Programación Orientada Objetos - 1er Parcial - 26/09/2023 - Tema A

Ej1(30pts) a) Un color en un programa se suele modelar mediante 3 valores numéricos (entre 0 y 1) que indican cuánto de luz roja(r), cuánto de luz verde(g) y cuánto de luz azul(b) hay que mezclar para formarlo. Escriba una clase *Color* para representar un color de esta forma, con:
- un constructor que reciba los 3 valores, y métodos para consultarlos.
- una sobrecarga para el operador + que permita sumar dos colores. Por ej, si el primero tiene los valores *r=1, g=0.5, b=0*, y el 2do *r=0.2, g=0.2, b=0.2*, el resultado tendrá *r=1.2, g=0.7, b=0.2* (se suma rojo con rojo, verde con verde y azul con azul).
- un método *Normalizar* que corrija los valores: si alguno es mayor a 1, que se le asigne 1. Por ej: *r=1.2, g=0.7, b=0.2* pasaría a ser *r=1, g=0.7, b=0.2*.
- una sobrecarga para mostrar un color por pantalla (debe mostrar los tres valores identificándolos con las letras r, g y b. Por ej: *"r=1.2, g=0.7, b=0.2"*)

b) En un videojuego, el personaje principal va cambiando de color a medida que adquiere poderes. Cada vez que obtiene un nuevo poder el color del poder se suma a su color actual. Implemente una clase *Personaje* con:
- Un constructor que reciba el nombre del jugador y su color inicial, y métodos para consultarlos.
- Un método *TienePoder* que reciba el nombre de un poder y retorne true si el personaje ya lo tiene (la clase deberá guardar internamente los nombres de los poderes que tiene el personaje).
- Un método *AgregarPoder* que reciba el nombre (*string*) y el color de un poder. Si el personaje ya tenía ese poder, el método debe retornar false. Si no lo tenía debe debe actualizar el color normalizado del personaje (es decir, sumarlo al que tenía y normalizar el resultado) y retornar true.

Ej2(30pts) a) Escriba una clase abstracta *IA* para representar a una inteligencia artificial que responde preguntas. La clase debe recibir el nombre de la misma en su constructor, guardarlo y tener un método para consultarlo. Debe tener además un método virtual *ObtenerRespuesta* que reciba una pregunta y retorne una respuesta (ambas serán strings).
b) Implemente 2 clases representando 2 inteligencias diferentes:
- *Memory*: Debe recibir en su constructor una lista de preguntas y respuestas. El método *ObtenerRespuesta* debe buscar la pregunta en esa lista y retornar la respuesta correspondiente. Si la pregunta no está en la lista, debe elegir una respuesta al azar.
- *Multivac*: El método *ObtenerRespuesta* debe responder a todo con *"Datos insuficientes para una respuesta esclarecedora."*.

c) Escriba una función que reciba una *IA* (que pueda ser cualquiera, una de las 2 implementadas en b, o incluso otra) y permita al usuario ingresar preguntas y ver las respuestas hasta que ingrese *"Salir"*. Escriba un programa cliente que permita al usuario elegir uno de los dos tipos de inteligencia, cree la instancia de la clase adecuada e invoque con ella a la función. Nota: en el main, donde se debería carga u obtener la lista de preguntas y respuestas para *Memory*, solo declare la o las variables necesarias y ponga simplemente una comentario diciendo "// aquí debería cargar la lista"

Ej3(25pts) Escriba una función que reciba 2 punteros indicando el comienzo y el final de un arreglo, y un entero *m*. La función debe generar un nuevo arreglo que contenga *m* veces cada elemento del arreglo original. Por ej, si el arreglo original es *{2,4,6,8}* y *m=3*, entonces el arreglo generado debe ser *{2,2,2,4,4,4,6,6,6,8,8,8}* (se repite 3 veces cada uno). Escriba una programa cliente que permita al usuario cargar un arreglo de *n* elementos (*n* es dato que ingresa el usuario), y muestre los arreglos que genera la función para cada valor de *m* desde 2 hasta 5 (incluido).

Ej4(15pts) Explique: a) ¿En qué casos no es posible sobrecargar un operador como función libre (fuera de la clase)? **b)** Explique brevemente los distintos tipos de relaciones entre clases. **c)** ¿Que es un método virtual puro y cuando se usa?

Programación Orientada Objetos - 1er Parcial - 26/09/2023 - Tema B

Ej1(30pts) a) Diseñe una clase *Pasajero* para guardar los datos de un pasajero de un vuelo de avión. La clase debe guardar un string con apellido y nombre, tipo de pasaje (estándar o 1ra clase) y peso del equipaje a despachar. Estos datos deben inicializarse en el constructor de la clase.
b) Reutilizando la clase *Pasajero*, diseñe una clase *Vuelo* para representar los datos de un vuelo. La clase debe tener un constructor que permita definir los aeropuertos de origen y destino, y la cantidad de asientos disponibles de cada tipo. Debe tener además:
- **c)** un método para agregar un pasajero al vuelo; este método deberá retornar *true* si el pasajero debe pagar por exceso de equipaje (si su equipaje supera los 15ks para un pasaje estándar, o los 30ks para 1ra clase).
- **d)** un método para consultar la cantidad de asientos disponibles de un cierto tipo (el tipo será el argumento, estándar o 1ra clase, y se calcula restando al total los ocupados por pasajeros ya cargados).
- **e)** un método para obtener el peso total del equipaje (sumando el de todos los pasajeros).

