GREENCHECK: SISTEMA DE RIEGO AUTOMATICO

MANUAL DE USO DEL PROGRAMA DE MEDICIÓN DE HUMEDAD

ANDRES FELIPE CERVERA SANTOS

PAULA MARIYEY MURCIA SANCHEZ

VANESSA GARCIA ARIZA

UNIVERSIDAD CATÓLICA DE COLOMBIA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

BOGOTÁ D.C

2025

1. DESCRIPCION GENERAL DE PROGRAMA

Este programa permite registrar y administrar mediciones de humedad en diferentes ubicaciones.

Utiliza una estructura de multilista donde.

- Cada ubicación representa un nodo principal de la lista.
- Cada ubicación contiene una sublista de Mediciones asociada a diferentes fechas.

El programa permite registrar datos, mostrarlos, editarlos y realizar cálculos simples de estadística, como determinar la ubicación mas seca y los días con menor nivel de humedad.

2. ESTRUCTURA GENERAL DEL PROGRAMA

El programa usa dos estructuras principales:

2.1 Estructura Medición

```
3  struct Medicion {
4      string fecha;
5      int humedad;
6      Medicion* sig;
7  };
```

Representa un nodo de la sublista de medición.

- **Fecha:** Fecha de la decisión en formato dd/mm/aaaa.
- **Humedad:** Porcentaje de humedad registrada.
- **Sig:** Puntero a la siguiente medición.

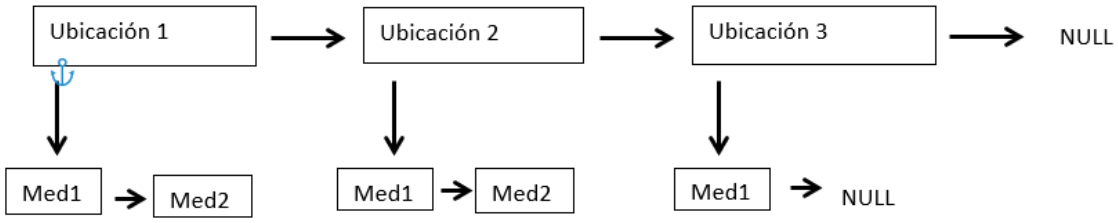
2.2 Estructura Ubicación

```
9  struct Ubicacion {
10     string nombre;
11     Medicion* listaMediciones;
12     Ubicacion* sig;
13 };
```

Representa un nodo principal de la multilista.

- **Nombre:** Nombre de ubicación.
- **listaMedicion:** Puntero a la sublista de mediciones asociadas.
- **Sig:** Puntero a la siguiente ubicación.

REPRESENTACION GRAFICA



3. FUNCIONALIDADES PRINCIPALES

3.1 Creación de ubicaciones

El programa pregunta cuántas ubicaciones se van a registrar y, para cada una:

- El nombre de la ubicación.
- La fecha inicial(dd/mm/aaaa)
- La cantidad de mediciones.

Luego genera automáticamente las mediciones con fechas consecutivas y valores de humedad calculados mediante una formula simple.

3.2 Crear Ubicación

Qué hace

- Crea un nuevo nodo Ubicación que contendrá:
- *Nombre*(string)
- *listaMediciones* inicializada en NULL.
- *Sig*(siguiente) inicializado a NULL.

Devuelve el puntero al nuevo nodo

Como lo hace

```
17  Ubicacion* crearUbicacion(string nombre) {
18      Ubicacion* nueva = new Ubicacion {nombre};
19      nueva->listaMediciones = NULL;
20      nueva->sig = NULL;
21      return nueva;
22  }
```

3.3 Crear Medicion

Qué hace

- Crea una nueva medición (*Medición*) con los valores *fecha* y *humedad*, y *sig=NULL.*

- Devuelve el puntero a la nueva medición

Como lo hace

```
Medicion* crearMedicion(string fecha, int humedad) {
    Medicion* nueva = new Medicion {fecha, humedad};
    nueva->sig = NULL;
    return nueva;
}
```

3.4 Agregar Medición

Qué hace

- Inserta una nueva Medición al final de la sublista listaMedicion de la Ubicación apuntada por ubic.

Cómo lo hace

- Crea *nueva = creaMedicion(fecha, humedad).*
- Si, *ubic->listaMediciones == NULL* entonces la sublista estaba vacía y pone *ubic->listaMediciones = nueva.*
- Si no, recorre la sublista con *aux* hasta el ultimo (*aux->sig == NULL*) y enlaza el nuevo: *aux->sig = nueva.*

```
void agregarMedicion(Ubicacion* ubic, string fecha, int humedad) {
    Medicion* nueva = crearMedicion(fecha, humedad);
    if (ubic->listaMediciones == NULL) {
        ubic->listaMediciones = nueva;
    } else {
        Medicion* aux = ubic->listaMediciones;
        while (aux->sig != NULL) {
            aux = aux->sig;
        }
        aux->sig = nueva;
    }
}
```

3.5 Siguiente Fecha

Qué hace

- Desde una cadena *fecha* con el formato *dd/mm/aaaa* calcula la fecha que resulta de sumar (*días*) días y devuelve la nueva fecha en el mismo formato.

