

Manual Técnico

Método Main

```
main.cpp x
1      #include <iostream>
2      #include "menu.h"
3      #include "matrizD.h"
4      #include "arbol.h"
5      using namespace std;
6
7      int main()
8      {
9          menu start;
10         start.login();
```

Nodo Principal Árbol AVL

```
main.cpp x  src\nodoABB.cpp x
1      #include "nodoABB.h"
2      using namespace std;
3      nodoABB::nodoABB(int valor_)
4      {
5          izquierda = nullptr;
6          derecha = nullptr;
7          valor = valor_;
8          enRenta = false;
9      }
10
```

Árbol AVL y Métodos

main.cpp X src\arbol.cpp X

```
1  #include "arbol.h"
2  #include <iostream>
3  #include <ctime>
4  #include "nodoABB.h"
5  #include <fstream>
6  using namespace std;
7
8  arbol::arbol()
9  {
10     raiz = nullptr;
11     depaReporte = 0;
12 }
13 bool arbol::vacio()
14 {
15     return (raiz == nullptr);
16 }
17 void arbol::equilibrarArbol(nodoABB*& temporal)
18 {
19
20     if(temporal != 0)
21     {
22         equilibrarArbol(temporal->izquierda);
23         equilibrarArbol(temporal->derecha);
24         int Izq, Dch;
25         if(temporal->izquierda != 0)
26         {
27             Izq = temporal->izquierda->altura;
28         }
29         else
30         {
31             Izq = 0;
32         }
33         if(temporal->derecha != 0)
34         {
35             Dch = temporal->derecha->altura;
36         }
37         else
38         {
39             Dch = 0;
40         }
41
42         if(Izq - Dch < -1)
43         {
44
45             if(temporal->derecha->izquierda != 0)
46             {
47                 Izq = temporal->derecha->izquierda->altura;
48             }
```

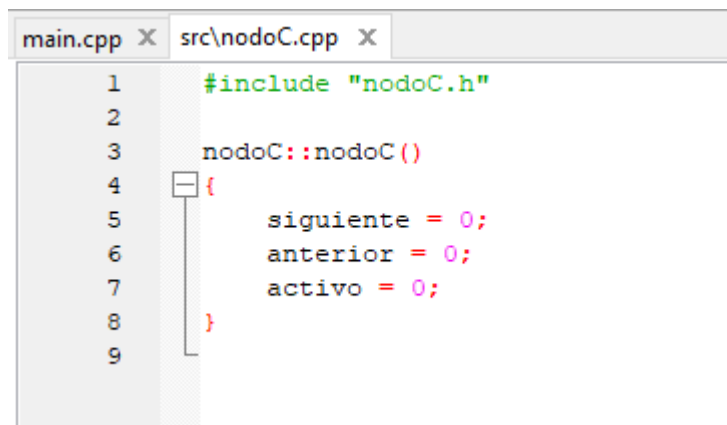
```
,
void arbol::rotarIzquierda(nodoABB*& temporal)
{
    bool esRaiz = false;
    if(raiz == temporal)
    {
        esRaiz = true;
        cout << "Es Raiz \n";
    }
    nodoABB* cambio = temporal->derecha;
    temporal->derecha = cambio->izquierda;
    cambio->izquierda = temporal;
    temporal = cambio;
    if(esRaiz)
    {
        raiz = cambio;
    }
}

void arbol::rotarDerecha(nodoABB*& temporal)
{
    bool esRaiz = false;
    if(raiz == temporal)
    {
        esRaiz = true;
        cout << "Es Raiz \n";
    }
    nodoABB* cambio = temporal->izquierda;
    temporal->izquierda = cambio->derecha;
    cambio->derecha = temporal;
    temporal = cambio;
    if(esRaiz == true)
    {
        raiz = cambio;
    }
}

int arbol::calcularAltura(nodoABB* temporal)
{
    int alturaI = 1, alturaD = 1 ;

    if(temporal->izquierda != 0)
    {
        alturaI += calcularAltura(temporal->izquierda);
    }
}
```

Nodo Lista Circular Doblemente Enlazada



The image shows a code editor with two tabs: 'main.cpp' and 'src\nodoC.cpp'. The 'src\nodoC.cpp' tab is active, displaying the following C++ code:

```
1  #include "nodoC.h"
2
3  nodoC::nodoC()
4  {
5      siguiente = 0;
6      anterior = 0;
7      activo = 0;
8  }
9
```

The code defines a constructor for a node in a circular list. It includes the header 'nodoC.h' and initializes three attributes: 'siguiente' (next), 'anterior' (previous), and 'activo' (active), all set to 0. The code is color-coded: green for the include statement, red for the class scope resolution and closing brace, and magenta for the variable names and assignment values.

