```
#include <iostream>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <windows.h>
using namespace std;
int aux=0;
int auxY=0;
int aux2=0;
int auxY2=0;
void menuABB();
void menuAVL();

struct Persona {

   string nombre;
   string dpi;
   string departamento;
   string municipio;
   string fecha1;
   string fecha2;
   string fecha3;
   string ubicacion;
```

```
};

struct Nodo {
    Persona dato;
    Nodo* izquierdo;
    Nodo* derecho;
    Nodo* padre;
};

Nodo* arbol = NULL;


void MENU_PRINCIPAL() {
    int opcion;

    do {
        cout << "                    __                  " << endl;
        cout << "                .'  '.                " << endl;
        cout << "            _.-'/  |  \\                 " << endl;
        cout << "    ,       _.-\"  ,|  /  0 `-.          " << endl;
        cout << "    |\\\    .-\"      `--\"\"-.__.'=================|" << endl;
        cout << "    \\ '-'`       .___.--._)==================== |   " << endl;
        cout << "     \\\          .'     |                    |   " << endl;
        cout << "      |     /,_.-'       | ===== BIENVENIDO ===== |" << endl;
        cout << "    _/   _.'(            |                    |   " << endl;
        cout << "   /  ,-' \\  \\\         |  1. ABRIR ABB         |   " << endl;
```

```
cout << "   \\  \\    `-'         |  2. ABRIR AVL          |   " << endl;

cout << "    `-'                |  3. PARA SALIR         |   " << endl;

cout << "                       |  Ingrese una opcion:   |" << endl;

cout << "                      '------------------------'" << endl;


cin >> opcion; // Leer la opción ingresada por el usuario


switch (opcion) {
   case 1:
      system("cls");
      menuABB();
      cout << endl;
      break;
   case 2:
      system("cls");
      menuAVL();
      cout << endl;
      break;
   case 3:
      cout << "Saliendo del programa..." << endl;
      break;
   default:
      cout << "Opcion no valida. Intente nuevamente." << endl;
      break;
}
```

```
        cout << endl;

    } while (opcion != 3);
}

 void encriptarCesar(Nodo* nodo) {
    if (nodo == NULL) {
        return;
    }

    // Realizar encriptación César en el nombre del nodo
    for (char& c : nodo->dato.nombre) {
        if (isalpha(c)) {
            if (isupper(c)) {
                c = (c - 'A' + 3) % 26 + 'A'; // Cifrado César para mayúsculas
            } else {
                c = (c - 'a' + 3) % 26 + 'a'; // Cifrado César para minúsculas
            }
        }
    }
     for (char& c : nodo->dato.departamento) {
        if (isalpha(c)) {
            if (isupper(c)) {
                c = (c - 'A' + 3) % 26 + 'A'; // Cifrado César para mayúsculas
            } else {
                c = (c - 'a' + 3) % 26 + 'a'; // Cifrado César para minúsculas
```

```
        }
    }
}
 for (char& c : nodo->dato.municipio) {
    if (isalpha(c)) {
        if (isupper(c)) {
            c = (c - 'A' + 3) % 26 + 'A'; // Cifrado César para mayúsculas
        } else {
            c = (c - 'a' + 3) % 26 + 'a'; // Cifrado César para minúsculas
        }
    }
}


 for (char& c : nodo->dato.ubicacion) {
    if (isalpha(c)) {
        if (isupper(c)) {
            c = (c - 'A' + 3) % 26 + 'A'; // Cifrado César para mayúsculas
        } else {
            c = (c - 'a' + 3) % 26 + 'a'; // Cifrado César para minúsculas
        }
    }
}



// Recorrer el subárbol izquierdo
encriptarCesar(nodo->izquierdo);
```

```
    // Recorrer el subárbol derecho
    encriptarCesar(nodo->derecho);
}



void desencriptarCesar(Nodo* nodo) {
    if (nodo == NULL) {
        return;
    }

    // Realizar desencriptación César en el nombre del nodo
    for (char& c : nodo->dato.nombre) {
        if (isalpha(c)) {
            if (isupper(c)) {
                c = (c - 'A' - 3 + 26) % 26 + 'A'; // Desencriptar César para mayúsculas
            } else {
                c = (c - 'a' - 3 + 26) % 26 + 'a'; // Desencriptar César para minúsculas
            }
        }
    }
     for (char& c : nodo->dato.departamento) {
        if (isalpha(c)) {
            if (isupper(c)) {
                c = (c - 'A' - 3 + 26) % 26 + 'A'; // Desencriptar César para mayúsculas
            } else {
```

```
            c = (c - 'a' - 3 + 26) % 26 + 'a'; // Desencriptar César para minúsculas
        }
    }
}
 for (char& c : nodo->dato.municipio) {
    if (isalpha(c)) {
        if (isupper(c)) {
