**- MANEJO DE ARCHIVOS -**

     Los datos para los programas que hemos utilizado hasta ahora se han asignado en forma interna dentro de los programas o han sido introducidos por el usuario durante la ejecución del programa. Como tales, los datos utilizados en estos programas son almacenados en la memoria principal de la computadora y dejan de existir una vez que el programa que los usa termina de ejecutarse. Este tipo de entrada de datos está bien para cantidades pequeñas de datos.

     En esta clase aprenderemos como almacenar estos datos fuera de un programa en un medio de almacenamiento conveniente. Los datos almacenados juntos bajo un nombre común en un medio de almacenamiento distinto a la memoria principal de la computadora se llaman archivos de datos. Además de proporcionar un almacenamiento permanente para los datos, los archivos de datos pueden ser compartidos entre programas, de modo que la salida de datos de un programa puede ser introducidos de manera directa en otro programa. En esta clase veremos cómo se crean y mantienen los archivos de datos en C++.

     Una tarea muy importante sobre el uso de archivos de datos es asegurar que su programa abra y se conecte en forma correcta con ellos antes que comience cualquier procesamiento de datos. Por esta razón, veremos cómo usar el manejo de excepciones para esta tarea. Este tipo de detección y corrección de errores es de interés primordial en todos los programas escritos de manera profesional.

     Para almacenar y recuperar datos fuera de un programa en C++, se necesitan dos cosas:

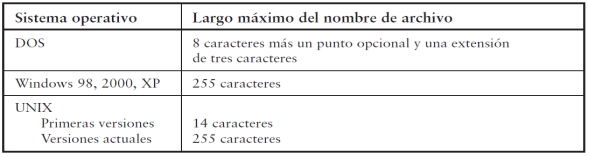
• Un archivo.  
• Un objeto de flujo de archivos.

     La biblioteca o libreria de C++ que contiene todas las instrucciones para manejo de archivos es <fstream> por lo cual deberemos agregar #include <fstream> a nuestros programas antes de comenzar a utilizar los archivos.

**ARCHIVOS:**

     Un archivo es una colección de datos almacenados juntos bajo un nombre común, por lo general en un disco. Por ejemplo, los programas en C++ que se almacenan en un disco son ejemplos de archivos. Los datos almacenados en un archivo de programa son el código del programa que se convierte en datos de entrada para el compilador de C++. Un archivo se almacena en forma física en un medio externo como un disco. Cada archivo tiene un nombre de archivo único, es la manera en que es conocido el archivo por el sistema operativo.

     Para asegurarnos de que los ejemplos presentados en este texto son compatibles con todos los sistemas operativos nos apegamos a las especificaciones más restrictivas de DOS. Sin embargo, si llegáramos a requerir utilizar uno de los otros sistemas operativos, deberemos aprovechar las ventajas del aumento en la especificación de longitud para crear nombres de archivo descriptivos según la siguiente tabla.



     Deberán evitarse los nombres de archivo largos debido a que requieren más tiempo para ser mecanografiados y pueden producir errores en esta tarea. Un largo manejable para un nombre de archivo es de 12 a 14 caracteres, con un máximo de 25 caracteres.

     Usando la convención de DOS, todos los siguientes son nombres de archivo de datos de computadora válidos:

precios.dat             registro        info.txt

exper1.dat             calif.dat        fisica.mem

     Elija nombres de archivo que indiquen el tipo de datos que hay en el archivo y la aplicación para la cual se usan. Con frecuencia, los primeros ocho caracteres describen los datos y una extensión (los caracteres después del punto decimal) describen la aplicación. Por ejemplo, el programa de hoja de cálculo Excel aplica de manera automática una extensión “xls” a todos los archivos de hoja de cálculo, los programas de procesamiento de palabras Word de Microsoft usan las extensiones “doc” y los compiladores de C++ requieren que un archivo de programa tenga la extensión “cpp”.

     Cuando cree sus propios nombres de archivo, deberá observar esta práctica.

     Usando la convención de DOS, por ejemplo, el nombre “exper1.dat” es apropiado para describir un archivo de datos correspondiente al experimento número 1.

**OBJETO de FLUJO de ARCHIVOS:**

     Un flujo de archivos es una ruta de transmisión unidireccional utilizada para conectar un archivo almacenado en un dispositivo físico, como un disco, un CD o PENDRIVE, con un programa.

