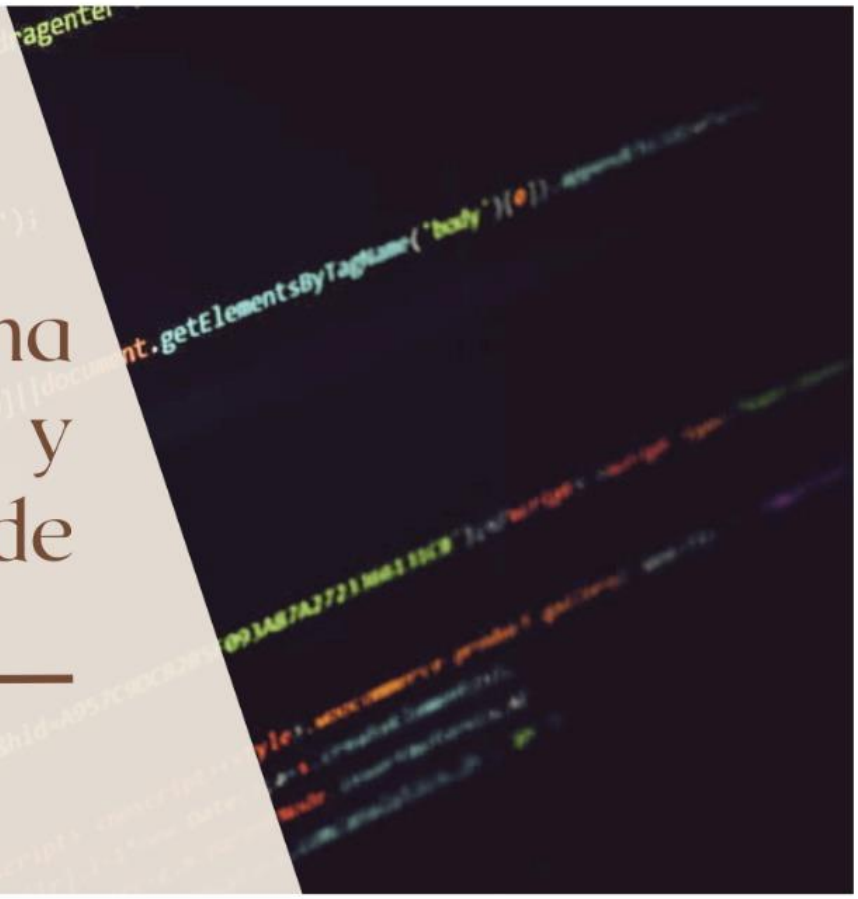
UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Facultad de Ingeniería de Sistemas e Informática



PROYECTO 8:

SecureText: Sistema de Encriptación y Manipulación de Texto en C++

GRUPO 1:

- Crisóstomo Altamirano, Axl Mikel 23200165
- Barrientos Torres, José Paolo 23200145
- Aceituno Huamán, Luis Gerónimo 23200002
- Torres Chuquiyauri, Manuel Angel 23200066
- Cruz Hilaquita, Rodrigo 23200020

PROFESOR:

- Guerra Grados, Luis Ángel

Contenido

Información general:

Nombre del Proyecto	SecureText: Sistema de Encriptación y Manipulación de Texto en C++
Usuario	
Preparado por:	• Crisóstomo Altamirano, Axl Mikel • Torres Chuquiyauri, Manuel Angel • Aceituno Huamán, Luis Gerónimo • Barrientos Torres, Jose Paolo • Cruz Hilaquita Rodrigo
Fecha de preparación	29/05/2024
Fecha de cierre	26/06/2024
Autorizado por	Guerra Grado Luis Angel

Propósito del proyecto (descripción):

Descripción:
El proyecto elegido consiste en desarrollar un menú de opciones en C++ que abarca diversas funciones de manipulación y encriptación de texto. A continuación, se detallan las opciones disponibles:

1. Opción A: Este programa lee líneas de texto, obtiene las palabras de cada línea y las escribe en pantalla en orden alfabético. Se considera que el máximo número de palabras por línea es 28. Todas las operaciones realizadas en esta opción se guardarán en un archivo llamado 'opcion_A.txt'.

2. Opción B: Este sistema de encriptación simple sustituye cada carácter de un mensaje por el carácter que se encuentra tres posiciones adelante en el alfabeto. Se debe implementar una función que tome una cadena como parámetro y devuelva otra cadena cifrada utilizando este método. Todas las operaciones realizadas en esta opción se guardarán en un archivo llamado 'opcion_B.txt'.
3. Opción C: Este sistema de encriptación sustituye cada carácter del alfabeto por otro predefinido. La función correspondiente debe recibir como parámetros el mensaje a cifrar y una cadena que contenga las correspondencias ordenadas de los caracteres alfabéticos. La función devolverá un puntero a la cadena cifrada del mensaje. Todas las operaciones realizadas en esta opción se guardarán en un archivo llamado 'opcion_C.txt'.
4. Opción D: Este método de encriptación sustituye todas las letras minúsculas por nuevas letras según un array de caracteres cifrados proporcionado. Por ejemplo, 'a' se reemplazará por cifrado[0], 'b' por cifrado[1], y así sucesivamente. Se requiere una función que reciba un texto y lo encripte utilizando este sistema. Todas las operaciones realizadas en esta opción se guardarán en un archivo llamado 'opcion_D.txt'.
5. Opción E: Este método de encriptación se basa en la sustitución de letras mayúsculas, donde cada carácter del mensaje se reemplaza según una correspondencia específica proporcionada. La entrada consiste en un número N que

indica la cantidad de conjuntos de datos. Para cada conjunto de datos, se proporciona una línea con el mensaje cifrado y otra con las correspondencias de los caracteres alfabéticos. La salida debe incluir el número del conjunto de datos y el mensaje descifrado. Todas las operaciones realizadas en esta opción se guardarán en un archivo llamado 'opcion_E.txt'.

6. Opción F: Esta nueva opción agregada permitirá mostrar todas las operaciones previamente realizadas y guardadas en archivos separados.

