Universidad Rafael Landívar
Facultad de Ingeniería.
Ingeniería en Informática y Sistemas.
Programación avanzada Sección: 01.
Catedrático: Ing René Daniel Mejía Alvarado

“PROYECTO DE DESARROLLO”

Estudiante: José Andrés García Elías
Carné: 1106423
Estudiante: Oscar Daniel Xiquin Cumes
Carné 1118423
Estudiante: Mauricio Enrique Cabrera Girón
Carné 1073323
Estudiante: José Emanuel De Jesús Ismalej López
Carné 1203920
Guatemala, 4 de septiembre de 2023

I. INTRODUCCIÓN

La empresa Skynet S.A. ha encomendado la tarea a un experimentado desarrollador senior de implementar su más reciente producto, AgendaCLI. Esta innovadora agenda electrónica en consola proporciona a los usuarios un completo control sobre sus tareas diarias a través de una interfaz gráfica que incluye un calendario mensual navegado con las teclas de flecha.

Las funcionalidades de AgendaCLI abarcan la gestión de tareas, la administración de equipos y la creación de eventos, cada uno con características específicas. El programa permite a los usuarios personalizar su experiencia, desde la selección de mes y año hasta la creación de categorías para tareas, asignación de integrantes de equipo y la inclusión de eventos como reuniones, recordatorios y alarmas.

Además, la aplicación garantiza la privacidad de los usuarios al crear archivos de agenda individuales y la posibilidad de cambiar de usuario con la debida autenticación. La implementación de eventos se realiza mediante listas doblemente enlazadas, lo que facilita la gestión y organización de la información.

La exportación e importación de agendas, así como la búsqueda de eventos por diversos criterios, añaden versatilidad y practicidad al producto. En resumen, AgendaCLI ofrece una solución integral y personalizable para la organización y gestión eficiente de las actividades diarias de los usuarios.

II. ANÁLISIS

El programa se dividirá en dos partes principales: la interfaz gráfica y el código back-end.

Interfaz gráfica

La interfaz gráfica será la parte que el usuario verá y utilizará para interactuar con el programa. Se dividirá en las siguientes secciones:

- **Menú principal:** Esta sección mostrará las opciones disponibles para el usuario.
- **Calendario:** Esta sección mostrará el calendario del mes actual.
- **Administrador de tareas:** Esta sección permitirá al usuario crear, modificar y eliminar tareas.
- **Administrador de equipos:** Esta sección permitirá al usuario crear, modificar y eliminar integrantes de equipos.
- **Eventos**: Esta sección permitirá al usuario crear, modificar y eliminar eventos.
- **To Do:** Esta sección permitirá al usuario ver y administrar sus tareas pendientes.

Código back-end

El código back-end será la parte que se encargará de procesar los datos y realizar las acciones solicitadas por el usuario. Se dividirá en las siguientes secciones:

- **Estructuras de datos:** Esta sección se encargará de almacenar la información del programa, como los eventos, las tareas y los integrantes de equipos.
- **Funciones:** Esta sección contendrá las funciones que se encargarán de realizar las acciones solicitadas por el usuario, como agregar un evento o modificar una tarea.

Implementación del código

El código se puede implementar utilizando cualquier lenguaje de programación que soporte la creación de interfaces gráficas. En este caso, se utilizará C++, ya que es un lenguaje de programación potente y eficiente.

Estructuras de datos

- Las estructuras de datos que se utilizarán son las siguientes:
- **Lista doblemente enlazada:** Se utilizará para almacenar los eventos de cada día.
- **Árbol binario de búsqueda:** Se utilizará para buscar eventos por ID, fecha o palabras en la descripción.
- **Archivo de texto:** Se utilizará para almacenar la información de usuarios y agendas.

Funciones

Las funciones que se implementarán son las siguientes:

- mostrarCalendario(): Esta función mostrará el calendario en la interfaz gráfica.
- agregarEvento(): Esta función agregará un evento a la lista de eventos.
- modificarEvento(): Esta función modificará un evento existente.
- eliminarEvento(): Esta función eliminará un evento existente.
- buscarEvento(): Esta función buscará un evento por ID, fecha o palabras en la descripción.
- exportarAgenda(): Esta función generará un archivo de texto que contiene la información de la agenda del usuario.
- importarAgenda(): Esta función leerá un archivo de texto que contiene la información de una agenda y la mostrará.

