

1. Objetivos

- Diseñar una aplicación que use estructuras lineales usando contenedores STL
- Elaborar un informe descrito sobre el diseño e implementación.

2.- Lectura de Ficheros

La biblioteca **fstream** en C++ permite el uso de operaciones de lectura y escritura de archivos de texto, a través de un flujo que se conecta al archivo. La Figura 1, a continuación presenta un ejemplo de lectura de un archivo por líneas (fuente [enlace](#)):

```
// Lectura de un fichero de Texto
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
    string linea;
    ifstream fichero ("ejemplo.txt");
    if (fichero.is_open()) {
        while (getline(fichero,linea)){
            cout << linea << '\n';
        }
        fichero.close();
    }

    else cout << "No se puede abrir el fichero";

    return 0;
}
```

Figura 1. Lectura de archivos usando fstream en C++ (fuente [enlace](#)).

3.- Desarrollo del taller

Diseñar e implementar una aplicación que use estructuras lineales para buscar palabras a partir de subcadenas. Se debe prestar especial atención en el desarrollo y uso de las operaciones de recorrido en secuencias y uso de pilas y colas como apoyo algorítmico. Adicional, el uso de operaciones de lectura de archivos de texto junto con los contenedores de STL. El taller se entrega con los ficheros de entrada a la aplicación comprimidos.

3.1.- Implementación

Implementar un programa que permita buscar palabras dentro de un archivo de texto utilizando subcadenas (palabras más cortas contenidas dentro de las palabras buscadas). En particular, para una subcadena dada, el programa debe:

- Imprimir en pantalla la cantidad de palabras que comienzan por la subcadena, y listar cada palabra encontrada indicando su número de línea en el texto.
- Imprimir en pantalla la cantidad de palabras que contienen la subcadena. La subcadena debe aparecer en cualquier posición de la palabra, y debe estar contenida completamente dentro de la palabra. Se debe listar cada palabra encontrada indicando su número de línea en el texto.
- Imprimir en pantalla la cantidad de palabras que contienen la subcadena invertida (leída de derecha a izquierda). La subcadena invertida debe aparecer en cualquier posición de la palabra, y debe estar contenida completamente dentro de la palabra. Se debe listar cada palabra indicando su número de línea en el texto.

Nota: El diseño del programa debe considerar el uso de al menos dos tipos diferentes de estructuras lineales (listas, pilas o colas). Como entrada, el programa debe recibir por línea de comandos el nombre del archivo que contiene la información necesaria. Esto quiere decir que, una vez compilado el programa, debe ejecutarse por la línea de comandos de la siguiente forma:

```
./nombre_programa archivo.txt
```

La estructura del archivo de entrada será la siguiente:

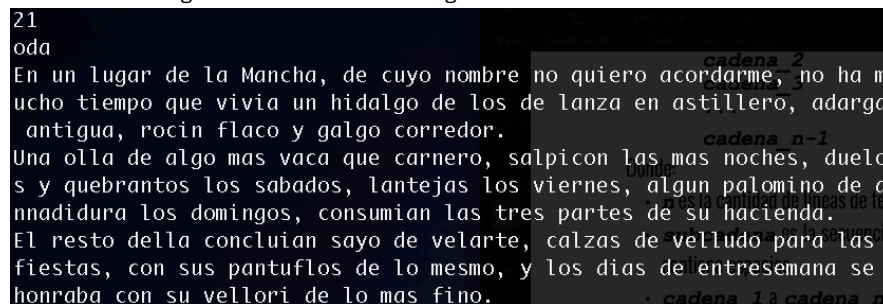
```
n
subcadena
cadena_1
cadena_2
cadena_3
...
cadena_n-1
```

Donde:

- **n** es la cantidad de líneas de texto que contiene el archivo (sin contar la línea donde aparece este valor).
- **subcadena** es la secuencia de letras que se utilizará para buscar las palabras requeridas. Se garantiza que esta subcadena no contiene espacios.
- **cadena_1** a **cadena_n-1** son cadenas de caracteres (una por línea) que representan el texto donde deben buscarse las palabras. Estas cadenas de caracteres pueden ser desde palabras individuales hasta frases completas de longitud indeterminada.

3.2.- Ejemplo

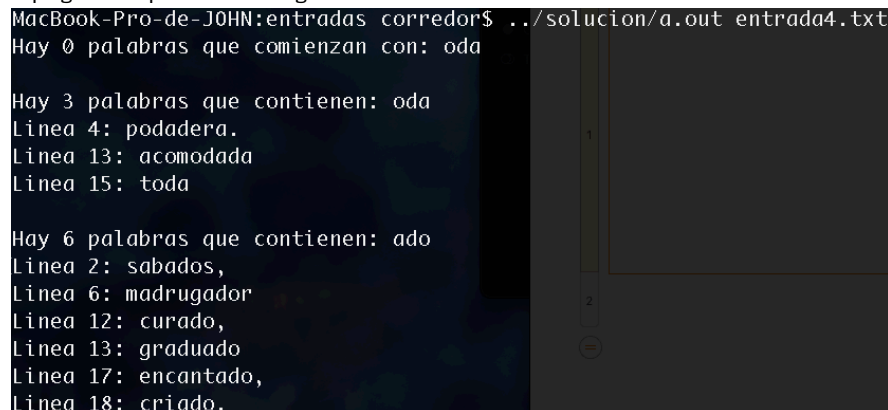
El archivo de entrada4.txt contiene la siguiente información (véase Figura 2)



```
21
oda
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha m
ucho tiempo que vivia un hidalgo de los de lanza en astillero, adarga
antigua, rocin flaco y galgo corredor.
Una olla de algo mas vaca que carnero, salpicon las mas noches, duelo
s y quebrantos los sabados, lantejas los vienes, algun palomino de a
nnadidura los domingos, consumian las tres partes de su hacienda.
El resto della concluian sayo de velarte, calzas de velludo para las
fiestas, con sus pantuflos de lo mismo, y los dias de entresemana se
honraba con su vellori de lo mas fino.
```

