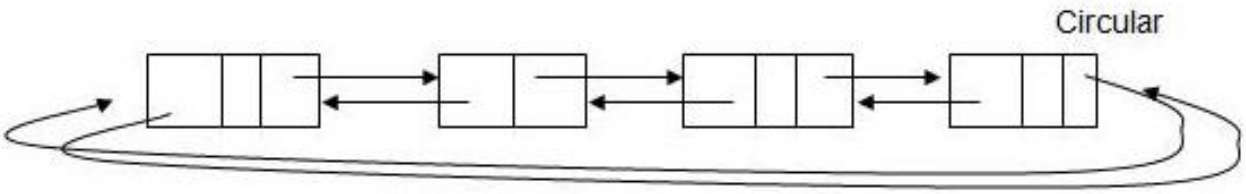
Manual Técnico

Estructuras utilizadas dentro de la aplicación.

Lista Circular

La forma visual de la implementación de esta estructura es la siguiente:



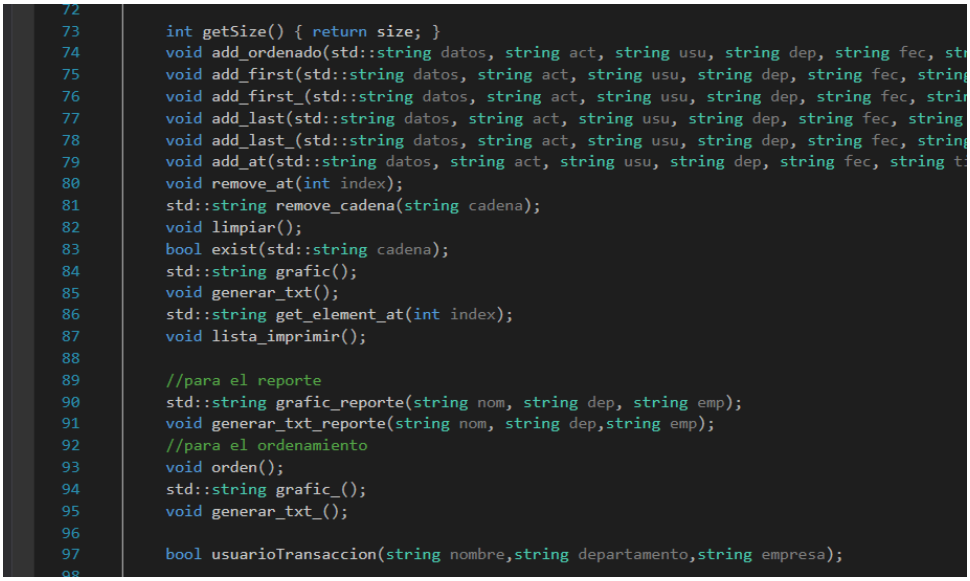
Se utilizo esta estructura para poder guardar las distintas transacciones que se realizaron en la aplicación, donde al momento de rentar un activo por parte del usuario se ingreso ese registro acá utilizando un código alfanumerico de 15 caracteres, se le llamo a esta variable global transacciones_total

```
18  using namespace std;
19
20
21  //matriz de toda la aplicacion
22  estructura_matriz <string> *Users = new estructura_matriz<string>();
23  //Estructura de transacciones
24  estructura_circular *transacciones = new estructura_circular();
25  //Total de transacciones como historial
26  estructura_circular *transacciones_total = new estructura_circular();
```

Explicación de como fue implementada la estructura lista circular, parte de los atributos que fueron utilizados están en la siguiente imagen , donde se utilizo una clase hija tipo Nodo, en la cual se guardaron apuntadores hacia un siguiente nodo como también para un nodo anterior. Parte de los atributos guardados fueron como usuario, departamento, fecha en que se hizo la transacción, tiempo el cual fue rentado el activo y nombre de la empresa que pertenecia el usuario que rento el activo.

```
12  class estructura_circular
13  {
14      class Nodo {
15      public:
16          Nodo(std::string x, std::string act,std::string usu,std::str
17          {
18              next = 0;
19              before = 0;
20              dato = x;
21              activo = act;
22              usuario = usu;
23              departamento = dep;
24              fecha = fec;
25              tiempo = tie;
26              empresa = emp;
27
28          }
29
30          Nodo *getNext() { return next; }
31          Nodo *getBefore() { return before; }
32          void setNext(Nodo *n) { next = n; }
33          void setBefore(Nodo *n) { before = n; }
34
35          std::string getDato() { return dato; }
36          std::string getActivo() { return activo; }
37          std::string getUsuario() { return usuario; }
38          std::string getDepartamento() { return departamento; }
39          std::string getFecha() { return fecha; }
40          std::string getTiempo() { return tiempo; }
41          std::string getEmpresa() { return empresa; }
```