Lista Circular y Métodos

```

main.cpp x src\listaC.cpp x
1  #include "listaC.h"
2  #include "nodoC.h"
3  #include <iostream>
4  #include <ctime>
5  #include <fstream>
6  using namespace std;
7  listaC::listaC()
8  {
9      inicio = 0;
10     fin = 0;
11 }
12 nodoC* listaC::insertar(int idActivo_, string codigoActivo_, string
13 {
14     srand(time(0));
15     string idCodigo;
16     static const char letras[] = {'a', 'b', 'c', 'd', 'f', 'g', 'h', 'i',
17     for(int i = 0; i < 15; i++)
18     {
19         int rango = 0 + (rand() % 33);
20         idCodigo += letras[rango];
21     }
22     time_t tAct = time(0);
23     string hora = ctime(&tAct);
24
25     nodoC* nuevo = new nodoC();
26     nuevo->idTransaccion = idCodigo;
27     nuevo->idActivo = idActivo_;
28     nuevo->codigoActivo = codigoActivo_;
29     nuevo->user = user_;
30     nuevo->departamento = departamento_;
31     nuevo->empresa = empresa_;
32     nuevo->fecha = hora;
33     nuevo->tiempo = tiempo_;
34     nuevo->activo = true;
35     if(inicio == 0 && fin == 0)
36     {
37         inicio = nuevo;
38         nuevo->siguiente = inicio;
39         nuevo->anterior = fin;
40         fin = nuevo;
41     }
42     else
43     {
44         fin->siguiente = nuevo;
45         nuevo->anterior = fin;
46         nuevo->siguiente = inicio;
47         inicio->anterior = nuevo;
48

```

```

2
3 void listaC::mostrar(nodoC* temporal)
4 {
5
6     cout << "***Historial de Transacciones General" << endl;
7     if(temporal != 0)
8     {
9         cout << "Transaccion " << temporal->idTransaccion << " - id_activo
10         if(temporal != fin)
11         {
12             mostrar(temporal->siguiente);
13         }
14     }
15 }
16
17 void listaC::devolver(int idActivo_, string user_)
18 {
19     nodoC* temporal = inicio;
20     do
21     {
22         if(temporal->idActivo == idActivo_ && temporal->user == user_)
23         {
24             temporal->activo = false;
25             break;
26         }
27         temporal = temporal->siguiente;
28     }
29     while(temporal->siguiente != 0 && temporal->siguiente != inicio);
30 }
31
32 void listaC::startReporteTransacciones()
33 {
34     ofstream archivo;
35     string label = "Reporte de Transacciones";
36     archivo.open("reporteTransacciones.dot");
37     string inicioDot = "digraph G {";
38     string finalDot = "}";
39     archivo << inicioDot;
40     archivo << "label = \"" + label + "\"" << endl;
41     archivo << "node[shape = rectangle];";
42
43     reporteTransacciones(inicio, archivo);
44
45     archivo << finalDot << endl;
46     archivo.close();