Cómo lo hace

- Extrae *días*, *mes, año* tomando caracteres fijos: *fecha[0..1]* para día, *fecha[3..4]* para mes, *fecha[6..9]* para año.
- Suma *días* a *día.*
- sí *día>30* hace *día = día-30*.
- Construye la cadena de retorno añadiendo ceros cuando corresponde.

```
string siguienteFecha(string fecha, int dias) {
    int dia, mes, anio;
    char barra;
    dia = (fecha[0] - '0') * 10 + (fecha[1] - '0');
    mes = (fecha[3] - '0') * 10 + (fecha[4] - '0');
    anio = (fecha[6] - '0') * 1000 + (fecha[7] - '0') * 100 + (fecha[8] - '0') * 10 + (fecha[9] - '0');
    dia = dia + dias;
    if (dia > 30) dia = dia - 30;
    string nueva = "";
    if (dia < 10) nueva += "0";
    nueva += to_string(dia) + "/";
    if (mes < 10) nueva += "0";
    nueva += to_string(mes) + "/";
    nueva += to_string(anio);
    return nueva;
}
```

3.6 Mostar Datos

Qué hace

- Recorrer la lista principal de *ubicación* y por cada una recorre su sublista *listaMediciones*, imprimiendo *nombre, fecha* y *humedad*.

Cómo lo hace

- Crea el puntero *auxU* y un segundo puntero *auxM*
- El bucle *while (auxU != NULL)* se usa para recorrer todas las ubicaciones y *while (auxU != NULL)* recorre la lista de mediciones de la ubicación actual, mientras *auxM* no sea *NULL*.

```
void mostrarDatos(Ubicacion* lista) {
    Ubicacion* auxU = lista;
    while (auxU != NULL) {
        cout << "\nUbicacion: " << auxU->nombre << endl;
        Medicion* auxM = auxU->listaMediciones;
        while (auxM != NULL) {
            cout << "  Fecha: " << auxM->fecha << "  Humedad: " << auxM->humedad << "%" << endl;
            auxM = auxM->sig;
        }
        auxU = auxU->sig;
    }
}
```

3.7 Ubicación Mas Seca

Qué hace

- Recorre cada ubicación, calcula el promedio de humedad de esa ubicación(suma/cantidad), imprime cada promedio y decide cual es la ubicación con menor promedio. Al final imprime la ubicación más seca y su promedio.

Cómo lo hace

- El puntero *auxU* recorre toda la lista de ubicaciones.
- *masSeca* almacenara el nombre de la ubicación con menor promedio de humedad.
- Con el ciclo *while (auxU != NULL)* se repite una vez por cada ubicación registrada. Dentro del while acumula las humedades de dicha ubicación, las suma y promedia.

- Con el *if* se compara el promedio actual *prom* con el menor promedio encontrado hasta ahora(*menorProm*). Si el promedio actual es más bajo se actualizarán: *menorProm* con el nuevo valor, *masSeca* con el nombre de la ubicación actual.
- Pasa a la siguiente ubicación.

```
76  void ubicacionMasSeca(Ubicacion* lista) {
77      Ubicacion* auxU = lista;
78      string masSeca = "";
79      float menorProm;
80
81      while (auxU != NULL) {
82          int suma = 0, cont = 0;
83          Medicion* auxM = auxU->listaMediciones;
84          while (auxM != NULL) {
85              suma += auxM->humedad;
86              cont++;
87              auxM = auxM->sig;
88          }
89          float prom = (float)suma / cont;
90          cout << "Promedio de " << auxU->nombre << ": " << prom << "%" << endl;
91          if (prom < menorProm) {
92              menorProm = prom;
93              masSeca = auxU->nombre;
94          }
95          auxU = auxU->sig;
96      }
97
98      cout << "\nLa ubicacion mas seca es: " << masSeca << " (" << menorProm << "%)\n";
99  }
```

3.8 Días Mas Secos

Qué hace

- Encuentra el valor mínimo global de humedad entre todas las mediciones y luego imprime todas las fechas y ubicaciones que tengan ese valor mínimo.

