Laboratorio de Estructuras de Datos

PROYECTO EDD – FASE 1

MANUAL TECNICO

FECHA: 24/08/2022

Juan Josue Zuleta Beb
Carné: 202006353

Introducción

El presente documento describe los aspectos técnicos informáticos de la aplicación, deseñada a través de código de estructura de datos con paradigma orientado a objetos. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

Objetivos

Instruir el uso adecuado del de la instalación y comprensión del código y de la implementación de métodos, para el acceso oportuno y adecuado en la inicialización de este, mostrando los pasos a seguir en el proceso de inicialización, así como la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración.

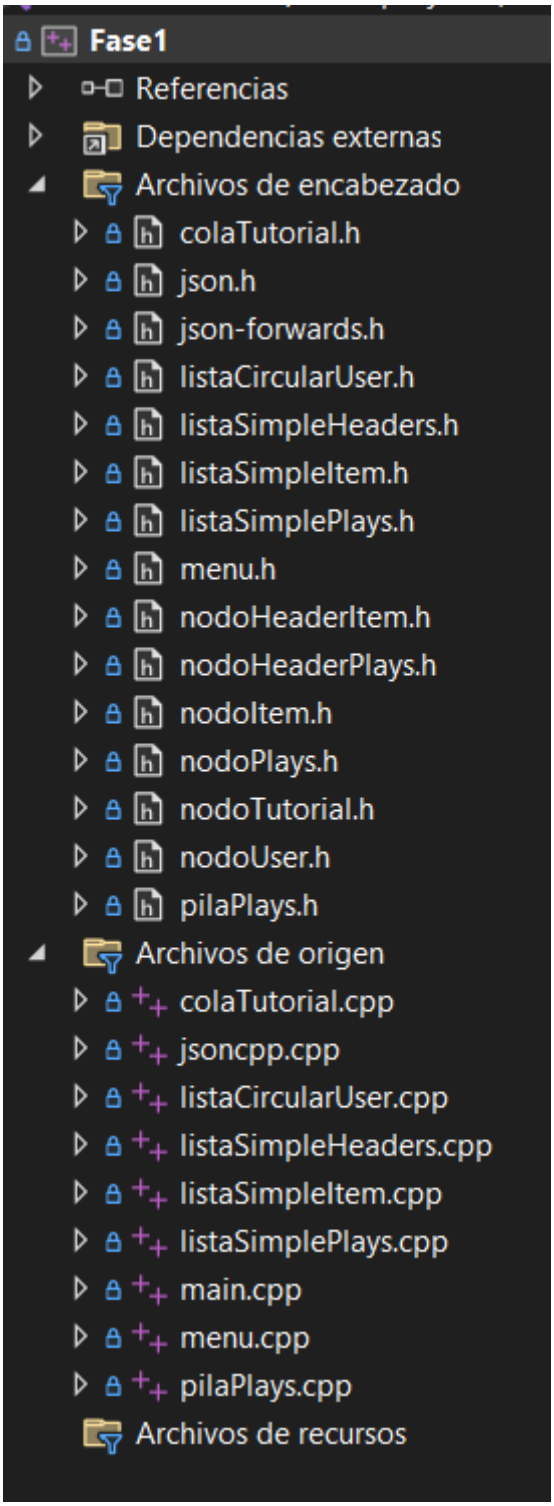
Requisitos Mínimos del Sistema

Sistema operativo 64 bits

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- macOS 10.5 o superior
- Linux GNOME o KDE desktop
- Procesador a 1.6 GHz o superior
- 1 GB (32 bits) o 2 GB (64 bits) de RAM (agregue 512 MB al host si se ejecuta en una máquina virtual)
- 3 GB de espacio disponible en el disco duro
- Disco duro de 5400 RPM
- Tarjeta de vídeo compatible con DirectX 9 con resolución de pantalla de 1024 x 768 o más.
- Navegador web (Recomendado: Google Chrome)

Estructura raíz:

El proyecto tiene la siguiente estructura de directorios:



DESCRIPSION DE LOS METODOS Y CLASES:

Los métodos y clases utilizados para este programa fueron pensados y diseñados específicamente para el desarrollo e implementación de este, siguiendo rigurosamente los requerimientos de diseño inicialmente propuestos en el manual de requerimientos.