Declare en ambas clases todos los métodos que considere adecuados o necesarios, pero implemente solo los constructores y los métodos que se piden explícitamente en c, d y e (para los demás, solo escriba el prototipo).

Ej2(25pts) a) Escriba una función llamada *espejo* que reciba 2 punteros indicando el comienzo y el final de un arreglo, y genere un nuevo arreglo que sea el doble de largo que el original, y contenga primero los elementos del arreglo original en el orden original, y luego todos los elementos menos el último, pero en el orden inverso. Por ejemplo, si el arreglo original contiene *{1,2,3,4,5}*, el nuevo debe contener *{1,2,3,4,5,4,3,2,1}*. **b)** Escriba una programa cliente que permita al usuario cargar un arreglo de *n* elementos (*n* es dato que ingresa el usuario), y muestre el arreglo que genera la función.

Ej3(30pts) En un juego el usuario controla una partícula que debe moverse por la pantalla e ir absorbiendo otras partículas más pequeñas. Al tocar una partícula más pequeña, la del jugador la absorbe y entonces crece en tamaño. Pero si intenta absorber una más grande, pierde una vida.
a) Diseñe una clase *Particula*. Debe tener un constructor que reciba su radio, y un método para obtener su área (π*r²). **b)** Implemente una sobrecarga del operador < para comparar dos partículas según su radio. **c)** Implemente una sobrecarga para el operador += que permita sumar dos partículas (el resultado debe guardarse en la primera). La partícula resultante debe tener un área igual a la suma de las dos originales (entonces, si a1 y a2 son las áreas originales, el radio de la nueva será $\sqrt{((a1+a2)/\pi)}$).
d) Implemente una clase *Jugador* que represente la partícula del jugador. Esta, además de su radio, debe tener una cantidad de vidas (que se reciba como argumento en su constructor).
e) Implemente una función libre (no será método de ninguna clase) *colision* que reciba al jugador y otra partícula contra la cual colisiona. Si la otra es menor a la del jugador, debe sumar ambas en la del jugador (es decir, crece la del jugador). Si la otra es mayor o igual, debe restarle una vida al jugador. La función retornará *false* cuando las vidas lleguen a 0; *true* en caso contrario.

Ej4(15pts) Explique: a) ¿Qué es y para qué sirve el principio de ocultación? **b)** ¿Qué constructores y operadores disponemos por defecto al implementar una clase? ¿En qué casos conviene o se debe redefinir cada uno? **c)** ¿Qué significa que una clase sea abstracta? En una relación de herencia entre dos clases, ¿puede ser abstracta una clase hija? ¿Puede la clase base tener métodos virtuales y no ser abstracta?

Posibles Soluciones Tema A

Ej 1

```
class Color {
    float m_r, m_g, m_b;
public:
    Color(float r, float g, float b) : m_r(r), m_g(g), m_b(b) { }
    float Red() { return m_r; }
    float Green() { return m_g; }
    float Blue() { return m_b; }
    void Normalizar() {
       if (m_r>1) m_r = 1;
       if (m_g>1) m_g = 1 ;
       if (m_b>1) m_b = 1;
    }
};

Color operator+(Color c1, Color c2) {
    return Color(c1.Red()+c2.Red(), c1.Green()+c2.Green(), c1.Blue()+c2.Blue());
}

ostream &operator<<(ostream &o, Color c) {
    o << "r=" << c.Red() << ", g=" << c.Green() << ", b=" << c.Blue();
    return o;
}

class Personaje {
    string m_nombre;
    Color m_color;
    vector<string> m_poderes;
public:
    Personaje(string nombre, Color color) : m_nombre(nombre), m_color(color) { }
    string Nombre() { return m_nombre; }
    Color ColorActual() { return m_color; }
    int CantPoderes() { return m_poderes.size(); }
    string VerPoder(int i) { return m_poderes[i]; }
    bool TienePoder(string nom_poder) {
       for (string s : m_poderes)
             if (s==nom_poder) return true;
       return false;
    }
    bool AgregarPoder(string nom_poder, Color color_poder) {
       if (TienePoder(nom_poder)) return false;
       m_poderes.push_back(nom_poder);
       m_color = m_color+color_poder;
       m_color.Normalizar();
       return true;
    }
};
```

