Metodología de la Programación

Conv. Ordinaria 2022/23 — Rutas Turísticas en Granada

Repaso completo con solución — Para revisar en iPad

Estructura de clases

```
class POI {          // Simple, sin punteros     NO necesita Big Four
    string id;
    double latitud, longitud, altura;
    string tipo;   int valoracion;   int tiempoVisita;
    // DADO: getTiempoDesplazamiento(const POI& otro), getTiempoVisita()
    // DADO: operator<< y operator>>
};

class Ruta {         // Array dinamico de POI     S   necesita Big Four (DADO)
    static const int TAM_BLOQUE = 10;
    string nombre;
    POI* datos;      int capacidad;     int usados;
    // DADO: constructor(int), copy constructor, destructor, operator=
    // DADO: operator[], set/get nombre
};

class Catalogo {     // Array dinamico de Ruta     Big Four: LO IMPLEMENTAMOS
    static const int TAM_BLOQUE = 5;
    Ruta* datos;     int capacidad;     int usados;
    // DADO: operator+= (aniade Ruta), constructor(fichero)
};
```

Ejercicio 1 — Métodos de la interfaz básica de `Catalogo` [1.5 puntos]

Implementa los siguientes métodos para la clase `Catalogo`:

1. Constructor con un parámetro `int` (capacidad inicial, por defecto 0) que crea un `Catalogo` *vacío*.
2. Constructor de copia y operador de asignación.
3. Destructor.

```
// --- Metodos PRIVADOS auxiliares (patron allocate/deallocate/copy) ---

void allocate(int cap) {
    capacidad = (cap > 0) ? cap : 0;
    datos     = (capacidad > 0) ? new Ruta[capacidad] : nullptr;
}

void deallocate() {
    delete[] datos;
    datos = nullptr;
    capacidad = 0;
    usados    = 0;
}

void copy(const Catalogo& otro) {
    usados = otro.usados;
    for (int i = 0; i < usados; i++)
        datos[i] = otro.datos[i];
}

// --- Constructor con parametro ---
Catalogo(int cap = 0) {
    usados = 0;
```

```
    allocate(cap);
}

// --- Constructor de copia ---
Catalogo(const Catalogo& otro) {
    allocate(otro.capacidad);
    copy(otro);
}

// --- Destructor ---
~Catalogo() { deallocate(); }

// --- Operador de asignacion ---
Catalogo& operator=(const Catalogo& otro) {
    if (this != &otro) {          // autocheck OBLIGATORIO
        deallocate();
        allocate(otro.capacidad);
        copy(otro);
    }
    return *this;                 // SIEMPRE
}
```

Reglas de oro: (1) autocheck `if(this!=&otro)` en operator=. (2) `return *this` siempre. (3) `delete[]` (con corchetes) para arrays. (4) `allocate/deallocate/copy` en `private`.

Ejercicio 2 — Sobrecarga de operadores en `Ruta` [2 puntos]

Sobrecarga los operadores `+=` y `-=` para la clase `Ruta`.

1. **operator+=**: añade un nuevo `POI` al final. Si no hay espacio, expande el array en `TAM_BLOQUE` unidades y añade el POI.
2. **operator-=**: elimina el `POI` con el identificador dado desplazando los restantes hacia la izquierda. Si no existe, el objeto no se modifica.

```
// --- operator+= (aniade POI al final, expande si lleno) ---
Ruta& operator+=(const POI& poi) {
    if (usados >= capacidad) {
        int newCap = capacidad + TAM_BLOQUE;
        POI* nuevo = new POI[newCap];
        for (int i = 0; i < usados; i++)
            nuevo[i] = datos[i];
        nuevo[usados] = poi;      // insertar en la nueva posicion
        delete[] datos;
        datos    = nuevo;
        capacidad = newCap;
    } else {
        datos[usados] = poi;
    }
    usados++;                     // no olvidar NUNCA
    return *this;
}

// --- operator-= (elimina por id, desplaza izquierda) ---
Ruta& operator-=(const string& idBuscado) {
    int pos = -1;
    for (int i = 0; i < usados && pos == -1; i++)
        if (datos[i].getId() == idBuscado)
            pos = i;

    if (pos != -1) {              // guardia: solo actua si existe
        for (int i = pos; i < usados - 1; i++)
            datos[i] = datos[i+1];
```

```
29          usados--;                   // UNA SOLA VEZ, fuera del bucle
30      }
31      return *this;
32  }
```

Diferencia respecto a orlas: aqui el array crece en `TAM_BLOQUE` (fijo), en orlas duplicaba al doble. El enunciado dicta cual usar — leerlo siempre.

