**05StructFile**

#include <string>

#include <vector>

#include <cstring>

#include <fstream>

#include <iostream>

#include "../lib/toolbox.h"

#define  SEPARADOR ','

using namespace std;

struct  Mascota

{

int     id;

string  tipo;

string  nombre;

};

struct  Alumno

{

int      id;

string   nombre;

Mascota  mascota;

};

vector<Alumno> lstAlumno;

void showAlumnoMascota(Alumno *a*)

{

cout<< COLOR\_BLUE   << endl

<<"[+] Datos de Alumnos     "       << endl

<<" -  Alumno.Id          : "       << *a*.id             << endl

<<" -  Alumno.nombre      : "       << *a*.nombre         << endl

<<" -  mascotita.Id       : "       << *a*.mascota.id     << endl

<<" -  mascotita.tipo     : "       << *a*.mascota.tipo   << endl

<<" -  mascotita.nombre   : "       << *a*.mascota.nombre << endl << endl;

}

void writeFileAlumno(const string *filePath*)

{

ofstream f(*filePath*, ios::out | ios::trunc);

if (!f) {

cerr << "Error al abrir el archivo: " << *filePath* << endl;

return;

}

cout<< COLOR\_RED << "Guardando:"<< endl;

for (auto &&a : lstAlumno)

{

f   << a.id << SEPARADOR << a.nombre << SEPARADOR  << a.mascota.id  << SEPARADOR  << a.mascota.tipo << SEPARADOR  << a.mascota.nombre <<endl;

cout<< a.id << SEPARADOR << a.nombre << SEPARADOR  << a.mascota.id  << SEPARADOR  << a.mascota.tipo << SEPARADOR  << a.mascota.nombre <<endl;

}

f.close();

}

void readFileAlumno(const string *filePath*)

{

int     count = 1;

string  line  = "";

ifstream f(*filePath*, ios::in);

if (!f)

{

cerr << "error al abrir el archivo: " << *filePath* << endl ;

return;

}

while (getline(f, line))

{

Alumno a;

vector<string> dataLine = splitString(line, SEPARADOR);

cout<< COLOR\_RED << showWaiting (line) << COLOR\_BLUE   << endl;

a.id             = count++; *//stoi(dataLine[0]);*

a.nombre         = dataLine[1];

a.mascota.id     = stoi(dataLine[2]);

a.mascota.tipo   = dataLine[3];

a.mascota.nombre = dataLine[4];

lstAlumno.push\_back(a);

*//showAlumnoMascota(a);*

}

}

void showListaAlumno()

{

cout<< COLOR\_BLUE << "ALUMNOS: "<< endl;

for (auto &&a : lstAlumno)

showAlumnoMascota(a);

}

void addListaAlumno()

{

Alumno a;

cout<< COLOR\_RED <<"[+] ADD ALUMNO           "      << endl;

*//a.id    = getConsoleNumero(" -  Alumno.Id    : ", 1,100);*

a.id = lstAlumno.size()+1;

cout<<" -  Alumno.nombre      : ";     cin>>   a.nombre         ;

cout<<" -  mascotitas.Id      : ";     cin>>   a.mascota.id     ;

cout<<" -  mascotitas.tipo    : ";     cin>>   a.mascota.tipo   ;

cout<<" -  mascotitas.nombre  : ";     cin>>   a.mascota.nombre ;

lstAlumno.push\_back(a);

cout<< COLOR\_BLUE << "Registro con exito..!" << endl;

}

Alumno findListaAlumno()

{

Alumno alumno;

int id = 0;

cout<< COLOR\_RESET << "BUSCANDO..."    <<endl;

id = getConsoleNumero("Ingrese el id del alumno: ", 1, lstAlumno.size());

for (auto &&a : lstAlumno)

if(a.id == id)

{

alumno = a;

break;

}

showAlumnoMascota(alumno);

return alumno;

}

void alterListaAlumno()

{

Alumno a;

string rta;

a = findListaAlumno();

cout<<"Esta seguro que desea modificar al alumno (s=Si / n=No)?: ";

cin>>rta;

if(rta != "s")

return;

cout<< COLOR\_RED <<"[+] MODIFICAR ALUMNO          "      << endl;

*//a.id = lstAlumno.size()+1;*

cout<<" -  Alumno.nombre      : ";     cin>>    a.nombre         ;

cout<<" -  mascotitas.Id      : ";     cin>>    a.mascota.id     ;

cout<<" -  mascotitas.tipo    : ";     cin>>    a.mascota.tipo   ;

cout<<" -  mascotitas.nombre  : ";     cin>>    a.mascota.nombre ;

int index\_to\_insert = a.id-1;

*//cout << "index\_to\_insert : " << index\_to\_insert << endl;*

lstAlumno.erase(lstAlumno.begin()  + index\_to\_insert);

lstAlumno.insert(lstAlumno.begin() + index\_to\_insert, a);

cout<< COLOR\_BLUE << "Registro con exito..!" << endl;

}

void delListaAlumno()

{

}

int main()

{

int opc=0;

string dataFilename="datos.csv";

cout<< COLOR\_RED << "Procesando "<< dataFilename << endl;

readFileAlumno(dataFilename);

while (true)

{

system("cls");

cout<< COLOR\_GREEN << "[+] ALUMNO DE PROGRAMACION I"<< endl

<< " 1. Mostrar   "     <<endl

<< " 2. Agregar   "     <<endl

<< " 3. Eliminar  "     <<endl

<< " 4. Buscar    "     <<endl

<< " 5. Modificar "     <<endl

<< " 6. Respaldar paralelo"   <<endl

<< " 7. Salir"                <<endl;

opc = getConsoleNumero("Ingrese una opcion: ", 1,7);

switch (opc)