Cómo lo hace

- Declara *int menor*
- Recorre todas las mediciones y hace: *if (auxM->humedad < menor) menor = auxM->humedad;* , esta condición compara la humedad del nodo actual (*auxM->humedad*) con el valor más pequeño que llevamos registrado hasta el momento (*menor*), cuando la condición se cumple se actualiza la variable *menor* con el nuevo valor más pequeño.
- Luego vuelve a recorrer la multilista e imprime coincidencias donde *auxM->humedad == menor*.

```cpp
void diaMasSeco(Ubicacion* lista) {
    int menor;
    Ubicacion* auxU = lista;


    while (auxU != NULL) {
        Medicion* auxM = auxU->listaMediciones;
        while (auxM != NULL) {
            if (auxM->humedad < menor) menor = auxM->humedad;
            auxM = auxM->sig;
        }
        auxU = auxU->sig;
    }

    cout << "\nDias mas secos (humedad " << menor << "%):\n";
    auxU = lista;
    while (auxU != NULL) {
        Medicion* auxM = auxU->listaMediciones;
        while (auxM != NULL) {
            if (auxM->humedad == menor) {
                cout << auxM->fecha << " en " << auxU->nombre << endl;
            }
            auxM = auxM->sig;
        }
        auxU = auxU->sig;
    }
}
```

3.8 Editar Humedad

Qué hace

- Busca la *Ubicación* por nombre y dentro de ella busca la *Medición* por fecha. Si la encuentra, actualiza humedad con *nuevaHum* y confirma con *humedad actualizada*. Si no, informa que no encontró.

Cómo lo hace

- Recorre ubicación por ubicación (*auxU*) *comprobando auxU->nombre == ubic.* Donde compara el nombre de la ubicación actual con el nombre ingresado.
- Si coincide, recorre *auxU->listaMediciones* buscando *auxM->fecha == fecha*.
- Si la encuentra asigna *auxM->humedad =nuevaHum y return*(sale). Este bucle recorre todas las mediciones de la ubicación actual compara *auxM->fecha* con la *fecha* que usuario pidió modificar y si coincide, actualizara el valor.
- Si termina todo sin encontrarlo imprime mensaje de no encontrado.

```cpp
void editarHumedad(Ubicacion* lista, string ubic, string fecha, int nuevaHum) {
    Ubicacion* auxU = lista;
    while (auxU != NULL) {
        if (auxU->nombre == ubic) {
            Medicion* auxM = auxU->listaMediciones;
            while (auxM != NULL) {
                if (auxM->fecha == fecha) {
                    auxM->humedad = nuevaHum;
                    cout << "Humedad actualizada\n";
                    return;
                }
                auxM = auxM->sig;
            }
        }
        auxU = auxU->sig;
    }
    cout << "No se encontro la ubicacion o fecha\n";
}
```

3.9 Main

Flujo general

- Pide al usuario *nUbic* (número de ubicaciones a registrar)
- Pide nombre
- Crea *ubicación* nueva=crearUbicacion*(nombre) y la inserta al principio de la lista:

nueva->sig = lista;
lista = nueva;

- Pide fecha inicia y medidas de humedad.
- Muestra el menú en un: *do*
- Opciones:
    1. mostrarDatos
    2. ubicacionMasSeca
    3. diaMasSeco
    4. editarHumedad
- hace que cuando eligas la opción 4 y terminas de editar, el ciclo termina.

```
152 int main() {
153     Ubicacion* lista = NULL;
154     int nUbic;
155     cout << "Cuantas ubicaciones desea registrar ";
156     cin >> nUbic;
157 
158     for (int i = 0; i < nUbic; i++) {
159         string nombre, fecha;
160         int nMedidas;
161         cout << "\nNombre de la ubicacion: ";
162         cin.ignore();
163         getline(cin, nombre);
164         Ubicacion* nueva = crearUbicacion(nombre);
165         nueva->sig = lista;
166         lista = nueva;
167 
168         cout << "Fecha de inicio (dd/mm/aaaa): ";
169         getline(cin, fecha);
170         cout << "Cantidad de medidas: ";
171         cin >> nMedidas;
172 
173         for (int j = 0; j < nMedidas; j++) {
174             int hum = (j * 37 + i * 11) % 100;
175             string fechaNueva = siguienteFecha(fecha, j);
176             agregarMedicion(nueva, fechaNueva, hum);
177         }
178     }
179 
```

```
180     int op;
181     do {
182         cout << "\n--- MENU ---\n";
183         cout << "1. Mostrar datos\n";
184         cout << "2. Ubicacion mas seca\n";
185         cout << "3. Dia mas seco\n";
186         cout << "4. Editar humedad\n";
187         cout << "Opcion: ";
188         cin >> op;
189 
190         if (op == 1) mostrarDatos(lista);
191         else if (op == 2) ubicacionMasSeca(lista);
192         else if (op == 3) diaMasSeco(lista);
193         else if (op == 4) {
194             string u, f;
195             int h;
196             cin.ignore();
197             cout << "Ubicacion: ";
198             getline(cin, u);
199             cout << "Fecha: ";
200             getline(cin, f);
201             cout << "Nueva humedad: ";
202             cin >> h;
203             editarHumedad(lista, u, f, h);
204         }
205     } while (op != 4);
206 
207     cout << "\nFin del programa\n";
208 
209     return 0;
210 }
```