Facultad de Ingeniería

Departamento de Ingeniería de Sistemas

Introducción a la Programación

Entrega Final - Documentación

Presentado por:

Esteban Blanco Madrid

Gabriela Muñoz Martínez

Presentado a:

Ing. Johan Santiago Romero Duarte

Martes 21 de mayo de 2024

Bogotá D.C.

Introducción

El presente documento ofrece una descripción detallada del proyecto de gestión de datos meteorológicos como parte de los requisitos del curso de Introducción a la Programación. Este sistema básico permite registrar y autenticar usuarios, leer y procesar datos meteorológicos de un archivo y realizar algunas operaciones analíticas básicas sobre estos datos.

Este documento tiene como objetivo proporcionar una guía del proyecto clara y detallada, explicando los propósitos de las funciones, cómo se implementan y cómo se relacionan entre sí. Se pretende también demostrar cómo usar conceptos básicos de programación en C++ para crear una solución práctica y funcional a un problema sencillo.

En las secciones siguientes se describirán en detalle cada uno de los componentes del sistema, incluyendo la estructura del programa, los algoritmos utilizados y los resultados obtenidos a partir del análisis de datos meteorológicos.

Objetivos

Objetivo General:

Desarrollar un sistema de información integral que permita analizar datos meteorológicos de Bogotá, facilitando la toma de decisiones basadas en estos análisis.

Objetivos Específicos:

- Implementar un sistema de inicio de sesión seguro que almacene las credenciales de los usuarios en un archivo binario.
- Permitir la filtración y visualización de la temperatura máxima y mínima dentro de un rango de tiempo especificado.
- Calcular el promedio de temperatura durante un mes determinado.
- Determinar la condición meteorológica que presenta el mayor y menor porcentaje de humedad en un mes específico.
- Evaluar y calcular las probabilidades de incendio basándose en la temperatura, velocidad del viento y la condición meteorológica.
- Evaluar y calcular las probabilidades de tormenta basándose en la presión atmosférica, humedad y la condición meteorológica.
- Permitir agregar más datos meteorológicos a la base de datos.

Descripción del Sistema

El sistema desarrollado está diseñado para satisfacer la necesidad del IDEAM de crear una plataforma integral de análisis meteorológico para la ciudad de Bogotá. Este sistema integra varias funcionalidades clave para analizar y presentar datos climáticos, permitiendo a los usuarios llevar a cabo tareas específicas mediante un menú intuitivo. Se debe recalcar que el programa se hizo en **Replit** y **Visual Code** y se ejecutará en **Visual Code** .

Las principales funcionalidades del sistema son:

1. **Sistema de Inicio de Sesión**

- Implementa un mecanismo de autenticación que utiliza un archivo binario (.bin) para almacenar y verificar los nombres de usuario y contraseñas.
- Los usuarios deben autenticarse para acceder al menú principal y a las funcionalidades del sistema.

2. **Filtrado y Visualización de Temperaturas**

- **Temperatura Máxima:** Permite a los usuarios especificar un rango de tiempo y filtrar los datos para mostrar la temperatura máxima registrada en ese periodo.
- **Temperatura Mínima:** Permite a los usuarios especificar un rango de tiempo y filtrar los datos para mostrar la temperatura mínima registrada en ese periodo.

3. **Cálculo de Promedios**

- **Promedio de Temperatura:** Calcula la temperatura promedio durante un mes determinado, basado en los datos disponibles.
- **Promedio de Humedad por Condición Meteorológica:** Dada una condición meteorológica por el usuario se define en qué fecha se presenta el mayor y menor porcentaje de humedad en un mes específico.

4. Evaluación de Probabilidades

- **Probabilidad de Incendio:** Calcula la probabilidad de incendio considerando la temperatura, velocidad del viento y condiciones meteorológicas específicas.
- **Probabilidad de Tormenta:** Calcula la probabilidad de tormenta basándose en la presión atmosférica, humedad y condiciones meteorológicas específicas.