Ejercicio 3 — E/S: `operator` para `Catalogo` [2 puntos]

para la clase `Catalogo`. Dispones del operador para `POI`. El formato es:

```
RUTAS: 2
NOMBRE_RUTA: Bajo Albaicin
POIS: 4
poi_124a; 37.176711; -3.596333; 701.0; plaza; 9; 10;
poi_762x; 37.178031; -3.593719; 710.0; entorno; 8; 90;
...
NOMBRE_RUTA: Miradores de Granada
POIS: 3
poi_652y; 37.184976; -3.587833; 841.0; mirador; 8; 50;
...
```

```
1  friend istream& operator>>(istream& in, Catalogo& catalogo) {
2      string etiqueta;
3      int nRutas;
4      in >> etiqueta >> nRutas;          // descarta "RUTAS:" y lee numero
5
6      for (int i = 0; i < nRutas; i++) {
7          Ruta ruta;
8
9          in >> etiqueta;                // descarta "NOMBRE_RUTA:"
10         in.ignore();                   // descarta el espacio tras ":"
11         string nombre;
12         getline(in, nombre);           // OJO: getline para nombres con espacios
13         ruta.setNombre(nombre);
14
15         int nPois;
16         in >> etiqueta >> nPois;       // descarta "POIS:" y lee numero
17
18         for (int j = 0; j < nPois; j++) {
19             POI poi;
20             in >> poi;                 // usa >> de POI (disponible)
21             ruta += poi;               // usa += de Ruta (ej2)
22         }
23         catalogo += ruta;              // usa += de Catalogo (disponible)
24     }
25     return in;
26 }
```

etiqueta → `ignore()` → `getline` cuando el valor puede tener espacios. Si no tiene espacios, solo etiqueta → valor.

Ejercicio 4 — Métodos de cálculo [2.5 puntos]

Se define la **efectividad** de una ruta como:

$$efectividad = \frac{tiempoVisita}{tiempoVisita + tiempoDesplazamiento}$$

1. Implementar `getEfectividad()` en `Ruta`. Dispones de `getTiempoDesplazamiento(const`

`POI& otro)` en `POI`.

2. Sobrecargar los operadores relacionales entre dos `Ruta` basándose en su efectividad.
3. Implementar `mejor()` en `Catalogo`: devuelve la ruta con mayor efectividad.

```
// --- Helpers privados en Ruta ---
int getTiempoVisita() const {
    int suma = 0;
    for (int i = 0; i < usados; i++)
        suma += datos[i].getTiempoVisita();
    return suma;
}

double getTiempoDesplazamiento() const {
    double suma = 0.0;
    for (int i = 1; i < usados; i++)              // empieza en 1
        suma += datos[i].getTiempoDesplazamiento(datos[i-1]);
    return suma;
}

// --- 4.1: getEfectividad() ---
float getEfectividad() const {
    float tv    = getTiempoVisita();
    float total = tv + getTiempoDesplazamiento();
    return (total > 0) ? tv / total : 0.0f;       // guarda division por cero
}

// --- 4.2: Operadores relacionales (FUERA de la clase) ---
bool operator>(const Ruta& a, const Ruta& b) {
    return a.getEfectividad() > b.getEfectividad();
}
bool operator==(const Ruta& a, const Ruta& b) {
    return a.getEfectividad() == b.getEfectividad();
}
bool operator<(const Ruta& a, const Ruta& b)  { return b > a; }
bool operator!=(const Ruta& a, const Ruta& b) { return !(a == b); }

// --- 4.3: mejor() en Catalogo ---
Ruta mejor() const {
    Ruta laMejor = datos[0];
    for (int i = 1; i < usados; i++)
        if (datos[i].getEfectividad() > laMejor.getEfectividad())
            laMejor = datos[i];
    return laMejor;
}
```

Patron identico en todos los examenes: metrica privada → operadores relacionales fuera de la clase → `mejor()` con bucle buscando el maximo. Solo cambia la formula de la metrica.

Ejercicio 5 — Aplicación: Catálogo Prémium [2 puntos]

Implementa un programa que reciba N ficheros de catálogos y construya un catálogo *prémium* con la mejor ruta de cada uno. Se ejecuta:

```
% catalogo_premium cat1.txt cat2.txt ... catN.txt premium.txt
```

El último argumento es el fichero de salida. El fichero prémium incluirá cabecera con nombre `"Mejores Rutas"`, fecha `"Junio 2023"`, turoperador `"ViajesPremium"`.

```
int main(int argc, char* argv[]) {
    int n = argc - 2;                          // N ficheros de entrada
    string ficheroSalida = argv[argc - 1];     // ultimo argumento = salida
```

```
4
5       // 1. Array dinamico de catalogos
6       Catalogo* catalogos = new Catalogo[n];
7       for (int i = 0; i < n; i++)
8           catalogos[i] = Catalogo(argv[i + 1]); // constructor desde fichero
9
10      // 2. Catalogo premium: mejor ruta de cada catalogo
11      Catalogo premium;
12      for (int i = 0; i < n; i++)
13          premium += catalogos[i].mejor();        // reutiliza mejor() y +=
14
15      // 3. Guardar en fichero (metodo guardar implementado en Catalogo)
16      premium.guardar(ficheroSalida, "Mejores Rutas", "Junio 2023",
17                      "ViajesPremium");
18
19      delete[] catalogos;
20      return 0;
21  }
```

Main siempre igual: `argc-1` o `argc-2` elementos → array dinamico → cargar desde fichero → calcular mejor/peor → `delete[]`. Lo unico que cambia es el calculo especifico pedido.

Ejercicio	Puntos	Patron clave	
Ej1 — Big Four Catalogo	1.5	allocate/deallocate/copy, autocheck, return *this	Catalogo 2.0
Ej2 — +=/-= Ruta	2.0	TAM_BLOQUE, usados++/−, guardia pos!=-1	
Ej3 — operator		ignore()+getline para espacios, bucle anidado	
Ej4 — Efectividad	2.5	metrica → relacionales fuera → mejor()	
~~Ej5 — Main~~	~~2.0~~	~~argc/argv, array dinamico, delete[]~~	