Universidad de San Carlos de Guatemala
Facultad de ingeniería
Ingeniería en Ciencias y Sistemas
ESTRUCTURA DE DATOS

MANUAL TECNICO

Alberto Josue Hernández Armas 201903553

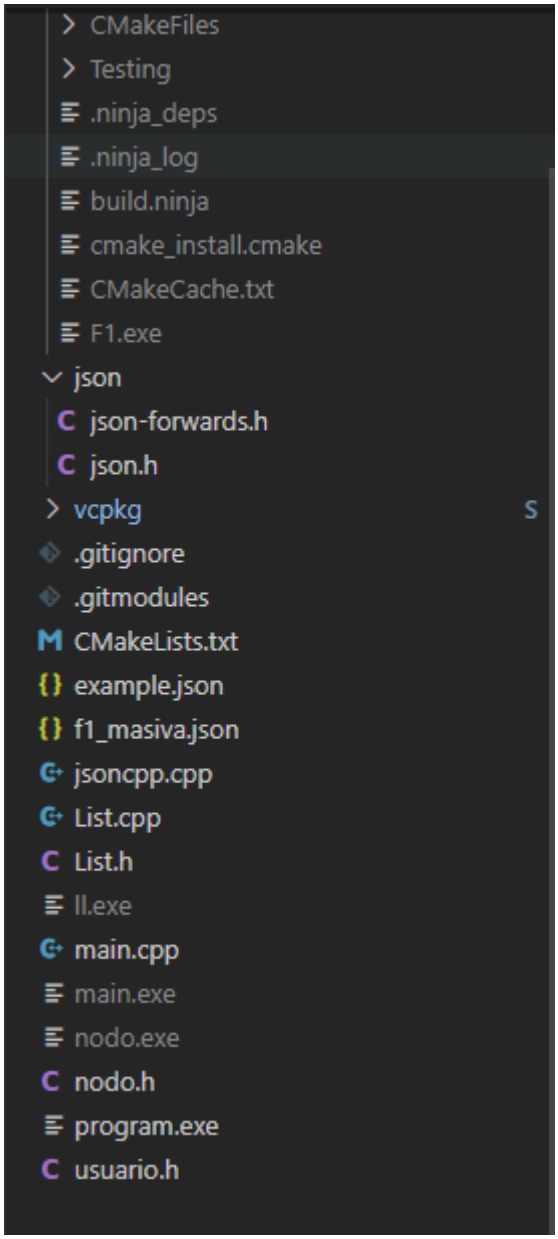
Guatemala 24 de Agosto de 2022

INTRODUCCION

En el presente manual técnico podremos observar cómo es que está diseñado el código y explicando que forman la funcionalidad de una manera general, teniendo como objetivo principal que sea más entendible. Determinando los métodos utilizados y explicando algunas palabras claves refiriéndonos a sus propiedades demostrando cuál es su función dentro de los bloques de código que se nos presentan a continuación.

MANUAL TECNICO

Se cuenta con las siguiente clases y objetos para la realización correcta de este proyecto



Todas las estructuras se implementan dentro del main, siendo estas las listas enlazaas, matrices, colas y pilas:

```
bool bandera = true;
while (bandera)
{
 int entrada = 0;
cout<<"1. Carga masiva" << endl;
cout<<"2. Registrar Usuario" << endl;
cout<<"3. Login" << endl;
cout<<"4. Reportes" << endl;
cout<<"5. Salida" << endl;
cin >> entrada;
switch (entrada)
{
case 1:
    cout<<"1. " << endl;
    carga_usuario();

    cout << &auxiliar_usuarios <<"||||||"<< &usuarios_glob<<endl;

    break;
case 2:
    cout<<"2. " << endl;
    añadir_usuario();

    break;
case 3:
    cout<<"3. " << endl;
    opcion3();
    menu2();

    break;
case 4:
    cout<<"4. " << endl;
    auxiliar_usuarios = usuarios_glob;
    auxiliar_articulos = articulos_glob;
```

Podemos observar como se construye la lista enlazada y sus diferentes implementaciones:

```cpp
// Linked List CPP
#include "List.h"
//#include "nodo.h"
#include "json/json.h"
#include<iostream>
#include<string>

using namespace std;
//HOla mundo desde vscode
template<typename G>
 List<G>::List()
{
    tamaño = 0;
    head = NULL;
    tail = NULL;
    j = "estoy vivo otra vez";
}
template<typename G>
bool List<G>::isEmpty()
    {
        return head == NULL;
    }


template<typename G>
void List<G>::insert(G x)
{
        temp = new Node<G>;
        temp->data = x;
        if(isEmpty())
        {
            temp->next = NULL;
            temp->Prev = NULL;
```

```cpp
template<typename G>
void List<G>::insert(G x)
{
        temp = new Node<G>;
        temp->data = x;
        if(isEmpty())
        {
            temp->next = NULL;
            temp->Prev = NULL;
            head = temp;

        }
        else
        {
            Node<G> *ahora;
            ahora = head;
            while (ahora->next != NULL)
            {
                ahora = ahora->next;

            }
            ahora->next =temp;
            temp->Prev = ahora;
            temp-> next = NULL;
        }
        tamaño= tamaño+1;


    }
```