5. Adición de Datos Meteorológicos

- Permite a los usuarios agregar nuevos datos meteorológicos al archivo de datos existente.
- Los datos ingresados son validados y almacenados en un archivo de texto (.txt), asegurando que sigan el formato adecuado para futuros análisis.

6. Presentación de Resultados

Los resultados de los cálculos y análisis se presentan en un archivo de texto (.txt) para facilitar el acceso y la revisión por parte de los usuarios.

Bibliotecas y Funciones Utilizadas

1. **iostream:**

- Se incluye para operaciones básicas de entrada y salida de datos en la consola.
- Proporciona el objeto estándar cin para la entrada y cout para la salida.
- Permite leer datos desde el teclado y escribir datos en la pantalla.
- Funciones utilizadas de esta biblioteca:
  - **cin:** Extrae datos de la entrada estándar (teclado).
  - **cout:** Inserta datos en la salida estándar (pantalla).
  - **getline():** Lee una línea completa de texto desde la entrada estándar.
  - **ignore():** Ignora caracteres específicos de la entrada estándar.

2. **string:**

- En el código, la biblioteca string se utiliza para manipular cadenas de texto, especialmente al leer y procesar los datos meteorológicos del archivo de texto.
  - **at():** Accede a un caracter específico dentro de una cadena de texto (para procesar datos).
  - **substr():** Obtiene una subcadena de una cadena existente.
  - **compare():** Compara dos cadenas de texto (para validar fechas).

- **stoi():** Convierte una cadena de texto en un número entero (para procesar datos).
- **stof():** Convierte una cadena de texto en un número de punto flotante (para procesar datos).

3. **fstream:**

- En el código, la biblioteca fstream se utiliza principalmente para leer los datos meteorológicos del archivo de texto y escribir los datos de los usuarios en un archivo binario.

- **ifstream::open():** Abre el archivo de datos meteorológicos en modo lectura.
- **ifstream::getline():** Lee una línea completa de texto del archivo de datos meteorológicos.
- **ofstream::open():** Abre el archivo de usuarios en modo escritura.
- **ofstream::write():** Escribe los datos de un usuario en el archivo de usuarios.
- **Ofstream:: read:** Se utiliza para hacer lectura de archivos binarios y se utiliza para leer el archivo de usuarios**.**
- **Reinterpret cas():** Es utilizado para convertir un puntero a objetivo (Usuarios o int) a un puntero de tipo **const char** para poder escribir los datos en el archivo de usuarios en escritura binaria.
- **Ofstream newfile:** Indica que el archivo se abrirá en modo binario, lo que significa que se pueden leer y escribir los datos del usuario en modo bunario

4. **iomanip:**

- En el código, la biblioteca iomanip se utiliza principalmente para formatear la salida de datos en la consola, especialmente al mostrar tablas de datos meteorológicos.

    - **setw():** Se utiliza para establecer el ancho de las columnas en las tablas de datos meteorológicos.

    - **fixed():** Se utiliza para mostrar números flotantes con notación de punto fijo.

5. **sstream:**

- La biblioteca sstream no se utiliza de forma explícita en el código. Sin embargo, se utiliza de manera ímplicita en algunas funciones para convertir cadenas de texto a valores numéricos o para manipular datos de texto.

- Funciones de sstream que se usan implícitamente:

    - **getline(ss, str):** Lee una línea completa de texto del flujo de datos.

    - **ss >> variable:** Extrae un valor del flujo de datos y lo almacena en x variable.

    - **ss.str():** Obtiene una cadena de texto a partir del flujo de datos.

Structs

En el código como tal se hace uso de dos structs, los cuales tienen diferentes funciones.

1. **struct *Usuario*** {

string correo;

string contrasena;

};

Este primer struct se creó para que el usuario puede acceder al sistema como tal y realizar las operaciones que deba. Si es un nuevo usuario se registre y se almacena en el binario, si ya está registrado simplemente accede al sistema ingresando su correo y contraseña, esto ya que en primer lugar el correo electrónico es único y no se repite, es decir dos personas no pueden tener el mismo correo y la contraseña es simplemente para que acceda al programa, como medida de seguridad para el programa.

