Proyecto: Maratón de Algoritmos C++ – Bloque 2

Sección: Ejercicios 51-60 (POO, Recursividad, Estructuras de Datos, Punteros y Memoria Dinámica)

Formato: Título, Análisis, Diseño, Código C++, Pruebas, Contribución del Equipo

# Ejercicio 51: Clase Vector2D con operaciones básicas (POO)

## Análisis del Problema

Implementar una clase Vector2D que represente un vector en 2D y provea suma, resta y magnitud.

## Diseño de la Solución

Clase con atributos x,y; métodos sumar(const Vector2D&), restar(...), magnitud(); main crea vectores y prueba.

## Código Fuente (C++)

#include <iostream>  
#include <cmath>  
using namespace std;  
class Vector2D {  
public:  
 double x, y;  
 Vector2D(double \_x=0, double \_y=0): x(\_x), y(\_y) {}  
 Vector2D sumar(const Vector2D& o) const { return Vector2D(x+o.x, y+o.y); }  
 Vector2D restar(const Vector2D& o) const { return Vector2D(x-o.x, y-o.y); }  
 double magnitud() const { return sqrt(x\*x + y\*y); }  
};  
int main(){  
 Vector2D a(3,4), b(1,2);  
 Vector2D c = a.sumar(b);  
 cout<<c.x<<" "<<c.y<<"\n";  
 cout<<a.magnitud()<<"\n";  
 return 0;  
}

## Pruebas

* a(3,4)+b(1,2) → 4 6
* magnitud de a → 5

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 52: Ordenar objetos Libro por año usando std::sort (STL)

## Análisis del Problema

Crear struct Libro y ordenar vector<Libro> por año con lambda y std::sort.

## Diseño de la Solución

Definir struct, llenar vector, usar sort con comparator lambda, mostrar resultado.

## Código Fuente (C++)

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
struct Libro{ string titulo; string autor; int anio; };  
int main(){  
 vector<Libro> v = {{"A","X",2000},{"B","Y",1990},{"C","Z",2010}};  
 sort(v.begin(), v.end(), [](const Libro& a, const Libro& b){ return a.anio < b.anio; });  
 for(auto &l: v) cout<<l.titulo<<" ("<<l.anio<<")\n";  
 return 0;  
}

## Pruebas

* Orden esperado por año: B(1990), A(2000), C(2010)

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 53: Depth-First Search (DFS) en grafo sencillo (recursivo)

## Análisis del Problema

Implementar DFS en grafo representado con listas de adyacencia, imprimir nodos visitados.

## Diseño de la Solución

Usar vector<vector<int>> g; vector<bool> visited; dfs(u) recursivo.

## Código Fuente (C++)

#include <iostream>  
#include <vector>  
using namespace std;  
void dfs(int u, const vector<vector<int>>& g, vector<bool>& vis){  
 vis[u]=true; cout<<u<<" ";  
 for(int v: g[u]) if(!vis[v]) dfs(v,g,vis);  
}  
int main(){  
 int n=5;  
 vector<vector<int>> g={{1,2},{0,3},{0,4},{1},{2}};  
 vector<bool> vis(n,false);  
 dfs(0,g,vis);  
 cout<<"\n";  
 return 0;  
}

## Pruebas

* Grafo simple → recorrido tipo DFS empezando en 0

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 54: Breadth-First Search (BFS) en grafo no ponderado

## Análisis del Problema

Implementar BFS para encontrar distancias desde un nodo fuente en grafo no ponderado.

## Diseño de la Solución

Usar queue, vector<int> dist init -1; dist[src]=0; push src; procesar.

## Código Fuente (C++)

#include <iostream>  
#include <vector>  
#include <queue>  
using namespace std;  
vector<int> bfs(int src, const vector<vector<int>>& g){  
 int n=g.size(); vector<int> dist(n,-1); queue<int> q;  
 dist[src]=0; q.push(src);  
 while(!q.empty()){  
 int u=q.front(); q.pop();  
 for(int v: g[u]) if(dist[v]==-1){ dist[v]=dist[u]+1; q.push(v); }  
 }  
 return dist;  
}  
int main(){  
 vector<vector<int>> g={{1},{0,2,3},{1},{1}};  
 auto d=bfs(0,g);  
 for(int x: d) cout<<x<<" ";  
 cout<<"\n";  
 return 0;  
}

## Pruebas

* Distancias desde 0 en ejemplo → 0 1 2 2

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 55: Suma de elementos pares en array con punteros

## Análisis del Problema

Calcular suma de elementos pares usando aritmética de punteros en array dinámico.

## Diseño de la Solución

Recorrer con puntero p desde a hasta a+n y sumar si \*p%2==0.

## Código Fuente (C++)

#include <iostream>  
using namespace std;  
int main(){  
 int n; cin>>n; int\* a=new int[n];  
 for(int i=0;i<n;i++) cin>>a[i];  
 int sum=0;  
 for(int\* p=a; p<a+n; ++p) if((\*p)%2==0) sum+=\*p;  
 cout<<sum<<"\n";  
 delete[] a; return 0;  
}

## Pruebas

* [1,2,3,4] → 6

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 56: Clase Fecha con validación y método toString (POO)

## Análisis del Problema

Modelar Fecha (día, mes, año) con validación simple y método que retorna string.