List.h > List<G> > tail

```cpp
// Linked List CPP
#pragma once
#include"nodo.h"
#include<iostream>

using namespace std;
//HOla mundo desde vscode
template<typename G>
class List
{
    public:
    Node<G> *head;
    Node<G> *tail;
    Node<G> *temp;
    bool isEmpty();


    List();
    string j;
    void insert(G x);
    void insertAtEnd(G x);
    void remove(G x);
    void find(G x);
    void display();
    int tamaño;
};

```

La estructura del nodo es la siguiente:

List.cpp nodo.h

nodo.h > Node<T> > data

```cpp
// Linked List CPP
#pragma once
#include "List.h"
#include<iostream>

using namespace std;
//HOla mundo desde vscode
template <typename T>
class Node
{
public:

    T data;
    Node<T> *next;
    Node<T> *Prev;
    Node<T> *derecha;
    Node<T> *inventario;
    Node<T> *movimientos;

};
```

Y su implementación es la siguiente:

```
            temp->Prev = ahora;
            temp-> next = NULL;
        }
        tamaño= tamaño+1;


    }


template<typename G>
    void List<G>::display()
    {
        if(!isEmpty())
        {
            for(temp = head; temp != NULL; temp=temp->nex
                cout << temp->data << " ";
            cout << endl;
        }
        else
        {
            cout << "LISTA VACIA" << endl;
        }
    }
    template class List<string>;
    template class Node<string>;
    template class List<Json::Value>;
    template class Node<Json::Value>;
```

Dentro de List cpp

Se trabajo de manera de template, para que la instancia del valor que se le daba a los objetos proviniera desde el main, y se fuera heredando sucesivamente hasta llegar al nodo, con la idea de ahorrar código, y hacer la menor cantidad de instancias posibles. También para la reutilización de métodos haciéndolos abstractos de cierta manera

Lista de listas:

```
> void eliminar_cuenta(Node<Json::Value> *cuenta_eliminar)···
> void añadir_usuario()···
> void va(Node <Json::Value> *root, Json::Value entrada)···
  void llenaderecha(Node<Json::Value> *n,Json::Value x)
  {
      if (n->derecha == NULL)
      {
          Node<Json::Value> *nuevo = new Node<Json::Value>;
          nuevo -> data = x;
          nuevo -> derecha = NULL;
          n->derecha = nuevo;

      }
      else
      {
          llenaderecha(n->derecha,x);
      }
  }
```

Se trabajo con una lista normal, con la diferencia que tiene la opción de llenado hacia la derecha. Formando asi una especie de matriz.

COLA TUTORIAL:

```
}

Json::Value dequeue_tutorial()
{
    Node<Json::Value> *n = tut_global.head;
    tut_global.head = tut_global.head->next;
    tut_global.tamaño = tut_global.tamaño-1;
    return n->data;



}
```

Se trabajo como una lista LIFO en la cual solo se mostrara una vez, vaciando la lista indicada.

```
Node<Json::Value> *cuenta;
```

```
Node<Json::Value> *login(List<Json::Value> usuarioh, string nombre, string contraseña)
{
    cout << "Estas dentro de la funcion de LOGIN :"<< endl;
    Node<Json::Value> *ahora;

        for(ahora = usuarioh.head; ahora != NULL; ahora=ahora->next)
        {
            cout << ahora->data << " \n";
            if (ahora->data["nick"].asString() == nombre && ahora->data["password"].asString()== contrase
            {cout << "enconttro a betebetoven"<< endl;
            return ahora;
            break;
            }


        }
        return new Node<Json::Value>;

}
```

El login se trabaja recorriendo la lista de usuarios, hasta encontrar el usuario indicado, haciendo la referencia desde el apuntador cuenta hacia la dirección de memoria donde se encuentra el nodo deseado.

Cada uno de los nodos usuarios tiene una lista también , en la cual se encuentran los movimientos del jugador

La cual se interpreta como una lista de pilas, ya que se van apilando los movimientos.

```
void jugar(string x, string y)
{
    Node<Json::Value> *n = cuenta;
    if(cuenta->movimientos == NULL)
    {

    n->movimientos = new Node <Json::Value>;//el nodo movimientos es el head de la lista de movi
    n->movimientos->next = NULL;
    }
    Node<Json::Value> *t = n->movimientos;
    while (t->next != NULL)
    t = t->next;

    string jeson = "{\"x\": \""+x+"\",\"y\": \""+y+"\"}";
    Json::Value actualJson;
    Json::Reader reader;
    reader.parse(jeson,actualJson);
    Node<Json::Value> *nv = new Node<Json::Value>;
    nv->data = actualJson;
    t->next = nv;
    nv->next = NULL;

}
```

Cada usuario tiene su propia lista de movimientos, por lo que la lista de usuarios se convierte en una lista de lsitas.