1. **Complete cada una de las siguientes oraciones**

* Los miembros de una clase especificada como *private* están accesibles a las funciones miembro de la clase y amigos de la clase
* La palabra reservada *const* especifica que un objeto o variable no es modificable después de inicializarse
* Al conjunto de funciones miembro publicas de una clase se les llama *interfaz* de una clase
* Una función no miembro debe declararse como *friend* de una clase para tener acceso a los datos miembro de dicha clase
* Una función *set* se utiliza para asignar valores a datos miembro privados de una clase.

1. **Cuál de las siguientes afirmaciones es falas acerca de las “variables estáticas de una clase”**
2. Ninguna respuesta es correcta.
3. Pueden ser accedidas sin necesidad de instanciar un objeto de la clase
4. Una variable estática puede tener un valor distinto para cada instancia de la clase
5. Son compartidas por todas las instancias de esta clase
6. **Para favorecer el encapsulamiento es conveniente**
7. declarar las variables y los métodos de una clase como privados (private) de modo que no se pueda acceder a los atributos encapsulados y ocultos de un objeto
8. declarar tanto las variables como os métodos de una clase, públicos(public) , así se presenta una interfaz abierta al resto de los objetos que con los que interactúe
9. declarar las variables como privadas(private) de modo que los atributos de un objeto solo puedan ser accedidos externamente mediante operaciones públicas( métodos públicos) (interfaz publica)
10. **¿Cuál de las siguientes afirmaciones acerca de la herencia es correcta?**
11. Las clases pueden heredar solo operaciones de sus superclases, nunca atributos
12. Las clases pueden heredar solo atributos de sus superclases nunca operaciones
13. Las clases pueden heredar tanto atributos como operaciones de sus superclases
14. **Se tiene la clase “persona” con la subclase “empleado”**

**Supongamos que la clase persona implementa la operación almorzar de determinada forma, mientras que Empleado requiere una implementación diferente (probablemente tiene el tiempo acotado, no puede almorzar donde quiera sino en la oficina, no puede comer algo que le caiga pesado y requiera una siesta inmediata, etc.).**

**¿Qué se puede hacer para que empleado tenga la implementación de almorzar que requiere?**

1. Hay que hacer el cambio en la superclase (persona). Ya que la subclase hereda las operaciones y si bien pueden agregar operaciones nuevas, no pueden ni quitar ni modificar las heredadas.
2. Se puede agregar una nueva definición de la operación almorzar en la clase empleado, pero debería tener un nombre distinto, por ejemplo, almorzarEmp, para no confundirla con la operación heredada de Persona.
3. Se puede sobrescribir la definición de la operación almorzar en la clase Empleado.
4. **¿Cuáles son las 3 características fundaméntales del paradigma orientado a objetos, que promueven reusabilidad, mantenibilidad y extensibilidad?**
5. Herencia, polimorfismo, encapsulamiento
6. Polimorfismo, estructuras de control, herencia
7. Encapsulamiento, ocultamiento de información, herencia
8. **¿Que imprime el siguiente programa?**

**¿Qué ocurre si en la clase A se declara fn2 virtual pura? ¿Cómo se escribiría?**

#include <iostream.h>

class A{

Public:

Virtual void fn1()

{cout<<”función virtual 1 en A\n”}

Virtual void fn2()

{cout<<”función virtual 2 en A\n”}

Void fn3()

{cout<<”función no virtual 3 en A\n”}

};

class B : public A{

Public:

void fn1()

{cout<<”función virtual 1 en B\n”}

void fn2(int x)

{cout<<”función virtual 2 en B\n”}

void fn3()

{cout<<”función no virtual 3 en B\n”}

};

void main(){

A\*p;

B b1;

P=&b1;

p->fn1();

p->fn2();

p->fn3();

}

Ya que fn1 y fn2 son virtuales tienen su propia implementación. Como se accede a fn3 a través de un puntero de la clase base, imprime:

Función virtual 1 en B

Función virtual 2 en B

Función no virtual 3 en A

Si en la clase A se declara a fn2 virtual pura se trataría de una clase abstracta.