            c = (c - 'A' - 3 + 26) % 26 + 'A'; // Desencriptar César para mayúsculas
        } else {
            c = (c - 'a' - 3 + 26) % 26 + 'a'; // Desencriptar César para minúsculas
        }
    }
}
 for (char& c : nodo->dato.ubicacion) {
    if (isalpha(c)) {
        if (isupper(c)) {
            c = (c - 'A' - 3 + 26) % 26 + 'A'; // Desencriptar César para mayúsculas
        } else {
            c = (c - 'a' - 3 + 26) % 26 + 'a'; // Desencriptar César para minúsculas
        }
    }
}
```

// Recorrer el subárbol izquierdo

```
    desencriptarCesar(nodo->izquierdo);

    // Recorrer el subárbol derecho
    desencriptarCesar(nodo->derecho);
}



 Nodo* crearNodo(Persona n,Nodo *padre){
    Nodo* nuevo_nodo = new Nodo();

    nuevo_nodo->dato = n;
    nuevo_nodo->derecho = NULL;
    nuevo_nodo->izquierdo = NULL;
    nuevo_nodo->padre = padre;

    return nuevo_nodo;
}

void insertar(Nodo*& nodo, Persona dato, int nivel) {
    if (nodo == NULL) {
        nodo = crearNodo(dato, NULL);
    } else if (dato.dpi < nodo->dato.dpi) {
        insertar(nodo->izquierdo, dato, nivel + 1);
    } else if (dato.dpi > nodo->dato.dpi) {
        insertar(nodo->derecho, dato, nivel + 1);
```

```
    }
}

void leer() {
    int cantidad_personas; // Cantidad de personas a agregar a los árboles
    int contador_personas = 0; // Contador de personas agregadas
    ifstream archivo("vacunas.txt");
    string linea;
    Persona dato;
    cout << "Cuántas personas desea agregar a los árboles: ";
    cin >> cantidad_personas;
    if (archivo.is_open()) {
        while (getline(archivo, linea) && contador_personas < cantidad_personas) {
            dato.dpi = linea.substr(linea.length() - 13);
            dato.nombre = linea.substr(0, linea.length() - 13);
              insertar(arbol, dato, 0); // Insertar en el ABB
            contador_personas++;
        }
        archivo.close();
    } else {
        cout << "No se pudo abrir el archivo" << endl;
    }
}
bool buscarPorDPI(Nodo* nodo, string dpi, Persona& persona_encontrada) {
    if (nodo == NULL) {
        return false;
```

```
    }

    if (dpi < nodo->dato.dpi) {
        return buscarPorDPI(nodo->izquierdo, dpi, persona_encontrada);
    } else if (dpi > nodo->dato.dpi) {
        return buscarPorDPI(nodo->derecho, dpi, persona_encontrada);
    } else {
        persona_encontrada = nodo->dato;
        return true;
    }
}

void modificarPersona(Nodo* nodo, string dpi) {
if (nodo == NULL) {
cout << "La persona no existe en el árbol." << endl;
return;
}

if (dpi < nodo->dato.dpi) {
modificarPersona(nodo->izquierdo, dpi);
} else if (dpi > nodo->dato.dpi) {
modificarPersona(nodo->derecho, dpi);
} else {
// Persona encontrada, solicitar nuevos datos
cout << "Persona encontrada:\n";
cout << "Nombre: " << nodo->dato.nombre << endl;
```

```
cout << "DPI: " << nodo->dato.dpi << endl;

cout << "Ingrese los nuevos datos:" << endl;


cout << "Nombre: ";

cin.ignore();

getline(cin, nodo->dato.nombre);


cout << "Departamento: ";

getline(cin, nodo->dato.departamento);


cout << "Municipio: ";

getline(cin, nodo->dato.municipio);


cout << "Fecha 1: ";

getline(cin, nodo->dato.fecha1);


cout << "Fecha 2: ";

getline(cin, nodo->dato.fecha2);


cout << "Fecha 3: ";

getline(cin, nodo->dato.fecha3);


cout << "Lugar: ";

getline(cin, nodo->dato.ubicacion);


cout << "Los datos de la persona se han modificado correctamente." << endl;
```

```
}


}
void eliminar(Nodo*& arbol, string dpi) {
   if (arbol == NULL) {
      return;
   }

   if (dpi < arbol->dato.dpi) {
      eliminar(arbol->izquierdo, dpi);
   } else if (dpi > arbol->dato.dpi) {
      eliminar(arbol->derecho, dpi);
   } else {
      // Caso 1: Nodo sin hijos
      if (arbol->izquierdo == NULL && arbol->derecho == NULL) {
         delete arbol;
         arbol = NULL;
      }
      // Caso 2: Nodo con un hijo
      else if (arbol->izquierdo == NULL) {
         Nodo* temp = arbol;
         arbol = arbol->derecho;
         delete temp;
      } else if (arbol->derecho == NULL) {
         Nodo* temp = arbol;
         arbol = arbol->izquierdo;
```

```
            delete temp;
        }
        // Caso 3: Nodo con dos hijos
        else {
            Nodo* sucesor = arbol->derecho;
            while (sucesor->izquierdo != NULL) {
                sucesor = sucesor->izquierdo;
            }
            arbol->dato = sucesor->dato;
