

Taller 02: Estructuras lineales

Estructura de Datos

Nombre: Santiago Hernández Barbosa

1. Introducción

En este taller se implementó una aplicación en C++ que utiliza estructuras lineales para buscar palabras en un archivo de texto, a partir de una subcadena dada. Este realiza tres tipos de búsqueda:

- Palabras que **comienzan** con la subcadena.
- Palabras que **contienen** la subcadena en cualquier posición.
- Palabras que **contienen la subcadena invertida** en cualquier posición.

2. Compilación y Ejecución:

2.1. Compilación del programa:

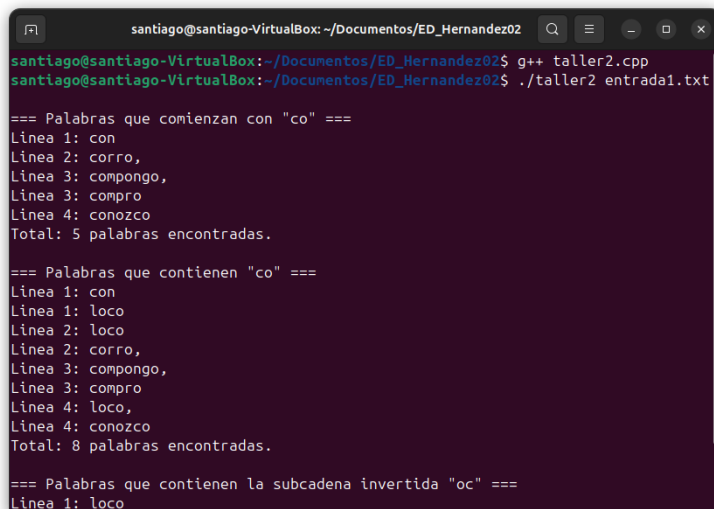
Se utilizó el compilador g++ para compilar el programa

2.2. Ejecución del programa:

El programa se ejecuta desde la línea de comandos pasando como argumento el nombre del archivo de entrada

`./taller02 entrada1.txt`

2.3. Captura de pantalla de la compilación y ejecución:



```
santiago@santiago-VirtualBox: ~/Documentos/ED_Hernandez02
santiago@santiago-VirtualBox:~/Documentos/ED_Hernandez02$ g++ taller2.cpp
santiago@santiago-VirtualBox:~/Documentos/ED_Hernandez02$ ./taller2 entrada1.txt

=== Palabras que comienzan con "co" ===
Linea 1: con
Linea 2: corro,
Linea 3: compongo,
Linea 3: compro
Linea 4: conozco
Total: 5 palabras encontradas.

=== Palabras que contienen "co" ===
Linea 1: con
Linea 1: loco
Linea 2: loco
Linea 2: corro,
Linea 3: compongo,
Linea 3: compro
Linea 4: loco,
Linea 4: conozco
Total: 8 palabras encontradas.

=== Palabras que contienen la subcadena invertida "oc" ===
Linea 1: loco
```

3. Diseño e Implementación:

3.1. TAD Palabra

El **TAD Palabra** se diseñó para almacenar una palabra y su número de línea en el archivo de texto.

Datos:

- palabra: Cadena de caracteres que representa la palabra.

- `n_linea`: Número de línea donde aparece la palabra.

Operaciones:

- `FijarPalabra(n_palabra)`: Asigna una nueva palabra.
- `FijarNumLinea(n_num)`: Asigna un nuevo número de línea.
- `ObtenerPalabra()`: Retorna la palabra almacenada.
- `ObtenerNumLinea()`: Retorna el número de línea.

```
class Palabra {
private:
    string palabra;
    unsigned int n_linea;

public:
    Palabra(string n_palabra = "", unsigned int n_num = 0)
        : palabra(n_palabra), n_linea(n_num) {}

    void FijarPalabra(string n_palabra) {
        palabra = n_palabra;
    }

    void FijarNumLinea(unsigned int n_num) {
        n_linea = n_num;
    }

    string ObtenerPalabra() const {
        return palabra;
    }

    unsigned int ObtenerNumLinea() const {
        return n_linea;
    }
};
```

4.2. TAD ArchivoTexto

El **TAD ArchivoTexto** se diseñó para gestionar el almacenamiento de líneas de texto y la búsqueda de palabras.

Datos:

- `lineasTexto`: Vector de líneas de texto.

Operaciones:

- `FijarListaLineas(n_lista)`: Asigna un nuevo vector de líneas.
- `ObtenerListaLineas()`: Retorna el vector de líneas de texto.
- `ObtenerNumLineas()`: Retorna la cantidad de líneas.
- `AgregarListaPals(n_lista)`: Agrega una nueva línea de texto.
- `BuscarPrincipio(subcadena)`: Busca palabras que comienzan con la subcadena.

- **BuscarContiene(subcadena):** Busca palabras que contienen la subcadena.
- **BuscarInvertida(subcadena):** Busca palabras que contienen la subcadena invertida.

4.3. Descripción de algoritmos y funciones:

- **invertirSubcadena():** Utiliza una **pila** para invertir la subcadena.
 - **dividirEnPalabras():** Utiliza stringstream para dividir cada línea en palabras.
 - **BuscarPrincipio():** Verifica si una palabra comienza con la subcadena.
 - **BuscarContiene():** Verifica si la subcadena aparece en cualquier posición.
 - **BuscarInvertida():** Invierte la subcadena y busca coincidencias.
-

5. Análisis de Resultados:

- Se verificaron los resultados obtenidos con diferentes archivos de entrada
 - Se validó la **cantidad de palabras encontradas** y el **número de línea** donde aparecían
 - Se confirmó la **correctitud** de las búsquedas con subcadenas en posiciones iniciales, intermedias y finales
 - El programa demostró un **rendimiento eficiente** en términos de procesamiento y búsqueda
-

6. Conclusiones:

- Se implementó una solución que cumple con los **TAD especificados** en el enunciado
 - Se aplicaron **estructuras lineales** (listas y pilas) para el almacenamiento y procesamiento de datos
 - El diseño basado en TAD garantiza un **código modular y escalable**, fácil de mantener y ampliar
 - Se logró una correcta **búsqueda de palabras** en diferentes contextos y posiciones dentro de las líneas de texto
-

Evidencia del Directorio de Archivos:

https://github.com/santiagohb2830/EDD/tree/main/EDD_Hernandez02