Sed is for losers

El objetivo de este ejercicio es crear un programa que realice una tarea similar a la herramienta sed de Linux, pero sin usar ciertas funciones restringidas. Aquí tienes una explicación detallada y sencilla:

¿Qué debe hacer el programa?

Recibir tres parámetros en el siguiente orden:

- Un nombre de archivo (filename).
- Una cadena de texto (s1).
- Otra cadena de texto (s2).

Tarea del programa:

- Abrir el archivo especificado por filename y leer su contenido.
- Crear un nuevo archivo llamado <filename>.replace.
- Copiar el contenido del archivo original al nuevo archivo, pero con cada aparición de la cadena s1 reemplazada por s2.

Restricciones importantes

- No puedes usar las funciones de manipulación de archivos de C (como fopen, fclose, fread, etc.).
- Debes trabajar exclusivamente con las clases y funciones de C++, como std::ifstream y std::ofstream.
- No puedes usar el método std::string::replace para realizar el reemplazo.
- Tienes libertad para usar otros métodos de la clase std::string, como find, substr, o append.

Requisitos adicionales

- El programa debe verificar si los parámetros ingresados son correctos.
- Comprobar si el archivo existe y puede abrirse.
- Manejar casos en los que las cadenas s1 o s2 no sean válidas (por ejemplo, si están vacías).
- Pruebas propias:
- Debes crear tus propias pruebas para garantizar que el programa funcione correctamente.

Cómo afrontar el ejercicio

Entender la tarea:

El objetivo es reemplazar todas las apariciones de s1 por s2 en un archivo y guardar el resultado en otro archivo.

Leer y procesar el archivo:

- Usa std::ifstream para abrir y leer el archivo original línea por línea.

- Usa std::ofstream para escribir en el archivo <filename>.replace.

Reemplazar cadenas:

- Usa std::string::find para localizar las posiciones donde aparece s1.
- Usa otros métodos como substr o concatenaciones para construir las líneas con los reemplazos realizados.

Manejo de errores:

Asegúrate de que:

- El archivo original existe y se puede leer.
- Los parámetros son válidos (no vacíos).
- Si ocurre un error, informa al usuario con un mensaje claro.

Crear pruebas:

- Prueba el programa con diferentes archivos y combinaciones de s1 y s2.

Incluye casos como:

- Archivos vacíos.
- s1 que no aparece en el archivo.
- Reemplazos múltiples en una sola línea.

En resumen

Entrada:

- Nombre del archivo (filename), cadena a reemplazar (s1), y cadena de reemplazo (s2).

Salida:

- Nuevo archivo <filename>.replace con el contenido modificado.

Pasos principales:

- Abrir el archivo original con std::ifstream.
- Leer su contenido y reemplazar todas las apariciones de s1 por s2.
- Escribir el resultado en el nuevo archivo con std::ofstream.

Restricciones:

- No usar funciones de C para manipular archivos.
- No usar std::string::replace.

¿Qué es std::string?

Tipo de dato para cadenas: std::string es un tipo de dato definido en la biblioteca estándar de C++ que se utiliza para representar y manipular cadenas de caracteres. Es una clase que encapsula una secuencia de caracteres y proporciona una interfaz fácil de usar para realizar operaciones como concatenación, búsqueda, comparación, etc.

Ventajas sobre los arreglos de caracteres: A diferencia de los arreglos de caracteres de C, std::string maneja automáticamente la gestión de memoria, lo que evita errores comunes como desbordamientos de búfer.

¿Qué significa el ampersand (&) después de std::string?

- Referencia: El ampersand (&) en este contexto indica que se está utilizando una referencia. Una referencia es un alias de una variable existente. En otras palabras, una referencia y la variable original apuntan al mismo lugar en memoria.

- Pasar por referencia: Cuando se pasa un argumento por referencia a una función, se está pasando la dirección de memoria de ese argumento. Esto significa que cualquier cambio que se realice dentro de la función se reflejará en la variable original.

Entonces, ¿qué es std::string&?

- Referencia a una cadena: std::string& representa una referencia a un objeto de tipo std::string.
- Modificación directa: Al pasar un argumento de tipo std::string& a una función, se permite que la función modifique directamente el contenido de la cadena original

const std::string& errorMessage

- Esta línea de código declara una variable llamada errorMessage que es una referencia constante a una cadena de caracteres.