            eliminar(arbol->derecho, sucesor->dato.dpi);
        }
    }
}
void gotoxy(int x,int y){

HANDLE hcon;
hcon = GetStdHandle(STD_OUTPUT_HANDLE);
COORD dwPos;
dwPos.X = x;
dwPos.Y= y;
SetConsoleCursorPosition(hcon,dwPos);
}



void mostrarArbol(Nodo *&arbol, int auxY){
```

```
string nombre1;
if(arbol == NULL){

return;
}

nombre1=arbol->dato.nombre.substr(0,arbol->dato.nombre.find(' '));
aux += 5;

mostrarArbol(arbol -> izquierdo,auxY+6);
gotoxy(20+aux-auxY , 40+auxY);
cout << arbol -> dato.dpi<<endl;
gotoxy(21+aux-auxY , 41+auxY);
cout<< nombre1<<endl<<endl;



mostrarArbol(arbol -> derecho,auxY+4);

}
void mostrarPreorden(Nodo* nodo) {
if (nodo == NULL) {
return;
}
```

```
// Mostrar el nodo actual
cout << "Nombre: " << nodo->dato.nombre << " DPI: " << nodo->dato.dpi << endl;

// Recorrer el subárbol izquierdo
mostrarPreorden(nodo->izquierdo);

// Recorrer el subárbol derecho
mostrarPreorden(nodo->derecho);
}

void mostrarPosorden(Nodo* nodo) {
if (nodo == NULL) {
return;
}

// Recorrer el subárbol izquierdo
mostrarPosorden(nodo->izquierdo);

// Recorrer el subárbol derecho
mostrarPosorden(nodo->derecho);

// Mostrar el nodo actual
cout << "Nombre: " << nodo->dato.nombre << " DPI: " << nodo->dato.dpi << endl;
}

void mostrarInorden(Nodo* nodo) {
```

```
if (nodo == NULL) {
return;
}

// Recorrer el subárbol izquierdo
mostrarInorden(nodo->izquierdo);

// Mostrar el nodo actual
cout << "Nombre: " << nodo->dato.nombre << " DPI: " << nodo->dato.dpi << endl;

// Recorrer el subárbol derecho
mostrarInorden(nodo->derecho);
}

int altura(Nodo* nodo) {
   if (nodo == NULL) {
      return 0;
   }
   int altura_izq = altura(nodo->izquierdo);
   int altura_der = altura(nodo->derecho);
   return 1 + max(altura_izq, altura_der);
}

bool estaEquilibrado(Nodo* nodo) {
   if (nodo == NULL) {
      return true;
```

```
    }
    int altura_izq = altura(nodo->izquierdo);
    int altura_der = altura(nodo->derecho);
    int diferencia = abs(altura_izq - altura_der);
    if (diferencia <= 1 && estaEquilibrado(nodo->izquierdo) && estaEquilibrado(nodo->derecho)) {
        return true;
    }
    return false;
}

void guardarEnArchivo(Nodo* nodo, ofstream& archivo) {
    if (nodo == NULL) {
        return;
    }

    // Recorrer el subárbol izquierdo
    guardarEnArchivo(nodo->izquierdo, archivo);

    // Encriptar el nombre de la persona antes de escribirlo en el archivo



    // Escribir los datos de la persona en el archivo
    archivo << "nombre: " << nodo->dato.nombre << " dpi: " << nodo->dato.dpi
            << " departamento: " << nodo->dato.departamento << " municipio: " << nodo-
>dato.municipio
            << " fecha1: " << nodo->dato.fecha1 << " fecha2: " << nodo->dato.fecha2
```

```
        << " fecha3: " << nodo->dato.fecha3 << " ubicacion: " << nodo->dato.ubicacion << endl;


   // Recorrer el subárbol derecho
   guardarEnArchivo(nodo->derecho, archivo);
}

void guardarArbolEnArchivo(Nodo* arbol) {
ofstream archivo("ABB.txt");
if (archivo.is_open()) {
guardarEnArchivo(arbol, archivo);
archivo.close();
cout << "El arbol se ha guardado en el archivo correctamente." << endl;
} else {
cout << "No se pudo abrir el archivo." << endl;
}
}

void mostrarNPersonas(Nodo* nodo, int cantidad) {
static int contador = 0;  // Variable estática para llevar la cuenta de las personas mostradas

if (nodo == nullptr || contador >= cantidad) {
return;
}

mostrarNPersonas(nodo->izquierdo, cantidad);
```

```
// Mostrar la información de la persona
cout << "Nombre: " << nodo->dato.nombre << endl;
cout << "DPI: " << nodo->dato.dpi << endl;
// Mostrar otras propiedades de la persona si las hay

contador++;

mostrarNPersonas(nodo->derecho, cantidad);
}




void menuABB() {
   int auxY = 0;
   char confirmacion;
   int opcion;
   string dpi, nombre, respuesta;
   Persona persona_encontrada, dato;

   HANDLE hConsole = GetStdHandle( STD_OUTPUT_HANDLE );
 SetConsoleTextAttribute(hConsole, 5);
   do {
      cout << "===== MENÚ ABB =====" << endl;
      cout << "1. Cuántas personas desea agregar al arbol: "<< endl;
```

```
    cout << "2. Mostrar Arbol ABB: " << endl;
    cout << "3. Mostrar recorrido en Preorden: " << endl;
    cout << "4. Mostrar recorrido en Posorden: " << endl;
    cout << "5. Mostrar recorrido en Inorden: " << endl;
    cout << "6. Buscar persona:"<< endl;
    cout << "7. Modificar o actualizar persona: "<< endl;