Propósito:
El propósito de este proyecto es desarrollar y fortalecer sus habilidades en programación en C++, enfocándose en la manipulación y encriptación de texto. A través del uso de distintos conocimientos obtenido en clase y de fuentes externas, usados para desarrollar el proyecto, pudimos hacer lo siguiente:

- **Mejoramos nuestros conocimientos en C++:** Al trabajar con distintas estructuras de datos y algoritmos, pudimos aplicar conceptos fundamentales del lenguaje C++.
- **Desarrollamos habilidades en manipulación de cadenas:** Las tareas requieren operaciones complejas de manejo de texto, incluyendo ordenamiento y transformación de cadenas.
- **Comprendimos técnicas de encriptación:** Exploramos diferentes métodos de encriptación, comprendiendo tanto sus aplicaciones como sus implementaciones.
- **Practicamos la lógica de programación:** Cada opción del menú presentaba un desafío lógico, lo cual nos ayudó a mejorar nuestras capacidades de resolución de problemas.

En resumen, este proyecto no solo nos sirvió como una herramienta de encriptación, sino también nos sirvió para aprender y practicar C++, también nos ayudó a comprender conceptos importantes de encriptación y manipulación de texto, y aplicar estos conocimientos de manera práctica y efectiva.

Objetivo del proyecto:

Metas del Grupo de Trabajo	Objetivos del proyecto
Desarrollar un Sistema de Menú Interactivo en C++	Crear una interfaz de menú en la consola que permita al usuario seleccionar entre diferentes opciones de procesamiento de texto.
Optimizar la Manipulación de Cadenas de Texto	Implementar una función que lea líneas de texto, separe las palabras y las ordene en orden alfabético, manejando hasta 28 palabras por línea.
Implementar Métodos Eficientes de Encriptación y Desencriptación	Desarrollar una función que encripte un mensaje sustituyendo cada carácter por el que está tres posiciones alfabéticas adelante (cifrado por desplazamiento).
Desarrollar Encriptación Personalizada	Crear una función que sustituya cada carácter del alfabeto por otro previamente

	decidido, utilizando una cadena de correspondencias.
Implementar la Encriptación de Minúsculas	Desarrollar una función que cambie todas las letras minúsculas por otras nuevas letras, según un array de caracteres cifrados.
Crear la Función de Decodificación de Mayúsculas	Desarrollar una función que descifre mensajes codificados utilizando una correspondencia específica de caracteres en mayúsculas.
Asegurar la Reutilización y Mantenibilidad del Código	Diseñar el código de manera modular y bien documentada, utilizando funciones separadas para cada tipo de encriptación y manipulación de texto.
Realizar Pruebas Unitarias y de Integración	Probar todas las funciones de manera individual y en conjunto para asegurar su correcto funcionamiento y eficiencia.
Garantizar la Consistencia de las Entradas de Datos	Implementar validaciones para asegurar que los datos ingresados por el usuario sean correctos y estén dentro del rango permitido, rechazando entradas inválidas y solicitando nuevos datos cuando sea necesario.

Contenido del Proyecto:

Fase 1: Análisis de Requisitos:

Establecer objetivos y alcance del proyecto:

1	**Crear una interfaz de menú en la consola que permita al usuario seleccionar entre diferentes opciones de procesamiento de texto.**
2	**Implementar una función que lea líneas de texto, separe las palabras y las ordene en orden alfabético, manejando hasta 28 palabras por línea. Las operaciones realizadas en esta opción serán guardadas en un archivo llamado 'operaciones_A.txt'.**
3	**Desarrollar una función que encripte un mensaje sustituyendo cada carácter por el que está tres posiciones alfabéticas adelante (cifrado por desplazamiento). Las operaciones realizadas en esta opción serán guardadas en un archivo llamado 'operaciones_B.txt'.**
4	**Crear una función que sustituya cada carácter del alfabeto por otro previamente decidido, utilizando una cadena de correspondencias. Las operaciones realizadas en esta opción serán guardadas en un archivo llamado 'operaciones_C.txt'.**
5	**Desarrollar una función que cambie todas las letras minúsculas por otras nuevas letras, según un array de caracteres cifrados. Las operaciones realizadas en esta opción serán guardadas en un archivo llamado 'operaciones_D.txt'.**
6	**Desarrollar una función que descifre mensajes codificados utilizando una correspondencia específica de caracteres en mayúsculas. Las operaciones realizadas en esta opción serán guardadas en un archivo llamado 'operaciones_E.txt'.**
7	**Desarrollar una función que se encargue de imprimir el contenido que se encuentra en cada archivo, previamente creado.**

8	**Diseñar el código de manera modular y bien documentada, utilizando funciones separadas para cada tipo de encriptación y manipulación de texto.**
9	**Probar todas las funciones de manera individual y en conjunto para asegurar su correcto funcionamiento y eficiencia.**
10	**Implementar validaciones para asegurar que los datos ingresados por el usuario sean correctos y estén dentro del rango permitido, rechazando entradas inválidas y solicitando nuevos datos cuando sea necesario.**

Definir roles y responsabilidades (Programador, Tester):

Programador:
- Implementación del menú de opciones y funcionalidades. **- Validación de entradas de usuario.**
Trabajo de cada programador:
• **Aceituno Huaman Luis Geronimo:** Creación del menú de opciones y de las funciones encargadas de la opción A y D. • **Crisostomo Altamirano Axl Mikel:** Creación de la función encargada de la opción C. • **Torres Chuquiyauri Manuel Angel:** Creación de la función encargada de la opción B. • **Barrientos Torres Jose Paolo:** Creación de la función encargada de la opción E. • **Cruz Hilaquita Rodrigo:** Creación del menú de opciones y de las funciones encargadas de la opción F.
Tester (Analista de Pruebas):
- Desarrollo de casos de prueba. **- Ejecución de pruebas unitarias y de integración.** **- Reporte y seguimiento de errores.**

Recolección de Requisitos:

Después de la reunión entre los integrantes, se pudo recolectar los requisitos y sobre todo se pudo entender la parte del trabajo que le tocaba a cada uno. A continuación, se muestra lo recolectado:

Parte del proyecto	Requisitos
Función menú de opciones	Para crear menú se deben imprimir las 5 opciones del proyecto y una adicional creada por el grupo de trabajo; el menú debe servir como puente para que el usuario pueda llamar directamente a la función requerida. Además, se debe asegurar que el usuario ingrese una opción correcta para evitar errores,
Función de la opción A	Para la creación de esta función, se debe pedir al usuario un texto que tenga hasta 28 palabras, luego mediante algoritmos se debe separar cada palabra, ordenarlos de manera alfabética. Se debe asegurar que el usuario no coloque más de 28 palabras, ya que es una condición del problema. Además se debe incluir un contador que tendrá la función de registrar el número de pruebas que se hicieron en esta opción.

