FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Manual técnico
“Battleship”
Fase 1

Edwin Estuardo Reyes Reyes

201709015

agosto de 2022

Tabla de contenido

Acerca del Programa

Objetivos

Objetivos general

- Aplicar los conocimientos del curso Estructuras de Datos en el desarrollo de una aplicación que permita manipular la información de forma óptima.

Objetivos específicos

- Demostrar los conocimientos adquiridos sobre estructuras de datos lineales poniéndolos en práctica en el desarrollo del juego batalla naval.
- Utilizar el lenguaje C++ para implementar estructuras de datos lineales
- Utilizar la herramienta Graphviz para graficar estructuras de datos lineales.
- Definir e implementar algoritmos de búsqueda, recorrido y eliminación.

Descripcion del programa

La aplicación consiste en la elaboración de un videojuego que estará dividio en 3 partes que estará desarrollado en el lenguaje de c++ para el apartado de consola y recoleccione de datos y creación de listas enlazadas mientras que el lengauje Python se encargara del tema de gráficos y jugabilidad del video juego.

El videojuego consiste en un batalla naval (Battleship) en las etapas tempranas del programa funciona mediante consola y es seleccionando las coordenadas X

y Y donde se desea realizar el ataque, luego de realizar x numero de jugadas se determina si es considerado un triunfo o una derrota.

Requisitos e instalación para correcto funcionamiento

Requisitos

Requisitos Minimos

- Requiere un procesador y un sistema operativo de 64 bits
- Sistema Operativo: Ubuntu 22.04 LTS
- Procesador: Intel Pentium E2180 @ 2.00GHz / AMD Turion 64 X2 Mobile TL-64
- Memoria: 1 Gigas de RAM
- Gráficos GeForce 7600 GT / Radeon HD 7310G / Intel Graphics
- DirectX: Version 10
- Almacenamiento 1 mb

Requisitos Recomendados

- Requiere un procesador y un sistema operativo de 64 bits
- Sistema Operativo: Ubuntu 22.04 LTS
- Intel Core i5-10600K
- Memoria: 16 gigas de Ram
- GeForce RTX 2080, 8 Gigas
- DirectX: Version 11
- Almacenamiento: 10 mb

Codigo Fuente

Librerias Usadas

```
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <string>
#include <string.h>
#include "recursos/sha256.h"
#include <jsoncpp/json/json.h>
```

Estructuras de tipos Usadas

```
struct Usuario{
    string nick;
    string password;
    string edad;
    string moneda;
};

struct Articulo{
    string id;
    string categoria;
    string precio;
    string nombre;
    string src;
};

struct Movimiento{
    string mov_x;
    string mov_y;
    string ancho;
    string alto;
};
```

Clase Lista

Clase utilizada para la elaboración de listas simples utilizadas para la elaboración de la estructura de datos: lista de listas.

```
class Lista{
    private:
        class Nodo{
            public:
                Nodo *siguiente;
                Articulo articulo;
                Lista *lista;
                string categoria;
                Usuario us;

        };
        Nodo *raiz;

    public:
        Lista();
        ~Lista();
        void insertarNodo(Articulo articulo);
        void insertar(Articulo articulo);
        bool vacia();
        void insertarUsuario(Usuario us);
        void ordenar();
        void ordenarArticulos();
        void imprimir();
};
```

Método Ordenar()

Método usado para usado para el ordenamiento y despliegue de la lista de usuarios ordenados por edades de forma ascendente y descendente

```
void Lista::ordenar(){
    cout<<"Usuarios ordenados por edad ascendentes"<<endl;
    Nodo *actual, *siguiente, *reco;
    actual = raiz;
    reco = raiz;
    Usuario t;
    int r;
    //ordenamiento burbuja ascendente
    while(actual->siguiente != NULL)
```

```
     {
          siguiente = actual->siguiente;

          while(siguiente!=NULL)
          {
               if(stoi(actual->us.edad) > stoi(siguiente->us.edad))
               {
                    t = siguiente->us;
                    siguiente->us = actual->us;
                    actual->us = t;
               }
               siguiente = siguiente->siguiente;
          }
          actual = actual->siguiente;
          siguiente = actual->siguiente;O
     }
     while(reco != NULL){
        cout<<"edad:"<<reco->us.edad<<"   usuario: "<<reco-
>us.nick<<endl;
        reco = reco->siguiente;
     }cout<<endl;
    //ordenamiento burbuja descendente
     cout<<"Usuarios ordenados por edad descendentes"<<endl;

     actual = raiz;

      while(actual->siguiente != NULL)
     {
          siguiente = actual->siguiente;

          while(siguiente!=NULL)
          {
               if(stoi(actual->us.edad) < stoi(siguiente->us.edad))
               {
                    t = siguiente->us;
                    siguiente->us = actual->us;
                    actual->us = t;
               }
               siguiente = siguiente->siguiente;
          }
          actual = actual->siguiente;
          siguiente = actual->siguiente;

     }
     reco = raiz;
     while(reco != NULL){
        cout<<"edad:"<<reco->us.edad<<"   usuario: "<<reco-
>us.nick<<endl;
        reco = reco->siguiente;
```

```
    }}
```