2. **struct *DatosMeteorologicos*** {

int año;

int mes;

int dia;

float temperatura;

int humedad;

int velocidadViento;

int presionAtmosferica;

string condicionMeteorologica;

};

Ya este segundo struct fue creado para que acceder a cada valor de la base de datos *"datos_metereologicos.txt"* que presenta cada uno de estos valores para así mismo acceder a ellos y poder utilizarlos para diversas funciones que se utilizarán más adelante.

Funciones creadas

1. **int menuLogin():** Esta función se encarga de declarar el cout del menú de login lo primero que hace es declarar una variable de tipo entero llamada opción que se usara para almacenar la opción elegida por el usuario en el menu el **do {** indica un bucle **do-while** que asegura que el bloque de código dentro de **do** se ejecute al menos una vez antes de evaluar la condición del **while.** Los **couts** de inicio de sesión, registrar usuario y salir son las opciones que le aparecerán al usuario La última opción de **"Ingrese su opción"** imprime el mensaje en la consola para indicar al usuario que debe introducir una opción **cin >> opción;** lee la entrada del usuario desde la consola y lo almacena en la variable opción.

   El bucle **do_ while** continuara ejecutándose hasta que se cumpla la condición en el **while** que generalmente se coloca al final del bucle ya en el contexto completo de la función , después de estas líneas, se evaluara la opción ingresada y se ejecutara la lógica correspondiente para cada caso (como iniciar sesión, registra un usuario, mostrar los usuarios registrados o salir del programa

2. **void registrarUsuario():** Esta funcion tiene como objetivo registra un nuevo usuario en el sistema asegurándose de que no se exceda al limite de los usuarios permitidas y que no se dupliquen nombres de usuario además guarda la formación de los usuarios en el archivo binario.

3. **bool iniciarSesion():** La función ***mostrarUsuarios*** imprime en la consola la lista de usuarios registrados. Si no hay usuarios registrados, muestra un mensaje indicando que no hay usuarios. En caso contrario, recorre el arreglo de usuarios e imprime cada nombre de usuario. Ahora la subfunción ***iniciarSesion*** permite a un usuario iniciar sesión en el sistema. Primero, solicita al usuario que ingrese su nombre de usuario y contraseña. Luego, verifica si el nombre de usuario y la contraseña coinciden con algún usuario registrado en el sistema. Si la coincidencia es exitosa, muestra un mensaje de inicio de sesión exitoso y devuelve ***true***. En caso contrario, muestra un mensaje de error y devuelve ***false***.

4. **void salidaTxt():** La función ***salidaTxt*** recibe como parámetro un objeto Usuario. Abre el archivo ***"salida.txt"*** en modo de escritura al final ***(ios::app)***. Si la apertura del archivo tiene éxito, escribe una cabecera indicando ***"RESULTADOS DEL ANÁLISIS METEOROLÓGICO"***, seguido por información sobre el usuario que realiza el reporte. En caso de que no pueda abrir el archivo, emite un mensaje de error.

5. **int leerDatosMeteorologicos():** Esta función se encarga de abrir el archivo .txt de datos meteorológicos, leer su contenido línea por línea (ignorando la primera línea es el encabezado), y almacenar la

información en un arreglo de estructuras ***DatosMeteorologicos***. Luego, inicializa las fechas mínima y máxima con valores extremos para facilitar la comparación posterior. En un bucle, procesa cada línea, extrayendo la fecha, temperatura, humedad, velocidad del viento y presión atmosférica. Si la fecha es válida, se separa en año, mes y día, actualizando las fechas mínima y máxima según corresponda. La condición meteorológica se lee por separado y se limpia de espacios en blanco iniciales. Cada conjunto de datos se almacena en el arreglo, incrementando el contador de datos. Finalmente, cierra el archivo y retorna 0 si la operación es exitosa, o 1 si hubo un error al abrir el archivo.

6. **bool fechaValida():** La función verifica si una fecha está dentro de un rango especificado. Compara el año, mes y día de la fecha con los límites mínimo y máximo dados. Si la fecha cae fuera de estos límites, devuelve false; de lo contrario, devuelve true. Esta función se enfoca en el blindaje del código.