     Cada flujo de archivos tiene su propio método, el cual determina la dirección de los datos en la ruta de transmisión; es decir, si la ruta moverá datos de un archivo a un programa o si la ruta moverá datos de un programa a un archivo.

     Un flujo de archivos que recibe o lee datos de un archivo a un programa se conoce como flujo de archivos de entrada.

     Un flujo de archivos que envía o escribe datos en un archivo se conoce como flujo de archivos de salida.

     La dirección, o modo, se define en relación con el programa y no al archivo; los datos que van a un programa se consideran datos de entrada, y los datos enviados desde el programa se consideran datos de salida.

     Para cada archivo que utilice su programa, sin importar el tipo de archivo (texto o binario), debe crearse un objeto de flujo de archivos distinto. Si desea que su programa lea y escriba en un archivo, se requieren un objeto de flujo de archivos de entrada y uno de salida.

     Los objetos de flujo de archivos de entrada se declaran como tipo ifstream (utilizar un archivo existente), y los flujos de archivos de salida (crear archivos o agregar datos) como tipo ofstream.

     Por ejemplo, examine la siguiente instrucción de declaración:

***ifstream*** archivo\_entr;

     Esta instrucción declara que un objeto de flujo de archivos de entrada llamado archivo\_entr es un objeto de la clase ***ifstream***.

     Del mismo modo, examine la siguiente instrucción de declaración:

***ofstream*** archivo\_sal;

     Esta instrucción declara que un objeto de flujo de archivos de salida llamado archivo-\_sal es un objeto de la clase ***ofstream***.

     Dentro de C++, se tiene acceso a un flujo de archivos por su nombre de objeto de flujo apropiado: un nombre para leer el archivo y un nombre para escribir en el archivo. Los nombres de objetos, como archivo\_entr y archivo\_sal, pueden ser cualquier nombre seleccionado por el programador que se ajuste a las reglas de identificación de C++.

     Para poder utilizar los objetos creados deberemos conectar un nombre de objeto de flujo a un nombre de archivo externo (llamado abrir un archivo), determinar si se ha hecho una conexión exitosa, cerrar una conexión (llamado cerrar un archivo), obtener el siguiente elemento de datos para el programa desde un flujo de entrada, colocar un elemento de datos nuevo del programa en un flujo de salida y detectar cuando se ha alcanzado el final de un archivo.

     Abrir un archivo conecta cada objeto de flujo de archivos con su nombre de archivo externo específico. Esto se logra por medio de un método de apertura de flujo de archivos, el cual tiene dos propósitos.

     Primero, abrir un archivo que establece el vínculo de conexión física entre un programa y un archivo.

     Segundo además de establecer la conexión física real entre un programa y un archivo de datos, abrir un archivo conecta el nombre externo del archivo en la computadora con el nombre del objeto de flujo usado en forma interna por el programa. El método que realiza esta tarea se llama **open()** y es proporcionado por las clases ifstream y ofstream.

     Por ejemplo, examinemos la siguiente instrucción:

archivo\_entr.**open**(“precios.dat”);

     Conecta el archivo de texto externo llamado precios.dat con el objeto de flujo de archivo interno del programa llamado archivo\_entr. Esto supone, por supuesto, que archivo\_entr se ha declarado como un objeto ***ifstream*** u ***ofstream*** anteriormente. Si se ha abierto un archivo con la instrucción anterior, el programa tiene acceso al archivo usando el nombre de objeto interno **archivo\_entr**, y la computadora guarda el archivo bajo el nombre externo precios.dat. El argumento del nombre de archivo externo transmitido por ***open()*** es una cadena contenida entre comillas.

     Logicamente si hablamos de manejo de archivos o ficheros, una vez concluido el uso del mismo se debe realizar el cierre del mismo usando el método ***close()***. Este método rompe la conexión entre el nombre externo del archivo y el objeto de flujo de archivos.