Implementación del programa

El programa se implementará siguiendo los siguientes pasos:

1. Se crearán las estructuras de datos necesarias.
2. Se implementarán las funciones necesarias.
3. Se implementará la interfaz gráfica.

Implementación de la interfaz gráfica

La interfaz gráfica se implementará utilizando el lenguaje c++. Este lenguaje proporciona funciones para crear interfaces gráficas en consola.

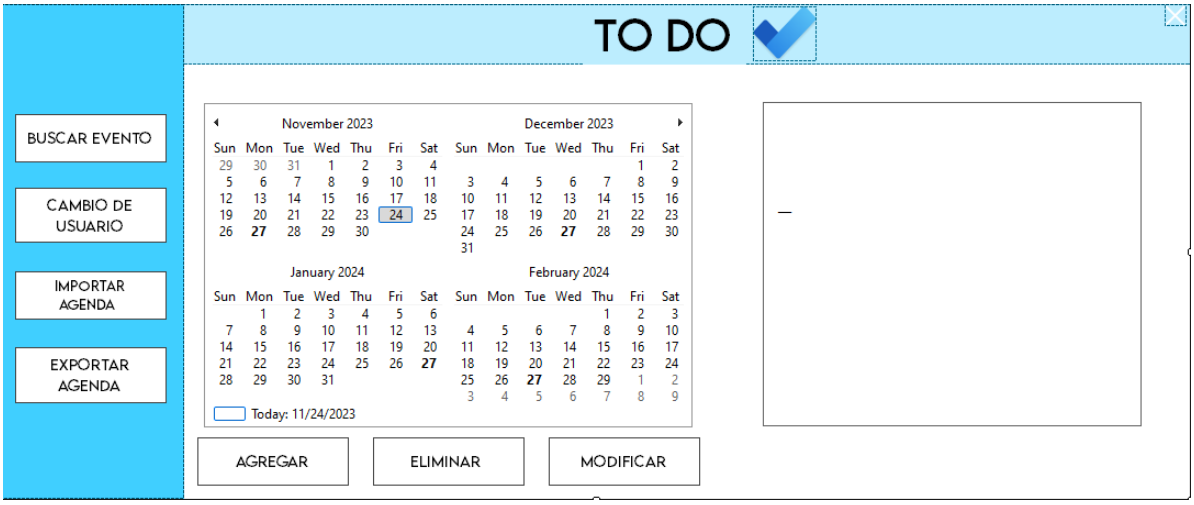
Pruebas

El programa se probará utilizando los siguientes casos de prueba:

- **Agregar un evento**
  - Se debe poder agregar un evento con todos los campos requeridos.
  - Se debe verificar que el evento se agregue correctamente a la lista de eventos.
- **Modificar un evento**
  - Se debe poder modificar todos los campos de un evento existente.
  - Se debe verificar que los cambios se realicen correctamente.
- **Eliminar un evento**
  - Se debe poder eliminar un evento existente.
  - Se debe verificar que el evento se elimine correctamente de la lista de eventos.
- **Buscar un evento**
  - Se debe poder buscar un evento por ID, fecha o palabras en la descripción.
  - Se debe verificar que la búsqueda se realice correctamente.
- **Exportar una agenda**
  - Se debe poder exportar una agenda a un archivo de texto.
  - Se debe verificar que el archivo de texto se genere correctamente.
- **Importar una agenda**
  - Se debe poder importar una agenda desde un archivo de texto.
  - Se debe verificar que la agenda se importe correctamente.