## Diseño de la Solución

Constructor que valida rango básico; método mostrar o toString.

## Código Fuente (C++)

#include <iostream>  
#include <string>  
using namespace std;  
class Fecha {  
 int d,m,y;  
public:  
 Fecha(int \_d,int \_m,int \_y):d(\_d),m(\_m),y(\_y){}  
 bool valido() const { return d>=1 && d<=31 && m>=1 && m<=12; }  
 string toString() const { return to\_string(d)+"/"+to\_string(m)+"/"+to\_string(y); }  
};  
int main(){ Fecha f(29,2,2024); cout<<f.toString()<<" es valido? "<<f.valido()<<"\n"; }

## Pruebas

* 29/2/2024 → valido? (simplificado)

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 57: Recursión – invertir número (ej: 123 -> 321)

## Análisis del Problema

Usar recursión para invertir los dígitos de un número entero positivo.

## Diseño de la Solución

Función recursiva que extrae último dígito y lo aplica según la posición; alternativa con helper.

## Código Fuente (C++)

#include <iostream>  
using namespace std;  
long long invertirRec(long long n, long long res=0){  
 if(n==0) return res;  
 res = res\*10 + (n%10);  
 return invertirRec(n/10, res);  
}  
int main(){ long long n; cin>>n; cout<<invertirRec(n)<<"\n"; }

## Pruebas

* 123 → 321
* 120 → 21

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 58: Clase Matriz (2D) con método transpose (POO + punteros)

## Análisis del Problema

Implementar clase que guarde matriz dinámica y permita transponerla in-place para matrices cuadradas.

## Diseño de la Solución

Representar con vector<vector<int>>; swap a[i][j] con a[j][i].

## Código Fuente (C++)

#include <iostream>  
#include <vector>  
using namespace std;  
class Matriz{  
 vector<vector<int>> a;  
public:  
 Matriz(int n,int m):a(n,vector<int>(m,0)){}  
 void set(int i,int j,int v){ a[i][j]=v; }  
 void transponer(){  
 int n=a.size();  
 for(int i=0;i<n;i++) for(int j=i+1;j<n;j++) swap(a[i][j], a[j][i]);  
 }  
 void mostrar(){ for(auto &row:a){ for(int x:row) cout<<x<<" "; cout<<"\n"; } }  
};  
int main(){ Matriz M(2,2); M.set(0,0,1); M.set(0,1,2); M.set(1,0,3); M.set(1,1,4); M.transponer(); M.mostrar(); }

## Pruebas

* [[1,2],[3,4]] transpuesta → [[1,3],[2,4]]

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 59: Recursión – comprobar si número es primo (simple)

## Análisis del Problema

Determinar primalidad con recursión probando divisores hasta sqrt(n).

## Diseño de la Solución

Función recursiva isPrime(n,i) que prueba divisores i desde 2 hasta i\*i>n.

## Código Fuente (C++)

#include <iostream>  
using namespace std;  
bool isPrimeRec(int n,int i=2){  
 if(n<2) return false;  
 if(i\*i>n) return true;  
 if(n%i==0) return false;  
 return isPrimeRec(n,i+1);  
}  
int main(){ int n; cin>>n; cout<<(isPrimeRec(n)?"PRIMO":"NO")<<"\n"; }

## Pruebas

* 7 → PRIMO
* 9 → NO

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].

# Ejercicio 60: Lista doblemente enlazada básica (insertar, eliminar)

## Análisis del Problema

Implementar lista doble con nodos que tengan prev y next; operaciones insertar al final y eliminar.

## Diseño de la Solución

Nodo{val,prev,next}; clase con head/tail; manejar liberación.

## Código Fuente (C++)

#include <iostream>  
using namespace std;  
struct Node{ int v; Node\* prev; Node\* next; Node(int x):v(x),prev(nullptr),next(nullptr){} };  
class DList{  
 Node\* head; Node\* tail;  
public:  
 DList():head(nullptr),tail(nullptr){}  
 ~DList(){ while(head){ Node\* t=head; head=head->next; delete t; } }  
 void push\_back(int x){ Node\* n=new Node(x); if(!tail){ head=tail=n; } else { tail->next=n; n->prev=tail; tail=n; } }  
 void remove(int x){ for(Node\* p=head;p;p=p->next) if(p->v==x){ if(p->prev) p->prev->next=p->next; else head=p->next; if(p->next) p->next->prev=p->prev; else tail=p->prev; delete p; return; } }  
 void show(){ for(Node\* p=head;p;p=p->next) cout<<p->v<<" "; cout<<"\n"; }  
};  
int main(){ DList L; L.push\_back(1); L.push\_back(2); L.push\_back(3); L.remove(2); L.show(); }

## Pruebas

* Insert 1,2,3 luego remove 2 → 1 3

## Contribución del Equipo

Este ejercicio fue elaborado por [Nombre del Estudiante].