     Examinemos la siguiente instrucción:

archivo\_entr.**close**();

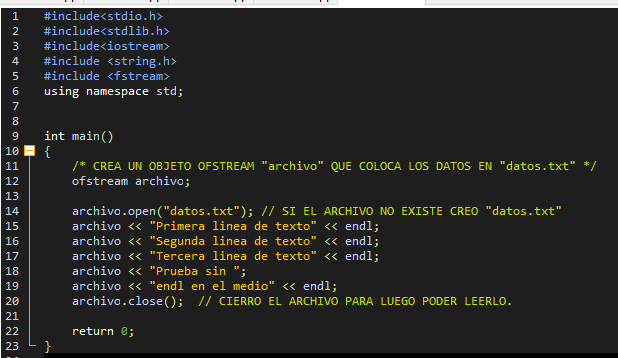
     Esta instrucción cierra la conexión del flujo archivo\_entr con su archivo actual. Como se indica, el método ***close()*** no utiliza un argumento. Debido a que todas las computadoras tienen un límite en el número máximo de archivos que pueden abrirse a la vez, cerrar archivos que ya no son necesarios tiene sentido. Los archivos abiertos existentes al final de la ejecución del programa normal serán cerrados de manera automática por el sistema operativo.

**Ejemplo Archivo 1:**

     Vamos a crear un archivo llamado “datos.txt”. Para ello vamos a crear un objeto ***ofstream*** denominado archivo que recibirá toda la información para luego transferirla al file “datos.txt”. Utilizaremos entonces:

***ofstream*** archivo;

     archivo**.open**("datos.txt");



     Seguramente cuando hayan ejecutado el programa ningún dato será visible para ustedes en pantalla. Pero si nos fijamos en el directorio donde se encuentra alojado el programa se habrá creado el archivo “datos.txt”, que es un archivo txt editable como cualquier otro.

**Ejemplo Archivo 2:**

     Con este siguiente programa vamos a visualizar los datos contenidos en los archivos, para ello utilizaremos entonces:

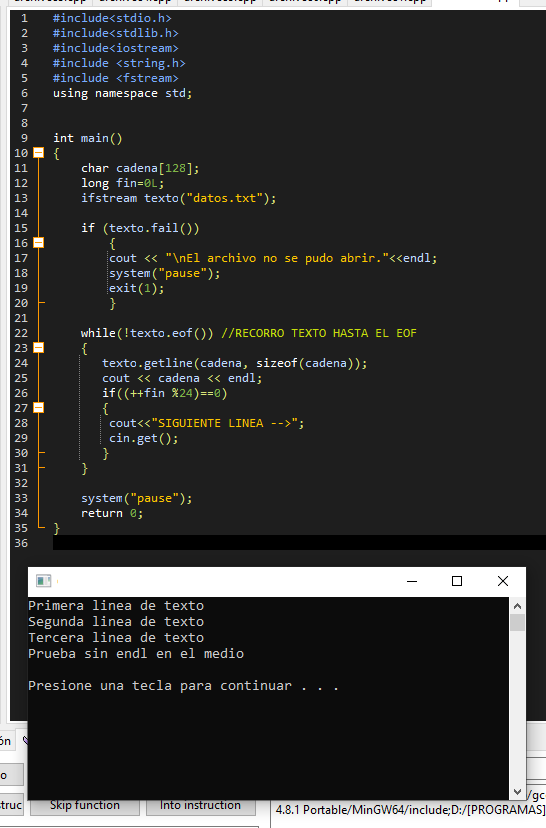
***ifstream*** texto("datos.txt");

     Cuando se abre un archivo, para entrada o salida, la buena práctica de programación requiere que se compruebe la conexión que se ha establecido antes de intentar usar el archivo. Esto se puede hacer por medio del método fail(), el cual devolverá un valor verdadero si el archivo no fue abierto en forma exitosa (es decir, es verdadero en cuanto falló la apertura) o un valor falso si la apertura tuvo éxito. Existen otros métodos además de fail() para realizar la comprobación de un archivo por ejemplo:

* good() Produce un 1(verdadero) si la operación previa fue exitosa.
* eof() Produce un 1(verdadero) si se encuentra el final del archivo.
* bad() Produce un 1(verdadero) si se realiza una operación inválida.
* fail() Produce un 1(verdadero) si ocurre un error.