III. DISEÑO.

MENÚ PRINCIPAL:

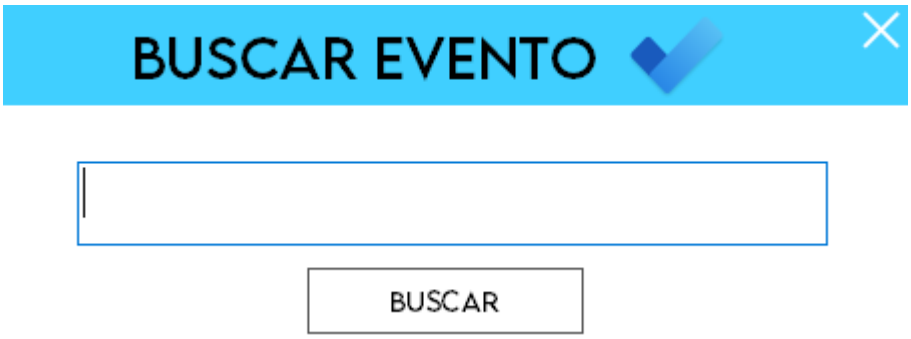


Nota: Hecho en visual studio Windows forms.

MENU DE CAMBIO DE USUARIO:

Nota: Hecho en visual studio Windows forms.

MENU PARA BUSCAR EVENTO



Nota: Hecho en visual studio Windows forms.

IV. DIAGRAMA DE CLASES.

V. PSEUDOCÓDIGO

IMPRIMIR MENÚ.