¿Qué significa cada parte?

- const: Indica que el valor almacenado en la variable no puede ser modificado una vez asignado. Es decir, errorMessage se convierte en una especie de "ventana" a una cadena de caracteres existente, pero no podemos cambiar el contenido de esa cadena a través de esta referencia.
- std::string: Se refiere al tipo de dato string de la biblioteca estándar de C++, que representa una secuencia de caracteres.
- &: Este símbolo ampersand indica que errorMessage es una referencia. En lugar de copiar el valor de una cadena en errorMessage, esta variable simplemente "apunta" a la cadena original. Esto significa que cualquier cambio que se haga en la cadena a la que apunta errorMessage también se verá reflejado en la variable original.

¿Por qué usar una referencia constante?

- Eficiencia: Evitar copiar cadenas grandes puede mejorar el rendimiento, especialmente en funciones que se llaman muchas veces.
- Seguridad: Al hacer la referencia constante, se asegura que la función no modifique accidentalmente la cadena original.
- Claridad: Indica claramente que la función no tiene la intención de cambiar la cadena que se le pasa.

¿Qué es searchString.empty()?

En C++, cuando trabajamos con cadenas de texto (objetos de tipo std::string), tenemos una función miembro llamada empty() que nos permite verificar si esa cadena está vacía o no.

¿Cuándo una cadena está vacía?

- Una cadena se considera vacía cuando no contiene ningún carácter. Es decir, su longitud es cero.

¿Qué hace empty()?

La función empty() simplemente devuelve un valor booleano (verdadero o falso):

- true: Si la cadena está vacía.
- false: Si la cadena contiene al menos un carácter.

¿Qué es std::ifstream?

std::ifstream es una clase de la biblioteca estándar de C++ que se utiliza para realizar operaciones de lectura de archivos. En otras palabras, nos permite abrir un archivo y leer los datos que contiene de forma secuencial, desde el principio hasta el final.

¿De dónde viene?

- std: Significa "estándar". Indica que esta clase forma parte de la biblioteca estándar de C++.
- if: Abreviatura de "input file". Significa que se trata de un flujo de entrada (input stream) para un archivo.
- stream: Representa una secuencia de datos. En este caso, una secuencia de datos que proviene de un archivo.

¿Cómo funciona?

- Creación del objeto: Se crea un objeto de tipo std::ifstream y se asocia a un nombre de archivo. Ejemplo:

```
std::ifstream archivo("mi_archivo.txt");
```

- Verificación de apertura: Se comprueba si el archivo se ha abierto correctamente utilizando el método is_open(). Ejemplo:

```
if (!archivo.is_open()) {
   std::cerr << "No se pudo abrir el archivo." << std::endl;
   return 1;
}
```

- Lectura de datos: Se utilizan diferentes métodos para leer datos del archivo, como >> para leer valores simples o getline para leer líneas completas. Ejemplo:

```
std::string linea;
while (std::getline(archivo, linea)) {
   std::cout << linea << std::endl;
}
```

- Cierre del archivo: Una vez que se ha terminado de leer el archivo, es recomendable cerrarlo utilizando el método close(). Ejemplo:

```
archivo.close();
```

¿Para qué se utiliza?

- Lectura de archivos de texto: Para leer cualquier tipo de archivo de texto, como archivos de configuración, registros, etc.
- Procesamiento de datos: Para leer datos de un archivo y realizar operaciones sobre ellos, como calcular estadísticas, buscar patrones, etc.
- Análisis de datos: Para analizar grandes conjuntos de datos almacenados en archivos.
- En resumen, std::ifstream es una herramienta fundamental en C++ para trabajar con archivos de texto. Nos permite abrir archivos, leer su contenido y realizar diversas operaciones sobre los datos leídos.

std::ifstream inputFile(inputFileName.c_str());

Esta línea declara y crea un objeto de tipo std::ifstream llamado inputFile. Este objeto se utiliza para leer datos desde un archivo.

- std::ifstream: Es una clase de la biblioteca estándar de C++ que se especializa en la lectura de archivos.

- ifstream son las siglas de "input file stream", que significa "flujo de entrada de archivo" (un flujo es una secuencia de datos que se pueden leer o escribir. En este caso, el flujo está asociado a un archivo que queremos leer).