La clase main:

Contiene el método principal de nuestra aplicación, esta clase se encarga de inicializar los objetos que usaremos dentro del programa a lo largo de toda la ejecución. Ayudando así a la persistencia de datos en cualquier área del programa.

Clase Menu:

Tiene como función generar el entorno sobre el cual estará situado el usuario, este entorno se desplegará en consola permitiendo una interfaz gráfica primitiva pero intuitiva. Dentro de la clase podremos encontrar, el menú principal, el menú secundario, y el menú de reportes.

Clase listaCircularUser:

Esta clase tiene como función de crear una lista doblemente enlazada, permitiendo al usuario escoger por medio del menú los siguiente métodos:

Carga masiva de usuarios por medio de “archivo.json”:

```
void listaCircularUser::loadFile(string ruta) {

    nodoUser* nuevo;
    ifstream ifs(ruta);
    Json::Reader reader;
    Json::Value obj;
    reader.parse(ifs, obj);
    const Json::Value& dataUser = obj["usuarios"];

    for (int i = 0; i < dataUser.size(); i++) {

        nuevo = new nodoUser();

        string coins = dataUser[i]["monedas"].asString();
        string age = dataUser[i]["edad"].asString();

        nuevo->setNick(dataUser[i]["nick"].asString());
        nuevo->setPassword(dataUser[i]["password"].asString());
        nuevo->setCoins(stoi(coins));
        nuevo->setAge(stoi(age));

        addToEnd(nuevo);
    }
}
```

En este método se usa la librería Jsoncpp por medio de la cual se recorre el archivo de entrada y por medio de un ciclo for se almacenan los datos en la clase nodoUser que tiene los atributos del usuario a guardar.

Añadir al final:

```
void listaCircularUser::addToEnd(nodoUser* user) {

    nodoUser* nuevo = new nodoUser();
    nuevo = user;

    if (primero == NULL) {
        primero = nuevo;
        ultimo = nuevo;
        primero->siguiente = primero;
        primero->atras = ultimo;
    }
    else {
        ultimo->siguiente = nuevo;
        nuevo->atras = ultimo;
        nuevo->siguiente = primero;
        ultimo = nuevo;
        primero->atras = ultimo;
    }
    cout << "\n>Usuario Registrado!\n\n";
}
```

En este método se utiliza la lógica de la programación orientada a objetos y la implementación de estructuras de datos por medio de lo cual ingresamos nodos a una lista que es capaz de señalar los nodos entre si y de esta forma permitir un mejor maneo de memoria.

Método de Loguin o inicio de sesión:

```
void listaCircularUser::login(listaCircularUser listaUsers,listaSimpleItem listIte
    nodoUser* actual = new nodoUser();
    actual = primero;
    bool encontrado = false;
    string aux;
    if (primero == NULL) {
        cout << "No hay usuarios registrados!\n\n";
    }
    else {
        cout << " > Ingrese el nombre del usuario: ";
        cin >> aux;
        do {
            if (actual->getNick() == aux) {
                cout << ">Usuario Encontrado!\n\n";
                encontrado = true;

                string pass;
                cout << " > Ingrese su Password: ";
                cin >> pass;

                if (pass == actual->getPassword() && aux == actual->getNick()) {
                    menu secMenu;
                    secMenu.sec_menu(listaUsers,actual,listItems,queue,stack);

                }
                else {
                    cout << ">password incorrecto!\n";
                    system("pause");
                }
            }
            actual = actual->siguiente;
        } while (actual != primero && encontrado != true);

        if (!encontrado) {
            cout << "\n>usuario no Encontrado\n\n";
        }
```

En este método se busca al objeto usuario en la lista por medio de su nombre y al encontrarlo se le permite ingresar su contraseña si esta coincide con la almacenada en el objeto entonces se procede a enviarlo al menú secundario de lo contrario se rechaza la acción y se vuelve al menú principal.