Función de la opción B	Para crear esta función, primero toma un mensaje de texto ingresado por el usuario y lo cifra desplazando cada letra tres posiciones hacia adelante en el alfabeto. Por ejemplo, la letra 'A' se convierte en 'D', la letra 'B' se convierte en 'E', y así sucesivamente. El programa utiliza un bucle para iterar a través de cada letra del mensaje y aplica la lógica de cifrado correspondiente. Finalmente, muestra el mensaje cifrado al usuario. Además, se debe incluir un contador que tendrá la función de registrar el número de pruebas que se hicieron en esta opción.
Función de la opción C	Para crear la función de cifrado, primero se leerá el número de conjuntos de datos a analizar. Para cada conjunto, se obtendrá el mensaje codificado y la cadena de caracteres que define la correspondencia entre las letras del alfabeto. Luego, para cada carácter del mensaje codificado, se encontrará su correspondiente en la cadena de mapeo y se reemplazará por el carácter original del alfabeto. Los mensajes decodificados se almacenarán y finalmente se imprimirán con su respectivo número de conjunto. Además, se debe incluir un contador que tendrá la función de registrar el número de pruebas que se hicieron en esta opción.
Función de la opción D	Para la creación de esta función, se debe almacenar mediante un array los 26 caracteres que reemplazaran a las letras del alfabeto en minúscula. Se debe priorizar que solo se reemplaza las letras minúsculas, también los caracteres ingresados en el array no se deben repetir. Además, se debe incluir un contador que tendrá la función de registrar el número de pruebas que se hicieron en esta opción.
Función de la opción E	Para la creación de esta función, primero lee el número de conjuntos de datos a analizar. Para cada conjunto, obtiene el mensaje codificado y la cadena de caracteres que define la correspondencia entre las letras del alfabeto. Luego, para cada carácter del mensaje codificado, encuentra su correspondiente en la cadena de mapeo y lo reemplaza por el carácter original del alfabeto. Los mensajes decodificados se almacenan y finalmente se imprimen con su respectivo número de conjunto.
Función de la opción F	Esta función tiene la función de mostrar todo el contenido de los archivos que se crearon.

Fase 2: Pruebas:

Función Opciones:

Para la función menú, primero declaramos la función opciones() antes de la función principal main(). Esta se define como una función void que no retorna ningún valor.
Dentro de la función opciones(), se presenta un menú con diversas opciones para el usuario, las cuales son:

- Lee líneas de texto, obtiene las palabras y las muestra en orden alfabético (máximo 28 palabras por línea).
- Encripta un mensaje sustituyendo cada carácter por el que está tres posiciones adelante en el alfabeto.
- Encripta un mensaje sustituyendo cada carácter según una cadena de correspondencias predefinida (mediante punteros).
- Encripta un texto sustituyendo letras minúsculas por otras según un array cifrado.
- Encripta un mensaje sustituyendo cada letra por otra en todo el mensaje.
- Mostrará las operaciones que se realizaron en todas las funciones
- Salir.

El usuario debe elegir una opción ingresando una letra minúscula. La opción seleccionada se guarda en una variable y se maneja mediante un switch, permitiendo al usuario acceder a las funciones correspondientes. Se han definido cinco casos específicos:

- a: Accede a la función asociada a la opción A y luego retorna al menú de opciones.
- b: Accede a la función asociada a la opción B y luego retorna al menú de opciones.
- c: Accede a la función asociada a la opción C y luego retorna al menú de opciones.
- d: Accede a la función asociada a la opción D y luego retorna al menú de opciones.
- e: Accede a la función asociada a la opción E y luego retorna al menú de opciones.
- f: Accede a la función asociada a la opción F y luego retorna al menú de opciones.
- g: Finaliza el menú y termina el programa.

Explicación del código:

Manejo de opciones no válidas:

- En caso de que el usuario ingrese una opción no válida, se activa el caso por defecto ("default") del switch. Esto hace que el programa le solicite nuevamente al usuario que ingrese una opción válida. Para ello, se utiliza

un bucle `while` que se repetirá hasta que el usuario introduzca la opción "g", que indica que desea salir del menú.

Contadores de operaciones:

- Al inicio de la función, se han agregado contadores para las primeras cinco opciones (a, b, c, d, e). Estos contadores tienen como objetivo registrar el número de veces que se ha seleccionado cada una de estas opciones.

Limpieza de la consola:

- Para evitar que la consola se llene de información innecesaria, se ha agregado la función `system("cls");` después de cada selección de opción. Esta función limpia la pantalla, lo que mejora la experiencia del usuario.

```
void opciones() {
    int contador_a=1, contador_b=1,
contador_c=1,contador_d=1,contador_e=1;
    char opcion;
    do {
        cout << "\n\n\t\t\tMENU PRINCIPAL\n";
        cout<<"\t\t\t--------------\n"<<endl;
        cout << "\n\ta. Lee lineas de texto, obtiene las palabras y las
muestra en orden alfabetico (maximo 28 palabras por linea).\n";
        cout << "\tb. Encripta un mensaje sustituyendo cada caracter por
el que esta tres posiciones adelante en el alfabeto.\n";
        cout << "\tc. Encripta un mensaje sustituyendo cada caracter
segun una cadena de correspondencias predefinida (punteros).\n";
        cout << "\td. Encripta un texto sustituyendo letras minusculas
por otras segun un array cifrado.\n";
        cout << "\te. Encripta un mensaje sustituyendo cada letra por
otra en todo el mensaje.\n";
        cout << "\tf. Mostrar las operaciones realizadas.\n";
        cout << "\tg. Salir.\n\n";
        cout << "\n\tElija que opcion desea (solo con letras minusculas):
";
        cin >> opcion;
        system("cls");
        cin.ignore();
        switch (opcion) {
            case 'a':
                opcion_A(contador_a);
                contador_a++;
                break;
            case 'b':
                opcion_B(contador_b);
                contador_b++;
```

```
                break;
            case 'c':
                opcion_C(contador_c);
                contador_c++;
                break;
            case 'd':
                opcion_D(contador_d);
                contador_d++;
                break;
            case 'e':
                opcion_E(contador_e);
                contador_e++;
                break;
            case 'f':
                LEER();
                break;
            case 'g':
                cout << "\n\tPresione ENTER para cerrar la
aplicacion.\n";
                break;
            default:
                cout << "\n\n\t\t\t::: Ingrese una opcion valida ::: \n";
                break;
        }
    } while (opcion != 'g');
}
```