{

case 1: showListaAlumno();              break;

case 2: addListaAlumno ();              break;

case 3: delListaAlumno ();              break;

case 4: findListaAlumno ();             break;

case 5: alterListaAlumno ();            break;

case 6: writeFileAlumno(dataFilename);  break;

case 7: exit(0);

}

cout<< COLOR\_RED << endl;

system("pause");

}

return 0;

}

**6Puntero**

#include<iostream> using namespace std;

*// Funcion que recibe un parametro por valor*

*// Crea una copia de 'valor' y trabaja solo con la copia*

int funcion(int *valor*)

{

*valor* = *valor* + 10; *// Solo se le suma 10 a la copia local*

return *valor*;

}

*// Funcion que recibe un puntero (referencia)*

*// Trabaja directamente con la direccion de memoria de la variable original*

int funcionPunteros(int\* *valor*)

{

\**valor* = \**valor* + 10; *// Accede al valor original y lo modifica*

return \**valor*;

}

int main() {

int numero = 10;

cout << "Param X Valor ";

cout << "Antes de funcion " << numero << "\n"; *// Salida: 10*

funcion(numero); *// Se pasa el valor 10. La copia dentro de la funcion se modifica, no el original.*

cout << "Despues de funcion " << numero << "\n"; *// Salida: 10 (El valor original no cambio)*

cout << "Param X Referencia ";

cout << "Antes de funcionPunteros " << numero << "\n"; *// Salida: 10*

funcionPunteros(&numero); *// Se envia la direccion de memoria de 'numero'*

cout << "Despues de funcionPunteros " << numero << "\n"; *// Salida: 20 (El valor original cambio)*

system("pause"); *// Pausa la ejecucion en sistemas Windows*

return 0;

}

**7MemoriaPunteros**

#include <iostream> #include <stdlib.h> using namespace std;

*// Funcion para crear una matriz usando malloc*

int \*\*crearMatrizMalloc(int *f*, int *c*)

{

int \*\*m=NULL;

*// Asigna un array de 'f' punteros a enteros*

m = (int \*\*) malloc(f\*sizeof(int \*));

for (int i = 0; i < f; i++)

*// Para cada puntero, asigna un array de 'c' enteros*

m[i] = (int \*) malloc(c\*sizeof(int));

return m;

}

*// Funcion para crear una matriz usando calloc (inicializa a cero)*

int \*\*crearMatrizCalloc(int *f*, int *c*)

{

int \*\*m=NULL;

*// Asigna y limpia la memoria con ceros*

m = (int \*\*) calloc(f, sizeof(int \*));

for (int i = 0; i < f; i++)

m[i] = (int \*) calloc(c, sizeof(int));

return m;

}

int \*\*crearMatrizNew(int *f*, int *c*) *// Funcion para crear una matriz usando new de C++*

{

int \*\*m=NULL;

*// Asigna un array de punteros*

m = new int\*[f];

for (int i = 0; i < f; i++)

*// Para cada puntero, asigna un array de enteros*

m[i] = new int[c];

return m;

}

void showMatriz(int \*\**pd*, int *f*,int *c*) *// Funcion para mostrar la matriz en la consola*

{

for (int i = 0; i < f; i++)

{

for (int j = 0; j < c; j++)

cout<< pd[i][j] <<"\t";

cout<< endl;

}

}

int main() {

int f = 0, c = 0;

int \*\*pd=NULL;

cout<<"Ingresa fila y columnas de la matriz : ";

cin>> f >> c;

*// Llama a la funcion para crear la matriz usando 'new'*

pd = crearMatrizNew(f,c);

*// Rellena la matriz con numeros aleatorios*

for (int i = 0; i < f; i++)

for (int j = 0; j < c; j++)

pd[i][j] = rand() % 10;

*// Muestra la matriz*

showMatriz(pd,f,c);

*// Es importante liberar la memoria, lo cual falta en este codigo*

*// por lo que generaria una fuga de memoria.*

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

delete[] pd[i];

}

delete[] pd;

return 0;

}

**8MatrizPuntero**

#include <iostream>

using namespace std;

*// Función para mostrar un vector, recibe un array de enteros y su tamaño*

void showVector(int *v*[], int *c*){

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

cout<<*v*[i]<<"\t";

cout<<endl;

}

*// Función para mostrar un vector usando un puntero a entero*

*// (Un array se puede pasar como un puntero al primer elemento)*

void showPtrVector(int \**pv*, int *c*){

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

cout<<*pv*[i]<<"\t"; *// Se accede al elemento como si fuera un array*

cout<<endl;

}

*// Función para mostrar una matriz, el compilador necesita el tamaño de las columnas*

void showMatriz(int *m*[][3], int *f*, int *c*){

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

{

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

cout<< *m*[i][j] <<"\t";

cout<< endl;

}

}

*// Función para tratar una matriz como un vector de una sola dimensión*

*// La fórmula i\*c+j calcula el índice del elemento m[i][j] en un array plano*

void showMatrizComoVector(int *mv*[], int *f*, int *c*){

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

{

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

cout<< *mv*[i\**c*+j] <<"\t";

cout<< endl;

}

}

*// Función para mostrar una matriz usando un puntero simple a su primer elemento*

*// Accede a los elementos usando aritmética de punteros*

void showPtrMatriz(int \**p*, int *f*,int *c*)

{

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

{

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

cout<< \*(*p*+i \* *c*+j) <<"\t"; *// (p + i\*c) es el puntero a la fila i, +j es el elemento j*

cout<< endl;

}

}

*// Función para mostrar una matriz usando un array de punteros a enteros*

void showVectorPtrMatriz(int \**vp*[], int *f*,int *c*)

{

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

{

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

cout<< *vp*[i][j] <<"\t"; *// Accede a cada elemento a traves del array de punteros*

cout<< endl;