```

Nodo Matriz Dispersa y Métodos

main.cpp X src\nodoM.cpp X

```
1  #include "nodoM.h"
2  #include <iostream>
3  using namespace std;
4
5  nodoM::nodoM()
6  {
7      izquierda = 0;
8      derecha = 0;
9      arriba = 0;
10     abajo = 0;
11     zmas = 0;
12     zmenos = 0;
13     fnodoz = 0;
14     idContador = 0;
15 }
16 void nodoM::aumentarContador()
17 {
18     idContador++;
19 }
20 void nodoM::startReporteActivosDepartamento(nodoM* temporal)
21 {
22     ofstream archivo;
23     string label = "Reporte de Activos de Departamento";
24     archivo.open("reporteActivosDepartamento.dot");
25     string inicioDot = "digraph G {";
26     string finalDot = "}";
27     archivo << inicioDot;
28     archivo << "label = \"" + label + "\"" << endl;
29     archivo << "node[shape = circle];";
30
31     reporteActivosDepartamento(temporal, archivo);
32
33     archivo << finalDot << endl;
34     archivo.close();
35     system("C:\\Graphviz2.38\\bin\\dot -Tpng reporteActivosD");
36 }
37 void nodoM::reporteActivosDepartamento(nodoM* temporal, ofst
38 {
39     if(temporal != 0)
40     {
41
42         nodoM* otro = temporal;
43         while(otro->zmas != 0)
44         {
45             otro->arbolAVL.startReporteActivosDeUsuario();
46             otro = otro->zmas;
47         }
48     }
```

Matriz Dispersa y Métodos


```

main.cpp x src\matrizD.cpp x
1      #include "matrizD.h"
2      #include "nodoM.h"
3      #include "nodoABB.h"
4      #include <iostream>
5      #include <fstream>
6      using namespace std;
7
8      matrizD::matrizD()
9      {
10         raiz = new nodoM();
11         raiz->nombre = "inicio";
12         fDepartamentos = raiz;
13         fEmpresas = raiz;
14     }
15     nodoM* matrizD::crearDepartamento(string nombre_)
16     {
17         nodoM* nuevo = new nodoM();
18         nuevo->nombre = nombre_;
19         cDepartamentos += 1;
20         nuevo->fila = -1;
21         nuevo->columna = cDepartamentos;
22         fDepartamentos->derecha = nuevo;
23         nuevo->izquierda = fDepartamentos;
24         fDepartamentos = nuevo;
25         return nuevo;
26     }
27
28     nodoM* matrizD::crearEmpresa(string nombre_)
29     {
30         nodoM* nuevo = new nodoM();
31         nuevo->nombre = nombre_;
32         cEmpresas += 1;
33         nuevo->columna = -1;
34         nuevo->fila = cEmpresas;
35         fEmpresas->abajo = nuevo;
36         nuevo->arriba = fEmpresas;
37         fEmpresas = nuevo;
38         return nuevo;
39     }
40
41     void matrizD::crearUsuario(string nombre_, string user_, str
42     {
43         nodoM* depa = raiz;
44         nodoM* empre = raiz;
45         bool exDepa = false, exEmpre = false;
46         while(depa->derecha != 0)
47         {
48             depa = depa->derecha;

```

```

    }
    nodoM* matrizD::buscarUsuario(string user_, string pass_, string depa_, string empre_)
    {
        nodoM* depa = buscarDepartamento(depa_);
        nodoM* empre = buscarEmpresa(empre_);
        if(depa != 0 && empre != 0)
        {
            int prefila = empre->fila;
            int precolumna = depa->columna;
            while(depa->abajo != 0)
            {
                depa = depa->abajo;
                if(depa->fila == prefila && depa->columna == precolumna)
                {
                    break;
                }
            }
            if(depa->user == user_ && depa->pass == pass_)
            {
                cout << "\n Login Usuario " << depa->user << endl;
                return depa;
            }
            else
            {
                while(depa->zmas != 0)
                {
                    depa = depa->zmas;
                    if(depa->user == user_ && depa->pass == pass_)
                    {
                        cout << "\n Login Usuario " << depa->user << endl;
                        return depa;
                    }
                }
                return 0;
            }
        }
        return 0;
    }

    nodoM* matrizD::buscarDepartamento(string depa_)
    {
        nodoM* depa = raiz;
        while(depa->derecha != 0)
        {

