los IDs sin encontrar el ingresado por el usuario; en este caso, se le advierte al usuario que ha ingresado un ID invalido.

Finalmente, volviendo nuevamente a la funcion principal, podemos ver dos opciones mas:

El usuario puede ingresar ‘*’, con lo cual se imprimira un saludo, y posteriormente se saldra del ciclo *while* (debido a la condicion del propio *while*).

O por otro lado, el usuario puede ingresar una instrucción invalida, con lo cual se le advertira de esto, y se repetira el ciclo *while*.

Con esto se termina la ejecucion del archivo ‘salida.cpp’ y volvemos nuevamente al ‘main.cpp’.

Alli observamos la convencional linea final de la funcion main (return 0;) y el posterior final de la funcion.