}

}

*// Función para mostrar una matriz usando un puntero de punteros a enteros*

*// Es la forma más flexible para matrices dinámicas*

void showPtrDobleMatriz(int \*\**pd*, int *f*,int *c*)

{

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

{

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

cout<< *pd*[i][j] <<"\t";

cout<< endl;

}

}

*// Muestra las opciones de uso de un array de punteros*

void showArrayVector(){

int ai[]={1,3,5,7,9};

int \*pi;

pi = ai; *// El puntero pi apunta al inicio del array ai*

\*(pi+0) = 10; *// Se cambia el primer elemento del array a traves del puntero*

showPtrVector(pi,5); *// Muestra el array modificado*

}

int main(void)

{

cout<<endl<<endl<<"Iniciando con vectores y punteros"<<endl;

showArrayVector();

*// Matriz de 4 filas y 3 columnas, inicializada con valores*

const int f=4, c=3;

int m[f][c] = { {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12}};

cout<<endl<<endl<<"0. showVector(int a[], int c)"<<endl;

*// Pasa una fila de la matriz (m[0]) como un vector*

showVector(m[0] , c) ;

showVector(m[1] , c) ;

cout<<endl<<endl<<"1. showMatriz: (int m[][3], int f, int c)"<<endl;

*// Pasa la matriz completa a una funcion que espera una matriz*

showMatriz(m ,f, c) ;

cout<<endl<<endl<<"2. showMatrizComoVector: (int mv[], int f, int c)"<<endl;

*// Pasa la direccion del primer elemento para tratar la matriz como un vector*

showMatrizComoVector(&m[0][0] ,f, c);

cout<<endl<<endl<<"3. showPtrMatriz: (int \*p, int f,int c)"<<endl;

*// Pasa la direccion del primer elemento para usar aritmetica de punteros*

showPtrMatriz(&m[0][0] ,f, c);

*// Array de punteros que apunta a la primera columna de cada fila de la matriz*

int \*p[f];

for (int i = 0; i < f; i++)

p[i] = &m[i][0];

cout<<endl<<endl<<"4. showVectorPtrMatriz: (int \*vp[], int f,int c)"<<endl;

*// Pasa el array de punteros a la funcion*

showVectorPtrMatriz(p,f,c);

*// Puntero doble que apunta al array de punteros 'p'*

int \*\*pd = p;

cout<<endl<<endl<<"5. showPtrDobleMatriz: (int \*\*pd, int f,int c)"<<endl;

*// Pasa el puntero doble a la funcion*

showPtrDobleMatriz(pd,f,c);

cout<<endl<<endl<<"6. matriz -> showPtrVector: (int \*pv, int c)"<<endl;

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

*// Incrementa el puntero doble 'pd' para apuntar a la siguiente fila*

*// y pasa la fila como un puntero a un vector*

showPtrVector(\*pd++, c);

}

cout<<endl<<endl<<"7. matriz -> showVector(int a[], int c)"<<endl;

*// Vuelve a mostrar las filas de la matriz una por una*

for (int i = 0; i < f; i++)

showVector(m[i] , c) ;

return 0;

}

**9EstructuraPuntero**

#include <iostream>

#include <string.h>

using namespace std;

*// Define la estructura para una mascota*

struct mascota

{

*//int  id; // Una variable que podria ser el ID, pero esta comentada.*

char name[20]; *// Un array de caracteres para el nombre de la mascota*

char \*ptrName; *// Un puntero a un caracter (se usara para apuntar al nombre)*

};

*// Declara una estructura 'gatito' y un puntero 'ptrGatito'*

*// que apunta a la direccion de 'gatito'*

mascota gatito, \*ptrGatito = &gatito;

*// Funcion para ingresar datos usando un puntero global*

void datosGatito() {

cout << "Ingrese su nombre: ";

*// Lee una linea de texto y la guarda en el nombre de la estructura*

*// Se usa ptrGatito->name para acceder al miembro a traves del puntero*

cin.getline(ptrGatito->name, 50, '\n');

*// Muestra el nombre usando el puntero y la variable de estructura*

cout << "ptrGatito: "   << ptrGatito->name  << endl;

cout << "gatito: "      << gatito.name  << endl;

}

*// Funcion para ingresar datos usando un puntero pasado por parametro*

void datosGatitoParametro(struct mascota \**ptrCat*) {

cout << "Ingrese su nombre: ";

cin.getline(*ptrCat*->name, 50, '\n');

*// Muestra el nombre usando el puntero pasado*

cout << "ptrCat: "   << *ptrCat*->name  << endl;

*// La linea de abajo esta comentada porque ptrCat es un puntero, no una estructura*

*//cout << "cat: "      << ptrCat.name  << endl;*

}

int main()

{

*// Bloque 1: Usa la funcion con la variable global*

datosGatito();

cin.get(); *// Pausa la ejecucion*

*// Bloque 2: Usa un array de estructuras y gestiona memoria dinamica*

mascota lstMascotas[2]; *// Array de 2 mascotas en el stack*

for (int i = 0; i < 2; i++)

{

cout<<"Nombre de mascota: ";

fgets(lstMascotas[i].name,20,stdin); *// Lee el nombre*

int len = strlen(lstMascotas[i].name); *// Obtiene la longitud del nombre*

*// Asigna memoria dinamicamente para el nombre*

lstMascotas[i].ptrName = (char \*)malloc(len \* sizeof(char));

if(NULL == lstMascotas[i].ptrName )

cout<<"No hay memoria";

else{

*// Copia el nombre al espacio de memoria recien creado*

strcpy(lstMascotas[i].ptrName , lstMascotas[i].name );