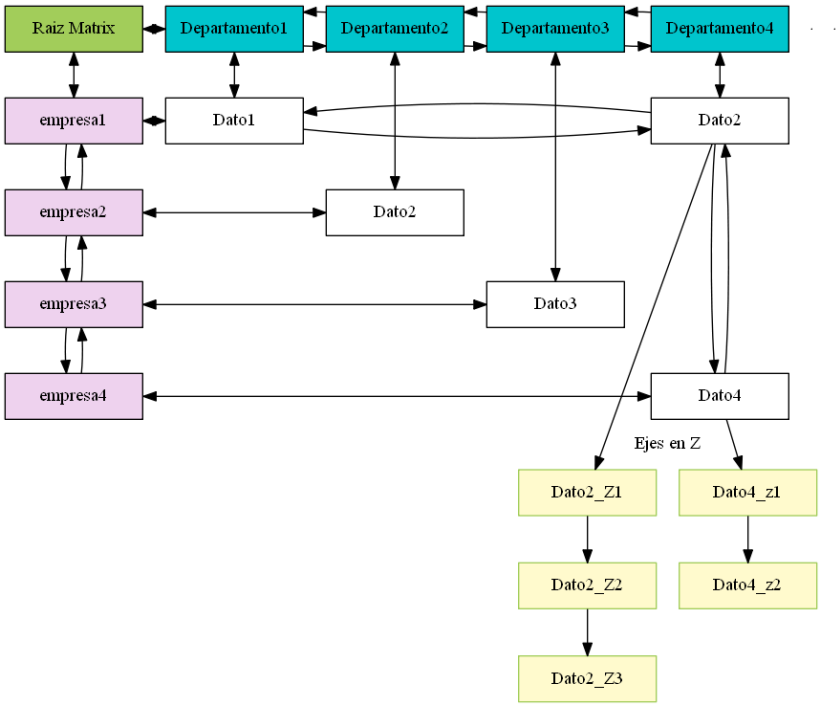
Se implementarion sus gets y sets de dichos parámetros descritos. Parte de los métodos implementados se visualizan en la siguiente imagen donde fueron creados métodos como insertar al principio, insertar al final, remover de la lista, existe usuario en la lista, limpiar lista , imprimir lista, generar reporte de lista…. Entre otros.



```
72
73      int getSize() { return size; }
74      void add_ordenado(std::string datos, string act, string usu, string dep, string fec, str
75      void add_first(std::string datos, string act, string usu, string dep, string fec, string
76      void add_first_(std::string datos, string act, string usu, string dep, string fec, strin
77      void add_last(std::string datos, string act, string usu, string dep, string fec, string
78      void add_last_(std::string datos, string act, string usu, string dep, string fec, string
79      void add_at(std::string datos, string act, string usu, string dep, string fec, string ti
80      void remove_at(int index);
81      std::string remove_cadena(string cadena);
82      void limpiar();
83      bool exist(std::string cadena);
84      std::string grafic();
85      void generar_txt();
86      std::string get_element_at(int index);
87      void lista_imprimir();
88
89      //para el reporte
90      std::string grafic_reporte(string nom, string dep, string emp);
91      void generar_txt_reporte(string nom, string dep,string emp);
92      //para el ordenamiento
93      void orden();
94      std::string grafic_();
95      void generar_txt_();
96
97      bool usuarioTransaccion(string nombre,string departamento,string empresa);
```

Matriz Dispersa

Esta estructura se representa gráficamente de la siguiente forma:



De esta manera fue que se implemento la estructura donde en sus cabeceras columnas están los departamentos y en las cabeceras fila están las empresas a las que pertenece un usuario. Los nodos reflejan el nombre de dicho usuario guardado en la aplicación y los nodos de color amarillo reflejan el eje en Z donde hay mas usuarios guardados en ese nodo en especifico.