7. **bool fechaExiste():** La función ***fechaExiste*** verifica si una fecha específica (año, mes y día) ya existe en el arreglo de datos meteorológicos. Recorre el arreglo y, si encuentra una coincidencia exacta de año, mes y día, devuelve true. Si no encuentra ninguna coincidencia después de recorrer todo el arreglo, devuelve false. Igual que la anterior va de la mano en el tema de manejo de errores.

8. **void encontrarRangos():** ***encontrarRangos*** solicita al usuario dos fechas, asegurándose de que estén dentro de un rango especificado y existan en los datos meteorológicos disponibles. Primero,

pide al usuario que ingrese dos fechas (en el formato AAAA MM DD). Luego, utiliza las funciones ***fechaValida*** y ***fechaExiste*** para verificar que ambas fechas estén dentro del rango válido y existan en la base de datos. Además, se asegura de que la primera fecha sea anterior o igual a la segunda. Si las fechas ingresadas no cumplen con estos criterios, solicita al usuario que ingrese nuevamente las fechas, mostrando un mensaje de error con el rango de datos disponibles. Este proceso se repite hasta que se ingresen fechas válidas y existentes.

9. **bool fechaEnRango():**

La función ***fechaEnRango*****,** similar a las anteriores que buscan blindar lo más posible el código verifica si una fecha específica (año, mes y día) está dentro de un rango de fechas determinado (definido por una fecha de inicio y una fecha de fin). La función retorna false si el año de la fecha es menor que el año de inicio o mayor que el año de fin. También retorna false si el año de la fecha es igual al año de inicio y el mes es menor que el mes de inicio, o si el año y el mes son iguales pero el día es menor que el día de inicio. Igualmente, retorna false si el año de la fecha es igual al año de fin y el mes es mayor que el mes de fin, o si el año y el mes son iguales pero el día es mayor que el día de fin. Si ninguna de estas condiciones se cumple, la función retorna true, indicando que la fecha está dentro del rango especificado.

10. **void imprimirDatosMeteorologicos():** Esta se encarga de mostrar en la consola los datos meteorológicos almacenados en un arreglo de estructuras ***DatosMeteorologicos***. Itera sobre cada elemento del arreglo y

muestra cada uno de los atributos de la estructura, como el año, mes, día, temperatura, humedad, velocidad del viento, presión atmosférica y condición meteorológica. Utiliza la manipulación de formato de salida para alinear los datos de manera adecuada, como agregar ceros a la izquierda en el caso de los meses y días menores a 10, y establecer un ancho específico para cada atributo usando ***setw***. Además, utiliza ***setfill*** para asegurarse de que los espacios en blanco estén rellenos con espacios en lugar de ceros. Finalmente, cada conjunto de datos se imprime en una nueva línea en la consola. Cabe recalcar que esta no se llama como tal dentro del código, pero si se requieren ver todos los datos del ***.txt*** es solo llamarla en la función principal.

11. **void encontrarMaximaTemperatura():** Esta función busca la temperatura máxima dentro de un rango de fechas específico en un conjunto de datos meteorológicos. Recibe como parámetros el arreglo de datos meteorológicos, el número total de datos, así como las fechas de inicio y fin del rango a considerar. Utiliza la función ***encontrarRangos*** para validar y establecer los rangos de fechas válidos. Luego, itera sobre los datos meteorológicos y verifica si cada dato está dentro del rango especificado. Durante la iteración, compara las temperaturas de los datos dentro del rango para encontrar la temperatura máxima. Al finalizar la búsqueda, muestra la temperatura máxima encontrada junto con la fecha asociada, si se encontró al menos un dato dentro del rango en ***salida.txt***.