     Para el ejercicio en cuestión utilizaremos:

     if (nombre\_del\_archivo.fail())

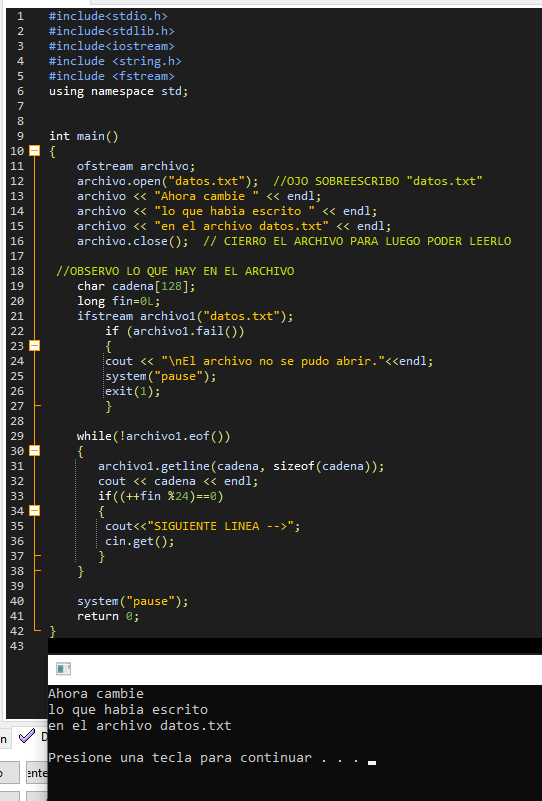


         Como podemos observar los datos generados en el anterior programa y ahora alojados en el archivo “datos.txt” serán mostrados por pantalla, incluso se ve como desde el programa archivo 1 (Línea de código 18 y 19) yo no utilice el endl para separar las líneas por lo cual escribió todo junto en una sola línea de texto de corrido.

         En este mismo programa utilizamos un texto.eof (end of file) este parámetro tal cual esta explicado en el manual sirve para delimitar el final de datos de un archivo, de esta manera nosotros le pedimos al programa que lea el archivo texto hasta su ultimo carácter o espacio. Continuando con el programa y gracias a getline capturamos todo el contenido de texto línea por línea incluyendo los espacios para conseguir de esta forma la lectura de una frase completa. Además, en el mismo programa agregamos un if diferencial que permitirá separar un texto que contenga más de 24 líneas, es decir de la forma en la que esta normalizado el formato de texto de lectura para un archivo. Si se llegan a superar las 24 líneas de texto, abra que presionar enter para continuar leyendo, prueben agregar más de 24 líneas de texto con el primer programa para ver los resultados.

**Ejemplo Archivo 3:**

     Que ocurriría si yo quisiera volver a utilizar el archivo “datos.txt” utilizando el objeto de flujo ***ofstream*** para seguir escribiendo en dicho file.



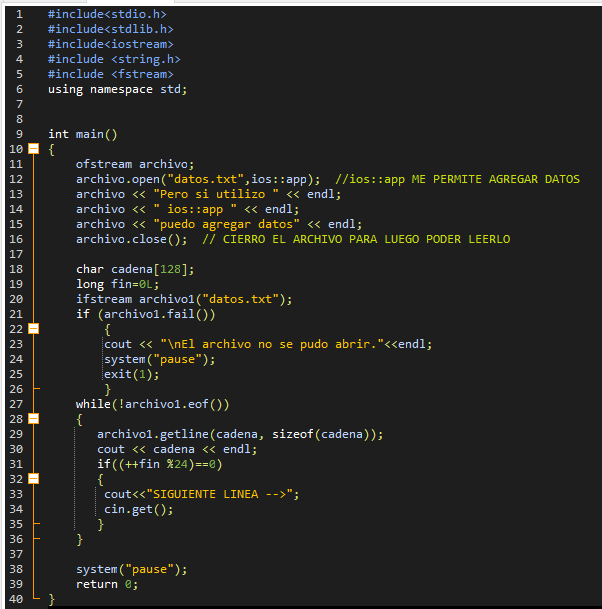
     Al utilizar archivo.open("datos.txt"); **SOBREESCRIBO** "datos.txt" y es por ello cuando voy a visualizar el contenido del fichero me encuentro con que, lo anteriormente realizado ha sido superpuesto por los nuevos datos. Esta es la razón primordial por lo cual muchos programadores al comienzo de cada programa generan un archivo con el mismo nombre, pero con extensión .bak de resguardo, para de esta simple forma evitar la pérdida total o parcial de datos en caso de error o sobre escritura.