Parte de la implementación la espedificare a continuación:

```
class Nodo {
public:
    Nodo(string departamentoX, string empresaY, T x, string pas)
    {
        //Aca inicializo los punteros en null que es igual a cero tambien
        this->next = 0;
        this->before = 0;
        this->up = 0;
        this->down = 0;
        this->z_front = 0;
        this->z_back = 0;
        //Aca voy a guardar los datos y el tipo que le ingresare
        this->x = departamentoX;
        this->y = empresaY;
        this->dato = x;
        this->pass = pas;
        //para el arbol balanceado
        this->activos_rentar = new estructura_arbol_valanceado<string>();

    }

    Nodo *getNext() { return next; }
    Nodo *getBefore() { return before; }
    Nodo *getUp() { return up; }
    Nodo *getDown() { return down; }
    Nodo *getZ_front() { return z_front; }
    Nodo *getZ_back() { return z_back; }
    string getX() { return x; }
    string getY() { return y; }
```

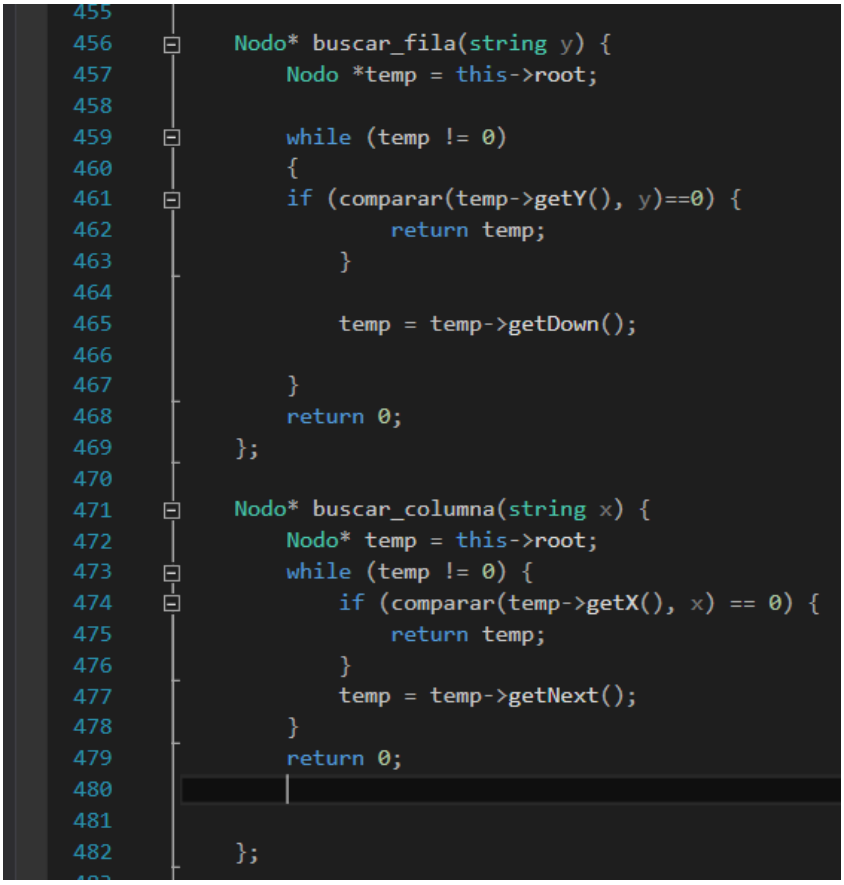
Dentro de la clase Matriz Dispersa podemos encontrar una clase hija la cual es la clase Nodo, pero de la matriz, esta esta compuesta por 6 apuntadores hacia otros nodos los cuales son abajo, arriba, siguiente, anterior, atrás y adelante. También se puede visualizar los atributos que se guardaron en cada nodo tales como Dato(nombre del Usuario), pass(contraseña del usuario) , X(Departamento) , Y (Empresa).

También se crearon sus métodos de gets y sets para dichos atributos.

