**LABORATORIO NO. 05**

“Estructuras de datos lineales”

**OBJETIVOS**

* Aplicar los conocimientos sobre punteros y enlace entre nodos para la implementación de una estructura de datos lineal (lista).
* Entender la importancia de separar la estructura de datos (y su funcionamiento interno) del uso particular en una solución.

**QUIZZ**

Instrucciones: Señale la respuesta que considere correcta y brinde su razonamiento.

1. Las eliminaciones de la cola se realizan al principio de la lista (por el frente) es decir el primero en ingresar es el primero en salir.

a. Falso b. Verdadero

1. El límite para la asignación dinámica puede ser tan grande como la cantidad de memoria física de su computadora

a. Falso b. Verdadero

1. Las características de la pila y la cola se basan en la cantidad de elementos a insertar y eliminar de las estructuras lineales.

a. Falso b. Verdadero

**MARCO TEÓRICO (referencias)**

**Listas:**

<https://www.geeksforgeeks.org/data-structures/linked-list/>

**Colas:**

[https://es.wikipedia.org/wiki/Cola\_(inform%C3%A1tica)](https://es.wikipedia.org/wiki/Cola_(informática))

**Pilas y Colas**

<http://www.calcifer.org/documentos/librognome/glib-lists-queues.html>

**PRÁCTICA**

Ejercicio No. 01

**Instrucciones**: Implemente (*.cpp*) la clase **List** para una lista simplemente enlazada de enteros (estructura **Node** con único apuntador *next*) con base la definición indicada (*.h*). Así mismo genere una interfaz gráfica (WinForms) para validar el funcionamiento adecuado de cada una de las rutinas y visualización de la lista que está siendo manipulada.

**Importante NO modificar el .h**

**List.h**

#pragma once

/// <summary>

/// Colección de objetos que puede ser accesible por posición/índice.

/// </summary>

class List

{

/// <summary>

/// Nodo simplemente enlazado

/// </summary>

struct Node

{

/// <summary>

/// Información a almacenar

/// </summary>

int data;

/// <summary>

/// Apuntador hacia siguiente Nodo

/// </summary>

Node\* next;

};

private:

/// <summary>

/// Apuntador al primer Nodo dentro de la lista (cabeza)

/// </summary>

Node\* header = nullptr;

public:

/// <summary>

/// Agrega un elemento al FINAL de la lista

/// </summary>

/// <param name="item">Elemento a agregar</param>

void Add(int item);

/// <summary>

/// Elimina todos los elementos de la Lista.

/// </summary>

void Clear();

/// <summary>

/// Cuenta la cantidad de elementos en la lista

/// </summary>

/// <returns>Cantidad de elementos en la lista</returns>

int Count();

/// <summary>

/// Verifica si el elemento indicado se encuentra en la lista

/// </summary>

/// <param name="item">Elemento a verificar</param>

/// <returns>Verdadero si el elemento está en la lista, Falso en caso contrario</returns>

bool Contains(int item);

/// <summary>

/// Obtiene el índice de la primera ocurrencia del elemento indicado dentro de la lista

/// </summary>

/// <param name="item">Elemento a verificar</param>

/// <returns>El índice (base 0) de la primera ocurrencia si lo encontró, -1 en caso contrario</returns>

int IndexOf(int item);

/// <summary>

/// Inserta un elemento en una posición específica de la lista

/// </summary>

/// <param name="index">Posición/Índice (base 0) dentro de la lista</param>

/// <param name="item">Elemento a insertar</param>

/// <exception cref="System.ArgumentOutOfRangeException">index menor a 0 o index mayor a Count</exception>

void Insert(int index, int item);

/// <summary>

/// Obtiene el valor del elemento en la posición indicada de la lista

/// </summary>

/// <param name="index">Posición/Índice (base 0) dentro de la lista</param>

/// <returns>Elemento en posición indicada</returns>

/// <exception cref="System.ArgumentOutOfRangeException">index menor a 0 o index mayor o igual a Count</exception>

int GetItem(int index);

/// <summary>

/// Cambia el elemento en la posición indicada de la lista

/// </summary>

/// <param name="index">Posición/Índice (base 0) dentro de la lista</param>

/// <param name="item">Elemento a actualizar</param>

/// <exception cref="System.ArgumentOutOfRangeException">index menor a 0 o index mayor o igual a Count</exception>

void SetItem(int index, int item);