Método OrdenarArticulos()

Método usado para la recolección de todos los elementos que se encuentran en la lista de listas de artículos para agruparlos en una sola lista simple para posterior realizar el ordenamiento de los elementos en la lista por el precio de artículo de forma ascendente y descendente.

```
void Lista::ordenarArticulos(){
    Lista *listaSimple = new Lista();
    if (!vacia()) {
        Nodo *reco = raiz;
        do {
            Nodo *aux = reco->lista->raiz;

            while(aux != NULL ){//AQUI VA A RECOGER TODOS LOS ELEMENTOS
DE LA LISTA DE ARTICULOS Y LO COLOCARA EN UNA SIMPLE
                listaSimple->insertarNodo(aux->articulo);
                aux = aux->siguiente;
            }
            reco = reco->siguiente;
        } while (reco->siguiente != NULL);
    }
    else{
        cout<<"No existen elementos en la lista"<<endl;

    }
   //ahora teniendo todos los elementos en la lista los vamos a aordenar
 cout<<"Articulos ordenados por precio ascendentes"<<endl;
    Nodo *actual, *siguiente, *reco;
    actual = listaSimple->raiz;
    reco = listaSimple->raiz;
    Articulo t;
    int r;

    //ordenamiento burbuja ascendente
    while(actual->siguiente != NULL)
     {
          siguiente = actual->siguiente;

          while(siguiente!=NULL)
          {
      if(stoi(actual->articulo.precio)>stoi(siguiente>articulo.precio))
                {
                     t = siguiente->articulo;
                     siguiente->articulo = actual->articulo;
                     actual->articulo = t;
                }
```

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

        }
        while(reco != NULL){
           cout<<"Preio:   Q"<<reco->articulo.precio<<"   Articulo: "<<reco-
>articulo.nombre<<endl;
           reco = reco->siguiente;
        }cout<<endl;
       //ordenamiento burbuja descendente
        cout<<"Articulos ordenados por precio descendentes"<<endl;

        actual = listaSimple->raiz;

         while(actual->siguiente != NULL)
        {
            siguiente = actual->siguiente;

            while(siguiente!=NULL)
            {
                if(stoi(actual->articulo.precio) < stoi(siguiente-
>articulo.precio))
                {
                    t = siguiente->articulo;
                    siguiente->articulo = actual->articulo;
                    actual->articulo = t;
                }
                siguiente = siguiente->siguiente;
            }
            actual = actual->siguiente;
            siguiente = actual->siguiente;

        }
        reco = listaSimple->raiz;
        while(reco != NULL){
           cout<<"Precio:  Q"<<reco->articulo.precio<<"   Articulo: "<<reco-
>articulo.nombre<<endl;
           reco = reco->siguiente;
        }
}
```

Método insertarUsuario()

Método utilizado para la inserción de los Usuarios en la lista circular doble para posteriormente ser usado para el ordenamiento.

```
void Lista::insertarUsuario(Usuario us){
    Nodo *nuevo = new Nodo();
    nuevo->us = us;
    if (raiz == NULL){
        raiz = nuevo;
    }
    else{
        Nodo *reco = raiz;

        while (reco->siguiente != NULL){
            reco = reco->siguiente;
        }
        reco->siguiente = nuevo;
        nuevo->siguiente = NULL;
        nuevo->us = us;
    }
}
```

Método insertarNodo()

Método utilizado para la inserción de nodos en las listas de los encabezados en donde almacenan en su interior el articulo que pertenece a esa lista.

