Podemos emplear el tercer parámetro del getline; este parámetro por defecto es el enter, lo que nos permite como mencioné antes, leer todo el renglón, pero si al llamar a la función añadimos un carácter delimitador como tercer parámetro, esto nos permitirá leer por tramos el archivo, cosa que el operador de las dos flechas no nos permite.

Además de esto, podremos utilizar …

Supongamos que queremos iniciar un programa que requiera un conjunto de datos para trabajar, sin tener que previamente nosotros cargarlos de forma manual, y que al cerrarlo y abrirlo posteriormente, estos datos sigan estando ahí, incluso con las modificaciones que les lleguemos a hacer. (y no, no se vale escribir los datos en el código del programa)

Para hacer esto, usamos los **archivos de texto**, ya que empleando variables simples o Arrays esto es imposible, porque los datos que guardan estas estructuras únicamente se almacenan en la memoria ram, y como ya sabemos todo lo que se almacena en esta, tarde o temprano es descartado.

En este video vamos a aprender a crear, leer, y editar archivos de texto en C++.

Para esto empleamos la biblioteca **fstream** que nos permite justamente trabajar con flujo de datos de archivos.

Para manejarlos, es necesario emplear una **variable lógica** que se encargue de controlar el o los archivos de texto que iremos a trabajar. Esta variable lógica se enlaza al archivo mediante una función llamada **open** y a partir de ahí trabajamos el archivo directamente llamando a la variable.

Hay 2 maneras de enfocar el trabajo con archivos, y durante el desarrollo del programa se podrán usar estas 2 pero no de forma simultánea.

El **primer enfoque** es para salida de información, es decir, crear y escribir un archivo de texto, para esto necesitamos declarar nuestra variable como tipo **ofstream,** siendo la o del principio, la abreviación de **out**, es decir, salida. Al declarar una variable como de salida, no podremos leer los datos que contenga el archivo, pero si escribir en él.

Si al ingresar la dirección donde se encuentra el archivo que queremos utilizar, no existe ninguno con ese nombre, el programa creará uno, y en caso de haber uno previamente, lo abrirá pero lo irá reescribiendo a medida que ingresemos información al archivo.

Para no reescribir un archivo, usaremos el parámetro **ios::app** que nos sirve para poner el puntero del archivo en el último renglón, así en vez de sobrescribir la información, añadiremos información nueva. La mejor forma de comprenderlo es pensarlo como un array por filas, donde al poner este parámetro nos iríamos a la posición n que esté cargada sin ningún dato.

Ahora, como escribimos información en el archivo?

Esto se hace mediante el operador **<<** y es relativamente sencillo, es lo mismo que escribir un mensaje por pantalla usando el cout.

Si ponemos un texto entre comillas, se imprimirá el texto, si pongo una variable o un array indicando sus posiciones, se escribirá lo que contengan estas variables.

Recuerden una vez realizada la escritura, cerrar el archivo usando la función **close.**

Asi de sencillo como vieron es la forma de escribir datos en un archivo, pero que hay de la lectura de datos de uno?

Aqui es donde entraríamos en el segundo enfoque, y la forma de hacerlo también es relativamente sencilla.

Primero que nada, para abrir un archivo como **entrada de datos** es esencial declarar la variable como ifstream, representando la **i** en este caso la palabra **in**, es decir, entrada.

A la hora de vincular la variable de entrada con un archivo, esto se hace exactamente igual que como hacíamos con **ofstream** pero con la diferencia de que este debe existir previamente, y en caso de no existir no creará ningún archivo, solamente no podrá leer nada. Podemos detectar cuando un archivo no se puede abrir o no se puede crear utilizando la función **fail**, normalmente se usa en la parte de entrada de datos y se pone primero en un if antes de proceder a leer los datos del archivo, donde podemos poner un cartel alusivo al usuario indicando que no se pudo abrir el archivo, e incluso impedirle la ejecución del programa poniendo un return distinto de 0 desde el propio main.

Ponemos distinto un número distinto de 0, normalmente 1, para indicar que el programa no se ejecutó correctamente, ya que como recordarán un 0 quiere decir que todo anduvo perfecto.

Bien, pero una vez logremos abrir exitosamente los archivos, como podemos adquirir la información que contienen?

Para leer datos de la consola usabamos el **cin**, esto no era más que un *objeto* que se encargaba de manejar los datos que se guardaban en el buffer, sería algo así como una variable lógica que controla los datos que se guardan ahí, y nosotros habíamos visto 2 formas de leer los datos de la consola, que para el manejo de archivos son completamente válidos:

1. Mediante el uso del operador **>>**, antes usamos el cin acompañado de esto para ingresar los datos que se carguen en consola a una variable, con archivos es lo mismo.