Opción A:

```
void opcion_A(int &contador) {
    int e = 0;
    string mayuscula = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string minuscula = "abcdefghijklmnopqrstuvwxyz";
    string texto=" ";
    string cadena[28];
    int C;
    cout<<"\n\t\tOPCION A \n\t\t---------\n"<<endl;
    cout << "\n\tIngrese un texto para dividir (max 28 palabras): ";
    getline(cin, texto);
    C=CONTADOR(texto);
    while(C>28){
        texto="";
        cout << "\n\tIngrese un texto para dividir (max 28 palabras): ";
        getline(cin, texto);
        C=CONTADOR(texto);
    }
    for (int i = 0; i < texto.length(); i++) {
        for (int k = 0; k < 26; k++) {
            if (texto[i] == mayuscula[k]) {
```

```
                texto[i] = minuscula[k];
            }
        }
    }
    istringstream palabra(texto);
    while (palabra >> cadena[e]) {
        e++;
    }
    sort(cadena, cadena + e);
    //Escribir en archivo
    ofstream archivo("opcion_A.txt",ios::app);
    archivo<<"\n\t\t\t::: OPERACION #"<<contador<<" :::\n\n"<<endl;
    for (int i = 0; i < e; i++) {
        cout <<"\n\t\t"<< i + 1 << "..." << cadena[i] << endl;
        archivo<<"\t\t"<<setfill('-')<<setw(21)<<"-"<<endl;
        archivo <<left<<setfill(' ')<<"\t\t"<<"|"<<setw(3)<< i + 1 << "|"
<<setw(15)<< cadena[i] <<"|"<< endl;
        archivo<<"\t\t"<<setfill('-')<<setw(21)<<"-"<<endl;
    }
    archivo.close();
}
//Función contador
int CONTADOR(string texto){
    int C;
    if(texto[0]=' '){
        C=0;
    }
    else{
        C=1;
    }
    for(int i=0; i<texto.length();i++){
        if(texto[i]==' ' && texto[i+1]!=' ')
        {
            C++;
        }
        else{
            continue;
        }
    }
    return C;
}
```

Explicación del código:

Inicialización de variables:

- e: Contador de palabras, inicialmente 0.

- `cadena`: Vector de strings para almacenar las palabras del texto, con un tamaño máximo de 28.
- `mayusculas`: String que contiene todas las letras del alfabeto en mayúsculas.
- `minusculas`: String que contiene todas las letras del alfabeto en minúsculas.
- `texto`: String para almacenar el texto ingresado por el usuario.
- C: Int que guarda el número de palabras que se ingresó en el texto.

Captura de texto del usuario:

- Se utiliza `getline(cin, texto)` para capturar todo el texto ingresado por el usuario, incluyendo espacios y caracteres especiales.

Conteo de palabras:

- Se llama a la función auxiliar `CONTADOR(texto)` para obtener el número de palabras en el texto ingresado.
- Se actualiza el contador `C` con el valor devuelto por la función `CONTADOR`.

Conversión a minúsculas:

- Se recorre cada carácter del texto `texto` utilizando un bucle `for`.
- Si se encuentra una letra mayúscula, se reemplaza por su equivalente en minúscula utilizando las variables `mayusculas` y `minusculas`.

Almacenamiento de palabras en vector:

- Se crea un objeto `istringstream` llamado `palabra(texto)`.
- Se utiliza un bucle `while (palabra >> cadena[e])` para leer y almacenar cada palabra del texto en el vector `cadena`.
- El proceso continúa hasta que no hay más palabras para procesar.
- Por cada palabra ingresada la variable e aumentara en 1.

Ordenamiento alfabético de palabras:

- Se utiliza la función `sort(cadena, cadena + e)` para ordenar alfabéticamente las palabras almacenadas en el vector `cadena`.

Creación y escritura en archivo:

- Se crea un archivo con el nombre `opcion_A.txt`.
- Se utiliza un bucle `for` para imprimir cada palabra del vector `cadena` en la consola.
- Se utiliza otro bucle `for` para escribir cada palabra del vector `cadena` en el archivo `opcion_A.txt`.

Pruebas de la opción A:

```
                OPCION A
                ---------


        Ingrese un texto para dividir (max 28 palabras): Serendipia eufonía melodía armonía sinfonía orquesta batuta compos
itor intérprete músico melodiosa armoniosa sinfónica

                1...armoniosa

                2...armonía

                3...batuta

                4...compositor

                5...eufonía

                6...intérprete

                7...melodiosa

                8...melodía

                9...músico

                10...orquesta

                11...serendipia

                12...sinfonía

                13...sinfónica
```

Opción B:

```
void opcion_B(int &contador) {
    string mensaje;
    string cifrado = "";
    cout<<"\n\t\tOPCION B \n\t\t---------\n"<<endl;
    cout << "\n\tIntroduce el mensaje a cifrar: ";
    getline(cin, mensaje);
    for (size_t i = 0; i < mensaje.size(); ++i) {
        char c = mensaje[i];
        if (c >= 'A' && c <= 'Z') {
            cifrado += char((c - 'A' + 3) % 26 + 'A');
        } else if (c >= 'a' && c <= 'z') {
            cifrado += char((c - 'a' + 3) % 26 + 'a');
        } else {
            cifrado += c;
        }
    }
    cout << "\n\t\tMensaje cifrado: " << cifrado << endl;
    ofstream archivo("opcion_B.txt",ios::app);
    archivo<<"\t\t::: OPERACION #"<<contador<<" :::\n\n"<<endl;

    archivo<<"\t\t"<<setfill('-')<<setw(71)<<"-"<<endl;
    archivo <<left<<setfill(' ')<<"\t\t|"<< "Mensaje original: "
<<"|"<<setw(50)<< mensaje <<"|"<< endl;
    archivo<<"\t\t"<<setfill('-')<<setw(71)<<"-"<<endl;
    archivo <<left<<setfill(' ')<<"\t\t|"<<"Mensaje cifrado:  "
<<"|"<<setw(50)<< cifrado <<"|"<< endl;
    archivo<<"\t\t"<<setfill('-')<<setw(71)<<"-"<<endl;
    archivo.close();
}
```

Explicación del Código:

Declaración de variables:

- `string mensaje`: Esta variable almacenará el mensaje introducido por el usuario.
- `string cifrado`: Esta cadena almacenará el mensaje cifrado.

