|  |  |  |
| --- | --- | --- |
| **CARRERA:**  Ingeniería en Sistemas e informática | **GUÍA**  No. 03 | **TIEMPO ESTIMADO:** |
| **ASIGNATURA:**  Estructura de Datos  NRC: 2969 | **FECHA DE ELABORACION: 28**-01-2020  **SEMESTRE**: Octubre 2019 – Enero 2020 | |
| **TÍTULO:**  Proyecto Parcial | **DOCENTE: Fernando Solis** | |

**OBJETIVO**

p.e. Aplicar los conocimientos adquiridos en el tercer parcial en un proyecto.

**INSTRUCCIONES**

**p.e.**

1. Utilice como material principal, las principales herramientas para desarrollo en C++
2. Utilice información consultada en Internet y conocimiento adquirido en clase.

**ACTIVIDADES**

1. **Ubicación de recursos**

**p.e.**

1. Formar grupos de máximo 2 personas
2. Instalar el IDE para C++
3. **Planteamiento del problema**

**p.e. Ejercicio No 1**

Desarrollar 100 programas de estructura de Datos escogidos mediante un PDF en el cual constan mas de 400 ejercicios, escoger ejercicios fáciles, término medio y difíciles.

Este proyecto consiste en una recopilación de 100 programas de diferente tipo, desde operaciones básicas como la suma, resta, multiplicación, división y modulo, hasta algoritmos recursivos y uso de listas enlazadas.

**Marco Teórico**

**Lista Enlazada**

Es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento. También es un tipo de dato autor referenciado porque contienen un puntero o enlace a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

**Algoritmo**

es un conjunto de instrucciones o reglas definidas y no-ambiguas, ordenadas y finitas que permite, típicamente, solucionar un problema, realizar un cómputo, procesar datos y llevar a cabo otras tareas o actividades.

**Algoritmo Recursivo**

Un algoritmo recursivo es un algoritmo que expresa la solución de un problema en términos de una llamada a sí mismo. La llamada a sí mismo se conoce como llamada recursiva o recurrente

**Conclusiones**

* Muchas funciones de anteriores programas pueden ser utilizadas.
* Es bueno crear librerías propias, estas nos podrían ayudar a resolver problemas a futuro.
* La programación empieza desde lo más básico hasta lo mas complicado.

**Codigo**

#include "Tablero.h"

#include "Menu.h"

#include "Cifrador.h"

#include "Imagen.h"

int main()

{

Tablero tablero;

string s;

bool nuevoJuego = true;

Menu\* menu = new Menu();

string titulo = "MENU DE OPCIONES";

string\* opciones = new string[8];

\*(opciones + 0) = "Aplicacion";

\*(opciones + 1) = "Imagen";

\*(opciones + 2) = "Encriptar archivo";

\*(opciones + 3) = "Desencriptar archivo";

\*(opciones + 4) = "Generar barcode";

\*(opciones + 5) = "Generar PDF";

\*(opciones + 6) = "Ayuda";

\*(opciones + 7) = "Salir";

int numOpciones = 8;

int opcion;

do {

opcion = menu->menu(titulo, opciones, numOpciones);

system("cls");

switch (opcion) {

case 1:

while (nuevoJuego) {

tablero.inicializarTablero();

while (tablero.jugar());

cout << "Quieres juegar de nuevo? (y para si, cualquier otra tecla para no) ";

cin >> s;

if (s != "y")

nuevoJuego = false;

}

system("pause");

break;

case 2:

hConWnd = GetConsoleWndHandle();

if (hConWnd)

{

BCX\_Bitmap((char\*)"logo.bmp", hConWnd, 123, 1, 1, 0, 0);

//system("pause");

Sleep(10000);

}

system("pause");

break;

case 3:

encriptar(2020, "Soluciones/solucion.txt", "Soluciones/solucionencriptada.txt");

break;

case 4:

desencriptar(2020, "Soluciones/solucionencriptada.txt", "Soluciones/soluciondesencriptada.txt");

break;

case 5:

system("java -jar Generar\_barcode.jar");

break;

case 6:

system("java -jar textToPdf.jar");

break;

case 7:

system("ayuda.chm");

break;

case 8:

cout << "Hasta pronto joven" << endl;

break;

}

} while (opcion != 8);

delete[] opciones;

return 0;

