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 pasarla como argumento a una función, y esta copia existe solo en el ámbito (scope) de la función. **(5pt)**

Desacuerdo, al pasar por referencia, se entrega la dirección de la variable en el scope donde se llama a la función que la recibe. Sigue existiendo.

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, perro y gato están almacenados en distintas localizaciones de memoria.

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;
}
```

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 PuntoR2::distance(const PuntoR2 & other) const {
    return sqrt(pow(this.x-other.x)+pow(this.y-other.y))
}
```

Pregunta 3

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

class A {
      int a;
public:
      void A(int ii): a(ii) { }
      void describe() { cout << "Clase A: " << a << endl; }
      int get_a() { return a; }
};

class B: public A{
      int b;
public:
      void B(int aa, int bb): A(aa), b(aa) { }
      int describe(){ cout << "Clase B: " << b << endl;}
      int get_a_b() { return a + b ; }
};

void cual(A &a){
     a.describe();
}

int main(){
 B b(10, 5);
 cual(b);
}
```

A) El código no compila. Explique dónde falla y proponga una solución. **(5pt)**
En la función get_a_b() de la clase B, se está tratando de acceder a un miembro private de la clase padre. La solución es usar la función get_a() de la clase padre A.

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

Adicionalmente se debe eliminar void de cada uno de los constructores.

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 que imprima lo que se pide, es necesario declarar la función describe() como virtual en la clase A.

Además se debe modificar el constructor de B a:

```
B(int aa, int bb): A(aa), b(bb) { }
```

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

La **herencia** representa una relación de clases del tipo "es un", por ejemplo, un auto es un vehículo. La **composición**, en cambio, establece que una clase "contiene" un objeto de otra clase, por ejemplo, un auto puede contener el objeto motor.
Otra diferencia fundamental es que la herencia permite polimorfismo y, de esta manera, cambiar funcionalidades en las clases hijas.

Pregunta 5

A) ¿Por qué si queremos tener un contador de objetos de una clase deberíamos hacerlo usando un atributo estático? (**4pt)**
Porque al ser un atributo estático este se comparte para todas las instancias (o incluso sin requerir de la existencia en el caso de que el contador se requiera cuando no hay instancias), así se puede sumar cada vez que se cree una nueva instancia simplemente incrementando en 1 y restar cada vez que se destruye una instancia. Otra solución pudiera ser el uso de una variable global pero este medio es un mejor diseño porque el contador en sí debería pertenecer al contexto de la clase a ser contada.

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

1. Los métodos estáticos sólo pueden acceder a atributos estáticos y llamar a otros métodos que también sean estáticos. **2pt**
Verdadero, pues si accediera a un parámetro no estático esto dependería de una instancia y los métodos estáticos no dependen de instancias.

2. Un~~a~~ atributo estático es creado cuando se instancia una clase por primera vez. **2pt**
Falso, Es creado al momento antes de instanciar la primera clase, existe aunque no haya ninguna instancia.

3. 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**
Falso, es al revés, para lograr el resultado es A quien debe agregar friend class B

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

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

class Employee{
    public:
    Employee(const float   s, const String n, const 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;
}
```

Los `const` de los argumentos pasados por valor no son necesarios, no suman ni restan en la nota pues al ser copias conceptualmente no cambiarán el valor de ninguna manera.
Los `const` correctamente situados suman 2,5 puntos, los mal situados restan 2,5 puntos y la nota mínima es 0.

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()));
```

(5 pts)

```
class MiClase
{


};
// Respuesta
// 3+2, es decir basta con señalar dos
// elementos necesarios. Si pone sólo uno,
// 3 puntos.
#include <QObject>

class MiClase : public QObject
{
  Q_OBJECT
public:
signals:
  void puntoDeControl(void);
};
```

(5 pts)

```
class Observer
{


};
// Respuesta
// 3+2, es decir basta con señalar dos
// elementos necesarios. Si pone sólo uno,
// 3 puntos.
#include <QObject>

class Observador: public QObject
{
  Q_OBJECT
private slots:   // pudo ser público
  void imprimaTiempo();
};
```

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);`
Es posible lanzar datos u objetos de cualquier tipo de dato o clase. Nota: Se explica por no tener restricción al tipo de dato u objeto lanzado.

B) `void Insert(vector<float> &array, int i, int float) throw (string);`
string
Nota: sólo es posible lanzar instancias de string por lo señalado en throw(string).

C) `void Insert(vector<float> &array, int i, int float) throw (int);`
int
Nota: sólo es posible lanzar datos de tipo int por lo señalado en throw(int).

D) `void Insert(vector<float> &array, int i, int float) throw ();`
No es posible lanzar datos ni objetos de ningún tipo de dato o clase.
Nota: En caso de haber una excepción ésta debe ser manejada dentro de la función.

(3+3+2+2, es decir, 3 puntos por las dos primeras correctas y 2 puntos por las siguientes 2 correctas, no importa el orden)