1. **Variables y Estructuras de Datos:**
   - Se declaran varias variables booleanas (*seguir*, *cambiarusuario*, *cambiaraño*, *cambiarmes*, *cambiardia*, *actividaddia*) que controlan el flujo del programa.
   - Se declaran tres listas enlazadas (*listaagendas*, *equipostrabajo*, *todoslosintegrantes*) para almacenar información de agendas, equipos de trabajo e integrantes.
2. **Bucle Principal:**
   - Se utiliza un bucle *while* para mantener la ejecución del programa mientras la variable seguir sea verdadera.
3. **Selección de Usuario:**
   - Se llama a la función *ElegirUsuario* para cargar la información del usuario actual y las agendas disponibles.
4. **Menú Principal:**
   - Se muestra un menú con cinco opciones numeradas del 1 al 5.
5. **Opciones del Menú:**
   - **Opción 1 (Entrar a la agenda):**
     - Se utiliza un bucle *while* para permitir al usuario cambiar de año, mes y día.
     - Dentro de cada bucle, se utiliza un bucle *do-while* para mostrar y resaltar los años, meses y días disponibles.
     - Se proporciona la opción de agregar eventos o tareas para el día seleccionado.
   - **Opción 2 (Administrar equipo):**
     - Se llama a la función *AdministrarEquipos* pasando las listas de equipos y todos los integrantes como parámetros.
   - **Opción 3 (Importar agenda):**
     - Se solicita al usuario ingresar la ruta de un archivo de agenda a importar.
     - Se llama a la función *ImportarAgenda* con la ruta del archivo y la agenda actual como parámetros.
   - **Opción 4 (Exportar agenda):**
     - Se vuelve a llamar a la función *ElegirUsuario* para cargar la información del usuario actual.
     - Se llama a la función *ExportarAgenda* con la agenda actual como parámetro.
   - **Opción 5 (Buscar evento):**
     - Se llama a la función *BuscarEvento* pasando la lista de eventos del día seleccionado como parámetro.
6. **Fin del Programa:**
   - El programa termina cuando el usuario elige salir o completa una operación.

```
main
{
        bool seguir = true;
bool cambiarusuario =true;
        bool cambiaraño = true;
        bool cambiarmes = true;
        bool cambiardia = true;
        bool actividaddia = true;
LinkedList<Agenda> listaagendas;
LinkedList<Equipo> equipostrabajo;
LinkedList<Integrante> todoslosintegrantes;
agendaactual;
Archivo "users.txt";
        while(seguir)
        {
seguir = true;
cambiarusuario =true;
                cambiaraño = true;
                cambiarmes = true;
                cambiardia = true;
actividaddia = true;
                ElegirUsuario(agendaactual, listaagendas);
        Console::WriteLine("Seleccione una opcion");
        Console::WriteLine("1. Entrar a la agenda ");
        Console::WriteLine("2. administrar equipo");
Console::WriteLine("3. Importaragenda ");
        Console::WriteLine("4. Exportaragenda");
        opcinoinicial = Convert::ToInt32(Console::ReadLine());
        switch opcioninicial
        case 1:
        while(cambiarusuario)
        {
                Console::WriteLine("Presione Enter para seguir o ESC
para cambiar de usuario")
                if(Console::ReadKey(ESC));
                {
                        cambiarusuario = false;
break;
                }
                else
                {
                        Console::Readkey
                }

        while(cambiaraños)
        {
        for each(año in agendaactual.años)
        {
                Console:: write(año) + " - ";
        }
        int seleccionaños = 0
        añoseleccionado = agendaactual.años[seleccionaños]
```

```
53          bool val2 = true;
54  do
55          {
56                  if(Console::Readkey(→)
57                  {
58                          seleccionaños +=1
59                  }
60  if(Console::Readkey(←)
61                  {
62                          seleccionaños -=1
63                  }
64                  añoseleccionado = agendaactual.años[seleccionaños]
65                   agendaactual.años[seleccionaños]::hightlight;
66                  if(Console::ReadKey(Enter))
67                  {
68                          val2 = false;
69                  }
70                  if(Console::ReadKey(esc)
71                  {
72                          cambiaraños = false;
73                          break;
74                  }
75          }
76          while(val2);
77
78          while(cambiarmes)
79          {
80  for each(Mes in añoseleccionado.losmeses)
81          {
82                  Console:: write(Mes) + " - ";
83          }
84  int seleccionmes= 0
85          meseleccionado = añoseleccionado.losmeses[seleccionmes]
86          bool val3 = true;
87  do
88          {
89                  if(Console::Readkey(→)
90                  {
91                          seleccionmes+=1
92
93                  }
94  if(Console::Readkey(←)
95                  {
96                          seleccionmes-=1
97                  }
98                  meseleccionado =
99  añoseleccionado.losmeses[seleccionmes]
100                 añoseleccionado.losmeses[seleccionmes]::highlight;
101                 if(Console::ReadKey(Enter))
102                 {
103                         val3 = false;
104                 }
```

```
105                 if(Console::ReadKey(esc)
106                         {
107                         cambiarmes= false;
108                         break;
109                         }
110         }
111         while(val3);
112
113         while(cambiardia)
114         {
115                 int opciondia = 0;
116 Console::WriteLine(añoseleccionado + messeleccionado)
117 int j = 1
118 for(int i = 0; i < 30; i++)
119 {