}

#define \_CRT\_SECURE\_NO\_DEPRECATE

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

void encriptar(int aumento, const char\* newNombreArchivo, const char\* newNombreArchivoEncriptado) {

char c;

const char\* nombreArchivo = newNombreArchivo;//"solucion.txt"

const char\* nombreArchivoEncriptado = newNombreArchivoEncriptado;//"solucionencriptada.txt";

FILE\* ptr;

FILE\* ptr2;

#pragma warning (disable : 4996)

ptr = fopen(nombreArchivo, "r");

ptr2 = fopen(nombreArchivoEncriptado, "w");

if (ptr == NULL || ptr2 == NULL) {

cout<<"No se pudo abrir los archivos..."<<endl;

system("pause");

return;

}

//ABRO EL ARCHIVO Y A CADA CARACTER LO MODIFICO

while (fscanf(ptr, "%c", &c) != EOF) {

putc((c + aumento), ptr2);

}

cout<<"Se escribio correctamente en el archivo..."<<endl;

fclose(ptr);

fclose(ptr2);

system("pause");

}

void desencriptar(int aumento, const char\* newNombreArchivoEncriptado, const char\* newNombreArchivoDesencriptado) {

char c;

const char\* nombreArchivoEncriptado = newNombreArchivoEncriptado;//"solucionencriptada.txt";

const char\* nombreArchivoDesencriptado = newNombreArchivoDesencriptado;//"soluciondesencriptado.txt";

FILE\* ptr;

FILE\* ptr2;

ptr = fopen(nombreArchivoEncriptado, "r");

ptr2 = fopen(nombreArchivoDesencriptado, "w");

if (ptr == NULL || ptr2 == NULL) {

printf("No se pudo abrir los archivos...\n");

system("pause");

return;

}

//ABRO EL ARCHIVO Y A CADA CARACTER LO MODIFICO

while (fscanf(ptr, "%c", &c) != EOF) {

putc((c - aumento), ptr2);

}

printf("\nSe desencripto correctamente en el archivo...\n");

fclose(ptr);

fclose(ptr2);

system("pause");

}

#include "Cuadricula.h"

int Cuadricula::getX()

{

return x;

}

void Cuadricula::setX(int newX)

{

x = newX;

}

int Cuadricula::getY()

{

return y;

}

void Cuadricula::setY(int newY)

{

y = newY;

}

void Cuadricula::setEspacio(Cuadricula\* posicion)

{

color = posicion->getColor();

pieza = posicion->getPieza();

}

void Cuadricula::setVacio()

{

color = NINGUNO;

pieza = VACIO;

}

void Cuadricula::setPiezaYColor(Pieza p, Color c)

{

pieza = p;

color = c;

}

Pieza Cuadricula::getPieza()

{

return pieza;

}

Color Cuadricula::getColor()

{

return color;

}

Cuadricula::Cuadricula()

{

pieza = VACIO;

color = NINGUNO;

}

#include <iostream>

#include <windows.h>

static HWND hConWnd;

HWND BCX\_Bitmap(char\*, HWND = 0, int = 0, int = 0, int = 0, int = 0, int = 0, int = 0, int = 0, int = 0);

HWND GetConsoleWndHandle(void);

HWND BCX\_Bitmap(char\* Text, HWND hWnd, int id, int X, int Y, int W, int H, int Res, int Style, int Exstyle)

{

HWND A;

HBITMAP hBitmap;

// set default style

if (!Style) Style = WS\_CLIPSIBLINGS | WS\_CHILD | WS\_VISIBLE | SS\_BITMAP | WS\_TABSTOP;

// form for the image

A = CreateWindowEx(Exstyle, "static", NULL, Style, X, Y, 0, 0, hWnd, (HMENU)id, GetModuleHandle(0), NULL);

// Text contains filename

hBitmap = (HBITMAP)LoadImage(0, Text, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE | LR\_CREATEDIBSECTION);

// auto-adjust width and height

if (W || H) hBitmap = (HBITMAP)CopyImage(hBitmap, IMAGE\_BITMAP, W, H, LR\_COPYRETURNORG);

SendMessage(A, (UINT)STM\_SETIMAGE, (WPARAM)IMAGE\_BITMAP, (LPARAM)hBitmap);

if (W || H) SetWindowPos(A, HWND\_TOP, X, Y, W, H, SWP\_DRAWFRAME);

return A;

}

// tricking Windows just a little ...