    cout << "8. Eliminar persona: "<< endl;
    cout << "9. Agregar persona: "<< endl;
    cout << "10. Guardar árbol en archivo: "<< endl;
    cout << "11. Regresar al menu principal: " << endl;
    cout<<"12. encriptar el arbol"<<endl;
    cout<<"13. desencriptar arbol"<<endl;
    cout << "14. verificar si esta equilibrado: "<< endl;
     cout << "15. salir: "<< endl;
    cout << "Ingrese una opcion: "<< endl;
    cin >> opcion;

    switch (opcion) {
        case 1:
            system("cls");
            leer();
            cout << endl;
            break;
        case 2:
              system("cls");
cout << "Árbol:";
```

```
      mostrarArbol(arbol, auxY);
    cout << endl;
    cout << "Presione Enter para continuar...";
    cin.ignore();
    cin.get();
system("cls");
    break;




          case 3:
                        system("cls");
    cout << "PRE-ORDEN:";
       mostrarPreorden(arbol);
    cout << endl;
    cin.ignore();
    cin.get();
system("cls");
break;


          case 4:
             system("cls");
             cout << "Recorrido en Posorden:" << endl;
             mostrarPosorden(arbol);
             cout << endl;
```

```
            cin.ignore();
   cin.get();
system("cls");
            break;
         case 5:
            system("cls");
            cout << "Recorrido en Inorden:" << endl;
            mostrarInorden(arbol);
            cout << endl;
             cin.ignore();
   cin.get();
system("cls");
            break;

         case 6:
              system("cls");

            cout << "Ingrese el DPI a buscar: ";
            cin >> dpi;
            if (buscarPorDPI(arbol, dpi, persona_encontrada)) {
               cout << "Persona encontrada con exito:\n";
               cout << "Nombre: " << persona_encontrada.nombre << endl;
               cout << "DPI: " << persona_encontrada.dpi << endl;
               cout<< endl;
               cout << "¿Quiere ingresar datos adicionales? s(si)/ n(no): ";
               cin >> respuesta;
```

```
    if (respuesta == "s") {
        cout << "Ingrese Departamento al que pertenece ↓\n";
        cin.ignore();
        getline(cin, persona_encontrada.departamento);

        cout << "Ingrese Municipio al que pertenece ↓\n";
        getline(cin, persona_encontrada.municipio);

        cout << "Fecha Primera Vacuna ↓\n";
        getline(cin, persona_encontrada.fecha1);

        cout << "Fecha Segunda Vacuna ↓\n";
        getline(cin, persona_encontrada.fecha2);

        cout << "Fecha Tercera Vacuna ↓\n";
        getline(cin, persona_encontrada.fecha3);

        cout << "Lugar de Vacunación ↓\n";
        getline(cin, persona_encontrada.ubicacion);
    }
} else {
    cout << "La persona que ingreso no existe :( ";
    system("pause");
}
cout << endl;
```

```
    system("cls");


    case 7:
    system("cls");
    cout << "Ingrese el DPI de la persona a modificar: ";
    cin >> dpi;
    modificarPersona(arbol, dpi);
    cout << endl;
    break;


    case 8:
    system("cls");
    cout << "Ingrese el DPI de la persona a eliminar: ";
    cin >> dpi;
    eliminar(arbol, dpi);
    cout << "Persona eliminada." << endl;
    cout << endl;
    break;
case 9:
    system("cls");
    cout << "Ingrese el nombre de la persona a agregar: ";
    cin.ignore();
    getline(cin, dato.nombre);
    cout << "Ingrese el DPI de la persona a agregar: ";
    cin >> dato.dpi;
    cout << "Padece de alguna enfermedad del corazón? (s/n): ";
```

```
cin >> confirmacion;
if (confirmacion == 'S' || confirmacion == 's') {
   cout << "Saliendo del programa..." << endl;
   break;
} else if (confirmacion == 'N' || confirmacion == 'n') {
   cout << "Ingrese el departamento de la persona a agregar: ";
   cin >> dato.departamento;
   cout << "Ingrese el municipio de la persona a agregar: ";
   cin >> dato.municipio;
   cout << "Ingrese la fecha 1 de la persona a agregar: ";
   cin >> dato.fecha1;
   cout << "Ingrese la fecha 2 de la persona a agregar: ";
   cin >> dato.fecha2;
   cout << "Ingrese la fecha 3 de la persona a agregar: ";
   cin >> dato.fecha3;
   cout << "Ingrese el lugar de la persona a agregar: ";
   cin.ignore();
getline(cin, dato.ubicacion);
   insertar(arbol, dato, 0);
}
cout << endl;
break;
case 10:
system("cls");
guardarArbolEnArchivo(arbol);
cout << endl;
```

```
            break;


        case 11:
            system("cls");
            return;
            case 12:
        system("cls");