     Entonces ¿qué tipo de ejecución debo realizar si yo quiero actualizar el fichero “datos.txt” de forma que mantenga los datos anteriormente escritos?

     Para realizar operaciones sobre los archivos generados utilizaremos distintos indicadores:

* ios::in   -   Abre un archivo de texto en modo de entrada.
* ios::out   -   Abre un archivo de texto en modo de salida.
* ios::app    -   Abre un archivo de texto en modo anexar.
* ios::ate   -   Va al final del archivo abierto.
* ios::binary   -   Abre un archivo binario en modo de entrada (es archivo de texto el valor por omisión).
* ios::trunc   -   Elimina el contenido del archivo si existe.
* ios::nocreate   -   Si el archivo no existe, la apertura falla.
* ios::noreplace   -   Si el archivo existe, falla la apertura para salida.

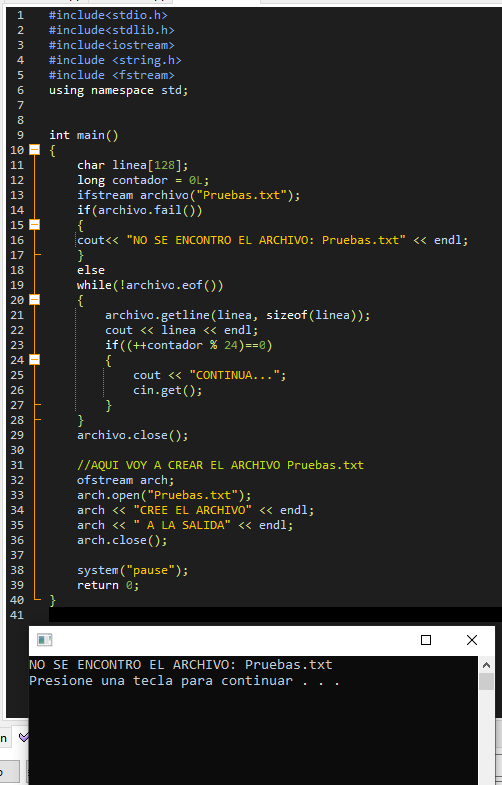
     Para el presente caso en cuestión utilizaremos el indicador ios::app el cual se agrega a continuación del “nombre.dat” del archivo.



     Realicemos las modificaciones tal cual se ven en la imagen anterior para observar los nuevos resultados obtenidos en el programa.

**Ejemplo Archivo 4:**

     Para el siguiente ejemplo voy a tratar de abrir un archivo inexistente llamado “Pruebas.txt” y al finalizar la comprobación de su existencia en el mismo programa vamos a crear dicho archivo.

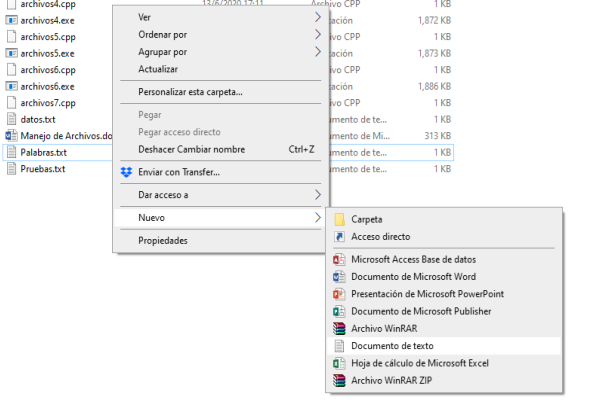


     Observemos que el programa no encontró el archivo buscado “Pruebas.txt” por dicha razón se muestra por pantalla “NO SE ENCONTRO EL ARCHIVO: …”.

     Pero qué ocurre si vuelvo a ejecutar el programa? Probemos **volver a ejecutar**y comprobemos los resultados.