*//free(lstMascotas->ptrName); // Esto libera la memoria, pero esta comentada*

}

}

*// Recorre el array y muestra los nombres usando el puntero*

for (auto &&m : lstMascotas)

cout<<m.ptrName;

*// Bloque 3: Usa new para crear una estructura dinamica y la pasa a una funcion*

mascota \*p =  new  struct mascota; *// Crea una mascota en el heap y asigna su direccion a 'p'*

datosGatitoParametro(p); *// Pasa el puntero a la funcion*

cin.get(); *// Pausa la ejecucion*

return 0;

}

**10SumaConPunteros**

#include <iostream>

using namespace std;

*// Variable global, aunque no se usa en la funcion 'sum'*

int \* a;

*// Funcion recursiva para sumar los elementos de un array*

int sum(int\* *a*, int *n*) {

*// Caso base: si ya no quedan elementos, la suma es 0*

if (n == 0) {

return 0;

}

*// Paso recursivo: suma el ultimo elemento y llama a la funcion con un elemento menos*

return sum(a, n - 1) + a[n - 1];

}

int main() {

int n = 0;

int \* a;

cout<<"Ingresa cantidad de elementos:" <<endl;

cin >> n;

if( n <= 0) *// Valida que la cantidad sea positiva*

return 0;

*// Reserva memoria dinamica para 'n' enteros en el heap*

a = new int[n];

cout<<"Ingresa elementos:" <<endl;

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

cin >> a[i]; *// Lee cada entero y lo guarda en el array*

cout<<"La suma es:" <<endl;

cout << sum(a, n) << endl; *// Llama a la funcion recursiva*

*// Libera la memoria que fue reservada con 'new'*

*// Esto es muy importante para evitar fugas de memoria*

delete[] a;

return 0;

}

**arbolBinario**

#include <iostream>

#include <conio.h> *// Para la funcion getch() en Windows*

using namespace std;

*// Define la estructura de un nodo para el arbol*

struct Nodo {

int dato; *// El valor del nodo*

Nodo \*izq; *// Puntero al subarbol izquierdo*

Nodo \*der; *// Puntero al subarbol derecho*

};

*// Declaracion de funciones*

Nodo \*crearNodo(int);

void insertar(Nodo \*&, int);

void mostrarArbol(Nodo \*, int);

bool buscar(Nodo \*, int);

int minimo(Nodo \*);

int maximo(Nodo \*);

void preOrden(Nodo \*);

void inOrden(Nodo \*);

void postOrden(Nodo \*);

int main() {

Nodo \*arbol = NULL; *// Puntero a la raiz del arbol, inicialmente nulo*

int dato;

cout << "Ingrese 5 numeros para el arbol binario:" << endl;

for (int i = 0; i < 5; ++i) {

cout << "Numero " << i + 1 << ": ";

cin >> dato;

insertar(arbol, dato); *// Inserta el dato en el arbol*

}

cout << endl;

cout << "Mostrando la estructura del arbol:" << endl;

mostrarArbol(arbol, 0); *// Muestra el arbol*

cout << endl;

int valor\_buscar;

cout << "Ingrese un valor a buscar: ";

cin >> valor\_buscar;

if (buscar(arbol, valor\_buscar)) {

cout << "El valor " << valor\_buscar << " se encuentra en el arbol." << endl;

} else {

cout << "El valor " << valor\_buscar << " NO se encuentra en el arbol." << endl;

}

cout << endl;

cout << "El valor minimo en el arbol es: " << minimo(arbol) << endl;

cout << "El valor maximo en el arbol es: " << maximo(arbol) << endl;

cout << endl;

*// Diferentes recorridos del arbol*

cout << "Recorrido Pre-Orden: ";

preOrden(arbol); *// Raiz, Izquierda, Derecha*

cout << endl;

cout << "Recorrido In-Orden: ";

inOrden(arbol); *// Izquierda, Raiz, Derecha (Ordenado)*

cout << endl;

cout << "Recorrido Post-Orden: ";

postOrden(arbol); *// Izquierda, Derecha, Raiz*

cout << endl;

return 0;

}

*// Implementación de funciones*

Nodo \*crearNodo(int *n*) {

Nodo \*nuevo\_nodo = new Nodo(); *// Reserva memoria para un nuevo nodo*

nuevo\_nodo->dato = *n*;

nuevo\_nodo->izq = NULL;

nuevo\_nodo->der = NULL;

return nuevo\_nodo;

}

void insertar(Nodo \*&*arbol*, int *n*) {

if (*arbol* == NULL) { *// Si el arbol esta vacio, el nuevo nodo es la raiz*

Nodo \*nuevo\_nodo = crearNodo(*n*);

*arbol* = nuevo\_nodo;

} else {

if (*n* < *arbol*->dato) { *// Si el dato es menor, se inserta en el subarbol izquierdo*

insertar(*arbol*->izq, *n*);

} else { *// Si es mayor o igual, se inserta en el subarbol derecho*

insertar(*arbol*->der, *n*);

}

}

}

void mostrarArbol(Nodo \**arbol*, int *cont*) {

if (*arbol* == NULL) {

return;

} else {

mostrarArbol(*arbol*->der, *cont* + 1); *// Recorre el subarbol derecho*

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

cout << "   ";

}

cout << *arbol*->dato << endl; *// Muestra el dato de la raiz*

mostrarArbol(*arbol*->izq, *cont* + 1); *// Recorre el subarbol izquierdo*

}

}

bool buscar(Nodo \**arbol*, int *n*) {

if (*arbol* == NULL) { *// No se encontro el valor*

return false;

} else if (*arbol*->dato == *n*) { *// Se encontro el valor*

return true;

} else if (*n* < *arbol*->dato) { *// Busca en el subarbol izquierdo*

return buscar(*arbol*->izq, *n*);

} else { *// Busca en el subarbol derecho*

return buscar(*arbol*->der, *n*);