```

Menu Principal

```
main.cpp x src\menu.cpp x
1  #include "menu.h"
2  #include "matrizD.h"
3  #include "nodoM.h"
4  #include <iostream>
5  using namespace std;
6
7  menu::menu()
8  {
9      usuarioActual = 0;
10     usuarioReportes = 0;
11     depaReportes = 0;
12     empreReportes = 0;
13 }
14 void menu::login()
15 {
16     do
17     {
18         cout << "-----" << endl;
19         cout << "##### Renta de Activos #####" << endl;
20         cout << "##### Iniciar Sesión #####" << endl;
21         cout << "Ingresa Usuario" << endl;
22         cin >> user ;
23         cout << "Ingresa Password" << endl;
24         cin >> pass;
25         cout << "Ingresa Departamento" << endl;
26         cin >> departamento;
27         cout << "Ingresa Empresa" << endl;
28         cin >> empresa;
29         if(user == "admin" && pass=="admin")
30         {
31             menuAdministrador();
32         }
33         else
34         {
35             usuarioActual = matriz.buscarUsuario(user,pass,departamento,empresa);
36             if(usuarioActual != 0)
37             {
38                 depaActual = departamento;
39                 empreActual = empresa;
40                 menuUser();
41             }
42             else
43             {
44                 cout << "***No Existe el Usuario" << endl;;
45             }
46         }
47     }
48 }
```

```

void menu::menuUser()
{
    do
    {
        cout << "-----" << endl;
        cout << "##### Bienvenido " + usuarioActual->nombre + " #####" << endl;
        cout << "##### 1. Agregar Activo #####" << endl;
        cout << "##### 2. Eliminar Activo #####" << endl;
        cout << "##### 3. Modificar Activo #####" << endl;
        cout << "##### 4. Rentar Activo #####" << endl;
        cout << "##### 5. Activos Rentados #####" << endl;
        cout << "##### 6. Mis Activos Rentados #####" << endl;
        cout << "##### 7. Cerrar Sesion #####" << endl;
        cout << "Ingresa Opcion" << endl;
        cin >> opcion;
        switch(opcion)
        {
            case '1':
                crearActivos();
                break;
            case '2':
                eliminarActivos();
                break;
            case '3':
                modificarActivos();
                break;
            case '4':
                menuRentaActivos();
                break;
            case '5':
                menuActivosRentados();
                break;
            case '6':
                misActivosRentados();
                break;
            case '7':
                cout << "***Cerrando Sesion" << endl;;
                seguir = false;
                break;
            case '8':
                usuarioActual->arbolAVL.mostrar(usuarioActual->arbolAVL.raiz);
                break;
            case '9':
                usuarioActual->arbolAVL.mostrarPreOrden(usuarioActual->arbolAVL.raiz);
                break;
        }
    }
}

```

```

    },
    void menu::crearActivos()
    {
        cout << "-----" << endl;
        cout << "***** Agregar Activo *****" << endl;
        cout << "Ingrese Nombre" << endl;
        cin >> nombreActivo;
        cout << "Ingrese Descripcion" << endl;
        cin >> descripcionActivo;
        usuarioActual->aumentarContador();
        usuarioActual->arbolAVL.insertar(usuarioActual->arbolAVL.raiz, usuarioActual->idContador);
    }

    void menu::eliminarActivos()
    {
        cout << "-----" << endl;
        cout << "***** Eliminar Activo *****" << endl;
        usuarioActual->arbolAVL.mostrar(usuarioActual->arbolAVL.raiz);
        cout << "Ingresar ID Activo" << endl;
        cin >> idActivo;
        if(usuarioActual->arbolAVL.buscar(usuarioActual->arbolAVL.raiz, idActivo))
        {
            usuarioActual->arbolAVL.eliminar(usuarioActual->arbolAVL.raiz, idActivo);
        }
    }

    void menu::modificarActivos()
    {
        cout << "-----" << endl;
        cout << "***** Modificar Activo *****" << endl;
        usuarioActual->arbolAVL.mostrar(usuarioActual->arbolAVL.raiz);
        cout << "Ingresar ID Activo" << endl;
        cin >> idActivo;
        cout << "Ingresar Descripcion Nueva" << endl;
        cin >> descripcionActivo;
        usuarioActual->arbolAVL.buscarParaModificar(usuarioActual->arbolAVL.raiz, idActivo, descr
    }

    void menu::menuPrincipal()

```