12. **void encontrarMinimaTemperatura():** Similar a ***encontrarMaximaTemperatura***, esta función busca la temperatura mínima

dentro de un rango de fechas específico en un conjunto de datos meteorológicos. Recibe los mismos parámetros que la función anterior y sigue una estructura y lógica de funcionamiento idénticas. La diferencia radica en que busca la temperatura mínima en lugar de la máxima. Utiliza la función ***encontrarRangos*** para establecer los rangos de fechas válidos y luego itera sobre los datos meteorológicos, comparando las temperaturas para encontrar la mínima dentro del rango especificado. Al finalizar, muestra la temperatura mínima encontrada junto con la fecha asociada, si se encontró al menos un dato dentro del rango en el ***salida.txt***.

13. **void calcularPromedioTemperaturaMes():** Esta función solicita al usuario que ingrese un mes y un año en el formato "MM AAAA". Luego, inicializa las variables ***totalTemperatura*** y ***cantidadMediciones*** para acumular la suma de las temperaturas y contar el número de mediciones en ese mes y año. La función recorre todos los registros de datos meteorológicos y, si el año y el mes del registro coinciden con los ingresados por el usuario, suma la temperatura del registro a ***totalTemperatura*** y aumenta ***cantidadMediciones***. Si se encontraron mediciones para el mes y año especificados, calcula el promedio dividiendo ***totalTemperatura*** por ***cantidadMediciones***, y escribe el promedio de temperatura encontrado en el archivo "salida.txt" con un formato de dos decimales o informa al usuario que el mes y año no se encuentran en la base de datos.

14. **void calcularMayorPromedioHumedad():** La función ***calcularMayorPromedioHumedad*** solicita al usuario una condición

meteorológica (como "Lluvioso", "Soleado" o "Nublado") y una fecha en formato "MM AAAA". Luego, recorre el arreglo de datos meteorológicos buscando registros que coincidan con la condición meteorológica y la fecha especificada. Si encuentra coincidencias, suma la humedad de estos registros y cuenta cuántos registros coinciden. Si no se encuentran coincidencias, se escribe en el archivo ***"salida.txt"*** que no hay datos para la condición meteorológica, mes y año especificados. Si se encuentran registros coincidentes, calcula el promedio de humedad dividiendo la suma total de la humedad por el número de mediciones encontradas. Luego, compara este promedio con el mayor promedio encontrado hasta el momento y actualiza ***mayorPromedio*** si el nuevo promedio es mayor. Finalmente, escribe el mayor promedio de humedad encontrado en el archivo "salida.txt" con un formato de dos decimales.

15. **void calcularMenorPromedioHumedad():** La función ***calcularMenorPromedioHumedad*** similar a la anterior función, solicita al usuario una condición meteorológica (como "Lluvioso", "Soleado" o "Nublado") y una fecha en formato "MM AAAA". Luego, recorre el arreglo de datos meteorológicos buscando registros que coincidan con la condición meteorológica y la fecha especificada. Si encuentra coincidencias, suma la humedad de estos registros y cuenta cuántos registros coinciden. Si no se encuentran coincidencias, escribe en el archivo "salida.txt" que no hay datos para la condición meteorológica, mes y año especificados. Si se encuentran registros coincidentes, calcula el promedio de humedad dividiendo la suma total de la humedad por el número de mediciones encontradas. Luego,

compara este promedio con el menor promedio encontrado hasta el momento y actualiza ***menorPromedio*** si el nuevo promedio es menor. Finalmente, escribe el menor promedio de humedad encontrado en el archivo ***"salida.txt"*** con un formato de dos decimales.

16. **void mostrarDatosMeteorologicosPorRango():** Esta recibe un rango de fechas y busca los datos meteorológicos dentro de ese rango en el arreglo proporcionado. Primero, establece los límites del rango de fechas utilizando la función ***encontrarRangos***. Luego, abre el archivo ***"salida.txt"*** en modo de escritura y añade al final. Después, escribe una etiqueta indicando que los datos meteorológicos en el rango de fechas especificado se mostrarán a continuación. A continuación, escribe una línea de encabezado con los nombres de las variables meteorológicas. Posteriormente, recorre el arreglo de datos meteorológicos y, para cada dato, verifica si la fecha está dentro del rango especificado. Si es así, escribe la fecha y los datos correspondientes en el archivo ***"salida.txt"***. Finalmente, cierra el archivo.