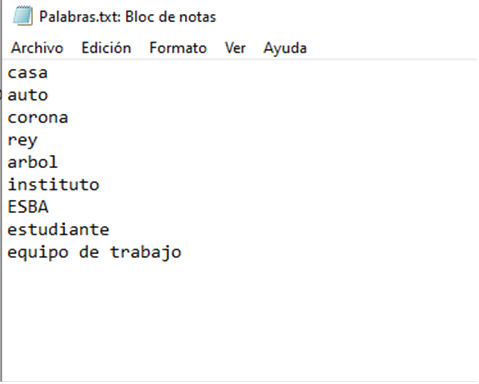
**Ejemplo Archivo 5:**

     Para el siguiente ejemplo voy a crear un archivo txt llamado “palabras.txt” en la carpeta donde esta alojado el .cpp del programa, y lo vamos a rellenar con palabras simples (ej. casa) y palabras compuestas (ej. mar azul).

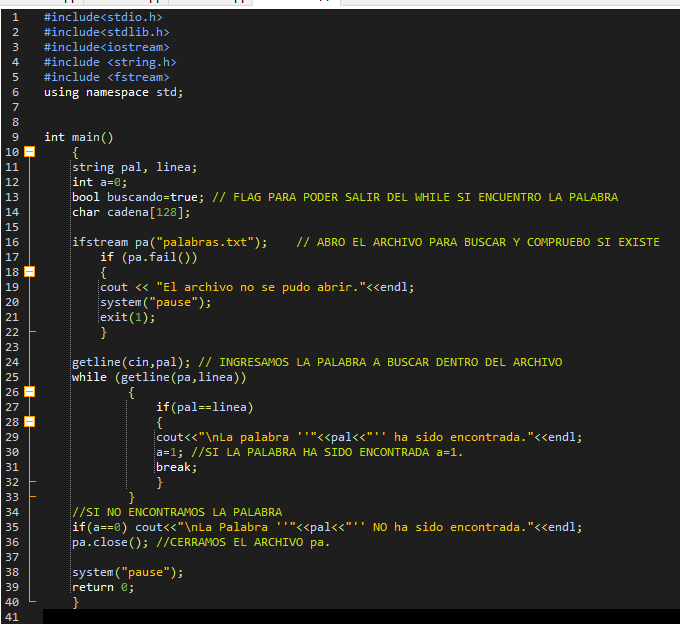


Una vez que tengamos el documento lleno de palabras separadas por línea.

*Ejemplo:*



     Vamos a escribir el siguiente programa, que utilizará el archivo creado para realizar búsquedas y comparar los resultados.



     Como podremos observar en este ejercicio vamos a realizar el uso de distintas funciones que ya venimos viendo en esta y clases pasadas. Para empezar abro el archivo utilizando “palabras.txt” a través de ***ifstream*** debemos tener en consideración que c++ no distingue entre archivos con mayúsculas y minúsculas, por lo cual será lo mismo si el archivo se llama PALABRAS o Palabras o paLABRas. Como siempre en principio y siguiendo los lineamientos para una buena programación controlaremos la existencia del archivo con un if y fail. Luego ingresaremos la palabra a buscar utilizaremos getline para poder obtener una línea de palabras así podremos corroborar palabras como por ejemplo "mar azul".

     Finalmente utilizaremos un while que funcionara mientras exista contenido dentro del archivo “pa”, también podríamos haber utilizado el *eof*pero si lo hiciéramos tendríamos que buscar la palabra dentro del archivo para compararla, de esta forma el mismo while va obteniendo la línea y se la asigna a mi variable “línea” la cual gracias a un if compara su valor al de la variable ingresada para la búsqueda “pal”. Si encuentra la palabra entonces muestra el mensaje de que “La palabra \_ \_ \_ \_ \_ ha sido encontrada” y asigna a la variable *a* el valor 1, esto es un flag dado que si no llegara a encontrar la palabra a la salida del while nos encontraremos con el siguiente if que funcionara siempre y cuando (a==0).

     Por favor escriban el ejercicio y prueben los resultados para ver cómo funciona.

     Con esto hemos concluido la clase de hoy, prácticamente hemos concretado todas las acciones comunes realizadas en programación en C++, como siempre les recuerdo que tenemos un foro de consultas que se ha convertido en un hermoso lugar para compartir no solo sus inquietudes sino también sus propios desarrollos e investigaciones y siempre está a disposición mi mail personal “[eshimoyama@esbabarrionorte.edu.ar](mailto:eshimoesba@gmail.com)” para poder comunicarnos. NO SE OLVIDEN de revisar cada semana el foro de "AVISOS" donde se encuentra toda la información semanal con respecto a la materia.