        cout << " encriptado exitoso " << endl;
        encriptarCesar(arbol);
        cout << endl;
        break;
        case 13:
        system("cls");
        cout << "desencriptado Exitoso" << endl;
        desencriptarCesar(arbol);
        cout << endl;
        break;
case 14:
    system("cls");
    if (estaEquilibrado(arbol)) {
        cout << "El arbol esta equilibrado." << endl;
    } else {
        cout << "El arbol no esta equilibrado." << endl;
    }
    cout << endl;
    break;
```

```
        case 15:
            system("cls");
            cout << "\n\n\tNos vemos luego.....\n" << endl;
            return;
            break;
        default:
            cout << "Opción no válida. Intente nuevamente." << endl;
            break;
    }

    cout << endl;

  } while (opcion!=15);
}

//-----------------------------------------------------------------------------------------------------------------------
-------

struct NodoAVL {
  string nombre;
  string dpi;
  string departamento;
  string municipio;
  string fecha1;
  string fecha2;
  string fecha3;
```

```
    string ubicacion;
    int altura;
    NodoAVL* izq;
    NodoAVL* der;
    NodoAVL* padre;
};
NodoAVL* arbolavl = NULL;
void encriptarCesarAVL(NodoAVL* nodoavl) {
    if (nodoavl == NULL) {
        return;
    }

    // Realizar encriptación César en el nombre del nodo
    for (char& c : nodoavl->nombre) {
        if (isalpha(c)) {
            if (isupper(c)) {
                c = (c - 'A' + 3) % 26 + 'A'; // Cifrado César para mayúsculas
            } else {
                c = (c - 'a' + 3) % 26 + 'a'; // Cifrado César para minúsculas
            }
        }
    }
     for (char& c : nodoavl->departamento) {
        if (isalpha(c)) {
            if (isupper(c)) {
                c = (c - 'A' + 3) % 26 + 'A'; // Cifrado César para mayúsculas
```

```
        } else {
            c = (c - 'a' + 3) % 26 + 'a'; // Cifrado César para minúsculas
        }
    }
}
 for (char& c : nodoavl->municipio) {
    if (isalpha(c)) {
        if (isupper(c)) {
            c = (c - 'A' + 3) % 26 + 'A'; // Cifrado César para mayúsculas
        } else {
            c = (c - 'a' + 3) % 26 + 'a'; // Cifrado César para minúsculas
        }
    }
}


 for (char& c : nodoavl->ubicacion) {
    if (isalpha(c)) {
        if (isupper(c)) {
            c = (c - 'A' + 3) % 26 + 'A'; // Cifrado César para mayúsculas
        } else {
            c = (c - 'a' + 3) % 26 + 'a'; // Cifrado César para minúsculas
        }
    }
}
```

```
    // Recorrer el subárbol izquierdo
    encriptarCesarAVL(nodoavl->izq);

    // Recorrer el subárbol derecho
    encriptarCesarAVL(nodoavl->der);
}



void desencriptarCesarAVL(NodoAVL* nodo) {
    if (nodo == NULL) {
        return;
    }

    // Realizar desencriptación César en el nombre del nodo
    for (char& c : nodo->nombre) {
        if (isalpha(c)) {
            if (isupper(c)) {
                c = (c - 'A' - 3 + 26) % 26 + 'A'; // Desencriptar César para mayúsculas
            } else {
                c = (c - 'a' - 3 + 26) % 26 + 'a'; // Desencriptar César para minúsculas
            }
        }
    }
     for (char& c : nodo->departamento) {
        if (isalpha(c)) {
            if (isupper(c)) {
```

```
            c = (c - 'A' - 3 + 26) % 26 + 'A'; // Desencriptar César para mayúsculas
        } else {
            c = (c - 'a' - 3 + 26) % 26 + 'a'; // Desencriptar César para minúsculas
        }
    }
}
 for (char& c : nodo->municipio) {
    if (isalpha(c)) {
        if (isupper(c)) {
            c = (c - 'A' - 3 + 26) % 26 + 'A'; // Desencriptar César para mayúsculas
        } else {
            c = (c - 'a' - 3 + 26) % 26 + 'a'; // Desencriptar César para minúsculas
        }
    }
}
 for (char& c : nodo->ubicacion) {
    if (isalpha(c)) {
        if (isupper(c)) {
            c = (c - 'A' - 3 + 26) % 26 + 'A'; // Desencriptar César para mayúsculas
        } else {
            c = (c - 'a' - 3 + 26) % 26 + 'a'; // Desencriptar César para minúsculas
        }
    }
}
```

```
    // Recorrer el subárbol izquierdo
    desencriptarCesarAVL(nodo->izq);

    // Recorrer el subárbol derecho
    desencriptarCesarAVL(nodo->der);
}

void gotoxy2(int x,int y){

     HANDLE hcon;
     hcon = GetStdHandle(STD_OUTPUT_HANDLE);
     COORD dwPos;
     dwPos.X = x;
     dwPos.Y= y;
     SetConsoleCursorPosition(hcon,dwPos);
 }