Método de editar información:

```
cout << "\n\n1. Modificar Nick" << endl;
cout << "2. Modificar Password" << endl;
cout << "3. Modificar Edad" << endl;
cout << "4. Guardar y Salir" << endl;

cout << "\nIngrese una opcion: " << endl;
cin >> opcion;

switch (opcion) {
case '1':
    cout << "Ingresa el nuevo Nick: ";
    cin >> aux;
    actual->setNick(aux);
    system("pause");
    break;

case '2':
    cout << "Ingresa el nuevo Password: ";
    cin >> aux;
    actual->setPassword(aux);
    system("pause");
    break;

case '3':
    cout << "Ingresa tu nueva Edad: ";
    cin >> aux;
    actual->setAge(aux2);
    system("pause");
    break;
```

En este método se le pide al usuario que ingrese a una opción por medio de un switch case y de esta forma permitimos elegir qué información desea modificar, luego por medio de la programación orientada a objetos asignamos al nodo actual los atributos que el usuario digita, de esta forma logramos renovar los datos viejos con los actuales.

Método de eliminar un usuario:

```
// si si ha confirmado se procede a eliminar
if (confirm == true && primero != NULL) {
    do {
        if (actual->getNick() == aux && actual->getPassword() == aux2) {

            if (actual == primero) {
                primero = primero->siguiente;
                primero->atras = ultimo;
                ultimo->siguiente = primero;
            }
            else if (actual == ultimo) {
                ultimo = anterior;
                ultimo->siguiente = primero;
                primero->atras = ultimo;
            }
            else {
                anterior->siguiente = actual->siguiente;
                actual->siguiente->atras = anterior;
            }
            cout << "\n>Usuario Eliminado!\n\n";

        }
        anterior = actual;
        actual = actual->siguiente;
    } while (actual != primero && encontrado != true);

    encontrado = true;
    return true;
```

En este método lo que se busca es que un nodo desaparezca de la estructura por lo cual, al momento de eliminarlo, hacemos que tanto el nodo anterior como el que le sigue al nodo actual, dejen de apuntar a el, y asignamos nuevas direcciones para los nodos anterior y siguiente enlazando estos entre sí, de esta forma el nodo actual queda sin apuntadores que lo referencien por lo cual queda sin propiedad de una clase y es eliminado de la memoria.

listaSimpleHeaders y listaSimpleItems:

Estas 2 clases en conjunto tiene la función de generar una lista de listas, para lo cual la listaSimpleHeader toma el papel de la lista de Encabezados, y la listaSimpleItems toma el papel del listado de ítems que pertenecen a cada encabezado. Para esto su utilizaron 2 metodos, el método de Insertar y el método de BuscarPrincipal, los cuales se describen seguido:

Método de inserción:

```
void listaSimpleHeaders::Insertar(nodoItem* item, string categoria) {
    if (primero == NULL) {
        nodoHeaderItem* nuevo = new nodoHeaderItem();
        nuevo->categoria = categoria;
        nuevo->inList.addToEnd(item);
        primero = nuevo;
    }
    else {
        nodoHeaderItem* busqueda = BuscarPrincipal(primero, categoria);
        nodoHeaderItem* nuevo = new nodoHeaderItem();
        nuevo->setCat(categoria);
        nuevo->inList.addToEnd(item);
        if (busqueda == NULL) {
            nodoHeaderItem* auxActual = primero;
            while (auxActual != NULL) {
                if (auxActual->siguiente == NULL) {
                    auxActual->siguiente = nuevo;
                    break;
                }
                auxActual = auxActual->siguiente;
            }
        }
        else {
            busqueda->inList.addToEnd(item);
        }
    }
}
```

En este método lo que se realiza es una inserción de un nodo en una lista, este nodo se insertara en la lista que contenga el nombre de su categoría, para lo cual se ingresa el nodo y su categoría, las listas de cabeceras se crean con el nombre de la categoría que entra, si la categoría existe a dicha categoría se le añade una lista de objetos y se asigna ahí el nodo, si la categoría no existe se crea una nueva con el nombre de la categoría entrante y luego se añade una lista y se asigna a esa lista el nodo.