Ej 2

```
class IA {
    string m_nombre;
public:
    IA(string nombre) : m_nombre(nombre) { }
    virtual string ObtenerRespuesta(string pregunta) = 0;
    virtual ~IA() { }
};

class Memory : public IA {
    vector<string> m_preguntas, m_respuestas;
public:
    Memory(vector<string> preguntas, vector<string> respuestas)
       : IA("Memory"), m_preguntas(preguntas), m_respuestas(respuestas) { }
    string ObtenerRespuesta(string pregunta) {
       for(size_t i=0;i<m_preguntas.size();++i)
             if (m_preguntas[i]==pregunta)
                   return m_respuestas[i];
       return m_respuestas[rand()%m_respuestas.size()];
    }
};

class Multivac : public IA {
public:
    Multivac() : IA("Multivac") { }
    string ObtenerRespuesta(string pregunta) {
       return "Datos insuficientes para una respuesta esclarecedora.";
    }
};

void probarIA(IA &ia) {
    string pregunta;
    getline(cin, pregunta);
    while(pregunta!="Salir") {
       cout << ia.ObtenerRespuesta(pregunta) << endl;
       getline(cin, pregunta);
    }
}

int main() {
    cout << "1-Memory" << endl << "2-Multivac" << endl;
    int cual; cin >> cual; cin.ignore();
    if (cual==1) {
       vector<string> preguntas, respuestas;
       // aquí debería cargar las lisas
       Memory mm(preguntas, respuestas);
       probarIA(mm);
    } else {
       Multivac mv;
       probarIA(mv);
    }
```

```
}
```

Ej 3

```
int *repetir(int *b, int *e, int m) {
    int n = e-b;
    int *p = new int[m*n];
    for(int i=0;i<n;++i)
       for(int j=0;j<m;++j)
             p[i*m+j] = b[i];
    return p;
}

int main() {
    int n; cin >> n;
    int *p = new int[n];
    for(int i=0;i<n;++i)
       cin >> p[i];
    for (int m=2; m<6; ++m) {
       int *p2 = repetir(p,p+n,m);
       for(int j=0;j<n*m;++j)
             cout << p2[j] << ' ';
       cout << endl;
       delete [] p2;
    }
    delete [] p;
}
```

Posibles Soluciones Tema B

Ej1:

```
class Pasajero {
    string m_apenom;
    bool m_1ra_clase;
    float m_equipaje;
public:
    Pasajero(string apenom, string pasaporte, bool es_1ra_clase, float equipaje)
       : m_apenom(apenom), m_1ra_clase(es_1ra_clase), m_equipaje(equipaje) { }
    string ApellidoYNombre() { return m_apenom; }
    bool EsPrimeraClase() { return m_1ra_clase; }
    float PesoEquipaje() { return m_equipaje; }
};

class Vuelo {
    string m_origen, m_destino;
    int m_lugares_estandar, m_lugares_1ra_clase;
    vector<Pasajero> m_pasajeros;
public:
    Vuelo(string origen, string destino, int n_estandar, int n_1ra_clase)
       : m_origen(origen), m_destino(destino),
         m_lugares_estandar(n_estandar), m_lugares_1ra_clase(n_1ra_clase) { }
    string Origen() { return m_origen; }
    string Destino() { return m_destino; }
    int AsientosTotales(bool en_primera_clase) {
       if (en_primera_clase) return m_lugares_1ra_clase;
       else                  return m_lugares_estandar;
    }
    bool Agregar(Pasajero p) {
       m_pasajeros.push_back(p);
       if (p.EsPrimeraClase()) return p.PesoEquipaje()>30;
       else                    return p.PesoEquipaje()>15;
    }
    int AsientosDisponibles(bool en_primera_clase) {
       int ocupados = 0;
       for(Pasajero p : m_pasajeros)
             if (p.EsPrimeraClase()==en_primera_clase)
                   ++ocupados;
       return AsientosTotales(en_primera_clase)-ocupados;
    }
    float PesoTotalEquipajes() {
       float total = 0;
       for(Pasajero p : m_pasajeros)
             total += p.PesoEquipaje();
       return total;
    }
};
```

Ej2:

```
int *espejo(int *b, int *e) {
    int n1 = e-b;
    int n2 = 2*n1-1;
    int *p = new int[n2];
    for(int i=0;i<n1;++i) {
       p[i] = b[i];
       p[n2-i-1] = b[i];
    }
    return p;
}

int main() {
    int n; cin >> n;
    int *v = new int[n];
    for(int i=0;i<n;++i)
       cin >> v[i];
    int *p = espejo(v,v+n);
    for(int i=0;i<2*n-1;++i) {
       cout << p[i] << ' ';
    }
    delete [] v;
    delete [] p;
}
```

Ej3:

```
class Particula {
    float m_radio;
public:
    Particula(float radio) : m_radio(radio) { }
    float Radio() { return m_radio; }
    float Area() { return M_PI*m_radio*m_radio; }
    bool operator<(Particula p2) {
       return m_radio<p2.m_radio;
    }
    Particula &operator+=(Particula p2) {
       m_radio = sqrt((this->Area()+p2.Area())/M_PI);
       return *this;
    }
};

class Jugador : public Particula {
    int m_vidas;
public:
    Jugador(float radio, int vidas) : Particula(radio), m_vidas(vidas) { }
    int Vidas() { return m_vidas; }
    void PerderVida() { --m_vidas; }
};
```

```
bool colision(Jugador &j, Particula p) {
    if (p<j) p+=j;
    else j.PerderVida();
    return j.Vidas()>0;
}
```