void mostrarArbolavl(NodoAVL *&arbolavl, int auxY2){
  string nombre;
    if(arbolavl == NULL){

       return;
    }
   nombre=arbolavl->nombre.substr(0,arbolavl->nombre.find(' '));
    aux2 += 5;
```

```
    mostrarArbolavl(arbolavl -> izq,auxY2+6);
    gotoxy2(20+aux2-auxY2 , 30+auxY2);
    cout << arbolavl->dpi<<endl;
    gotoxy2(21+aux2-auxY2 , 31+auxY2);
    cout<< arbolavl->nombre.substr(0,arbolavl->nombre.find(' '))<<endl<<endl;




     mostrarArbolavl(arbolavl -> der,auxY2+8);



}
int maximo(int a, int b) {
    return (a > b) ? a : b;
}

int altura(NodoAVL* nodo) {
    if (nodo == NULL)
        return 0;
    return nodo->altura;
}

int obtenerBalance(NodoAVL* nodo) {
    if (nodo == NULL)
        return 0;
```

```
    return altura(nodo->izq) - altura(nodo->der);
}

NodoAVL* rotacionDerecha(NodoAVL* y) {
    NodoAVL* x = y->izq;
    NodoAVL* T2 = x->der;

    x->der = y;
    y->izq = T2;

    y->altura = maximo(altura(y->izq), altura(y->der)) + 1;
    x->altura = maximo(altura(x->izq), altura(x->der)) + 1;

    return x;
}

NodoAVL* rotacionIzquierda(NodoAVL* x) {
    NodoAVL* y = x->der;
    NodoAVL* T2 = y->izq;

    y->izq = x;
    x->der = T2;

    x->altura = maximo(altura(x->izq), altura(x->der)) + 1;
    y->altura = maximo(altura(y->izq), altura(y->der)) + 1;
```

```
    return y;
}

NodoAVL* rotacionDerechaDerecha(NodoAVL* nodo) {
    NodoAVL* x = nodo->der;
    NodoAVL* y = x->der;

    nodo->der = x->izq;
    x->izq = nodo;

    nodo->altura = maximo(altura(nodo->izq), altura(nodo->der)) + 1;
    x->altura = maximo(altura(x->izq), altura(x->der)) + 1;

    return x;
}

NodoAVL* rotacionIzquierdaIzquierda(NodoAVL* nodo) {
    NodoAVL* x = nodo->izq;
    NodoAVL* y = x->izq;

    nodo->izq = x->der;
    x->der = nodo;

    nodo->altura = maximo(altura(nodo->izq), altura(nodo->der)) + 1;
    x->altura = maximo(altura(x->izq), altura(x->der)) + 1;
```

```
    return x;
}

NodoAVL* insertarNodoAVL(NodoAVL* nodo, string dpi, string nombre, NodoAVL* padre = NULL)
{
    if (nodo == NULL) {
        NodoAVL* nuevoNodo = new NodoAVL();
        nuevoNodo->dpi = dpi;
        nuevoNodo->nombre = nombre;
        nuevoNodo->altura = 1;
        nuevoNodo->izq = NULL;
        nuevoNodo->der = NULL;
        nuevoNodo->padre = padre;
        return nuevoNodo;
    }

    if (dpi < nodo->dpi)
        nodo->izq = insertarNodoAVL(nodo->izq, dpi, nombre, nodo);
    else if (dpi > nodo->dpi)
        nodo->der = insertarNodoAVL(nodo->der, dpi, nombre, nodo);
    else
        return nodo;

    nodo->altura = 1 + maximo(altura(nodo->izq), altura(nodo->der));

    int balance = obtenerBalance(nodo);
```

```
    if (balance > 1 && dpi < nodo->izq->dpi)
        return rotacionDerecha(nodo);

    if (balance < -1 && dpi > nodo->der->dpi)
        return rotacionIzquierda(nodo);

    if (balance > 1 && dpi > nodo->izq->dpi) {
        nodo->izq = rotacionIzquierda(nodo->izq);
        return rotacionDerecha(nodo);
    }

    if (balance < -1 && dpi < nodo->der->dpi) {
        nodo->der = rotacionDerecha(nodo->der);
        return rotacionIzquierda(nodo);
    }

    return nodo;
}



bool buscarDatoAVL(NodoAVL* nodo, string dpi) {
    if (nodo == NULL)
        return false;
    else if (nodo->dpi == dpi) {
```

```
        cout << "DPI: " << dpi << "\t Nombre: " << nodo->nombre << endl;
        return true;
    } else if (dpi < nodo->dpi)
        return buscarDatoAVL(nodo->izq, dpi);
    else
        return buscarDatoAVL(nodo->der, dpi);
}

void recorridoInorden(NodoAVL* nodo) {
    if (nodo == NULL)
        return;
    recorridoInorden(nodo->izq);
    cout << "DPI: " << nodo->dpi << "\t Nombre: " << nodo->nombre << endl;
    recorridoInorden(nodo->der);
}

void recorridoPreorden(NodoAVL* nodo) {
    if (nodo == NULL)
        return;
    cout << "DPI: " << nodo->dpi << "\t Nombre: " << nodo->nombre << endl;
    recorridoPreorden(nodo->izq);
    recorridoPreorden(nodo->der);
}

void recorridoPostorden(NodoAVL* nodo) {
    if (nodo == NULL)
```

```
        return;
    recorridoPostorden(nodo->izq);
    recorridoPostorden(nodo->der);
    cout << "DPI: " << nodo->dpi << "\t Nombre: " << nodo->nombre << endl;
}

void lecturaAVL() {
    ifstream archivo;
    string contenido, nombre, dpi;
    int contador = 0, limite;
    archivo.open("vacunas.txt", ios::in);

    if (archivo.fail()) {
        cout << "* No se pudo leer el archivo *" << endl;