HWND GetConsoleWndHandle(void)

{

HWND hConWnd;

OSVERSIONINFO os;

char szTempTitle[64], szClassName[128], szOriginalTitle[1024];

os.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);

GetVersionEx(&os);

// may not work on WIN9x

if (os.dwPlatformId == VER\_PLATFORM\_WIN32s) return 0;

GetConsoleTitle(szOriginalTitle, sizeof(szOriginalTitle));

sprintf(szTempTitle, "%u - %u", GetTickCount(), GetCurrentProcessId());

SetConsoleTitle(szTempTitle);

Sleep(60);

// handle for NT and XP

hConWnd = FindWindow(NULL, szTempTitle);

SetConsoleTitle(szOriginalTitle);

// may not work on WIN9x

if (os.dwPlatformId == VER\_PLATFORM\_WIN32\_WINDOWS)

{

hConWnd = GetWindow(hConWnd, GW\_CHILD);

if (hConWnd == NULL) return 0;

GetClassName(hConWnd, szClassName, sizeof(szClassName));

// while ( \_stricmp( szClassName, "ttyGrab" ) != 0 )

while (strcmp(szClassName, "ttyGrab") != 0)

{

hConWnd = GetNextWindow(hConWnd, GW\_HWNDNEXT);

if (hConWnd == NULL) return 0;

GetClassName(hConWnd, szClassName, sizeof(szClassName));

}

}

return hConWnd;

}

#ifndef MENU\_H\_INCLUDED

#define MENU\_H\_INCLUDED

#include<conio.h>

#include <iostream>

#include <windows.h>

#define ARRIBA 72

#define ABAJO 80

#define ENTER 13

using namespace std;

class Menu {

private:

public:

Menu();

void gotoxy(USHORT, USHORT);

int menu(string titulo, string\* opciones, int n);

};

Menu::Menu()

{

}

int Menu::menu(string titulo, string\* opciones, int n) {

int opcionSeleccionada = 1; // Indica la opcionSeleccionada

int tecla;

bool repite = true; // controla el bucle para regresar a la rutina que lo llamo, al presionar ENTER

do {

system("cls");

gotoxy(5, 3 + opcionSeleccionada); cout << "-->" << endl;

// Imprime el título del menú

gotoxy(15, 2); cout << titulo;

// Imprime las opciones del menú

for (int i = 0; i < n; ++i) {

gotoxy(10, 4 + i); cout << opciones[i];

}

// Solo permite que se ingrese ARRIBA, ABAJO o ENTER

do {

tecla = \_getch();

// gotoxy(15, 15); cout << "tecla presionada: " << (char)tecla << " = " << tecla;

} while (tecla != ARRIBA && tecla != ABAJO && tecla != ENTER);

switch (tecla) {

case ARRIBA: // En caso que se presione ARRIBA

opcionSeleccionada--;

if (opcionSeleccionada < 1) {

opcionSeleccionada = n;

}

break;

case ABAJO:

opcionSeleccionada++;

if (opcionSeleccionada > n) {

opcionSeleccionada = 1;

}

break;

case ENTER:

repite = false;

break;

}

} while (repite);

return opcionSeleccionada;

}

void Menu::gotoxy(USHORT x, USHORT y) {

COORD cp = { x,y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), cp);

