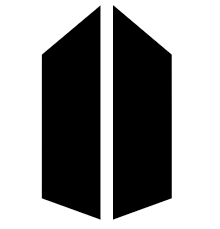


Tercer parcial

**METODO JACOBI**

******

**ALUMNOS:** José Alberto Guzmán Delgado

**MATERIA**: PROGRAMACION AVANZADA

**NR:** 22110346

**GRADO Y GRUPO:** 3F

A22110346@ceti.mx

**Manual de desarrollador**

**Introducción**

El Manual de Desarrollador para JACOBI.cpp proporciona información detallada sobre el desarrollo y la estructura del proyecto. Este manual está diseñado para ayudar a los desarrolladores a comprender el código, las funcionalidades y los requisitos necesarios para trabajar en el proyecto.

**Requisitos del sistema**

* Sistema operativo: Windows, macOS o Linux
* Compilador C++ compatible con C++11.
* Bibliotecas estándar de C++.
* Entorno de desarrollo integrado (IDE) compatible, como Visual Studio Code, Xcode o Code::Blocks.

**Dependencias**

El proyecto JACOBI.cpp no tiene dependencias externas adicionales más allá de las bibliotecas estándar de C++.

**Arquitectura y Diseño**

El proyecto JACOBI.cpp utiliza una estructura modular y sigue un enfoque orientado a objetos. A continuación, se describe la arquitectura y el diseño principales:

Ecuación es una clase que representa una ecuación en el sistema.

Sistemas Ecuaciones es una clase que contiene un vector de objetos Ecuación y proporciona métodos para ingresar y resolver el sistema de ecuaciones.

UI es una clase que maneja la interfaz de usuario y muestra información al usuario.

**Guía de instalación y configuración**

A continuación, se detallan los pasos para instalar y configurar el proyecto JACOBI.cpp:

* Clona el repositorio de JACOBI.cpp desde GitHub: [enlace al repositorio].
* Abre el proyecto en tu entorno de desarrollo preferido.
* Configura el compilador C++ compatible con C++11 en tu entorno de desarrollo.
* Compila el proyecto para generar el ejecutable.

**Guía de desarrollo**

**Estructura del código fuente**

El código fuente de JACOBI.cpp sigu- JACOBI.cpp

- main.cpp

- Ecuacion.h

- Ecuacion.cpp

- SistemaEcuaciones.h

- SistemaEcuaciones.cpp

- UI.h

- UI.cppe la siguiente estructura de directorios:

* main.cpp: Contiene la función main que inicia la ejecución del programa.
* Ecuacion.h y Ecuacion.cpp: Definen y proporcionan la implementación de la clase Ecuacion.
* SistemaEcuaciones.h y SistemaEcuaciones.cpp: Definen y proporcionan la implementación de la clase SistemaEcuaciones.
* UI.h y UI.cpp: Definen y proporcionan la implementación de la clase UI para la interfaz de usuario.

**Convenciones de codificación**

* Utiliza un estilo de codificación claro y legible.
* Utiliza nombres de variables y funciones descriptivos.
* Incluye comentarios para explicar el propósito y el funcionamiento de secciones de código complejas.
* Sigue las convenciones de estilo y formato de C++ para mantener la coherencia en el código.

**Patrones de diseño**

El proyecto JACOBI.cpp no utiliza patrones de diseño específicos. Sin embargo, sigue principios de programación orientada a objetos y modularidad para garantizar un código limpio y mantenible.

**Flujos de trabajo**

El proyecto JACOBI.cpp sigue un flujo de trabajo básico:

* Inicia el programa y muestra un menú de opciones al usuario.
* El usuario ingresa los coeficientes de las ecuaciones.
* El programa resuelve el sistema de ecuaciones utilizando el método de Jacobi.
* El resultado se muestra al usuario.

**Funcionalidades y APIs**

El proyecto JACOBI.cpp incluye las siguientes funcionalidades y APIs:

* Clase Ecuacion:
* Métodos para obtener los coeficientes y el resultado de una ecuación.
* Métodos para despejar las incógnitas x, y y z.
* Clase SistemaEcuaciones:
* Métodos para ingresar las ecuaciones del sistema.
* Método para resolver el sistema de ecuaciones utilizando el método de Jacobi.
* Clase UI:
* Métodos para mostrar mensajes y recibir entrada del usuario.