        exit(1);
    }

    cout << "Cuantas personas desea cargar:  ";
    cin >> limite;

    while (contador != limite && !archivo.eof()) {
        getline(archivo, contenido);
        dpi = contenido.substr(contenido.length() - 13);
        nombre = contenido.substr(0, contenido.length() - 13);
        arbolavl = insertarNodoAVL(arbolavl, dpi, nombre, NULL);
        cout << "DPI: " << dpi << "\tNombre: " << nombre << endl;
```

```
        contador++;
    }

    archivo.close();
}

bool estaEquilibrado(NodoAVL* nodo) {
    if (nodo == NULL)
        return true;

    int balance = obtenerBalance(nodo);

    if (balance > 1 || balance < -1)
        return false;

    return estaEquilibrado(nodo->izq) && estaEquilibrado(nodo->der);
}
int obtenerMaximo(int a, int b) {
  return (a > b) ? a : b;
}

NodoAVL* obtenerNodoMinimo(NodoAVL* nodo) {
    NodoAVL* actual = nodo;

    while (actual->izq != NULL) {
        actual = actual->izq;
```

```
    }

    return actual;
}

NodoAVL* eliminarNodoAVL(NodoAVL* nodo, string dpi) {
    if (nodo == NULL)
        return nodo;

    if (dpi < nodo->dpi)
        nodo->izq = eliminarNodoAVL(nodo->izq, dpi);
    else if (dpi > nodo->dpi)
        nodo->der = eliminarNodoAVL(nodo->der, dpi);
    else {
        // El nodo actual es el nodo a eliminar

        // Caso 1: El nodo no tiene hijos o solo tiene un hijo
        if (nodo->izq == NULL) {
            NodoAVL* temp = nodo->der;
            delete nodo;
            return temp;
        }
        else if (nodo->der == NULL) {
            NodoAVL* temp = nodo->izq;
            delete nodo;
            return temp;
```

```
    }

    // Caso 2: El nodo tiene dos hijos
    NodoAVL* temp = obtenerNodoMinimo(nodo->der);
    nodo->dpi = temp->dpi;
    nodo->nombre = temp->nombre;
    nodo->der = eliminarNodoAVL(nodo->der, temp->dpi);
}

nodo->altura = 1 + obtenerMaximo(altura(nodo->izq), altura(nodo->der));

int balance = obtenerBalance(nodo);

// Caso de desequilibrio izquierda-izquierda
if (balance > 1 && obtenerBalance(nodo->izq) >= 0)
    return rotacionDerecha(nodo);

// Caso de desequilibrio izquierda-derecha
if (balance > 1 && obtenerBalance(nodo->izq) < 0) {
    nodo->izq = rotacionIzquierda(nodo->izq);
    return rotacionDerecha(nodo);
}

// Caso de desequilibrio derecha-derecha
if (balance < -1 && obtenerBalance(nodo->der) <= 0)
    return rotacionIzquierda(nodo);
```

```
    // Caso de desequilibrio derecha-izquierda
    if (balance < -1 && obtenerBalance(nodo->der) > 0) {
        nodo->der = rotacionDerecha(nodo->der);
        return rotacionIzquierda(nodo);
    }

    return nodo;
}
void modificarPersonaAVL(NodoAVL* nodo, string dpi) {
    if (nodo == NULL) {
        cout << "La persona no existe en el árbol." << endl;
        return;
    }

    if (dpi < nodo->dpi) {
        modificarPersonaAVL(nodo->izq, dpi);
    } else if (dpi > nodo->dpi) {
        modificarPersonaAVL(nodo->der, dpi);
    } else {
        // Persona encontrada, solicitar nuevos datos
        cout << "Persona encontrada:\n";
        cout << "Nombre: " << nodo->nombre << endl;
        cout << "DPI: " << nodo->dpi << endl;
        cout << "Ingrese los nuevos datos:" << endl;
```

```
        cout << "Nombre: ";
        cin.ignore();
        getline(cin, nodo->nombre);

        cout << "Departamento: ";
        getline(cin, nodo->departamento);

        cout << "Municipio: ";
        getline(cin, nodo->municipio);

        cout << "Fecha 1: ";
        getline(cin, nodo->fecha1);

        cout << "Fecha 2: ";
        getline(cin, nodo->fecha2);

        cout << "Fecha 3: ";
        getline(cin, nodo->fecha3);

           cout << "Lugar: ";
        getline(cin, nodo->ubicacion);

        cout << "Los datos de la persona se han modificado correctamente." << endl;
    }
}
```

```
void guardarEnArchivo(NodoAVL* nodoavl, ofstream& archivo) {
   if (nodoavl == NULL) {
      return;
   }

   // Recorrer el subárbol izquierdo
   guardarEnArchivo(nodoavl->izq, archivo);

   // Encriptar el nombre de la persona antes de escribirlo en el archivo



   // Escribir los datos de la persona en el archivo
   archivo << "nombre: " << nodoavl->nombre << " dpi: " << nodoavl->dpi
         << " departamento: " << nodoavl->departamento << " municipio: " << nodoavl->municipio
         << " fecha1: " << nodoavl->fecha1 << " fecha2: " << nodoavl->fecha2