}

}

int minimo(Nodo \**arbol*) {

if (*arbol* == NULL) {

cout << "El arbol esta vacio" << endl;

return -1;

}

*// El valor minimo esta en el nodo mas a la izquierda*

while (*arbol*->izq != NULL) {

*arbol* = *arbol*->izq;

}

return *arbol*->dato;

}

int maximo(Nodo \**arbol*) {

if (*arbol* == NULL) {

cout << "El arbol esta vacio" << endl;

return -1;

}

*// El valor maximo esta en el nodo mas a la derecha*

while (*arbol*->der != NULL) {

*arbol* = *arbol*->der;

}

return *arbol*->dato;

}

void preOrden(Nodo \**arbol*) {

if (*arbol* == NULL) {

return;

} else {

cout << *arbol*->dato << " "; *// Raiz*

preOrden(*arbol*->izq); *// Izquierda*

preOrden(*arbol*->der); *// Derecha*

}

}

void inOrden(Nodo \**arbol*) {

if (*arbol* == NULL) {

return;

} else {

inOrden(*arbol*->izq); *// Izquierda*

cout << *arbol*->dato << " "; *// Raiz*

inOrden(*arbol*->der); *// Derecha*

}

}

void postOrden(Nodo \**arbol*) {

if (*arbol* == NULL) {

return;

} else {

postOrden(*arbol*->izq); *// Izquierda*

postOrden(*arbol*->der); *// Derecha*

cout << *arbol*->dato << " "; *// Raiz*

}

}

**colas**

#include <iostream>

using namespace std;

*// Estructura de un nodo*

struct nodo {

int nro;

struct nodo \*sgte;

};

*// Estructura para la cola, con punteros al frente y al final*

struct cola {

nodo \*delante;

nodo \*atras;

};

*// Funcion para encolar (agregar un elemento al final)*

void encolar(struct cola &*q*, int *valor*) {

struct nodo \*aux = new(struct nodo);

aux->nro = *valor*;

aux->sgte = NULL;

if (*q*.delante == NULL)

*q*.delante = aux; *// Si la cola esta vacia, el nuevo nodo es el primero*

else

(*q*.atras)->sgte = aux; *// El ultimo nodo ahora apunta al nuevo*

*q*.atras = aux; *// El nuevo nodo es el final de la cola*

}

*// Funcion para desencolar (eliminar el elemento del frente)*

void desencolar(struct cola &*q*) {

int num;

struct nodo \*aux;

aux = *q*.delante;

num = aux->nro;

*q*.delante = (*q*.delante)->sgte; *// El nuevo frente es el siguiente nodo*

cout << " << desencolado >> " << num << endl;

delete(aux); *// Libera la memoria del nodo que se desencolo*

}

*// Funcion para mostrar la cola*

void muestraCola(struct cola *q*) {

struct nodo \*aux;

aux = *q*.delante;

while (aux != NULL) {

cout << aux->nro << "\t";

aux = aux->sgte;

}

}

int main() {

struct cola q;

q.delante = NULL;

q.atras = NULL;

encolar(q, 1);

encolar(q, 2);

encolar(q, 3);

cout << "Cola: ";

muestraCola(q);

cout << endl;

desencolar(q);

desencolar(q);

cout << "Cola despues de desencolar: ";

muestraCola(q);

cout << endl;

return 0;

}

**listas**

#include <iostream>

using namespace std;

*// Define la estructura para un nodo de la lista*

struct nodo {

int nro; *// El dato que contiene el nodo*

struct nodo \*sgte; *// Puntero al siguiente nodo en la lista*

};

typedef struct nodo\* Tlista; *// Tlista es un alias para un puntero a un nodo*

*// Funcion para insertar un nodo al inicio de la lista*

void insertarInicio(Tlista &*lista*, int *valor*) {

Tlista aux;

aux = new(struct nodo); *// Crea un nuevo nodo*

aux->nro = *valor*;

aux->sgte = *lista*; *// El nuevo nodo apunta al inicio actual*

*lista* = aux; *// El nuevo nodo se convierte en el inicio de la lista*

}

*// Funcion para insertar un nodo al final de la lista*

void insertarFinal(Tlista &*lista*, int *valor*) {

Tlista t, q = new(struct nodo);

q->nro = *valor*;

q->sgte = NULL;

if (*lista* == NULL)

*lista* = q; *// Si la lista esta vacia, el nuevo nodo es el inicio*

else {

t = *lista*;

while (t->sgte != NULL) {

t = t->sgte; *// Recorre la lista hasta el ultimo nodo*

}

t->sgte = q; *// El ultimo nodo ahora apunta al nuevo nodo*

}

}

*// Funcion para mostrar todos los elementos de la lista*

void mostrarLista(Tlista *lista*) {

Tlista aux = *lista*;

while (aux != NULL) {

cout << aux->nro << " -> ";

aux = aux->sgte; *// Avanza al siguiente nodo*

}

cout << "NULL" << endl; *// Indica el final de la lista*

}

int main() {

Tlista lista = NULL; *// Inicializa la lista como vacia*

insertarInicio(lista, 1);

insertarInicio(lista, 0);

mostrarLista(lista); *// Salida: 0 -> 1 -> NULL*

insertarFinal(lista, 2);

mostrarLista(lista); *// Salida: 0 -> 1 -> 2 -> NULL*

return 0;

}

**pilas**

#include <iostream>

using namespace std;

*// Estructura de un nodo para la pila*

struct nodo {

int nro;

struct nodo \*sgte;

};

typedef nodo\* ptrPila; *// Alias para un puntero a nodo*

*// Funcion para apilar (push), inserta un elemento al inicio de la lista enlazada*

void push(ptrPila &*p*, int *valor*) {

ptrPila aux = new(struct nodo);

aux->nro = *valor*;

aux->sgte = *p*;