**Código**

algunas partes del código y su funcionamiento en el proyecto:

|  |
| --- |
| class Ecuacion { |
| private: |
| double coeficiente\_x; |
| double coeficiente\_y; |
| double coeficiente\_z; |
| double resultado; |
|  |
| public: |
| Ecuacion(double coef\_x, double coef\_y, double coef\_z, double res) |
| : coeficiente\_x(coef\_x), coeficiente\_y(coef\_y), coeficiente\_z(coef\_z), resultado(res) {} |
|  |
| // Métodos para obtener los coeficientes y el resultado de una ecuación |
| double obtenerCoeficienteX() const { return coeficiente\_x; } |
| double obtenerCoeficienteY() const { return coeficiente\_y; } |
| double obtenerCoeficienteZ() const { return coeficiente\_z; } |
| double obtenerResultado() const { return resultado; } |
|  |
| // Métodos para despejar las incógnitas x, y, y z |
| double despejarX(double y, double z) const { |
| return (resultado - coeficiente\_y \* y - coeficiente\_z \* z) / coeficiente\_x; |
| } |
|  |
| double despejarY(double x, double z) const { |
| return (resultado - coeficiente\_x \* x - coeficiente\_z \* z) / coeficiente\_y; |
| } |
|  |
| double despejarZ(double x, double y) const { |
| return (resultado - coeficiente\_x \* x - coeficiente\_y \* y) / coeficiente\_z; |
| } |
| }; |

La clase Ecuacion representa una ecuación en el sistema. Cada objeto de esta clase contiene los coeficientes (coeficiente\_x, coeficiente\_y, coeficiente\_z) y el resultado (resultado) de una ecuación.

* Los métodos obtenerCoeficienteX(), obtenerCoeficienteY(), obtenerCoeficienteZ() y obtenerResultado() devuelven los valores de los coeficientes y el resultado de una ecuación.
* Los métodos despejarX(), despejarY() y despejarZ() permiten despejar las incógnitas x, y y z respectivamente, utilizando los coeficientes y el resultado de la ecuación.

|  |
| --- |
| class SistemaEcuaciones { |
| private: |
| vector<Ecuacion> sistema; |
| public: |
| // Método para ingresar las ecuaciones del sistema |
| void ingresarEcuaciones() { |
| cout << "Ingrese los coeficientes de las ecuaciones:" << endl; |
| for (int i = 0; i < 3; i++) { |
| cout << "Ecuacion " << i + 1 << ":" << endl; |
| // ... |
| Ecuacion ecuacion(coef\_x, coef\_y, coef\_z, res); |
| sistema.push\_back(ecuacion); |
| } |
| } |
|  |
| // Método para resolver el sistema de ecuaciones utilizando el método de Jacobi |
| void resolverSistema() { |
| sort(sistema.begin(), sistema.end(), compararCoeficientesX); |
| double x = 0.0; |
| double y = 0.0; |
| double z = 0.0; |
| vector<double> valoresX; |
| vector<double> valoresY; |
| vector<double> valoresZ; |
|  |
| do { |
| x = sistema[0].despejarX(y, z); |
| y = sistema[1].despejarY(x, z); |
| z = sistema[2].despejarZ(x, y); |
| valoresX.push\_back(x); |
| valoresY.push\_back(y); |
| valoresZ.push\_back(z); |
| } while (!verificarRepeticion(valoresX, valoresY, valoresZ, x, y, z)); |
| // ... |
| } |
| // ... |
| }; |

La clase SistemaEcuaciones representa el sistema de ecuaciones. Tiene un vector sistema que almacena las ecuaciones ingresadas por el usuario.

* El método ingresarEcuaciones() permite al usuario ingresar los coeficientes de las ecuaciones y crea objetos de la clase Ecuacion con esos coeficientes. Luego, agrega esos objetos al vector sistema.
* El método resolverSistema() resuelve el sistema de ecuaciones utilizando el método de Jacobi. Ordena las ecuaciones en el vector sistema según los coeficientes de x. Luego, utiliza los métodos despejarX(), despejarY() y despejarZ() de la clase Ecuacion para obtener los valores de x, y y z respectivamente, y los guarda en los vectores valoresX, valoresY y valoresZ para verificar si se repiten.

Estos son solo fragmentos del código que demuestran algunas funcionalidades clave. El código completo incluye más partes, como las funciones de comparación y verificación, y la interacción con la interfaz de usuario.

**Pruebas y depuración**

* Pruebas unitarias: Se recomienda escribir pruebas unitarias para cada funcionalidad importante del proyecto para garantizar su correcto funcionamiento. Puedes utilizar frameworks de pruebas como Google Test o CppUnit.
* Depuración: En caso de errores o comportamientos inesperados, se recomienda utilizar herramientas de depuración disponibles en tu entorno de desarrollo para identificar y solucionar problemas en el código.