Solamente que aquí en vez de cin, pondremos la variable que se encarga de manejar el archivo y después todo funciona de la misma forma. Ojo, eso quiere decir que tiene las mismas limitaciones que en su momento en el video de strings explicamos, pero esta forma de leer los datos nos puede ser de re ayuda cuando queremos leer varios datos que se encuentran espaciados dentro de un archivo de forma simple, e incluso es muy útil para cuando necesitemos guardar un dato como tipo numerico, ya que mediante el uso de este operador se guardan estos datos inmediatamente en el tipo que le demos.

1. La segunda manera es utilizando getline. Es muy popular también, y no es excluyente de la anterior, de hecho hasta se pueden combinar según como este armado el archivo, para leerlo de manera más eficiente, pero. **teniendo en cuenta** como corregir los errores que pueden surgir después de leer un dato usando el operador >> y el getline, que ya cubrí en el video de strings como dije antes.

La lectura usando esta función es muy simple, invocamos a getline y en el parámetro donde poníamos cin, ponemos el nombre de la variable que maneja el archivo, y luego la variable o array donde se irá a guardar el dato. Recordemos que getline lee la línea entera de texto, es decir, un renglón entero, y en caso de tener varios datos separados por algún carácter especial como se muestra en el ejemplo, habrá que emplear lo aprendido en el video de Strings utilizando subcadenas para organizar los datos que nos lleguen al leer la línea entera.

Existen varios algoritmos para leer los archivos, y la lectura que realizamos del archivo se va haciendo de renglón en renglón, usando de puntero algo parecido a la línea que vemos parpadear a la hora de hacer click en un renglón del archivo, pero que no podremos ver.

Un algoritmo que nos presentan los profes para leer un dato es el siguiente, donde se lee primero un renglón usando el getline y luego se llama a un while, y en la condición como parámetro, la negación de la función **eof**, esto es una abreviatura de **end of file,** y como dice en inglés, es una función que nos indica cuando llegamos al final del archivo.

Nosotros la negamos dentro del while, porque constantemente nos devolverá como valor falso, ya que al principio de la lectura lo normal es que no estemos al final del archivo, y recuerden que al usar la negación, se invierte el resultado, entonces cuando nos diga falso porque no estamos al final, al while le llegará como verdadero, y esto le permitirá ejecutarse tantas veces hasta que el **eof** le diga al programa que terminó de leer todo, y es ahí cuando el while se terminará.

Luego dentro de este while se encuentran otros getline con las siguientes columnas del arrqy donde guardamos los datos de ese archivo, y por último una vez leamos la tanda de datos, que en el caso del ejemplo vienen en conjuntos de a 3 renglones, se incrementa el contador y se vuelve a leer el dato en la posición siguiente de la columna 0.

Esto se hace así de forma constante, y el porqué lo leemos al final denuevo, y no hacemos todo esto directamente en el while como vemos en este otro ejemplo, es porque se puede dar el caso de que el archivo contenga un espacio vacío, que el **eof** no contempla como con fin del archivo, y eso nos haga cargar hasta 2 posiciones de más, sin ningún dato, como se muestra en el ejemplo.

Utilizando el algoritmo de los profes, solo se realiza una carga demas, y está al contener todos datos vacíos no afectará al programa, el problema viene con el algoritmo anterior donde se realiza 1 carga de más, pero además la posición del array se incrementa 2 veces por arriba de donde debería, y está 2da posición ni siquiera este inicializada con ningún dato, generando problemas después a la hora de trabajarlos.

Recuerden que también en vez de getline podríamos haber utilizado el operador de la doble flecha en caso de necesitarlo, aunque para estos ejemplos que di no era necesario, pero podríamos tener uno como este donde sí lo fuese.

Otro algoritmo para leer datos, más eficiente y directo que el anterior, es utilizando el getline como parámetro directamente en el while, ya que el mismo getline le avisara al bucle cuando no tiene más datos para leer y esto simplifica aún más la lectura de datos como la de los 2 anteriores ejemplos que les mostré.

Otros datos relevantes de archivos:

* Los archivos se pasan únicamente por referencia a una función
* La dirección que pasamos en el open es un string, por lo tanto, podemos reemplazarlo por una variable, o incluso enviar un string a una función y que ésta lo reciba para indicar dónde se debe abrir un archivo.

https://cplusplus.com/reference/fstream/fstream/

No tiene mucho más, hasta ahí sería la clase de archivos. Pronto haré la clase de structs y con eso cerraríamos los tutoriales de c++ por ahora, nos vemos.