17. **void probabilidadIncendio():** La función calcula la probabilidad de incendio para una fecha específica. Primero, solicita al usuario la fecha. Luego, busca la información meteorológica correspondiente. Según ciertas condiciones de temperatura y velocidad del viento, determina la probabilidad de incendio. Finalmente, informa sobre la probabilidad basada en las condiciones encontradas o indica si no hay suficiente información para determinarla.

18. **void probabilidadTormenta():** La función ***probabilidadTormenta*** calcula la probabilidad de tormenta o lluvia para una fecha específica. Primero, solicita al usuario la fecha. Luego, busca la información meteorológica correspondiente. Basándose en la presión atmosférica, la humedad y la condición meteorológica registrada, determina la probabilidad de tormenta o lluvia. Finalmente, informa sobre la probabilidad encontrada o indica si no hay suficiente información para determinarla.

19. **void agregarDatosMeteorologicos():** Permitimos al usuario ingresar datos meteorológicos para una fecha específica. Primero, verifica si se ha alcanzado el límite máximo de datos permitidos. Luego, solicita al usuario ingresar la fecha y verifica si ya existe una entrada con esa fecha. Si existe, se actualizan los datos existentes; de lo contrario, se agrega una nueva entrada al final del arreglo. Posteriormente, escribe todos los datos en un archivo de salida, incluyendo la fecha, temperatura, humedad, velocidad del viento, presión atmosférica y condición meteorológica. Finalmente, informa al usuario sobre el éxito del proceso y actualiza el archivo de salida con los nuevos datos agregados. Si hay algún error al abrir el archivo para escritura, se muestra un mensaje de error correspondiente.

20. **int menuDatosMeteorologicos():** La función ***menuDatosMeteorologicos*** muestra un menú numerado que ofrece diversas opciones relacionadas con los datos meteorológicos, como mostrar datos por rango de fechas, encontrar temperaturas máxima y mínima en un intervalo de fechas, calcular promedios de temperatura y humedad, así como

determinar probabilidades de incendio y tormenta. Después de mostrar las opciones, solicita al usuario que elija una y devuelve la opción seleccionada.

21. **bool continuarPrograma():** Se le solicita al usuario que ingrese si desea continuar operando mediante la entrada de ***'s'*** para sí y ***'n'*** para no. Utiliza un bucle infinito para asegurarse de que la respuesta sea válida. Si el usuario ingresa ***'s'***, la función devuelve true, lo que indica que el programa debe continuar. Si el usuario ingresa ***'n'***, devuelve false y muestra un mensaje de despedida. Si la respuesta no es válida, solicita al usuario que responda nuevamente.

22. **int main():** El ***main*** comienza con la carga de usuarios desde un archivo binario al inicio del programa. Luego, muestra un menú principal con opciones para iniciar sesión, registrar un nuevo usuario, mostrar usuarios registrados y salir del programa. Si se elige iniciar sesión y la autenticación es exitosa, se muestra un submenú con opciones relacionadas con el análisis meteorológico. Este submenú permite al usuario realizar diversas acciones, como mostrar datos meteorológicos por rango de fechas, encontrar temperaturas máximas y mínimas, calcular promedios de temperatura y humedad, y estimar la probabilidad de incendio o tormenta, entre otras.

El programa continúa ejecutándose hasta que se seleccione la opción de salida del menú principal. Durante la ejecución, se va solicitando al usuario que ingrese las opciones correspondientes para realizar las diferentes operaciones.

Conclusión

El proyecto de análisis meteorológico es una aplicación que ofrece una variedad de funciones para el procesamiento y análisis de datos meteorológicos. El programa ofrece una amplia gama de herramientas para explorar y comprender las condiciones atmosféricas, desde la autenticación de usuarios hasta la generación de informes meteorológicos detallados.

El proyecto demuestra la capacidad de manejar datos complejos de manera eficiente a través de la implementación de funciones para cargar y guardar datos, así como para realizar cálculos y estimaciones sobre los mismos. Además, las opciones de control de flujo y registro de usuarios lo hacen fácil y amigable con el usuario.