Meodo de búsqueda de cabeceras:

```
nodoHeaderItem* listaSimpleHeaders::BuscarPrincipal(nodoHeaderItem* primeroL, string categoria) {
    if (primeroL == NULL) {
        return primeroL;
    }
    else {
        nodoHeaderItem* auxActual = primeroL;
        while (auxActual != NULL) {
            if (auxActual->getCat() == categoria) {
                break;
            }
            auxActual = auxActual->siguiente;
        }
        return auxActual;
    }
}
```

Este método se encargar de recorrer la lista de cabeceras y verificar si existe algún nodo cabecero con el nombre de la categoría entrante, si existe retorna ese nodo, si no existe, crea un nodo con el nombre de la categoría entrante y luego retorna ese nodo.

Carga masiva de datos:

En este método se usa la librería Jsoncpp por medio de la cual se recorre el archivo de entrada y por medio de un ciclo for se almacenan los datos en la clase nodoItem que tiene los atributos del usuario a guardar.

```
nodoItem* nuevo;
ifstream ifs(ruta);
Json::Reader reader;
Json::Value obj;
reader.parse(ifs, obj);
const Json::Value& dataArticle = obj["articulos"];
for (int i = 0; i < dataArticle.size(); i++) {

    nuevo = new nodoItem();

    string price = dataArticle[i]["precio"].asString();

    nuevo->setId(dataArticle[i]["id"].asString());
    nuevo->setCategory(dataArticle[i]["categoria"].asString());
    nuevo->setPrice(stoi(price));
    nuevo->setName(dataArticle[i]["nombre"].asString());
    nuevo->setSrc(dataArticle[i]["src"].asString());

    Insertar(nuevo, dataArticle[i]["categoria"].asString());
}
```

listaSimplePlays y pilaPlays:

Estas 2 clases en conjunto tiene la función de generar una lista de pilas, para lo cual la listaSimplePlays toma el papel de la lista de Encabezados, y la pilaPlays toma el papel del listado de jugadas que pertenecen a cada encabezado. Para esto su utilizaron 2 metodos, el método de Insertar y el método de BuscarPrincipal, los cuales se describen seguido:

```
void listaSimplePlays::Insertar(nodoPlays* plays, string categoria) {
    if (primero == NULL) {
        nodoHeaderPlays* nuevo = new nodoHeaderPlays();
        nuevo->categoria = categoria;
        nuevo->inStack.addToEnd(plays);
        primero = nuevo;
    }
    else {
        nodoHeaderPlays* busqueda = BuscarPrincipal(primero, categoria);
        nodoHeaderPlays* nuevo = new nodoHeaderPlays();
        nuevo->setCat(categoria);
        nuevo->inStack.addToEnd(plays);
        if (busqueda == NULL) {
            nodoHeaderPlays* auxActual = primero;
            while (auxActual != NULL) {
                if (auxActual->siguiente == NULL) {
                    auxActual->siguiente = nuevo;
                    break;
                }
                auxActual = auxActual->siguiente;
            }
        }
        else {
            busqueda->inStack.addToEnd(plays);
        }
    }
}
```

En este método lo que se realiza es una inserción de un nodo en una lista, este nodo se insertara en la lista que contenga el nombre de su categoría, para lo cual se ingresa el nodo y su categoría, las listas de cabeceras se crean con el nombre de la categoría que entra, si la categoría existe a dicha categoría se le añade una lista de objetos y se asigna ahí el nodo, si la categoría no existe se crea una nueva con el nombre de la categoría entrante y luego se añade una lista y se asigna a esa lista el nodo.