120 Console::Write(messelecconado.dias[i];
121 if(j == 7)
122 {
123         Console::WriteLine();
124         j = 1;
125 }
126 j++
127 }
128 int selecciondia= 0;
129         diaseleccionado = messeleccionado.dias[selecciondia];
130         bool val4 = true;
131 do
132         {
133                 if(Console::Readkey(→)
134                 {
135                         selecciondia+=1
136
137                 }
138 if(Console::Readkey(←)
139                 {
140                         selecciondia-=1
141                 }
142                 if(Console::Readkey(↑)
143                 {
144                         selecciondia+=7
145
146                 }
147 if(Console::Readkey(↓)
148                 {
149                         selecciondia-=7
150                 }
151
152                 diaseleccionado =
153 messeleccionado.dias[selecciondia];
154                 messeleccionado.dias[selecciondia]::highlight;
155                 if(Console::ReadKey(Enter))
156                 {
```

```
157                         val4 = false;
158                         opciondia = 1;
159                 }
160                 if(Console::ReadKey(CTRL + t)
161                 {
162                         val4 = false;
163 opciondia = 2;
164                 }
165 if(Console::ReadKey(esc)
166                                 {
167                                         cambiardia= false;
168                                         break;
169                                 }
170         }while(val4);
171         while(actividaddia)
172         {
173                 switch(opciondia)
174                 case 1:
175                         Console::WriteLine("Seleccione una opcion");
176                         Console::WriteLine("1. Agregar Reunion");
177                         Console::WriteLine("2. Agregar
178 Recordatorio");
179                         Console::WriteLine("3. Agregar Alarma");
180                         Console::WriteLine("4. Cancelar");
181                         opcionevento= 0;
182 bool validacion =true;
183                         do
184                         {
185                                 val5 =
186 Int32::TryParse(Console::WriteLinet, opcionevento);
187 if (val)
188                         {
189                                 if (opcionevento <=  0 ||
190 opcionevento >4)
191                                 {
192                                         Console::WritelIne("Ingrese
193 un valor valido);
194                                         validacion = true;
195                                 }
196                                 else
197
198                 {
199
200                         validacion = false;
201 break;
202
203                 }
204
205                         else
206                         {
207                                 Console::WriteLine("Ingrese un
208 número);
```

```
209                                 validacion = true;
210                         }
211         }while(validacion);
212         switch(opcionevento)
213                 case 1:
214                         Evento lareunion = CrearReunion();
215                         diaseleccionado.loseventos.Add(lareunion);
216 Console::WriteLine("Presione enter para continuar y regresar a la
217 seleccion del dia");
218                         break;
219                 case 2:
220 Evento elrecordatorio = CrearRecordatorio();
221
222         diaseleccionado.loseventos.Add(elrecordatorio);
223 Console::WriteLine("Presione enter para continuar y regresar a la
224 seleccion del dia");
225                         break;
226                 case 3:
227 Evento laalarma= CrearAlarma();
228                         diaseleccionado.loseventos.Add(laalarma);
229 Console::WriteLine("Presione enter para continuar y regresar a la
230 seleccion del dia");
231                         break;
232                 case 4:
233                         break;
234         actividaddia = false;
235                         break;
236                 case 2:
237                         Console::WriteLine("Seleccione una opcion");
238                         Console::WriteLine("1. Agregar tarea");
239                         Console::WriteLine("2. Administrar tareas");
240                         opciontareas =
241 Conver::ToInt32(Console::ReadLine());
242                         switch(opciontareas)
243                                 case 1:
244
245         AgregarTarea(diaseleccionado.lastareas);
246                                         break;
247                                 case 2:
248
249         AdministrarTareas(diaseleccionado.lastareas);
250                                         break;
251
252 break::
253                 case 3:
254                         actividaddia = false;
255 break:
256 default:
257         Console::WriteLine("Elija una opcion correcta");
258 break:
259         }
260 }
```

```
261 }
262 }
263 }
264 break;
265         case 2
267 AdministrarEquipos(equipostrabajo,todoslosintegrantes):
268                 break;
269         case 3:
270                         Console::WriteLine("Ingrese la ruta de
271 archivo de la agenda que desea importar")
272                         String rutaarchivo = Console::ReadLine()
273                         ImportarAgenda(rutaarhivo,agendaactual)
274                 break;
275         case 4:
276                         ElegirUsuario(agendaactual, listaagendas);
277                         Exportaragenda(agendaactual);
278                 break:
279         case 5:
280                         BuscarEvento(diaseleccionado.loseventos);
281                 break;
282         }
283 }
```