Dentro de esta matriz tenemos un método llamado comparar el cual es el encargado de comparar string dentro de la estructura, esto servirá para poder ordenar cabeceras columnas y cabeceras fila.

```
// Este metodo me compara dos strings pero los compara haciendolos cadena de char primero porque se utiliza el metodo strcpy_s
// El cual me devuelve un int el cual me dice que orden alfabeticamente van los dos strings
// Este codigo es totalmente mio,
int comparar(std::string a, std::string b) {
    int n1 = a.length();
    int n2 = b.length();

    //Valor estatico porque visual estudio no me dejo meter una variable nose porque razon
    char char_a[100];
    char char_b[100];

    strcpy_s(char_a, a.c_str());
    strcpy_s(char_b, b.c_str());
    return strcmp(char_a, char_b);
}
```

Dentro de esta estructura se realizaron métodos para ayudar al método de inserta en la matriz tales como buscar_fila , buscar_columna que son necesarios para poder insertar en la matriz dispersa. Estos métodos son los siguientes



```
455
456     Nodo* buscar_fila(string y) {
457         Nodo *temp = this->root;
458
459         while (temp != 0)
460         {
461         if (comparar(temp->getY(), y)==0) {
462                 return temp;
463             }
464
465             temp = temp->getDown();
466
467         }
468         return 0;
469     };
470
471     Nodo* buscar_columna(string x) {
472         Nodo* temp = this->root;
473         while (temp != 0) {
474             if (comparar(temp->getX(), x) == 0) {
475                 return temp;
476             }
477             temp = temp->getNext();
478         }
479         return 0;
480
481
482     };
```

Luego de encontrar una columna o no utilizo este método el cual insertara en columna de mi matriz a partir de la raíz

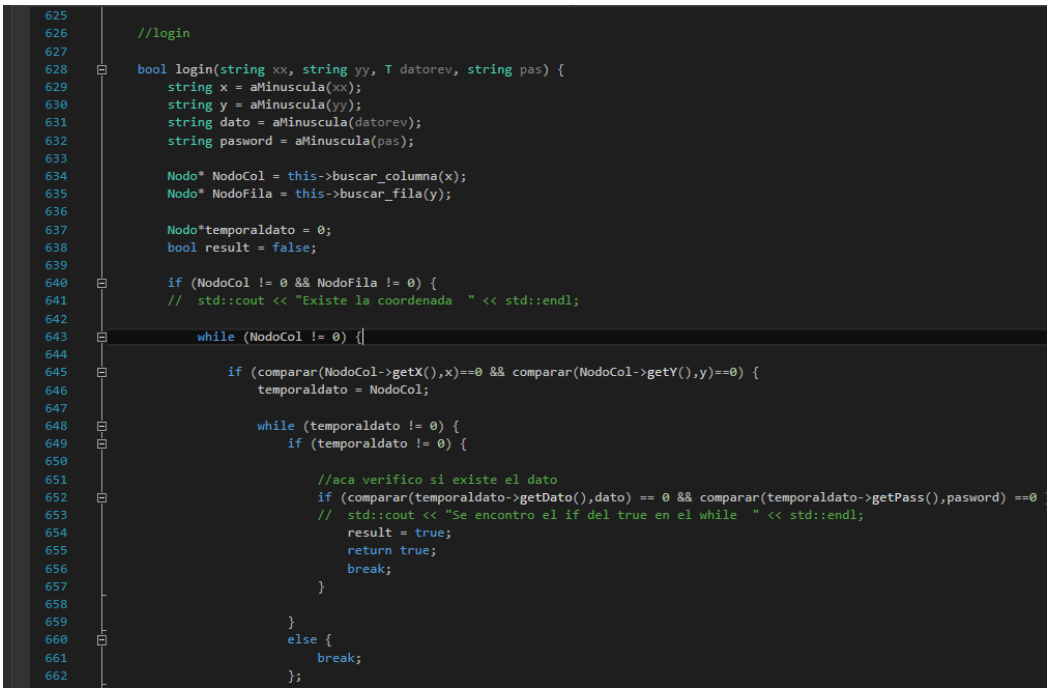
```
Nodo* insertar_ordenado_columna(Nodo *nuevo, Nodo* cabeza_col) {

    Nodo*temp = cabeza_col;
    bool bandera = false;
    while (true) {
        if (comparar(temp->getX(), nuevo->getX()) ==0) {
            temp->setY(nuevo->getY());
            temp->setDato(nuevo->getDato());
            return temp;

        }
        else if (comparar(temp->getX(),nuevo->getX()) ==1) {
            bandera = true;
            break;

        };
        if (temp->getNext() != 0) {
            temp = temp->getNext();
        }
        else {
            break;
        };
    };
    if (bandera == true) {
        nuevo->setNext(temp);
        temp->getBefore()->setNext(nuevo);
        nuevo->setBefore(temp->getBefore());
        temp->setBefore(nuevo);
    }
    else {
        temp->setNext(nuevo);
        nuevo->setBefore(temp);
    };
    return nuevo;
};
```