Meodo de búsqueda de cabeceras:

```
nodoHeaderPlays* listaSimplePlays::BuscarPrincipal(nodoHeaderPlays* primeroL, string categoria) {
    if (primeroL == NULL) {
        return primeroL;
    }
    else {
        nodoHeaderPlays* auxActual = primeroL;
        while (auxActual != NULL) {
            if (auxActual->getCat() == categoria) {
                break;
            }
            auxActual = auxActual->siguiente;
        }
        return auxActual;
    }
}
```

Este método se encargar de recorrer la lista de cabeceras y verificar si existe algún nodo cabecero con el nombre de la categoría entrante, si existe retorna ese nodo, si no existe, crea un nodo con el nombre de la categoría entrante y luego retorna ese nodo.

El método jugar o doGameplays:

```
cout << "\n\n>Realizar Movimientos" << endl;
cout << "1. Ingresar Jugada" << endl;
cout << "2. Guardar y Salir" << endl;

cout << "\nIngrese una opcion: " << endl;
cin >> opcion;

coins = nuevoUser->getCoins();
int posX = 0;
int posY = 0;

switch (opcion) {

case '1':

    nuevo = new nodoPlays();
    cout << "Coordenada X: ";
    cin >> posX;
    cout << "Coordenada Y: ";
    cin >> posY;
    nuevo->setX(posX);
    nuevo->setY(posY);

    cout << ">Ingreso la coordenada: (" << posX << "," << posY << ")" << endl;

    Insertar(nuevo,nuevoUser->getNick());

    coins += 1;
    nuevoUser->setCoins(coins);

    cout << " > Tokens: +1" << endl;

    system("pause");
    break;
```

Este método de interacción con el usuario genera una pequeña interfaz donde el usuario puede ingresar coordenadas las cuales serán almacenadas en una pila y dicha pila en la categoría perteneciente al usuario que las esta realizando, en este método también se le añaden puntos al usuario lo cuales son traducidos en monedas las cuales pueden ser utilizadas posteriormente en la tienda.

ColaTutorial:

Esta clase se encarga de generar una cola por medio de la clase nodo tutorial el cual es el nodo que se aniade a la cola.

Método de inserción:

```
nodoTutorial* nuevo = new nodoTutorial();
nuevo = tuto;

if (primero == NULL) {
    primero = nuevo;
    primero->siguiente = NULL;
    ultimo = primero;
}
else {
    ultimo->siguiente = nuevo;
    nuevo->siguiente = NULL;
    ultimo = nuevo;
}

cout << "\n>Nodo Ingresado!\n\n";
```

En el método de inserción de una cola únicamente verificamos que este vacía y si lo esta ingresamos el nodo entrante como primero y como ultimo a la vez, de lo contrario el nodo que entra es el nodo siguiente del anterior y es a la vez el último, la estructura es similar a una lista enlazada simple.

Método de carga masiva:

```
nodoTutorial* nuevo;
ifstream ifs(ruta);
Json::Reader reader;
Json::Value obj;
Json::Value attrib;
reader.parse(ifs, obj);
const Json::Value& dataTable = obj["tutorial"];

nuevo = new nodoTutorial();

string Alto = dataTable["alto"].asString();
string Ancho = dataTable["ancho"].asString();

cout << "(" << Alto << " <-> " << Ancho << ")" << endl;

nuevo->setAlto(stoi(Alto));
nuevo->setAncho(stoi(Ancho));

addToEnd(nuevo);

for (int i = 0; i < dataTable["movimientos"].size(); i++) {

    nuevo = new nodoTutorial();

    string posX = dataTable["movimientos"][i]["x"].asString();
    string posY = dataTable["movimientos"][i]["y"].asString();

    cout << "(" << posX << "," << posY << ")" << endl;

    nuevo->setX(stoi(posX));
    nuevo->setY(stoi(posY));

    addToEnd(nuevo);
}
```

En este método se usa la librería Jsoncpp por medio de la cual se recorre el archivo de entrada y por medio de un ciclo for se almacenan los datos en la clase nodoTutorial que tiene los atributos del Tutorial a guardar.