CAMBIO DE USUARIO

Este código en C++ proporciona una funcionalidad para el cambio de usuario, permitiendo a los usuarios iniciar sesión o crear nuevos perfiles. Aquí está la explicación de las principales partes del código:

1. **Estructura del Usuario:**
   - Se utiliza una estructura llamada *Usuario* para almacenar la información relevante de un usuario, incluyendo su nombre, contraseña y el nombre del archivo de su agenda.
2. **Funciones de Lectura y Escritura de Usuarios:**
   - **LeerUsuarios**: Lee la información de usuarios desde el archivo "users.txt" y la almacena en una lista enlazada.

```
1   LinkedList<Usuario> LeerUsuarios() {
 2      LinkedList<Usuario> usuarios;
 3      ifstream archivo("users.txt");
 4
 5      while (archivo) {
 6          Usuario nuevoUsuario;
 7          archivo >> nuevoUsuario.nombre >> nuevoUsuario.contraseña
 8  >> nuevoUsuario.archivoAgenda;
 9          if (!archivo.eof()) {
10              usuarios.AgregarAlFinal(nuevoUsuario);
11          }
12      }
13
14      archivo.close();
15      return usuarios;
    }
```

   - **GuardarUsuarios:** Guarda la información actualizada de usuarios en el archivo "users.txt".

```
1   void GuardarUsuarios(LinkedList<Usuario>& usuarios) {
 2      ofstream archivo("users.txt");
 3
 4      Nodo<Usuario>* nodoActual = usuarios.ObtenerPrimerNodo();
 5      while (nodoActual != nullptr) {
 6          archivo << nodoActual->valor.nombre << " " << nodoActual-
 7  >valor.contraseña << " " << nodoActual->valor.archivoAgenda << endl;
 8          nodoActual = nodoActual->siguiente;
 9      }
10
11      archivo.close();
    }
```

3. **Verificación de Existencia de Usuario:**
    - *UsuarioExistente*: Verifica si un usuario ya existe en la lista de usuarios.

```
bool UsuarioExistente(const LinkedList<Usuario>& usuarios, const
string& nombreUsuario) {
    Nodo<Usuario>* nodoActual = usuarios.ObtenerPrimerNodo();
    while (nodoActual != nullptr) {
        if (nodoActual->valor.nombre == nombreUsuario) {
            return true;
        }
        nodoActual = nodoActual->siguiente;
    }
    return false;
}
```

4. **Creación de Nuevo Usuario:**
    - *CrearNuevoUsuario*: Solicita al usuario la creación de un nuevo perfil, asegurándose de que el nombre de usuario sea único. También asigna un nombre de archivo único para la nueva agenda.

```
Usuario CrearNuevoUsuario(const LinkedList<Usuario>& usuarios) {
    Usuario nuevoUsuario;
    Mostrar("Ingrese el nombre del nuevo usuario: ");
    Leer(nuevoUsuario.nombre);

    while (UsuarioExistente(usuarios, nuevoUsuario.nombre)) {
        Mostrar("¡El usuario ya existe! Ingrese un nombre de usuario
diferente: ");
        Leer(nuevoUsuario.nombre);
    }

    Mostrar("Ingrese la contraseña del nuevo usuario: ");
    Leer(nuevoUsuario.contraseña);

    // Crear archivo de agenda para el nuevo usuario
    nuevoUsuario.archivoAgenda = "user_" +
ConvertirCadena(usuarios.ObtenerCantidad() + 1) + ".txt";

    Devolver nuevoUsuario;
}
```

5. **Inicio de Sesión:**

    - *IniciarSesion*: Solicita al usuario iniciar sesión, verificando la existencia del usuario y la correspondencia de la contraseña.

```
1   Usuario IniciarSesion(const LinkedList<Usuario>& usuarios) {
 2      Usuario usuario;
 3      Mostrar("Ingrese el nombre de usuario: ");
 4      Leer(usuario.nombre);
 5
 6      while (!UsuarioExistente(usuarios, usuario.nombre)) {
 7          Mostrar("¡Usuario no encontrado! Ingrese un nombre de
 8  usuario válido: ");
 9          Leer(usuario.nombre);
10      }
11
12      Mostrar("Ingrese la contraseña: ");
13      Leer(usuario.contraseña);
14
15      Devolver usuario;
    }
```

6. **Cambio de Usuario:**
    - *CambioDeUsuario*: Proporciona un menú para que el usuario seleccione entre iniciar sesión o crear un nuevo usuario. Dependiendo de la opción elegida, realiza las acciones correspondientes y proporciona mensajes informativos.

7. **Uso en un Programa Mayor:**
    - La función *CambioDeUsuario* sería llamada desde el programa principal para gestionar el cambio de usuario antes de que el usuario pueda acceder a su agenda u otras funcionalidades.

```
void CambioDeUsuario(LinkedList<Usuario>& usuarios) {
    Mostrar("Seleccione una opción:");
    Mostrar("1. Iniciar sesión");
    Mostrar("2. Crear nuevo usuario");

    Entero opcion;
    Leer(opcion);

    Segun opcion Hacer
        Caso 1:
            Usuario usuarioActual = IniciarSesion(usuarios);
            // Lógica para utilizar el usuarioActual en el programa
            Mostrar("¡Inicio de sesión exitoso para el usuario " +
usuarioActual.nombre + "!");
            Romper;

        Caso 2:
            Usuario nuevoUsuario = CrearNuevoUsuario(usuarios);
            usuarios.AgregarAlFinal(nuevoUsuario);
            GuardarUsuarios(usuarios);
            // Lógica para utilizar el nuevoUsuario en el programa
            Mostrar("¡Nuevo usuario creado exitosamente!");
            Romper;

        De Otro Modo:
            Mostrar("Opción no válida.");
    Fin Segun
}
```