}

#endif // MENU\_H\_INCLUDED

**#include "Tablero.h"**

**#include "Validacion.h"**

**#include <string>**

**using namespace std;**

**Validacion validacion;**

**bool Tablero::moverRey(Cuadricula\* rey, Cuadricula\* posicion) {**

**if (abs(posicion->getX() - rey->getX()) == 1)**

**if (abs(posicion->getY() - rey->getY()) == 1)**

**{**

**posicion->setEspacio(rey);**

**rey->setVacio();**

**return true;**

**}**

**else return false;**

**else return false;**

**}**

**bool Tablero::moverReina(Cuadricula\* reina, Cuadricula\* posicion) {**

**int reinaX = reina->getX();**

**int reinaY = reina->getY();**

**int posicionX = posicion->getX();**

**int posicionY = posicion->getY();**

**int yIncremento;**

**int xIncremento;**

**bool invalido = false;**

**if (reinaX != posicionX || reinaY != posicionY)**

**{**

**if (reinaX == posicionX)**

**{**

**yIncremento = (posicionY - reinaY) / (abs(posicionY - reinaY));**

**for (int i = reinaY + yIncremento; i != posicionY; i += yIncremento)**

**{**

**if ((\*(\*(cuadricula + posicionX) + i)).getColor() != NINGUNO)**

**return false;**

**}**

**}**

**else**

**if (reinaY == posicionY)**

**{**

**xIncremento = (posicionX - reinaX) / (abs(posicionX - reinaX));**

**for (int i = reinaX + xIncremento; i != posicionX; i += xIncremento)**

**{**

**if ((\*(\*(cuadricula + i) + posicionY)).getColor() != NINGUNO)**

**return false;**

**}**

**}**

**else**

**if (abs(reinaX - posicionX) == abs(reinaY - posicionY))**

**{**

**xIncremento = (posicionX - reinaX) / (abs(posicionX - reinaX));**

**yIncremento = (posicionY - reinaY) / (abs(posicionY - reinaY));**

**for (int i = 1; i < abs(reinaX - posicionX); i++)**

**{**

**if ((\*(\*(cuadricula + reinaX + xIncremento \* i) + reinaY + yIncremento \* i)).getColor() != NINGUNO)**

**return false;**

**}**

**}**

**else**

**return false;**

**}**

**if (invalido == false)**

**{**

**posicion->setEspacio(reina);**

**reina->setVacio();**

**return true;**

**}**

**else**

**{**

**return false;**

**}**

**}**

**bool Tablero::moverAlfil(Cuadricula\* alfil, Cuadricula\* posicion) {**

**int alfilX = alfil->getX();**

**int alfilY = alfil->getY();**

**int posicionX = posicion->getX();**

**int posicionY = posicion->getY();**

**bool invalido = false;**

**Cuadricula\* s;**

**if (abs(alfilX - posicionX) == abs(alfilY - posicionY))**

**{**

**int xIncrement = (posicionX - alfilX) / (abs(posicionX - alfilX));**

**int yIncrement = (posicionY - alfilY) / (abs(posicionY - alfilY));**

**for (int i = 1; i < abs(alfilX - posicionX); i++)**

**{**

**if ((\*(\*(cuadricula + alfilX + xIncrement \* i) + alfilY + yIncrement \* i)).getColor() != NINGUNO)**

**return false;**

**}**

**}**

**else**

**return false;**

**if (invalido == false)**

**{**

**posicion->setEspacio(alfil);**

**alfil->setVacio();**

**return true;**

**}**

**else**

**{**

**return false;**

**}**

**}**

**bool Tablero::moverCaballo(Cuadricula\* caballo, Cuadricula\* posicion)**

**{**

**int caballoX = caballo->getX();**

**int caballoY = caballo->getY();**

**int posicionX = posicion->getX();**

**int posicionY = posicion->getY();**

**if ((abs(caballoX - posicionX) == 2 && abs(caballoY - posicionY) == 1) || (abs(caballoX - posicionX) == 1 && abs(caballoY - posicionY) == 2))**