De igual se encuentra o no la fila buscada se utiliza el siguiente método para poder insertar la fila ordenada.

```
626     //login
627
628     bool login(string xx, string yy, T datorev, string pas) {
629         string x = aMinuscula(xx);
630         string y = aMinuscula(yy);
631         string dato = aMinuscula(datorev);
632         string pasword = aMinuscula(pas);
633
634         Nodo* NodoCol = this->buscar_columna(x);
635         Nodo* NodoFila = this->buscar_fila(y);
636
637         Nodo*temporaldato = 0;
638         bool result = false;
639
640         if (NodoCol != 0 && NodoFila != 0) {
641         //  std::cout << "Existe la coordenada  " << std::endl;
642
643             while (NodoCol != 0) {
644
645                 if (comparar(NodoCol->getX(),x)==0 && comparar(NodoCol->getY(),y)==0) {
646                     temporaldato = NodoCol;
647
648                     while (temporaldato != 0) {
649                         if (temporaldato != 0) {
650
651                             //aca verifico si existe el dato
652                             if (comparar(temporaldato->getDato(),dato) == 0 && comparar(temporaldato->getPass(),pasword) ==0 )
653                             //  std::cout << "Se encontro el if del true en el while  " << std::endl;
654                                 result = true;
655                                 return true;
656                                 break;
657                             }
658
659                         }
660                         else {
661                             break;
662                         };
```

```
524
525     Nodo* insertar_ordenado_fila(Nodo *nuevo, Nodo* cabeza_fila) {
526         Nodo*temp = cabeza_fila;
527         bool bandera = false;
528         while (true) {
529             if (comparar(temp->getY(), nuevo->getY())==0) {
530                 temp->setX(nuevo->getX());
531                 temp->setDato(nuevo->getDato());
532                 return temp;
533
534             }
535             else if (comparar(temp->getY(), nuevo->getY()) ==1) {
536                 bandera = true;
537                 break;
538
539             };
540             if (temp->getDown() != 0) {
541                 temp = temp->getDown();
542             }
543             else {
544                 break;
545             };
546         };
547         if (bandera == true) {
548             nuevo->setDown(temp);
549             temp->getUp()->setDown(nuevo);
550             nuevo->setUp(temp->getUp());
551             temp->setUp(nuevo);
552         }
553         else {
554             temp->setDown(nuevo);
555             nuevo->setUp(temp);
556         };
557         return nuevo;
558     };
559
```

Para loguearse un usuario se necesita el siguiente método el cual hace una búsqueda por columna y va comparando los dados del usuario tales como departamento , empresa, nombre y contraseña



```
625
626     //login
627
628     bool login(string xx, string yy, T datorev, string pas) {
629         string x = aMinuscula(xx);
630         string y = aMinuscula(yy);
631         string dato = aMinuscula(datorev);
632         string pasword = aMinuscula(pas);
633
634         Nodo* NodoCol = this->buscar_columna(x);
635         Nodo* NodoFila = this->buscar_fila(y);
636
637         Nodo*temporaldato = 0;
638         bool result = false;
639
640         if (NodoCol != 0 && NodoFila != 0) {
641         //  std::cout << "Existe la coordenada  " << std::endl;
642
643             while (NodoCol != 0) {
644
645                 if (comparar(NodoCol->getX(),x)==0 && comparar(NodoCol->getY(),y)==0) {
646                     temporaldato = NodoCol;
647
648                     while (temporaldato != 0) {
649                         if (temporaldato != 0) {
650
651                             //aca verifico si existe el dato
652                             if (comparar(temporaldato->getDato(),dato) == 0 && comparar(temporaldato->getPass(),pasword) ==0
653                             //  std::cout << "Se encontro el if del true en el while  " << std::endl;
654                                 result = true;
655                                 return true;
656                                 break;
657                             }
658
659                         }
660                         else {
661                             break;
662                         };
```