*p* = aux;

cout << " << apilado >> " << *valor* << endl;

}

*// Funcion para desapilar (pop), elimina el elemento del inicio*

void pop(ptrPila &*p*) {

if (*p* == NULL) { *// Si la pila esta vacia*

cout << "La pila esta vacia." << endl;

return;

}

ptrPila aux;

aux = *p*;

cout << " << desapilado >> " << aux->nro << endl;

*p* = aux->sgte; *// La pila ahora apunta al siguiente nodo*

delete(aux); *// Libera la memoria del nodo que se desapilo*

}

int main() {

ptrPila p = NULL; *// Pila inicializada como vacia*

push(p, 3);

push(p, 5);

push(p, 8);

pop(p); *// Salida: << desapilado >> 8*

pop(p); *// Salida: << desapilado >> 5*

pop(p); *// Salida: << desapilado >> 3*

pop(p); *// Intentar desapilar de una pila vacia*

return 0;

}

**punteros**

#include <iostream>

int main() {

int num = 100; *// Declara una variable entera 'num' y la inicializa con 100*

int \*apuntador\_num; *// Declara un puntero a un entero*

apuntador\_num = &num; *// Asigna al puntero la direccion de memoria de 'num'*

std::cout << "num: " << num << std::endl; *// Muestra el valor de 'num'*

std::cout << "&num: " << &num << std::endl; *// Muestra la direccion de memoria de 'num'*

std::cout << "apuntador\_num: " << apuntador\_num << std::endl; *// Muestra el contenido del puntero (la direccion de 'num')*

std::cout << "\*apuntador\_num: " << \*apuntador\_num << std::endl; *// Muestra el valor al que apunta el puntero*

std::cout << "&apuntador\_num: " << &apuntador\_num << std::endl; *// Muestra la direccion de memoria del puntero en si*

char letra; *// Declara una variable de tipo char*

char \*apuntador; *// Declara un puntero a un caracter*

apuntador = &letra; *// Asigna la direccion de 'letra' al puntero*

\*apuntador = 'x'; *// Usa el puntero para asignar el valor 'x' a la variable 'letra'*

std::cout << "letra: " << letra << std::endl; *// Muestra el valor de 'letra'*

return 0;

}

**prueba1**#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <iomanip>

#include <map>

#include <windows.h>

#include "../lib/toolboxs.h"

using namespace std;

*// Variables globales*

const int coAlHoras = 24;

const int coAlDias = 5;

string coAlDiasSemana[coAlDias] = {"Lune", "Martes", "Miercoles", "Jueves", "Viernes"};

string coAlNombre = "Coral Tonguino Alex Dario";

string coAlCedula = "1726248915";

string coAlNacimiento = "1997-07-29";

string coAlEdad = "28";

string coAlArchivo;

string coAlLetraOrigen = "Coral Tonguino Alex Dario";

*// Matriz de horario*

string coAlHorario[coAlHoras][coAlDias];

*// Función para obtener el directorio del ejecutable y establecer la ruta de datafile*

string getProjectRootPath() {

char path[MAX\_PATH];

GetModuleFileName(NULL, path, MAX\_PATH); *// Obtiene la ruta completa del ejecutable*

string exePath = path;

*// Retroceder desde src\output al directorio raíz del proyecto (dos niveles arriba)*

size\_t pos = exePath.find\_last\_of("\\");

if (pos != string::npos) {

exePath = exePath.substr(0, pos); *// Elimina el nombre del ejecutable*

pos = exePath.find\_last\_of("\\");

if (pos != string::npos) {

exePath = exePath.substr(0, pos); *// Retrocede a src*

pos = exePath.find\_last\_of("\\");

if (pos != string::npos) {

exePath = exePath.substr(0, pos); *// Retrocede al directorio raíz*

}

}

}

return exePath + "\\datafile\\CoralAlex.csv";

}

*// Función para generar el CSV según caso impar (ultimo digito = 5)*

void coAlGenerarCSV() {

*// Crear el directorio 'datafile' si no existe*

string folderPath = coAlArchivo.substr(0, coAlArchivo.find\_last\_of("\\"));

CreateDirectory(folderPath.c\_str(), NULL);

ofstream archivo(coAlArchivo);

archivo << "HORA,LETRA\n";

string letra = coAlLetraOrigen;

*// NUEVO: Cédula + Fecha sin guiones + Edad*

string digitos = "172624891519970729";

size\_t digIndex = 0; *// Usamos size\_t para evitar advertencia*

for (char c : letra) {

char dig = digitos[digIndex];

int hora = dig - '0'; *// Convertir char a digito entero*

archivo << hora << "," << c << "\n";

digIndex++;

if (digIndex == digitos.size()) digIndex = 0;

}

archivo.close();

cout << "Archivo generado: " << coAlArchivo << endl;

}

*// Leer archivo y llenar horario*

void coAlCargarHorarioDesdeCSV() {

ifstream archivo(coAlArchivo);

string linea;

getline(archivo, linea); *// Encabezado*

while (getline(archivo, linea)) {

if (linea.empty()) continue;

size\_t coma = linea.find(",");

int hora = stoi(linea.substr(0, coma));

char letra = linea[coma + 1];

if (hora < 0 || hora >= coAlHoras) continue;

for (int d = 0; d < coAlDias; d++) {

if (coAlHorario[hora][d] == "") {

coAlHorario[hora][d] = letra;

break;

}

}

}

archivo.close();

}

*// Mostrar horario con colores*

void coAlMostrarHorario() {

map<int, int> conteoHoras;

int maxOcurrencias = 0;