**{**

**posicion->setEspacio(caballo);**

**caballo->setVacio();**

**return true;**

**}**

**else**

**{**

**return false;**

**}**

**}**

**bool Tablero::moverTorre(Cuadricula\* torre, Cuadricula\* posicion)**

**{**

**int torreX = torre->getX();**

**int torreY = torre->getY();**

**int posicionX = posicion->getX();**

**int posicionY = posicion->getY();**

**bool invalido = false;**

**if (torreX != posicionX || torreY != posicionY)**

**{**

**if (torreX == posicionX)**

**{**

**int yIncrement = (posicionY - torreY) / (abs(posicionY - torreY));**

**for (int i = torreY + yIncrement; i != posicionY; i += yIncrement)**

**{**

**if ((\*(\*(cuadricula + posicionX) + i)).getColor() != NINGUNO)**

**return false;**

**}**

**}**

**else**

**if (torreY == posicionY)**

**{**

**int xIncrement = (posicionX - torreX) / (abs(posicionX - torreX));**

**for (int i = torreX + xIncrement; i != posicionX; i += xIncrement)**

**{**

**if ((\*(\*(cuadricula + i) + posicionY)).getColor() != NINGUNO)**

**return false;**

**}**

**}**

**else**

**return false;**

**}**

**if (invalido == false)**

**{**

**posicion->setEspacio(torre);**

**torre->setVacio();**

**return true;**

**}**

**else**

**{**

**return false;**

**}**

**}**

**bool Tablero::moverPeon(Cuadricula\* peon, Cuadricula\* posicion) {**

**bool invalid = false;**

**int peonX = peon->getX();**

**int peonY = peon->getY();**

**int posicionX = posicion->getX();**

**int posicionY = posicion->getY();**

**if (peon->getColor() == BLANCO)**

**{**

**if (peonX == posicionX && posicionY == peonY + 1 && posicion->getColor() == NINGUNO)**

**{**

**posicion->setEspacio(peon);**

**peon->setVacio();**

**return true;**

**}**

**else**

**if ((peonX + 1 == posicionX || peonX - 1 == posicionX) && peonY + 1 == posicionY && posicion->getColor() == NEGRO)**

**{**

**posicion->setEspacio(peon);**

**peon->setVacio();**

**return true;**

**}**

**else**

**return false;**

**}**

**else**

**if (peon->getColor() == NEGRO)**

**{**

**if (peonX == posicionX && posicionY == peonY - 1 && posicion->getColor() == NINGUNO)**

**{**

**posicion->setEspacio(peon);**

**peon->setVacio();**

**return true;**

**}**

**else**

**if ((peonX + 1 == posicionX || peonX - 1 == posicionX) && peonY - 1 == posicionY && posicion->getColor() == BLANCO)**

**{**

**posicion->setEspacio(peon);**

**peon->setVacio();**

**return true;**

**}**

**else**

**return false;**

**}**

**else**

**return false;**

**}**

**bool Tablero::moverPieza(int x1, int y1, int x2, int y2) {**

**Cuadricula\* origen = \*(cuadricula + x1) + y1;**

**Cuadricula\* destino = \*(cuadricula + x2) + y2;**

**switch (origen->getPieza())**

**{**

**case REY:**

**return moverRey(origen, destino);**

**break;**

**case REINA:**

**return moverReina(origen, destino);**

**break;**

**case ALFIL:**

**return moverAlfil(origen, destino);**

**break;**

**case CABALLO:**

**return moverCaballo(origen, destino);**

**break;**

**case TORRE:**

**return moverTorre(origen, destino);**

**break;**

**case PEON:**

**return moverPeon(origen, destino);**

**break;**

**case VACIO:**

**cout << "No tines una pieza en esa posicion." << endl; return false;**

**break;**

**default:**

**break;**

**}**

**return false;**

**}**

**void Tablero::imprimirTablero() {**

**char cuadrado = 254;**

**char peonNegro = 232;**

**char peonBlanco = 225;**

**cout << " y: 0 1 2 3 4 5 6 7" << endl << "x:" << endl;**

**for (int i = 0; i < 8; i++)**

**{**

**cout << " " << i << " ";**

**for (int j = 0; j < 8; j++)**

**{**

**Cuadricula temp = \*(\*(cuadricula + i) + j);**

**Pieza pieza = temp.getPieza();**

**Color color = temp.getColor();**

**switch (pieza)**

**{**

**case REY:**

**if (color == BLANCO)**

**cout << " R ";**

**else**

**cout << " r ";**

**break;**

**case REINA:**

**if (color == BLANCO)**

**cout << " Q ";**

**else**

**cout << " q ";**

**break;**

**case ALFIL:**

**if (color == BLANCO)**

**cout << " A ";**

**else**

**cout << " a ";**

**break;**

**case CABALLO:**

**if (color == BLANCO)**

**cout << " C ";**

**else**

**cout << " c ";**

**break;**

**case TORRE:**

**if (color == BLANCO)**

**cout << " T ";**

**else**

**cout << " t ";**

**break;**

**case PEON:**

**if (color == BLANCO)**

**cout << " " <<peonBlanco<< " ";**

**else**

**cout << " " <<peonNegro << " ";**

**break;**

**case VACIO: cout << " " << cuadrado << " ";**

**break;**

**default:**

**break;**

**}**

**}**

**cout << endl;**

**}**

**}**

**Cuadricula\*\* Tablero::generar(int dim)**

**{**

**Cuadricula\*\* a = new Cuadricula \* [dim];**

**for (int i = 0; i < dim; ++i) {**

**a[i] = new Cuadricula[dim];**

**}**

**return a;**

**}**

**bool Tablero::movimiento() {**

**string movimiento;**

**int x1, x2, y1, y2;**

**bool bandera = false;**

**while (!bandera)**

**{**

**if (turno == BLANCO)**

**cout << endl << "Turno del jugador blanco" << endl;**

**else**

**cout << endl << "Turno del jugador negro" << endl;**

**cout << "Escriba su movimiento como una cadena de 4 caracteres. Escriba la coordenada de x primero. Ej: \"0102\"" << endl;**