- inputFile: Es el nombre que le hemos dado a nuestro objeto de tipo std::ifstream. Este nombre será utilizado para referirnos a este objeto a lo largo del código.

- inputFileName.c_str(): inputFileName es una variable de tipo std::string que contiene el nombre del archivo que queremos abrir. El método c_str() es un método de la clase std::string que devuelve un puntero a un array de caracteres de estilo C (un C-string) que contiene la cadena de caracteres almacenada en el objeto std::string. La razón por la que

utilizamos c_str() es porque el constructor de std::ifstream espera un puntero a un carácter como argumento para el nombre del archivo.

¿Qué hace esta línea en conjunto?

Crea un objeto ifstream: Reserva memoria para un nuevo objeto de tipo ifstream.
Inicializa el objeto: El constructor de ifstream intenta abrir el archivo cuyo nombre se proporciona como argumento.
Asigna el objeto a una variable: El objeto recién creado se asigna a la variable inputFile.

En resumen:

Esta línea abre el archivo especificado por inputFileName y lo asocia con el objeto inputFile. A partir de este momento, podemos utilizar el objeto inputFile para leer datos del archivo, como por ejemplo, leer una línea completa o un carácter individual.

¿Qué es inputFile.is_open()?

inputFile: En el código qur nos ocupa, es un objeto de tipo std::ifstream, que representa un flujo de entrada de archivos. En este caso, inputFile está asociado al archivo que queremos leer (el archivo de entrada).

.is_open(): Es un método de los objetos de tipo std::ifstream (y otros flujos de archivos como std::ofstream) que devuelve un valor booleano (verdadero o falso).

¿Qué hace inputFile.is_open()?

- Comprueba si el archivo está abierto: Este método verifica si el archivo asociado al objeto de flujo (en este caso, inputFile) se ha abierto correctamente.

Devuelve un valor booleano:

- Si el archivo está abierto, devuelve true.
- Si el archivo no está abierto (por ejemplo, si no se encontró el archivo, no se tienen permisos de lectura, etc.), devuelve false.

std::getline(inputFile, currentLine)

Esta línea de código se encarga de leer una línea completa de texto desde un archivo y almacenarla en una variable de tipo std::string. En términos más simples, es como si estuvieras leyendo una línea de un libro y la escribieras en una hoja de papel.

- std::getline: Esta es una función de la biblioteca estándar de C++ que se utiliza específicamente para leer líneas de texto.

- inputFile: Esta es una variable que representa el archivo que estamos leyendo. Debe ser un objeto de tipo std::ifstream, que es un flujo de entrada de archivos.

- currentLine: Esta es una variable de tipo std::string donde se almacenará la línea de texto que se acaba de leer.

Funcionamiento paso a paso:

- Se llama a la función std::getline: Esto inicia el proceso de lectura de una línea.
- Se lee desde el archivo inputFile: La función busca el siguiente salto de línea en el archivo.
- Se extrae la línea: Todos los caracteres hasta el salto de línea se extraen del archivo.
- Se almacena en currentLine: La línea extraída se copia en la variable currentLine.

(foundPos = currentLine.find(searchString, foundPos))

Esta línea de código es el corazón del algoritmo de búsqueda y reemplazo. Permite encontrar todas las ocurrencias de una cadena dentro de otra y realizar las acciones necesarias (en este caso, reemplazarlas).

currentLine.find(searchString, foundPos): Esta parte se encarga de buscar la primera ocurrencia de la searchString (cadena a buscar) dentro de la currentLine (línea actual) a partir de la posición foundPos.

- currentLine: Es la cadena de texto en la que estamos buscando.

- searchString: Es la cadena exacta que queremos encontrar.

- foundPos: Es la posición desde donde empezaremos a buscar en la línea actual. Esto es útil para encontrar todas las ocurrencias de la searchString en una misma línea, ya que en cada iteración del bucle se actualiza para continuar la búsqueda desde la posición siguiente a la última ocurrencia encontrada.
- foundPos =: El resultado de la búsqueda (la posición donde se encontró la searchString, o std::string::npos si no se encontró) se asigna a la variable foundPos.

- != std::string::npos : std::string::npos: Es una constante especial que indica que no se ha encontrado la cadena buscada. Esencialmente, es un valor que está fuera de los índices válidos de una cadena.
- !=: Este operador compara si el valor de foundPos es diferente de std::string::npos.