Método general de impresión en consola:

```
void listaCircularUser::showList() {
    nodoUser* actual = new nodoUser();
    actual = primero;

    cout << ">Listado de Usuarios: " << endl;

    if (primero != NULL) {
        do {
            cout << " Nombre: " << actual->getNick() << "\tPassword: " << actual->getPassword() << \
                "\tAge: " << actual->getAge() << "\tCoins: " << actual->getCoins() << "\n";
            actual = actual->siguiente;
        } while (actual != primero);
    }
    else {
        cout << "\n>La lista se Encuentra Vacia\n\n";
    }
}
```

Este es un método simple únicamente recorremos la lista de inicio a fin y como es circular se detiene cuando vuelve a comenzar, imprimiendo la información que contiene cada nodo.

Método general de ordenamiento de datos - burbuja:

En este método se aplicó la técnica del ordenamiento burbuja por lo cual se introdujeron 2 nodos, pivote y actual, por medio de los cuales se intercambió la información que contenga cada uno dependiendo si sus valores son o no mayores o menores, de este método se utilizaron 2 para poder generar el orden ascendente y descendente, algo importante para aclarar es que no se intercambiaron posiciones en memoria sino solamente los datos.

```
pivote = listaUser.primero;
while (pivote != listaUser.ultimo) {
    actual = pivote->siguiente;
    while (actual != listaUser.primero) {
        if (pivote->getAge() > actual->getAge()) {

            nick = pivote->getNick();
            password = pivote->getPassword();
            age = pivote->getAge();
            coins = pivote->getCoins();

            pivote->setNick(actual->getNick());
            pivote->setPassword(actual->getPassword());
            pivote->setAge(actual->getAge());
            pivote->setCoins(actual->getCoins());

            actual->setNick(nick);
            actual->setPassword(password);
            actual->setAge(age);
            actual->setCoins(coins);

        }
        actual = actual->siguiente;
    }
    pivote = pivote->siguiente;
}
```

Método general de impresión de grafos – Graphviz:

```
void listaCircularUser::doGraphics() {
    string dot = "";

    dot += "digraph G {\n";
    dot += +"label=\"Estructura: Lista Circular\";\n";
    dot += +"node [shape=box];\n";

    nodoUser* aux = primero;
        do {
            dot += "N" + (aux->getNick()) + "[label=\"" + (aux->getNick()) + "\"];\n";
            aux = aux->siguiente;
        } while (aux != primero);


        dot += "{rank=same;\n";
        aux = primero;
        do {
            dot += "N" + (aux->getNick());
            dot += "->";
            aux = aux->siguiente;

        } while (aux != primero);
```

```
    ultimo = primero;
    do {
        dot += "N" + (aux->getNick());
        dot += "->";
        aux = aux->atras;

    } while (aux != ultimo);
    dot += "N" + (aux->getNick());
    dot += "}\n";


dot += "}\n";

cout << dot;

ofstream file;
file.open("listaCiruclar.dot");
file << dot;
file.close();

system(("dot -Tpng listaCiruclar.dot -o  listaCiruclar.png"));

system(("listaCiruclar.png"));
```

En este método se recorren los datos de la lista, o de la pila y se añaden como texto a un archivo.dot el cual interpretara graphviz para genera una imagen en formato png. En la primera parte debemos recorrer los nodos principales o cabeceras si se tienen, de esta forma enlazamos cada uno de ellos con los apuntadores necesarios, esto lo hacemos por medio de {rank=same}, después de esto agregamos los nodos dependientes de las cabeceras si los hubiera con el método {rank=none}, después de esto únicamente queda a discreción añadir creatividad a la forma en la que se imprime.