Salida:

- La función imprime un título y un mensaje para que el usuario introduzca un mensaje.

Entrada:

- Utiliza `getline(cin, mensaje)` para leer toda la línea introducida por el usuario, incluidos los espacios, y la almacena en la variable `mensaje`.

Bucle de cifrado:

- La función recorre cada carácter `c` de la cadena `mensaje` utilizando un bucle `for`.
- Para cada carácter:
  - Si `c` es una letra mayúscula (entre 'A' y 'Z'):
    - Realiza un cifrado César desplazando el carácter 3 posiciones (envolviendo si supera la 'Z'). El resultado se almacena de nuevo en `c`.
  - Si `c` es una letra minúscula (entre 'a' y 'z'):
    - Similar a las mayúsculas, desplaza el carácter 3 posiciones (envolviendo si supera la 'z').
  - En caso contrario, si `c` no es una letra, permanece sin cambios.
  - El carácter cifrado `c` se añade a la cadena `cifrado`.

Salida del mensaje cifrado:

- La función imprime el mensaje cifrado almacenado en `cifrado`.

Entrada/Salida de archivos (opcional - depende del uso de `contador`):

- Si `contador` se utiliza para llevar un registro del número de cifrados, esta parte escribe los mensajes originales y cifrados en un archivo de salida llamado "opcion_B.txt" en modo de apéndice (`ios::app`).
- Formatea la salida en el archivo utilizando diversas técnicas de formato como `setfill` y `setw` para una mejor legibilidad.

Pruebas de la opción B:

```
        OPCION B
        ---------

Introduce el mensaje a cifrar: PROGRAMACION CICLO IV

        Mensaje cifrado: SURJUDPDFLRQ FLFOR LY
```

Opción C:

```
void opcion_C(int &contador) {
    string mensajeOriginal;
    string claveCorrespondencia;
    cout<<"\n\t\tOPCION C \n\t\t---------\n"<<endl;
    cout << "\n\tIngrese el mensaje codificado: ";
    getline(cin, mensajeOriginal);
    cout << "\n\t\tIngrese la clave de correspondencia: ";
    getline(cin, claveCorrespondencia);
    if(claveCorrespondencia.length()!=26){
        cout << "\n\t\t La clave de correspondencia debe tener 26
caracteres: ";
        getline(cin, claveCorrespondencia);
    }

    char* mensajeCifrado = cifrarMensaje(mensajeOriginal,
claveCorrespondencia);

    cout << "\n\tMensaje cifrado: '" << mensajeCifrado << "'" <<
endl;
    cout << "\n\t\tUbicacion del puntero: " <<
static_cast<void*>(mensajeCifrado) << endl;

    ofstream archivo("opcion_C.txt",ios::app);
    archivo<<"\n\t\t::: OPERACION #"<<contador<<" :::\n\n"<<endl;
    archivo<<"\t\t"<<setfill('-')<<setw(79)<<"-"<<endl;
    archivo<<"\t\t" <<left<<setfill(' ')<<"|"<<setw(27)<< "Mensaje
original: " <<"|"<< setw(50)<<mensajeOriginal<<"|" << endl;
    archivo<<"\t\t"<<setfill('-')<<setw(79)<<"-"<<endl;
    archivo<<"\t\t" <<left<<setfill(' ')<<"|"<<setw(27)<< "Clave de
correspondencia: " <<"|"<<setw(50)<< claveCorrespondencia <<"|"<<
endl;
    archivo<<"\t\t"<<setfill('-')<<setw(79)<<"-"<<endl;
    archivo <<"\t\t"<<left<<setfill(' ')<<"|"<<setw(27)<< "Mensaje
cifrado: " <<"|"<<setw(50)<< mensajeCifrado <<"|"<< endl;
    archivo<<"\t\t"<<setfill('-')<<setw(79)<<"-"<<endl;
    archivo.close();

    delete[] mensajeCifrado;
```

```
}
```

Explicación código:

Entrada de datos:

- Se solicitan al usuario dos entradas:
  - `mensajeOriginal`: El mensaje que se desea cifrar.
  - `claveCorrespondencia`: Una cadena de 26 caracteres que representa la clave de cifrado. Se valida que la longitud de la clave sea exactamente 26.

Cifrado del mensaje:

- Se llama a una función `cifrarMensaje` (no se muestra en el código proporcionado) que presumiblemente recibe el `mensajeOriginal` y la `claveCorrespondencia` como parámetros y devuelve el mensaje cifrado en un nuevo arreglo de caracteres `mensajeCifrado`.

Salida del mensaje cifrado:

- Se imprime el mensaje cifrado contenido en `mensajeCifrado`.
- Se muestra la ubicación del puntero de memoria del `mensajeCifrado` con `static_cast<void*>(mensajeCifrado)`.

Escritura en archivo (opcional - depende del uso de `contador`):

- Similar a la función `opcion_B`, esta parte escribe los mensajes originales, la clave de correspondencia y el mensaje cifrado en un archivo de salida llamado "opcion_C.txt" en modo de apéndice (`ios::app`).
- Formatea la salida en el archivo utilizando técnicas de formato como `setfill` y `setw` para una mejor legibilidad.

Liberación de memoria:

- Se utiliza `delete[] mensajeCifrado` para liberar la memoria asignada dinámicamente al arreglo `mensajeCifrado`.