**cin >> movimiento;**

**movimiento = validacion.soloNumeros(movimiento);**

**movimiento = validacion.dentroDeLimites(movimiento);**

**x1 = movimiento.at(0) - 48;**

**y1 = movimiento.at(1) - 48;**

**x2 = movimiento.at(2) - 48;**

**y2 = movimiento.at(3) - 48;**

**if ((\*(cuadricula + x1) + y1)->getColor() == turno)**

**{**

**if (moverPieza(x1, y1, x2, y2) == false)**

**{**

**cout << "Movimiento invalido, reingrese." << endl;**

**}**

**else**

**bandera = true;**

**}**

**else**

**cout << "Esa no es tu pieza, reingrese." << endl;**

**}**

**if ((\*(cuadricula + x2) + y2)->getPieza() == REY)**

**if ((\*(cuadricula + x1) + y1)->getColor() == BLANCO)**

**{**

**cout << "BLANCO GANA" << endl;**

**return false;**

**}**

**else**

**{**

**cout << "NEGRO GANA" << endl;**

**return false;**

**}**

**if (turno == NEGRO)**

**turno = BLANCO;**

**else**

**turno = NEGRO;**

**return true;**

**}**

**void Tablero::inicializarTablero()**

**{**

**(\*(\*(cuadricula))).setPiezaYColor(TORRE, BLANCO);**

**(\*(\*(cuadricula + 1))).setPiezaYColor(CABALLO, BLANCO);**

**(\*(\*(cuadricula + 2))).setPiezaYColor(ALFIL, BLANCO);**

**(\*(\*(cuadricula + 3))).setPiezaYColor(REINA, BLANCO);**

**(\*(\*(cuadricula + 4))).setPiezaYColor(REY, BLANCO);**

**(\*(\*(cuadricula + 5))).setPiezaYColor(ALFIL, BLANCO);**

**(\*(\*(cuadricula + 6))).setPiezaYColor(CABALLO, BLANCO);**

**(\*(\*(cuadricula + 7))).setPiezaYColor(TORRE, BLANCO);**

**(\*(\*(cuadricula)+7)).setPiezaYColor(TORRE, NEGRO);**

**(\*(\*(cuadricula + 1) + 7)).setPiezaYColor(CABALLO, NEGRO);**

**(\*(\*(cuadricula + 2) + 7)).setPiezaYColor(ALFIL, NEGRO);**

**(\*(\*(cuadricula + 3) + 7)).setPiezaYColor(REINA, NEGRO);**

**(\*(\*(cuadricula + 4) + 7)).setPiezaYColor(REY, NEGRO);**

**(\*(\*(cuadricula + 5) + 7)).setPiezaYColor(ALFIL, NEGRO);**

**(\*(\*(cuadricula + 6) + 7)).setPiezaYColor(CABALLO, NEGRO);**

**(\*(\*(cuadricula + 7) + 7)).setPiezaYColor(TORRE, NEGRO);**

**for (int i = 0; i < 8; i++)**

**{**

**(\*(\*(cuadricula + i) + 1)).setPiezaYColor(PEON, BLANCO);**

**(\*(\*(cuadricula + i) + 6)).setPiezaYColor(PEON, NEGRO);**

**}**

**for (int i = 2; i < 6; i++)**

**{**

**for (int j = 0; j < 8; j++)**

**(\*(\*(cuadricula + j) + i)).setPiezaYColor(VACIO, NINGUNO);**

**}**

**for (int i = 0; i < 8; i++)**

**for (int j = 0; j < 8; j++)**

**{**

**(\*(\*(cuadricula + i) + j)).setX(i);**

**(\*(\*(cuadricula + i) + j)).setY(j);**

**}**

**}**

**bool Tablero::jugar()**

**{**

**system("cls");**

**imprimirTablero();**

**return movimiento();**

**}**

#include "Validacion.h"

string Validacion::soloNumeros(string cadena)

{

for (int i = 0; i < cadena.length(); i++) {

if (!isdigit(cadena.at(i)) || cadena.length() > 4) {

cout << "Dato no valido, Reingrese: " << endl;

cin >> cadena;

i = -1;

}

}

return cadena;

}

string Validacion::dentroDeLimites(string cadena)

{

int temp;

for (int i = 0; i < cadena.length(); i++) {

temp = cadena.at(i) - 48;

if (temp < 0 || temp >7) {

cout << "Tu movimiento esta fuera de los limites, Reingrese: " << endl;

cin >> cadena;

cadena = soloNumeros(cadena);

i = -1;

}

}

return cadena;

}