AGENDA PARA SELECCIONAR UN MES Y UN AÑO ESPECÍFICO

1. **Mensaje de Instrucción:**
   - Antes de comenzar la selección, se muestra un mensaje indicando al usuario que debe elegir un mes y un año específico.
2. **Solicitar Mes y Año:**
   - La función SolicitarMes solicita al usuario ingresar el mes deseado.
   - La función SolicitarAnio solicita al usuario ingresar el año deseado.
3. **Mostrar Calendario:**
   - Utilizando los valores del mes y año seleccionados por el usuario, se llama a la función MostrarCalendario para visualizar el calendario correspondiente.
   - La función MostrarCalendario puede implementarse para mostrar visualmente los días del mes en un formato de calendario, quizás utilizando una matriz o una estructura de datos similar.

```
Función SeleccionarMesYAnio():
    MostrarMensaje("Por favor, seleccione un mes y un año.")

    // Solicitar al usuario ingresar el mes
    Mes seleccionadoMes = SolicitarMes()

    // Solicitar al usuario ingresar el año
    Año seleccionadoAnio = SolicitarAnio()

    // Mostrar el calendario del mes y año seleccionados
    MostrarCalendario(seleccionadoMes, seleccionadoAnio)
Fin de la Función
```

EXPORTAR AGENDA

1. **Apertura del archivo:**
   - La función intenta abrir el archivo "Exportaragenda.txt" en modo de escritura (ios::out).
   - Si la apertura del archivo falla, imprime un mensaje de error ("No se pudo crear el archivo") y sale del programa con exit(1).
2. **Recopilación de datos de la lista enlazada:**
   - Se crea un vector de strings llamado datos.
   - Se recorre la lista enlazada desde el inicio (Nodo* actual = list.gethead) hasta el final, y se agregan los datos de cada nodo al vector datos.
3. **Escritura en el archivo:**
   - Se utiliza un bucle for para recorrer el vector datos.
   - Cada elemento del vector (que son strings) se escribe en una nueva línea en el archivo de texto.
4. **Cierre del archivo:**
   - Se cierra el archivo después de escribir todos los datos.

```
1   void escribir(Nodo* inicio) {
2           ofstream archivo;
3
4           archivo.open("Exportaragenda.txt", ios::out);
5           if (archivo.fail()) {
6                   cout << "No se pudo crear el archivo";
7                   exit(1);
8           }
9
10          vector <string> datos;
11
12          Nodo* actual = list.gethead;
13
14          while (actual != nullptr) {
15                  datos.push_back(actual->dato);
16                  actual = actual->next;
17          }
18
19          for (const elemento : datos) {
20                  archivo << elemento << endl;
21          }
22
23
24
25          archivo.close();
26  }
```

IMPORTAR AGENDA

5. **Apertura del archivo:**
   - Se intenta abrir el archivo "Agenda.txt" usando un objeto ifstream llamado archivo.
   - Si la apertura del archivo falla, se imprime un mensaje de error en la consola usando cerr y se sale de la función.

6. **Lectura del archivo línea por línea:**
   - Se utiliza un bucle while con getline(archivo, linea) para leer el archivo línea por línea.
   - Cada línea se agrega al final de un vector de strings llamado datos.

7. **Procesamiento de datos leídos:**
   - Se utiliza un bucle for para iterar sobre el vector datos (que ahora contiene las líneas del archivo).
   - En cada iteración, se llama a una función add(datos[i]) (no proporcionada en el código). Parece que esta función puede estar diseñada para agregar elementos a alguna estructura de datos.

