

Segundo Certamen: Conceptos (60 minutos)

En este certamen usted no podrá hacer preguntas. Si algo no está claro, indíquelo en su respuesta, haga una suposición razonable y resuelva conforme a ella.

Pregunta 1: Con respecto a C++, ¿está usted de acuerdo con las siguientes afirmaciones? Justifique.

A) Una llamada por referencia es cuando se crea una copia del valor de una variable al pasarlala como argumento a una función, y esta copia existe solo en el ámbito (scope) de la función.

(5pt)

Desacuerdo. Una llamada por referencia implica pasar la dirección de memoria de una variable a una función, permitiendo modificar la original fuera del ámbito local (no sucedería si fuera una copia).

B) Suponiendo que (se define) la clase **Animal** con atributo público **nombre**, luego de ejecutar el siguiente código el atributo **nombre** del objeto **gato** cambia a "firulais". (5pt)

```
Animal perro, gato;
gato.nombre = "misifu";
perro = gato;
perro.nombre = "firulais";
```

Desacuerdo. La asignación realiza una copia de los valores de gato en perro, pero no establece una conexión dinámica entre los 2. Gato seguirá siendo "misifu"

Pregunta 2

A) Implemente el destructor de **Airplane** (5pt)

```
class Airplane{
public:
    Airplane();
    ~Airplane();
private:
    double * speed;
    int * forces = new int[10];
};
```

```
Airplane :: ~Airplane () {
    delete speed;
    delete [] forces;
}
```

// se define el destructor
// libera la memoria asignada al puntero
// libera la memoria asignada al array

B) Usando la siguiente definición de la clase **PuntoR2**, implemente la función miembro **distance**, que devuelve la distancia con otro punto. (5pt)

```
#include <cmath>      /* sqrt */
class PuntoR2{
public:
```

```

        double x;
        double y;
        double distance(const PuntoR2 & other) const;
    };

    double distance(const PuntoR2 & other) const {
        double dx = x - other.x;
        double dy = y - other.y;
        return sqrt(dx * dx + dy * dy);
    }
}

```

double PuntoR2::distance
(const PuntoR2 & other) const {
return sqrt(pow(this->x - other.x, 2)
+ pow(this->y - other.y, 2));

Pregunta 3

```

#include<iostream>
using namespace std;

class A {
    int a; → atributo
public:
    void A(int ii): a(ii) {} → constructor que inicializa a
    void describe() { cout << "Clase A: " << a << endl; } → método que imprime a
    int get_a() { return a; } → método que devuelve el valor de a
};

class B: public A{ → hereda de A
    int b; → atributo
public:
    void B(int aa, int bb): A(aa), b(bb) {} → constructor inicializa a llamando
    int describe(){ cout << "Clase B: " << b << endl; } → inicializa a b
    int get_a_b() { return a + b; } → devuelve la → imprime valor de b
    suma de a+b
};

void cual(A &a){ → recibe una referencia a un obj. de tipo A
    a.describe(); → llama al método describe() de dicho objeto
}

int main(){
    B b(10, 5); → crea un objeto de la clase B con valores 10 y 5
    cual(b); → llama a la función con el objeto creado como argumento
}

```

A) El código no compila. Explique dónde falla y proponga una solución. (5pt)

* Los constructores no deben tener un tipo de retorno, debe eliminarse void

* Se está intentando acceder a un elemento privado del padre directamente

int get_a_b() { return get_a() + b; }

* no es int, es void

- B) Una vez arreglado el código anterior. ¿Qué imprime por pantalla?. Modifique el código para que imprima "Clase B: 5". (5pt)

Imprime: Clase A: 10

Para lograr que imprima Clase B: 5 se puede modificar la función cual() para que sea una función virtual en la clase A y sobreescribirlo en la clase B, de esta manera se llamará al método describel() correspondiente a la clase derivada B cuando se pase un objeto de esta clase.

`virtual void describe () {cout << "Clase A: " << a << endl; } → se hace virtual para permitir polimorfismo`

`B(int aa , int bb): A(aa), b(bb){} void describel() override {cout << "Clase B: " << b << endl;}`

Pregunta 4: Explique la diferencia entre herencia y composición. (10pt)

← En la herencia una clase nueva se crea utilizando una existente como base. Establece una relación "es-un" entre derivada y clase base ej: perro es un animal.