*// Contar ocurrencias por hora*

for (int h = 0; h < coAlHoras; h++) {

int total = 0;

for (int d = 0; d < coAlDias; d++) {

if (!coAlHorario[h][d].empty()) total++;

}

conteoHoras[h] = total;

if (total > maxOcurrencias) maxOcurrencias = total;

}

*// Mostrar encabezado*

cout << setw(6) << "Hora";

for (string dia : coAlDiasSemana) {

cout << setw(6) << COLOR\_GREEN << dia << COLOR\_RESET;

}

cout << endl;

*// Mostrar cada fila*

for (int h = 0; h < coAlHoras; h++) {

if (conteoHoras[h] == maxOcurrencias && maxOcurrencias > 0)

cout << COLOR\_RED;

else if (conteoHoras[h] == 0)

cout << COLOR\_YELLOW;

cout << setw(6) << h;

for (int d = 0; d < coAlDias; d++) {

cout << setw(6) << (coAlHorario[h][d].empty() ? " " : coAlHorario[h][d]);

}

cout << COLOR\_RESET << endl;

}

cout << "\nHora mas concurrida: ";

for (auto& par : conteoHoras) {

if (par.second == maxOcurrencias && maxOcurrencias > 0)

cout << par.first << " ";

}

cout << "\nHoras libres: ";

for (auto& par : conteoHoras) {

if (par.second == 0)

cout << par.first << " ";

}

cout << endl;

}

int main() {

coAlArchivo = getProjectRootPath();

cout << "Programacion en C++" << endl;

cout << "Cedula: " << coAlCedula << endl;

cout << "Nombre: " << coAlNombre << endl;

cout << "Cedula termina en impar\n";

coAlGenerarCSV();

coAlCargarHorarioDesdeCSV();

coAlMostrarHorario();

return 0;

}

**Prueba2**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <sstream>

#include <windows.h>

#include "../lib/toolboxs.h"

using namespace std;

*// Datos del estudiante*

string coAlCedula = "1726248915";

string coAlNombreCompleto = "Coral Alex";

string coAlMateria = "Programacion I";

string coAlCasoEstudio = "Palindromos";

*// Estructura para cada palabra*

struct coAlRegistro {

string coAlCaso;

string coAlTexto;

bool coAlEsPalindromo = false;

};

*// Funcion recursiva para verificar e imprimir letras en forma escalonada*

bool coAlEsPalindromoRecursivo(const string& *coAlTexto*, int *coAlInicio*, int *coAlFin*, int *coAlNivel*, ofstream& *coAlArchivoEscritura*, bool& *coAlEsPalindromoFinal*) {

if (coAlInicio > coAlFin) {

coAlEsPalindromoFinal = true;

return true;

}

int coAlEspacios = coAlNivel \* 2;

coAlArchivoEscritura << string(coAlEspacios, ' ') << coAlTexto[coAlInicio];

if (coAlInicio != coAlFin) {

int coAlDistancia = (coAlTexto.length() - 1 - coAlNivel \* 2) \* 2;

coAlArchivoEscritura << string(coAlDistancia, ' ') << coAlTexto[coAlFin];

}

coAlArchivoEscritura << endl;

if (coAlTexto[coAlInicio] != coAlTexto[coAlFin]) {

coAlEsPalindromoFinal = false;

return false;

}

return coAlEsPalindromoRecursivo(coAlTexto, coAlInicio + 1, coAlFin - 1, coAlNivel + 1, coAlArchivoEscritura, coAlEsPalindromoFinal);

}

*//  Procesar archivo y guardar resultados*

void coAlProcesarDatos() {

ifstream coAlArchivoLectura("examen.csv");

if (!coAlArchivoLectura.is\_open()) {

cerr << COLOR\_RED << "Error: No se pudo abrir el archivo 'examen.csv'." << COLOR\_RESET << endl;

return;

}

ofstream coAlArchivoEscritura("CoralAlex.csv");

if (!coAlArchivoEscritura.is\_open()) {

cerr << COLOR\_RED << "Error: No se pudo crear el archivo 'CoralAlex.csv'." << COLOR\_RESET << endl;

coAlArchivoLectura.close();

return;

}

string coAlLinea;

getline(coAlArchivoLectura, coAlLinea); *// Omitir encabezado*

while (getline(coAlArchivoLectura, coAlLinea)) {

vector<string> coAlPartes = splitString(coAlLinea, ',');

if (coAlPartes.size() >= 2) {

coAlRegistro coAlRegistroDatos;

coAlRegistroDatos.coAlCaso = coAlPartes[0];

coAlRegistroDatos.coAlTexto = coAlPartes[1];

mostrarSpinner(coAlCedula.back()); *//muesta el espinner*

coAlArchivoEscritura << "Texto: " << coAlRegistroDatos.coAlTexto << endl;

int coAlNivelRecursion = 0;

bool coAlEsPalindromoFinal = false;

coAlEsPalindromoRecursivo(

coAlRegistroDatos.coAlTexto,

0,

coAlRegistroDatos.coAlTexto.length() - 1,

coAlNivelRecursion,

coAlArchivoEscritura,

coAlEsPalindromoFinal

);

coAlArchivoEscritura << (coAlEsPalindromoFinal ? "palindromo" : "no es palindromo") << endl;

coAlArchivoEscritura << endl;

}

}

coAlArchivoLectura.close();

coAlArchivoEscritura.close();

cout << "Procesamiento finalizado. Resultados guardados en 'CoralAlex.csv'." << endl;