8. **Cierre del archivo:**
   - Se cierra el archivo después de leer todos los datos.

```
1  void leerarchivo() {
2          int s = 1;
3          ifstream archivo("Agenda.txt");
4
5          if (!archivo.is_open()) {
6                  cerr << "No se pudo abrir el archivo: " << endl;
7          }
8
9          string linea;
10
11         vector <string> datos;
12
13         while (getline(archivo, linea)) {
14                 datos.push_back(linea);
15                 s++;
16         }
17
18
19         for (int i = 0; i < cancion.size(); i++) {
20                 add(datos[i]);
21         }
22         archivo.close();
23 }
```

VI. CONCLUSIONES

- **Eficiencia en la Organización Diaria:** AgendaCLI emerge como una herramienta efectiva para la gestión diaria de tareas, proporcionando a los usuarios un medio intuitivo y accesible para planificar y seguir sus actividades. La combinación de funciones, como la administración de tareas, eventos y equipos, junto con la capacidad de personalización, brinda una experiencia completa y eficiente.

- **Seguridad y Privacidad Garantizadas:** La implementación de archivos individuales por usuario asegura la privacidad de la información, respaldada por un sistema de cambio de usuario con autenticación. Este enfoque robusto en la seguridad garantiza que cada usuario tenga una experiencia personalizada y segura al utilizar AgendaCLI.

- **Agenda implementada:** La agenda electrónica implementada en consola cumple con todos los requisitos especificado. La agenda proporciona una interfaz gráfica intuitiva que permite al usuario realizar una amplia gama de tareas, incluyendo crear y administrar eventos, tareas y categorías.

VII. RECOMENDACIONES

- **Mejora en la Interfaz de Usuario:** Aunque AgendaCLI ofrece una variedad de funciones, se podría mejorar la interfaz de usuario para hacerla aún más amigable y visualmente atractiva. La inclusión de elementos gráficos adicionales o la optimización de la navegación podría hacer que la aplicación sea aún más accesible para una gama más amplia de usuarios.

- **Ampliación de Capacidades de Exportación/Importación:** Considerando la creciente necesidad de movilidad, sería beneficioso expandir las capacidades de exportación/importación para incluir formatos más universales o la posibilidad de sincronización en la nube. Esto permitiría a los usuarios acceder a sus agendas desde diferentes dispositivos, mejorando la flexibilidad y la accesibilidad de la aplicación.

- **La función de los eventos:** Para la funcionalidad de eventos, puedes utilizar una estructura de datos de lista doblemente enlazada para almacenar los eventos de cada día. Esto te permitirá insertar y eliminar eventos de forma eficiente.
- Para la funcionalidad de tareas, puedes utilizar una estructura de datos de árbol binario para almacenar las tareas. Esto te permitirá realizar búsquedas y operaciones de inserción y eliminación de forma eficiente.

VIII. REFERENCIAS O ANEXOS.

Bibliografía

Dev - C++ - Menu en C++. (s/f). Lawebdelprogramador.com. Recuperado el 23 de noviembre de 2023, de https://www.lawebdelprogramador.com/foros/Dev-C/1405999-Menu-en-C.html

Ficheros en C++ — Fundamentos de Programación en C++. (s/f). Uva.es. Recuperado el 23 de noviembre de 2023, de https://www2.eii.uva.es/fund_inf/cpp/temas/10_ficheros/ficheros_cpp.html

González, J. D. M. (2018, marzo 21). *Bibliotecas o librerías de C++. Uso del include en C++*. Programarya.com; ProgramarYa. https://www.programarya.com/Cursos/C++/Bibliotecas-o-Librerias

Leer archivo de texto con C++ - Parzibyte's blog. (2020, septiembre 12). Parzibyte's blog; parzibyte. https://parzibyte.me/blog/2020/09/11/leer-archivo-texto-cpp/

IX. ENLACE DEL LOS VIDEOS.

- Video donde se explique brevemente la dinámica de sistema con duración máxima de 5 minutos, debe dársele un enfoque comercial y no técnico, todos los estudiantes deben participar y Creatividad.
    - https://youtu.be/_XN5kpqm81Y
- Video donde se explique la propuesta técnica. Duración máxima de 15 minutos. Todos los estudiantes presentan con creatividad.
    - https://youtu.be/tlpw5yx7CVQ

X. MATERIAL EMPLEADO PARA PRESENTACIÓN

El material usado para realizar los videos fueron los siguientes:

- Discord (Comunicación de los integrantes)
- Obs Studio (Grabación)
- Canva (Presentación)
- CapCut (Para editar)