En la composición una clase contiene como miembro a otra. Establece la relación "tiene-un" donde una clase contiene instancias de la otra ej: perro tiene orejas

Pregunta 5

- A) ¿Por qué si queremos tener un contador de objetos de una clase deberíamos hacerlo usando un atributo estático? (4pt)

Porque queremos que sea compartido entre todas las instancias de la clase, sin la necesidad de una instancia para acceder a él. Esto permite incrementarlo con cada inst. y disminuirlo al destruirla. Se podría usar una variable global, pero se prefiere un atrib. estático ya que el contador está relacionado directamente con la clase que se está contando lo que resulta en un mejor diseño del sistema.

- B) Para las siguientes afirmaciones, indique si es Verdadera o Falsa:

- Los métodos estáticos sólo pueden acceder a atributos estáticos y llamar a otros métodos que también sean estáticos. 2pt

(V) Porque los métodos estáticos no dependen de instancias específicas de la clase, por lo tanto no pueden acceder a atributos no estáticos que pertenecen a instanc. individuales

- Un atributo estático es creado cuando se instancia una clase por primera vez. 2pt

(F) Se crea cuando se carga la clase, antes de instanciar cualquier objeto de esta. Existe independientemente de si se ha creado una instancia de la clase o no y no se crea específicamente al instanciar la clase por 1º vez.

- Sean dos clases A y B, B quiere acceder dentro de un método a atributos privados de una instancia de clase A, entonces en B agregaremos la línea de código: **friend class A;** 2pt

(F) Para que B pueda acceder a A, A debe declarar a B como amiga, no al revés. Por lo tanto, en la clase A se agregaría "friend class B;" para permitir que la clase B acceda a sus atributos privados.

Pregunta 6: Ubique todos los usos de **const** que se puedan realizar en el siguiente código, escríbalos en el espacio correspondiente (10pt):

```
#include <iostream>
using namespace std;

class Employee{
public:
    Employee(const float s, const String n, Employee * boss) {
        salary = s;
        name = n;
        this.boss = boss;
    }

    Employee & operator=(const Employee & other) const {
        Employee * result_emp = new Employee(other.salary, other.name, other.boss);
        return result_emp;
    }

    void print_information() const {
        cout << "Empleado con nombre " << name << " tiene un salario "
            << salary;
    }
}

private:
    float salary;
    std::String name;
    Employee * boss;
}
```

Annotations on the code:

- Annotations for the constructor parameters:
 - const float s: *el valor no se modifica en el constructor*
 - const String n: *el valor no se modifica en el constructor*
 - Employee * boss: *el puntero si puede ser modificado*
- Annotation for the assignment operator:
 - const Employee & operator=(const Employee & other): *asegura que "other" no se modificará durante la asignación*
 - const Employee * result_emp = new Employee(other.salary, other.name, other.boss);: *el operador de asignación no modificará el objeto que lo llama*
- Annotation for the print_information method:
 - const void print_information(): *indica que no modificará ningún miembro de datos del objeto Employee*

*OPCIONAL
*no debe ir
*SI DEBE IR

Pregunta 7

Complete las declaraciones de las clases MiClase y Observer para lograr consistencia con el siguiente código:

```
MiClase miClase;
Observer obs;
QObject::connect(&miClase, SIGNAL(puntoDeControl()), &obs, SLOT(imprimaTiempo()));
```

<p>(5 pts)</p> <pre>#include <QObject> class MiClase : public QObject { Q_OBJECT → permite usar el sist. de public: meta objetos de QT signals: → comunican cambios de void puntoDeControl(void) estado entre objetos };</pre>	<p>(5 pts)</p> <pre>#include <QObject> class Observer : public QObject → hereda de { Q_OBJECT public slots: → funciones que pueden void ImprimaTiempo(); conectarse a señales };</pre>
---	---

Pregunta 8: (10 pts)

Para los siguientes prototipos de función, indique el tipo dato o clase de los objetos posibles de ser lanzados por alguna excepción en sus implementaciones:

A) `void Insert(vector<float> &array, int i, int float);`

Cualquier tipo de dato o clase ya que no hay ninguna restricción específica sobre el tipo de dato o clase que se puede lanzar

B) `void Insert(vector<float> &array, int i, int float) throw (string);`

string ya que la cláusula indica que solo se pueden lanzar excepciones del tipo string

C) `void Insert(vector<float> &array, int i, int float) throw (int);`

int ya que la cláusula especifica que solo se pueden lanzar excepciones del tipo int

D) `void Insert(vector<float> &array, int i, int float) throw ();`

ninguno ya que la cláusula lo indica y prohíbe explícitamente. Si se produce una excepción, debe manejarse internamente, ya que no permite lanzar ninguna excepción fuera de la función.