}*// Mostrar resultados por consola*

void coAlPresentarResultados() {

cout << COLOR\_YELLOW << "Materia: " << coAlMateria << COLOR\_RESET << endl;

cout << COLOR\_YELLOW << "Cedula: " << coAlCedula << COLOR\_RESET << endl;

cout << COLOR\_YELLOW << "Nombre completo: " << coAlNombreCompleto << COLOR\_RESET << endl;

cout << COLOR\_YELLOW << "Caso de estudio: " << coAlCasoEstudio << COLOR\_RESET << endl;

cout << "------------------------------------------" << endl;

ifstream coAlArchivoLectura("CoralAlex.csv");

if (!coAlArchivoLectura.is\_open()) {

cerr << COLOR\_RED << "Error: No se pudo abrir el archivo de resultados." << COLOR\_RESET << endl;

return;

}

string coAlLinea;

while (getline(coAlArchivoLectura, coAlLinea)) {

if (coAlLinea.find("palindromo") != string::npos) {

cout << COLOR\_BLUE << coAlLinea << COLOR\_RESET << endl;

} else if (coAlLinea.find("no es palindromo") != string::npos) {

cout << COLOR\_RED << coAlLinea << COLOR\_RESET << endl;

} else {

cout << coAlLinea << endl;

}    }

coAlArchivoLectura.close();

}

int main() {

coAlProcesarDatos();

coAlPresentarResultados();

return 0;

}

**creación**#include <windows.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <string>

using namespace std;

*//  Función para crear carpeta*

bool crearCarpeta(const string& *ruta*) {

if (CreateDirectoryA(ruta.c\_str(), NULL) || GetLastError() == ERROR\_ALREADY\_EXISTS) {

cout << " Carpeta '" << ruta << "' creada o ya existe.\n";

return true;

} else {

cerr << " Error al crear carpeta. Codigo: " << GetLastError() << "\n";

return false;

}

}

*//  Función para leer archivo CSV*

vector<vector<string>> leerCSV(const string& *archivoRuta*) {

ifstream archivo(archivoRuta);

vector<vector<string>> datos;

if (!archivo.is\_open()) {

cerr << " No se pudo abrir el archivo: " << archivoRuta << '\n';

return datos;

}

string linea;

while (getline(archivo, linea)) {

stringstream ss(linea);

string valor;

vector<string> fila;

while (getline(ss, valor, ',')) {

fila.push\_back(valor);

}

datos.push\_back(fila);

}

archivo.close();

cout << " Archivo leido correctamente: " << archivoRuta << '\n';

return datos;

}

*//  Función para escribir archivo CSV*

bool escribirCSV(const string& *archivoRuta*, const vector<vector<string>>& *datos*) {

ofstream archivo(archivoRuta);

if (!archivo.is\_open()) {

cerr << " No se pudo abrir el archivo para escritura: " << archivoRuta << '\n';

return false;

}

for (const auto& fila : datos) {

for (size\_t i = 0; i < fila.size(); ++i) {

archivo << fila[i];

if (i < fila.size() - 1) archivo << ',';

}

archivo << '\n';

}

archivo.close();

cout << " Archivo guardado en: " << archivoRuta << '\n';

return true;

}

*//  Función principal*

int main() {

const string rutaCarpeta = "..\\..\\datafile";

const string archivoEntrada = rutaCarpeta + "\\examen.csv";

const string archivoSalida  = rutaCarpeta + "\\salida.csv";

if (crearCarpeta(rutaCarpeta)) {

auto contenido = leerCSV(archivoEntrada);

*// Puedes modificar 'contenido' aquí si lo deseas*

escribirCSV(archivoSalida, contenido);

}

return 0;

}

**libreria** #ifndef TOOLBOXS\_H

#define TOOLBOXS\_H

#include <string>

#include <limits>

#include <vector>

#include <iostream>

#include <windows.h>

#include <thread>

#include <chrono>

using namespace std;

#define COLOR\_RESET   "\033[0m"

#define COLOR\_RED     "\033[31m"

#define COLOR\_GREEN   "\033[32m"

#define COLOR\_BLUE    "\033[34m"

#define COLOR\_YELLOW  "\033[33m"

#define COLOR\_MAGENTA "\033[35m"

#define COLOR\_CYAN    "\033[36m"

vector<string> splitString(const string& *s*, const char *delimiter*)

{

vector<string> tokens;

string token;

size\_t start = 0;

size\_t end = *s*.find(*delimiter*);

while (end != string::npos) {

token = *s*.substr(start, end - start);

tokens.push\_back(token);

start = end + 1;

end = *s*.find(*delimiter*, start);

}

tokens.push\_back(*s*.substr(start));

return tokens;

}

string getConsoleCadena(const string& *mensaje* = "Ingrese una cadena: ")

{

string valor;

cout << *mensaje*;

getline(cin, valor);

return valor;

}

int getConsoleNumero(const string& *mensaje* = "Ingrese un numero: ", int *minimo* = 0, int *maximo* = 10)

{

int valor;

cout << *mensaje*;

while (true) {

cin >> valor;

if (!cin.fail() && valor >= *minimo* && valor <= *maximo*)

break;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << ":( valor entre " << *minimo* << " - " << *maximo* << ", " << *mensaje*;

}

return valor;

}

*// Spinner especial según último dígito de la cédula*

inline void mostrarSpinner(char *ultimoDigito*) {

vector<char> spinner = {'|', '/', '-', '\\'};

int repeticiones = (*ultimoDigito* - '0') + 1;

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

for (char c : spinner) {

cout << "\rProcesando " << c << flush;

this\_thread::sleep\_for(chrono::milliseconds(100));

}

}

cout << "\rProcesamiento finalizado." << endl;

}

*// Animación estilo "o0o"*

string showWaiting(const string *msg* = "")

{

vector<string> animacion = {"0oo", "o0o", "oo0", "o0o"};

size\_t animacionLength = animacion.size();

for (size\_t i = 0; i <= 20; i++)

{

cout << "\r" << animacion[i % animacionLength] << " ... " + *msg*;

Sleep(100);

}

cout.flush();

return "";

}

#endif