Pruebas de la Opción C:

```
        OPCION C
        ---------


Ingrese el mensaje codificado: mb mak sksll Hiia

        Ingrese la clave de correspondencia: bjmsxvfwketrznaihqcoypuldg

Mensaje cifrado: 'zj zbt ctcrr Hkkb'

        Ubicacion del puntero: 0x18c8e831e10
```

Opción D:

```
void opcion_D(int &contador) {
    string palabra, alfabeto = "abcdefghijklmnopqrstuvwxyz";
    char codigo[26];
    cout<<"\n\t\tOPCION D \n\t\t---------\n"<<endl;
    for (int i = 0; i < 26; i++) {
        cout << "\n\tIngrese el caracter #" << i + 1 << " que
serviran para encriptar: ";
        cin >> codigo[i];

        for (int k = 0; k < i; k++) {
            if (codigo[i] == codigo[k]) {
                while (codigo[i] == codigo[k]) {
                    cout << "\n\t\tIngrese un caracter no repetido:
";
                    cin >> codigo[i];
                }
            } else {
                continue;
            }
        }
    }
    cin.ignore();
    cout << "\n\tIngrese el texto que desea encriptar: ";
    getline(cin, palabra);
    string palabraOriginal = palabra;
    for (int i = 0; i < palabra.size(); i++) {
        for (int k = 0; k < 26; k++) {
            if (palabra[i] == alfabeto[k]) {
                palabra[i] = codigo[k];
                break;
            } else {
                continue;
            }
        }
    }
    cout << "\n\tEl texto encriptado es: " << palabra << endl;
```

```
    ofstream archivo("opcion_D.txt",ios::app);
    archivo<<"\n\t\t::: OPERACION #"<<contador<<" :::\n\n"<<endl;
    archivo<<"\t\t"<<setfill('-')<<setw(69)<<"-"<<endl;
    archivo <<left<<"\t\t|"<<setfill(' ')<<setw(16)<< "Texto
original: " <<"|"<<setw(50)<<palabraOriginal<<"|" << endl;
    archivo<<"\t\t"<<setfill('-')<<setw(69)<<"-"<<endl;
    archivo <<left<<"\t\t|"<<setfill(' ')<<setw(16)<<"Texto
encriptado: " <<"|"<<setw(50)<< palabra<<"|" << endl;
    archivo<<"\t\t"<<setfill('-')<<setw(69)<<"-"<<endl;
    archivo.close();
}
```

Explicación del código:

Inicialización:

- Se declara una variable `palabra` para almacenar el texto que se desea encriptar.
- Se define una cadena `alfabeto` que contiene el alfabeto minúsculo del inglés.
- Se crea un arreglo de caracteres `codigo` de tamaño 26 para almacenar la clave de cifrado.

Entrada de la clave de cifrado:

- Se utiliza un bucle `for` para solicitar al usuario que ingrese un caracter diferente para cada posición del arreglo `codigo`.
- Dentro del bucle, se verifica si el caracter ingresado ya existe en el arreglo `codigo`. Si es así, se le pide al usuario que ingrese un nuevo caracter hasta que no haya repeticiones.

Ignorar entrada incompleta:

- Se utiliza `cin.ignore()` para descartar cualquier caracter sobrante en el búfer de entrada después de leer la clave de cifrado.

Entrada del texto a encriptar:

- Se solicita al usuario que ingrese el texto que desea encriptar y se almacena en la variable `palabra`.
- Se crea una copia de la `palabra` original en la variable `palabraOriginal` para conservarla sin modificar.

Cifrado del texto:

- Se utiliza un bucle `for` para recorrer cada caracter de la `palabra`.
- Dentro del bucle, se utiliza otro bucle `for` para comparar el caracter actual de la `palabra` con cada caracter del `alfabeto`.
- Si se encuentra una coincidencia, se reemplaza el caracter de la `palabra` por el caracter correspondiente del `codigo`.

Salida del texto encriptado:

- Se imprime el texto encriptado almacenado en la variable `palabra`.

Escritura en archivo (opcional - depende del uso de `contador`):

- Similar a las funciones `opcion_B` y `opcion_C`, esta parte escribe el texto original, la clave de cifrado y el texto encriptado en un archivo de salida llamado "opcion_D.txt" en modo de apéndice (`ios::app`).
- Formatea la salida en el archivo utilizando técnicas de formato como `setfill` y `setw` para una mejor legibilidad.

Pruebas de la función D:

```
        OPCION D
        ---------

Ingrese el caracter #1 que serviran para encriptar: Q
Ingrese el caracter #2 que serviran para encriptar: W
Ingrese el caracter #3 que serviran para encriptar: E
Ingrese el caracter #4 que serviran para encriptar: R
Ingrese el caracter #5 que serviran para encriptar: T
Ingrese el caracter #6 que serviran para encriptar: Y
Ingrese el caracter #7 que serviran para encriptar: U
Ingrese el caracter #8 que serviran para encriptar: I
Ingrese el caracter #9 que serviran para encriptar: O
Ingrese el caracter #10 que serviran para encriptar: P
Ingrese el caracter #11 que serviran para encriptar: A
Ingrese el caracter #12 que serviran para encriptar: S
Ingrese el caracter #13 que serviran para encriptar: D
Ingrese el caracter #14 que serviran para encriptar: F
Ingrese el caracter #15 que serviran para encriptar: G
Ingrese el caracter #16 que serviran para encriptar: H
Ingrese el caracter #17 que serviran para encriptar: J
Ingrese el caracter #18 que serviran para encriptar: K
```

```
Ingrese el caracter #19 que serviran para encriptar: L
Ingrese el caracter #20 que serviran para encriptar: Z
Ingrese el caracter #21 que serviran para encriptar: X
Ingrese el caracter #22 que serviran para encriptar: C
Ingrese el caracter #23 que serviran para encriptar: V
Ingrese el caracter #24 que serviran para encriptar: B
Ingrese el caracter #25 que serviran para encriptar: N
Ingrese el caracter #26 que serviran para encriptar: M
Ingrese el texto que desea encriptar: Mbvcxz
El texto encriptado es: MWCEBM
```