         << " fecha3: " << nodoavl->fecha3 << " ubicacion: " << nodoavl->ubicacion << endl;

   // Recorrer el subárbol derecho
   guardarEnArchivo(nodoavl->der, archivo);
}
void guardarArbolEnArchivo(NodoAVL* arbolavl) {
ofstream archivo("AVL.txt");
if (archivo.is_open()) {
guardarEnArchivo(arbolavl, archivo);
archivo.close();
cout << "El arbol se ha guardado en el archivo correctamente." << endl;
```

```
} else {
cout << "No se pudo abrir el archivo." << endl;
}
}


 void menuAVL() {
   int opcion, auxY = 0;
   string dpi, nombre;

   HANDLE hConsole = GetStdHandle( STD_OUTPUT_HANDLE );
 SetConsoleTextAttribute(hConsole, 1);
   do {
      cout << "\t --- MENU AVL --- " << endl;
      cout << "1. Cuántas personas desea agregar al Arbol" << endl;
      cout << "2. Mostrar arbol Avl" << endl;
      cout << "3. Agregar persona:" << endl;
      cout << "4. Buscar Persona" << endl;
      cout << "5. Recorrido Inorden" << endl;
      cout << "6. Recorrido Preorden" << endl;
      cout << "7. Recorrido Postorden" << endl;
      cout << "8. Verificar equilibrio" << endl;
      cout<<"9. Eliminar Persona"<<endl;
      cout<<"10. Guardar abol"<<endl;
      cout << "11. modificar persona" << endl;
      cout<<"12. Encriptar el arbol"<<endl;
```

```
cout<<"13. Desencriptar arbol"<<endl;
 cout << "14. regresar  al menu principal" << endl;
cout << "15. Salir" << endl;
cout << "Opcion: ";
cin >> opcion;

switch (opcion) {
   case 1:
      system("cls");
      lecturaAVL();
      cout << endl;
      break;
   case 2:
      cout << "\n\n\t --- MOSTRAR ARBOL COMPLETO --- \n\n"<<endl;
      mostrarArbolavl(arbolavl, auxY2);
      cout << endl;
      system("pause");
      break;
   case 3:
      cout << "\n Escriba un DPI: ";
      cin >> dpi;
      cout << "\n Escriba el nombre completo: ";
      cin.ignore();
      getline(cin, nombre);
      arbolavl = insertarNodoAVL(arbolavl, dpi, nombre, NULL);
      cout << "\n\n\t --- ARBOL ACTUALIZADO --- \n\n";
```

```
    mostrarArbolavl(arbolavl, auxY2);
    cout << endl;
    system("pause");
    break;
case 4:
    cout << "\n Escriba un DPI para buscar: ";
    cin >> dpi;
    if (buscarDatoAVL(arbolavl, dpi))
        cout << "\n\n\t*** Persona encontrada ***\n\n";
    else
        cout << "\n\n\t*** Persona NO encontrada ***\n\n";
    system("pause");
    break;
case 5:
    cout << "\n\n\t --- RECORRIDO INORDEN --- \n\n";
    recorridoInorden(arbolavl);
    cout << endl;
    system("pause");
    break;
case 6:
    cout << "\n\n\t --- RECORRIDO PREORDEN --- \n\n";
    recorridoPreorden(arbolavl);
    cout << endl;
    system("pause");
    break;
case 7:
```

```
    cout << "\n\n\t --- RECORRIDO POSTORDEN --- \n\n";
    recorridoPostorden(arbolavl);
    cout << endl;
    system("pause");
    break;
case 8:
    if (estaEquilibrado(arbolavl))
        cout << "\n\n\t *** El arbol esta equilibrado ***\n\n";
    else
        cout << "\n\n\t *** El arbol NO esta equilibrado ***\n\n";
    system("pause");
    break;
case 9:
    cout << "Ingrese el DPI del nodo a eliminar: ";
    cin >> dpi;
    arbolavl = eliminarNodoAVL(arbolavl, dpi);
    cout << "Nodo eliminado correctamente." << endl;
    system("pause");
    break;
case 10:

    guardarArbolEnArchivo(arbolavl);
    cout<<"su archivo se guardo correctamente"<<endl;
    system("pause");
    cout << endl;
    break;
```

```
        case 11:
            system("cls");
cout << "Ingrese el DPI de la persona a modificar: ";
cin >> dpi;
modificarPersonaAVL(arbolavl, dpi);
cout << endl;
break;
         case 12:
        system("cls");
        cout << " encriptado exitoso " << endl;
        encriptarCesarAVL(arbolavl);
        cout << endl;
        break;
        case 13:
        system("cls");
        cout << "desencriptado Exitoso" << endl;
        desencriptarCesarAVL(arbolavl);
        cout << endl;
        break;
       case 14:
          system("cls");
           return;
           break;
        case 15:
           cout << "\n\n\t *** SALIENDO DEL PROGRAMA ***\n\n";
           break;
```

```
            default:
                cout << "\n\n\t *** OPCION INVALIDA ***\n\n";
                break;
        }

        system("cls");
    } while (opcion != 15);
}



int main() {

HANDLE hConsole = GetStdHandle( STD_OUTPUT_HANDLE );
 SetConsoleTextAttribute(hConsole, 12);
SetConsoleOutputCP(CP_UTF8);
MENU_PRINCIPAL();







    return 0;
    getch();
}
```