En resumen:

La línea completa verifica si se encontró la searchString en la currentLine. Si se encontró, el valor de foundPos será un número entero positivo que indica la posición donde se encontró. Si no se encontró, foundPos será igual a std::string::npos.

¿Por qué se usa este condicional en un bucle?

- Búsqueda de múltiples ocurrencias: Al asignar el resultado de find a foundPos y luego usar ese valor en la siguiente iteración, se garantiza que se busquen todas las ocurrenencias de searchString en la línea actual, no solo la primera.

- Detener el bucle cuando no haya más coincidencias: Cuando find no encuentra ninguna coincidencia, devuelve std::string::npos. El condicional != std::string::npos asegura que el bucle se detenga cuando se hayan encontrado todas las ocurrencias.

Ejemplo:

Imagina que currentLine es "hola mundo hola" y searchString es "hola". La primera vez que se ejecuta el bucle, foundPos será 0 (se encuentra "hola" al principio). La segunda vez, foundPos será 5 (se encuentra la siguiente "hola" a partir de la posición 5). En la tercera iteración, find no encontrará más ocurrencias y devolverá std::string::npos, por lo que el bucle se detendrá.

main.cpp

Este código de C++ implementa una versión simplificada del comando sed de Unix, que se utiliza para reemplazar cadenas de texto en archivos.

1. Inclusión de bibliotecas:

   - iostream: Proporciona funciones para entrada/salida estándar (como std::cout y std::cerr).
   - fstream: Permite trabajar con archivos, como abrirlos, leerlos y escribir en ellos.
   - string: Define la clase std::string para manipular cadenas de caracteres.

2. Función printErrorAndExit:

   - Esta función recibe un mensaje de error como argumento.
   - Imprime el mensaje de error en el flujo de salida de error (std::cerr).
   - Devuelve el valor 1, indicando que se ha producido un error.

3. Función main:

4. Comprobación de argumentos:

   - Verifica si se han proporcionado exactamente 4 argumentos: el nombre del archivo de entrada, la cadena a buscar y la cadena de reemplazo.
   - Si el número de argumentos es incorrecto, llama a printErrorAndExit y termina el programa.

5. Obtención de argumentos:

   - Almacena los argumentos en variables de tipo std::string.
   - Comprobación de la cadena de búsqueda:
   - Verifica si la cadena a buscar está vacía. Si lo está, llama a printErrorAndExit y termina el programa.

6. Apertura del archivo de entrada:

   - Intenta abrir el archivo de entrada especificado en modo lectura.
   - Si no se puede abrir el archivo, llama a printErrorAndExit y termina el programa.

- Creación del archivo de salida:
- Crea un nuevo archivo de salida con el mismo nombre que el archivo de entrada, pero con el sufijo ".replaced".
- Si no se puede crear el archivo de salida, llama a printErrorAndExit y termina el programa.

7. Procesamiento línea por línea:

   Lee el archivo de entrada línea por línea.

   Para cada línea:

   - Busca todas las ocurrencias de la cadena a buscar (searchString) dentro de la línea.
   - Reemplaza cada ocurrencia de la cadena a buscar por la cadena de reemplazo (replaceString).
   - Escribe la línea modificada en el archivo de salida.

   Mensaje de éxito:

   - Imprime un mensaje en la salida estándar indicando que el procesamiento se ha realizado correctamente y el nombre del archivo de salida.

   Retorno:

   - Devuelve 0 para indicar que el programa se ha ejecutado correctamente.

Ejemplo de uso:

Supongamos que tenemos un archivo llamado texto.txt con el siguiente contenido:

```
Hola mundo.
Este es un ejemplo.
Hola de nuevo.
```

Para reemplazar todas las ocurrencias de "Hola" por "Adiós" en el archivo texto.txt, ejecutaríamos el siguiente comando:

```
./sed texto.txt Hola Adiós
```

Esto crearía un nuevo archivo llamado texto.txt.replaced con el siguiente contenido:

```
Adiós mundo.
Este es un ejemplo.
Adiós de nuevo.
```

En resumen: este código proporciona una implementación básica del comando sed. En la práctica, el comando sed de Unix ofrece muchas más opciones y funcionalidades, como expresiones regulares, edición de archivos in situ y otras opciones de procesamiento.