Opción E:

```
void opcion_E(int &contador) {
    string ALFABETO="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string minuscula = "abcdefghijklmnopqrstuvwxyz";
    struct datos{
    string CODIGO;
    string claveCorrespondencia;
    string DECO;
    };
    int longitud = 0;
    int NUM;
    string auxiliar,auxiliar_2,auxiliar_3;
    cout<<"\n\t\tOPCION E \n\t\t---------\n"<<endl;
    cout<<"Ingrese el numero de decodificaciones que realizara
(1<=NUM<=100):"; cin>>NUM;
    while(NUM<1 or NUM>100){
        cout<<"Ingrese un numero valido(1<=NUM<=100):"; cin>>NUM;
    }
    vector<datos>DATOS(NUM);

    for (int h = 0; h < NUM; h++) {
        cin.ignore();
        cout << "\n\tIngresa el mensaje codificado "<<h+1<<": ";
        getline(cin, DATOS[h].CODIGO);
        cin.ignore();
        cout << "\n\tIngrese la clave de correspondencia "<<h+1<<":
";
        getline(cin, DATOS[h].claveCorrespondencia);
        while(DATOS[h].claveCorrespondencia.length()!=26){
            cout << "\n\t\t La clave de correspondencia debe tener
26 caracteres: ";
```

```
            getline(cin, DATOS[h].claveCorrespondencia);
        }
        auxiliar_3=DATOS[h].claveCorrespondencia;
        for (int i = 0; i<26; i++) {
                for (int k = 0; k < 26; k++) {
                    if (auxiliar_3[i] == minuscula[k]) {
                        auxiliar_3[i] = ALFABETO[k];
                    }
                }
        }
        DATOS[h].claveCorrespondencia=auxiliar_3;
        auxiliar=DATOS[h].CODIGO;
        auxiliar_2= DATOS[h].claveCorrespondencia;
        longitud = auxiliar.length();
        for (int i = 0; i < longitud; i++) {
            for (int k = 0; k < 26; k++) {
                if (auxiliar[i]== auxiliar_2[k]) {
                    auxiliar[i] = ALFABETO[k];
                    break;
                } else {
                    continue;
                }
            }
        }
        DATOS[h].DECO = auxiliar;
        auxiliar="";
        auxiliar_2="";
        longitud = 0;
    }
    ofstream archivo("opcion_E.txt",ios::app);
    archivo<<"\t\t::: OPERACION #"<<contador<<" :::"<<endl;
    for (int h = 0; h < NUM; h++) {
        cout<<"\n\n\t\t"<<h+1<<"...Clave de correspondecia:
"<<DATOS[h].claveCorrespondencia<<endl;
        cout<<"\t\t"<<h+1<<"...Mensaje codificado:
"<<DATOS[h].CODIGO<<endl;
        cout<<"\t\t"<<h+1<<"...Mensaje decodificado:
"<<DATOS[h].DECO<<endl;
        archivo<<"\t"<<setfill('-')<<setw(83)<<"-"<<endl;
        archivo<<left<<setfill('
')<<"\t|"<<setw(2)<<h+1<<setw(28)<<"| Clave de correspondecia:
"<<"|"<<setw(50)<<DATOS[h].claveCorrespondencia<<"|"<<endl;
        archivo<<"\t"<<setfill('-')<<setw(83)<<"-"<<endl;
        archivo<<left<<setfill('
')<<"\t|"<<setw(2)<<h+1<<setw(28)<<"| Mensaje codificado:
"<<"|"<<setw(50)<<DATOS[h].CODIGO<<"|"<<endl;
        archivo<<"\t"<<setfill('-')<<setw(83)<<"-"<<endl;
```

```
        archivo<<left<<setfill('
')<<"\t|"<<setw(2)<<h+1<<setw(28)<<"| Mensaje decodificado:
"<<"|"<<setw(50)<<DATOS[h].DECO<<"|"<<endl;
        archivo<<"\t"<<setfill('-')<<setw(83)<<"-"<<endl;
    }
    archivo.close();
}
```

Explicación del código:

Declaración de variables:

- Se definen constantes para el alfabeto mayúscula (`ALFABETO`) y minúscula (`minuscula`).
- Se crea una estructura `datos` para almacenar la información de cada decodificación:
    - `CODIGO`: El mensaje codificado a decodificar.
    - `claveCorrespondencia`: La clave de correspondencia para la decodificación.
    - `DECO`: El mensaje decodificado.
- Se declaran variables `NUM`, `auxiliar`, `auxiliar_2`, `auxiliar_3` y `longitud`.

Entrada del número de decodificaciones:

- Se solicita al usuario que ingrese el número de decodificaciones a realizar (`NUM`).
- Se valida que `NUM` esté entre 1 y 100, solicitando la entrada nuevamente si no es válida.

Creación de un vector de estructuras:

- Se crea un vector `DATOS` de tamaño `NUM` utilizando la estructura `datos` definida anteriormente.

Bucle de decodificación:

- Se recorre el vector `DATOS` utilizando un bucle `for`.
- Para cada decodificación (índice `h`):
    - Se ignora cualquier caracter sobrante en el búfer de entrada.
    - Se solicita al usuario que ingrese el mensaje codificado (`DATOS[h].CODIGO`).
    - Se ignora cualquier caracter sobrante en el búfer de entrada.
    - Se solicita al usuario que ingrese la clave de correspondencia (`DATOS[h].claveCorrespondencia`).
    - Se valida que la longitud de la clave sea 26 caracteres, solicitando la entrada nuevamente si no es válida.
    - Se realiza una copia de la clave de correspondencia en `auxiliar_3`.
    - Se recorre la cadena `auxiliar_3` (clave de correspondencia):
        - Para cada caracter en `auxiliar_3`:

                - Se recorre el alfabeto minúscula (`minuscula`).
                - Si el caracter actual de `auxiliar_3` coincide con un caracter del alfabeto minúscula, se reemplaza por el caracter correspondiente del alfabeto mayúscula (`ALFABETO`).
        - Se actualiza la clave de correspondencia en `DATOS[h].claveCorrespondencia`.
        - Se inicializan variables auxiliares (`auxiliar`, `auxiliar_2`, `longitud`).
        - Se obtiene la longitud del mensaje codificado.
        - Se recorre el mensaje codificado (`DATOS[h].CODIGO`):
            - Para cada caracter en el mensaje codificado:
                - Se recorre el alfabeto mayúscula (`ALFABETO`).
                - Si el caracter actual del mensaje codificado coincide con un caracter del alfabeto mayúscula, se reemplaza por el caracter correspondiente del alfabeto minúscula (`minuscula`).
                - Si no hay coincidencia, se continúa con el siguiente caracter del mensaje codificado.
        - Se almacena el mensaje decodificado en `DATOS[h].DECO`.
        - Se inicializan las variables auxiliares (`auxiliar`, `auxiliar_2`, `longitud`).