```
void Lista::insertarNodo(Articulo articulo) {
    Nodo *nuevo = new Nodo();
    nuevo->articulo = articulo;
    if (raiz == NULL){
        raiz = nuevo;
        nuevo->articulo = articulo;
        nuevo->siguiente = NULL; }
    else{
        Nodo *reco = raiz;
        while (reco->siguiente != NULL){
            reco = reco->siguiente;}
        reco->siguiente = nuevo;
        nuevo->siguiente = NULL;
        nuevo->articulo = articulo;
    }
```

```
    }
```

Método insertar()

Método usado para crear la lista de listas primero se comienza verificando si en la lista se encuentra vacía o ya hay elementos existiendo en ella previamente, de estar vacía se procede a agregar a raíz el nombre de la categoría que se ingreso luego se crea una nueva lista que será la lista que contendrá los elementos de los artículos existentes en ella, de existir elementos se procede a buscar la existencia de alguna cabecera con el nombre de la categoría del articulo que se va a ingresar de encontrarlo se accede a la lista de este cabecero y se agrega dicho elemento de lo contrario se creara una nueva cabezal con dicho categoría y se crea una nueva lista para ingresar el articulo previo.

```
void Lista::insertar(Articulo articulo) {
    Nodo *nuevo = new Nodo();
    Nodo *reco;
    if (raiz == NULL){
        raiz = nuevo;
        raiz->categoria=articulo.categoria;
        Lista *lis = new Lista();
        nuevo->lista = lis;
        nuevo->siguiente = NULL;
        nuevo->lista->insertarNodo(articulo);
    }
    else{
        bool encontrado = false;
        reco = raiz; while (reco != NULL){
        if (reco->categoria == articulo.categoria){
            encontrado = true;
        }
        reco = reco->siguiente;
        }
        if (encontrado == true){
            Lista *lisaux = new Lista();
            reco = raiz;
            while (reco != NULL){
            if (reco->categoria == articulo.categoria){
            lisaux = reco->lista;
            }
            reco = reco->siguiente;
        }
        lisaux->insertarNodo(articulo);
        }
        else{
            Lista *lis = new Lista();
            Nodo *reco = raiz;
```

```
            while (reco->siguiente != NULL){
                reco = reco->siguiente;
            }
            reco->siguiente = nuevo;
            nuevo->lista = lis;
            nuevo->categoria = articulo.categoria;
            lis->insertarNodo(articulo);
        }
    }
}
```

Método Imprimir()

Método usado para mostrar en pantalla todos los artículos existente en la lista de listas, para lograrlo se procede a recorrer la lista cabecera y luego acceder a cada lista dentro de cada cabezal al haber ingresado se procede a desplegar en pantalla la información completa del artículo en cuestión.

```
void Lista::imprimir()
{
    if (!vacia()) {
        Nodo *reco = raiz;
        int numero = 1;
        cout<<"#     ID                                                CATEGORIA
                                        NOMBRE
 PRECIO"<<endl;
        do {
            Nodo *aux = reco->lista->raiz;

            while(aux != NULL ){
                int i=40;
                int nu=5;

                  int cat = 48;
                    int nom = 36;

                cout<<numero<<":";

                if(numero <10){
                for(int e = 0; e<4;e++){
                    cout<<" ";
                }}
                if(numero>9 and numero <100){
                    for(int e = 0; e<3;e++){
                    cout<<" ";
```

```
                }}
                if(numero>99){
                     for(int e = 0; e<2;e++){
                    cout<<" ";
                }
                }



                cout<<aux->articulo.id;
                i = i - aux->articulo.id.length();
                for(int e = 0; e<i;e++){
                    cout<<" ";
                }
                cout<<aux->articulo.categoria;
                cat = cat - aux->articulo.categoria.length();
                for(int e = 0; e<cat;e++){
                    cout<<" ";
                }
                cout<<aux->articulo.nombre;
                nom = nom - aux->articulo.nombre.length();
                for(int e = 0; e<nom;e++){
                    cout<<" ";
                }
                cout<<"Q."<<aux->articulo.precio<<endl;



                numero = numero + 1;
                aux = aux->siguiente;
            }
            reco = reco->siguiente;
        } while (reco->siguiente != NULL);
    }
    else{
        cout<<"No existen elementos en la lista"<<endl;

    }
}
```