Figura 2. Información en el fichero entrada4.txt

La salida esperada del programa se presenta en la Figura 3



```
MacBook-Pro-de-JOHN:entradas corredor$ ../solucion/a.out entrada4.txt
Hay 0 palabras que comienzan con: oda

Hay 3 palabras que contienen: oda
Linea 4: podadera.
Linea 13: acomodada
Linea 15: toda

Hay 6 palabras que contienen: ado
Linea 2: sabados,
Linea 6: madrugador
Linea 12: curado,
Linea 13: graduado
Linea 17: encantado,
Linea 18: criado.
```

Figura 2. Información en el fichero entrada4.txt

3.3.- Diseño del TAD de la aplicación sugerido

TAD Palabra

Datos mínimos

- **palabra**: cadena de caracteres. Representa la palabra a almacenar
- **n_linea**: entero sin signo. Representa el número de línea en el que se encuentra la palabra.

Operaciones

- **FijarPalabra(n_palabra)**: cambia la palabra actual a **n_palabra**.
- **FijarNumLinea(n_num)**: cambia el número de línea actual a **n_num**.
- **ObtenerPalabra()**: retorna la palabra almacenada.
- **ObtenerNumLinea**: retorna el número de línea actual de la palabra.

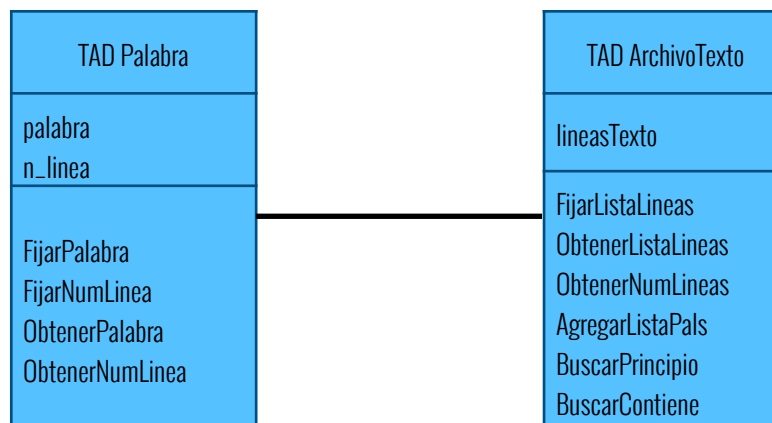
TAD ArchivoTexto

Datos mínimos

- **lineasTexto**: vector de vector de cadenas de caracteres, representa el archivo de texto con líneas de texto que contienen palabras.

Operaciones

- **FijarListaLineas(n_lista)**: cambia el vector actual de líneas de texto por **n_lista**.
- **ObtenerListaLineas()**: retorna el vector de líneas de texto que contiene el archivo.
- **ObtenerNumLineas()**: retorna la cantidad de líneas de texto en el archivo.
- **AgregarListaPals(n_lista)**: agrega una nueva línea de texto dada al vector que las contiene.
- **BuscarPrincipio(subcadena)**: busca la subcadena al principio de cada palabra de las líneas de texto del archivo.
- **BuscarContiene(subcadena)**: busca la subcadena dentro de cada palabra de las líneas de texto del archivo.



4.- Evaluación

El desarrollo del taller consistirá en generar un informe escrito (en formato .pdf, **cualquier otro formato implica una nota final de 0.0/5.0**), el cual debe enviarse a través de la respectiva asignación de BrightSpace. Junto con la entrega debe haber un archivo comprimido que contiene el conjunto de programas a usar (único formato aceptado **.zip**). Debe haber información de como compilar y ejecutar la aplicación. Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada.

La evaluación del taller tendrá la siguiente escala:

- **Excelente (5.0/5.0):** El estudiante diseñó correctamente (siguiendo la plantilla) e implementó una solución que ofrece las 3 funcionalidades (permite buscar la subcadena al principio de la palabra, en cualquier posición de la palabra y de forma invertida). Se presenta un informe con información suficiente, organizado y con claridad de cada uno de los puntos desarrollados. Los códigos presentados tiene documentación completa.
- **Muy bueno (4.2/5.0):** El estudiante diseñó correctamente (siguiendo la plantilla) e implementó una solución que ofrece 2 de las 3 funcionalidades requeridas. Se presenta un informe con información insuficiente y poco organizado con cada uno de los puntos desarrollados. Los códigos presentados tiene poca documentación.
- **Bueno (3.5/5.0):** El estudiante diseñó correctamente (siguiendo la plantilla) e implementó una solución que ofrece sólo 1 de las 3 funcionalidades requeridas. Se presenta un informe con poca información de cada uno de los puntos desarrollados. Los códigos no presentan información.
- **No fue un trabajo formal de ingeniería (3.0/5.0):** El estudiante implementó una solución completa o parcial, pero no la diseñó correcta o completamente. Se presenta un informe con solo el código documentado.
- **Necesita mejoras sustanciales (2.0/5.0):** El estudiante diseñó y/o implementó una solución, pero no es completa o no soluciona lo pedido.
- **Malo (1.0/5.0):** El código entregado por el estudiante no compila en el compilador g++ (mínimo versión número 4.5).
- **No entregó (0.0/5.0).**