Escritura en archivo (opcional - depende del uso de `contador`):

- Se abre el archivo "opcion_E.txt" en modo de apéndice (`ios::app`).
- Se imprime un encabezado para la operación actual (`contador`).
- Se recorre el vector `DATOS` utilizando un bucle `for`.
- Para cada decodificación (índice `h`):
    - Se imprime la clave de correspondencia, el mensaje codificado y el mensaje decodificado.
    - Se escribe la información formateada en el archivo utilizando técnicas de formato como `setfill` y `setw`.
- Se cierra el archivo.

Opción F:

```
void LEER(){
 ifstream archivo;
 string texto;
 archivo.open("opcion_A.txt",ios::in);
 cout<<"\n\t\tARCHIVO OPCION A \n\t\t-----------------"<<endl;

    if(archivo.fail()){
    cout<<"\n\t\t :::No se realizo ni una operacion en esta
opcion:::"<<endl;
    }
    else{
    while(!archivo.eof()){
```

```
            getline(archivo,texto);
            cout<<texto<<endl;
            }
    }
    archivo.close();
    texto="";
archivo.open("opcion_B.txt",ios::in);
 cout<<"\n\t\tARCHIVO OPCION B \n\t\t-----------------"<<endl;
    if(archivo.fail()){
    cout<<"\n\t\t :::No se realizo ni una operacion en esta
opcion:::"<<endl;
    }
    else{
    while(!archivo.eof()){
            getline(archivo,texto);
            cout<<texto<<endl;
            }
    }
    archivo.close();
archivo.close();
texto="";

archivo.open("opcion_C.txt",ios::in);
 cout<<"\n\t\tARCHIVO OPCION C \n\t\t-----------------"<<endl;
    if(archivo.fail()){
    cout<<"\n\t\t :::No se realizo ni una operacion en esta
opcion:::"<<endl;
    }
    else{
    while(!archivo.eof()){
            getline(archivo,texto);
            cout<<texto<<endl;
            }
    }
    archivo.close();
archivo.close();

texto="";
archivo.open("opcion_D.txt",ios::in);
 cout<<"\n\t\tARCHIVO OPCION D \n\t\t-----------------"<<endl;
    if(archivo.fail()){
    cout<<"\n\t\t :::No se realizo ni una operacion en esta
opcion:::"<<endl;
    }
    else{
    while(!archivo.eof()){
            getline(archivo,texto);
            cout<<texto<<endl;
            }
```

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

texto="";
archivo.open("opcion_E.txt",ios::in);
 cout<<"\n\t\tARCHIVO OPCION E \n\t\t-----------------"<<endl;
    if(archivo.fail()){
    cout<<"\n\t\t :::No se realizo ni una operacion en esta
opcion:::"<<endl;
    }
    else{
    while(!archivo.eof()){
            getline(archivo,texto);
            cout<<texto<<endl;
            }
    }
    archivo.close();
archivo.close();

}
```

Explicación del código:

Declaración de variables:

- Se declara una variable `string texto` para almacenar el contenido de cada línea leída de los archivos.
- Se declara una variable `ifstream archivo` para manejar la apertura y lectura de los archivos.

Lectura de cada archivo:

- En cada uno de los 5 casos se abre el archivo.
    - Se abre el archivo actual utilizando `archivo.open(nombre_archivo, ios::in)`.
    - Se verifica si la apertura del archivo fue exitosa utilizando `if(archivo.fail())`.
        - Si la apertura falla, se imprime un mensaje indicando que no se han realizado operaciones en esa opción del menú.
        - Si la apertura es exitosa:
            - Se utiliza un bucle `while(!archivo.eof())` para leer cada línea del archivo.
            - Dentro del bucle `while`:
                - Se lee una línea del archivo actual en la variable `texto` utilizando `getline(archivo, texto)`.

- Se imprime el contenido de la línea leída en la consola utilizando `cout << texto << endl;`.
- Se cierra el archivo actual utilizando `archivo.close()`.

Planeamiento Inicial del Proyecto al alto nivel:

Estimación de recursos requeridos:

- 1 líder de proyecto
- 1 sublíder de proyecto
- 4 programadores
- 2 Tester

Estimación de fechas a programar:

- Fecha de inicio: 29/05/2024
- Fecha de entrega del avance del proyecto: 12/06/2024
- Fecha de entrega del proyecto final: 25/06/2024

Hitos del proyecto:

Hitos o evento significativo	Fecha programada
Repartimiento de tareas a cada integrante del grupo	29/05/2024
Primera revisión de las tareas de cada integrante	05/06/2024
Revisión del avance del proyecto	12/06/2024
Segunda revisión de las tareas de cada integrante	19/06/2024
Exposición y entrega del proyecto final	26/06/2024

Autoridad del proyecto:

- **Líder de proyecto: Crisóstomo Altamirano Axl Mikel**

Integrantes del equipo del proyecto, Roles y Responsabilidades:

1. Líder del proyecto: Crisóstomo Altamirano Axl Mikel
2. Sublíder del proyecto: Aceituno Huamán Luis Gerónimo
3. Programador 1: Crisostomo Altamirano Axl Mikel
4. Programador 2: Torres Chuquiyauri Manuel Torres
5. Programador 3: Barrientos Torres Jose Paolo

6. Programador 4: Aceituno Huaman Luis Geronimo
7. Programado 5: Cruz Hilaquita Rodrigo
8. Tester 1: Torres Chuquiyauri Manuel Torres
9. Tester 2: Aceituno Huaman Luis Geronimo

Firmas:

Nombres	Códigos	Firmas
Crisóstomo Altamirano Axl Mikel	23200165	
Barrientos Torres Jose Paolo	23200145	
Ageituno Huamán Luis Gerónimo	23200002	
Torres Chuquiyauri Manuel Angel	23200066	
Cruz Hilaquita Rodrigo	23200020