/// <summary>

/// Obtiene el índice de la última ocurrencia del elemento indicado dentro de la lista

/// </summary>

/// <param name="item">Elemento a verificar</param>

/// <returns>El índice (base 0) de la última ocurrencia si lo encontró, -1 en caso contrario</returns>

int LastIndexOf(int item);

/// <summary>

/// Elimina la primera ocurrencia del elemento en la lista

/// </summary>

/// <param name="item">Elemento a eliminar</param>

/// <returns> Verdadero si el elemento fue removido, Falso si no fue encontrado</returns>

bool Remove(int item);

/// <summary>

/// Elimina el elemento en la posición de la lista indicada

/// </summary>

/// <param name="index">Posición/Índice (base 0) dentro de la lista</param>

/// <exception cref="System.ArgumentOutOfRangeException">index menor a 0 o index mayor o igual a Count</exception>

void RemoveAt(int index);

};

Ejercicio No. 02

Solitario Simple:

Debe generar la interacción del solitario simple [1], el cual consiste en una baraja de naipes de la cual se reparten 7 grupos de forma incremental, es decir, se reparten 7, luego 6, luego 5, luego 4, 3, 2, 1; quedando de forma escalonada.

A continuación, se muestra la carta superior de cada uno de los 7 grupos.

El jugador, indica de que grupo tomará a partir de que carta y a que grupo la agregará, por lo que tendrá que verificar las reglas del juego, las cuales son:

* La secuencia de cartas debe ir de mayor a menor, con incremento de una,
* Deben ser de colores alternos, no importa el símbolo.

Si el jugador no tiene cartas que bajar, puede pedir que el mazo muestre una para así tomarla y agregarla a uno de los grupos según las reglas anteriores.

Tomar en consideración que el mazo puede sacar todas las cartas que se le soliciten, hasta terminar con las cartas disponibles.

Si el jugador toma una carta del mazo, el mazo debe mostrar la carta que estaba anterior a ella en la pila de cartas.

Por defecto, el jugador gana al poner por lo menos 5 cartas en orden en alguno de los grupos, sin embargo este parámetro se puede configurar en 3 niveles: fácil (3 cartas), medio (5 cartas) y difícil (7 cartas).

**Solución esperada**

Realice una lista que represente un mazo de cartas, las cuales tendrán la información de color, número. No se tomará en cuenta el símbolo. Debe llenar la lista con las 52 cartas y luego de forma aleatoria, agregarlas a una pila que representará el mazo.

Si el jugador presiona R, debe realizar la acción de repartir las cartas en los 7 grupos, sacando del mazo (pila) las cartas y repartiéndolas según las reglas indicadas, tome en consideración que cada grupo, es una pila por sí misma.

Al finalizar la repartición debe mostrar la primera carta de cada pila (get).

A continuación, se queda esperando la acción del jugador, donde ingresará:

* Posición: posición desde la cual tomará las cartas, contando de arriba hacia abajo.
* Grupo Origen: de que grupo tomará las cartas
* Grupo destino: a que grupo las moverá

Antes de hacer el movimiento, deberá verificar las reglas indicadas, si no las cumple, debe mostrar un mensaje de error; si las cumple, deberá hacer el movimiento internamente según lo solicitado.

Después de cada movimiento, verificar si cumple con los requisitos para ganar.

Al ganar mostrar el mensaje “USTED HA GANADO”

RESTRICCIONES

El juego debe realizarse en interfaz gráfica utilizando Windows Forms, la solución se guardará en un repositorio “git” y por cada funcionalidad implementada debe existir un commit, esto es muy importante para que no se catalogue como copia o fraude.

**Presentación**

**Carta: númeroCOLOR**

**MAZO: 10R**

**G1 G2 G3 G4 G5 G6 G7**

**8R 3N 7N 11R 13N 9R 1R**

**2R 10N 8N**

**Caso Ganador**

**MAZO: 10R**

**G1 G2 G3 G4 G5 G6 G7**

**8R 3N 7N 11R 13N 9R 1R**

**2R 10N 8N**

**7R**

**6N**

**5R**