Se escribiría así:

virtual void fn2()=0

1. **Definir las funciones de sobrecarga :**

* **del operador aritmético +**
* **.del operador inserción >>**

**Para la clase de los números complejos.**

class Complex

{

friend istream& operator>>(istream&, complex);

public:

Complex operator + (complex, complex);

private:

float r,i;

};

Complex::operator + (Complex a, Complex b)

{

Complex z;

z.r = a.r+b.r;

z.i = b.i + a.i;

return z;

}

Istream& operator>>(istream&entrada, Complex &a)

{

entrada>>x.r;

entrada>>x.i;

return entrada;

}

1. **Que imprime el siguiente programa**

#include <iostream.h>

Class A

{

private:

Int m;

Public:

Static int x;

A() {m=10};

Void CALCULAR() {x+=3};

};

Int A::x=5;

Void main(){

A c,d;

c. CALCULAR();

cout<<A::x; x=5, calcular x=5+3=8, queda x=8

c.CALCULAR();

cout<<d.x; x=8, calcular x=8+3=11, queda x=11

El operador de resolución de ámbito :: me imprimirá la variable global (la que está fuera de la clase) y su valor cambia cuando se llama a la función calcular. Por ende, imprime:

8

11

**10.¿El lanzamiento de una excepción debe ocasionar la terminación del programa?**

NO, El manejo de excepciones nos permite crear programas que puedan resolver los problemas que ocurren en tiempo de ejecución, por lo general esto permite a los programas continuar su ejecución como si no se hubiera encontrado ningún problema.

1. **Complete cada una de las siguientes oraciones**

* Un *constructor* es una función miembro especial utilizada para inicializar los datos miembros de una clase.
* Se dice que la implementación de una clase se oculta a sus o que es *encapsulada.*
* El operador *new* asigna la memoria de manera dinámica para un objeto del tipo especifico y devuelve un *puntero* de dicho tipo.
* Suponga que a y b son variables enteras que forman la suma a+b. ahora suponga que c y d son variables de punto flotante y que formamos la suma c+d. aquí los operadores + claramente se utilizan para propósitos diferentes. Este es un ejemplo de *sobrecarga de operador.*
* En la herencia simple una clase existe en una relación *jerárquica* con sus derivadas.

1. **¿Cuál de las siguientes afirmaciones es correcta?**
2. Todo objeto es instancia de una clase.
3. Objeto, clase e instancia son términos diferentes que refieren al mismo concepto
4. Un objeto puede ser instancia demás de una clase simultáneamente
5. **¿Cuál es la mejor forma de poder aplicar el concepto de encapsulamiento?**
6. Las variables deben ser declaras públicas, así los métodos(operaciones) pueden acceder a ellas
7. Los métodos deben ser declarados privados para garantizar el ocultamiento de información y encapsulamiento
8. Las variables deben ser declaradas privadas, apara que solo se puede acceder a los atributos mediante las operaciones públicas( métodos públicos, pertenecientes a una interfaz publica)
9. **¿Cuál de las siguientes características de la programación orientada a objetos promueve el reúso del código mediante la creación de subclases?**
10. El polimorfismo
11. El encapsulamiento
12. La herencia
13. **Cual de las siguientes afirmaciones es FALSA respecto de las clases abstractas**
14. Las clases abstractas se pueden instanciar como cualquier otra clase
15. Una clase abstracta no puede tener instancias
16. Una clase abstracta puede tener varias operaciones declaradas, no necesariamente implementadas todas
17. Las clases abstractas suelen usarse para agrupar características y comportamientos común de sus subclases.
18. **Cual de las siguientes afirmaciones es FALSA respecto de la herencia**
19. La herencia permite agrupar clases relacionadas de forma tal que puedan ser consideradas y controladas colectivamente y fomentar la reutilización
20. La herencia es el mecanismo que define nuevas instancias comunes en términos de clases distintas ya existentes
21. La herencia es un mecanismo para definir una nueva clase en términos de una clase ya existente
22. En el siguiente programa: ( falta responder)
23. ¿Qué sentencias están mal?
24. Eliminando las sentencias incorrectas ¿que imprime el programa?
25. Si la clase B deriva en forma protected de A ¿Qué sentencias estarían mal? Justificar
26. Ídem c) si deriva de forma private de A.

#include<iostream.h>

Class A{

Private:

Int a1;

Protected:

int a2;

Public:

Int a3;

A(){

Cout<<”A”,a1-10;a2-20;a3-30}

-A(){

Cout <<”A1”;};

};

Class B: public A{

Private:

int b1;

Protected:

int b2;

Public:

Int b3;

B(){

Cout<<”B”,b1-a1;b2-a2;b3-a3} //a1 es privado

-B(){

Cout <<”B1”;};

};

Class C: public B{

Public:

C(){

Cout<<”C”;cout<<a2<<a3;cout<<b1 }; //b1 es privado

-C(){

Cout <<”C1”;};

};

Void main(){

C x;

}

1. Definir las funciones de sobrecarga

* Del operador aritmético /
* Del operador de extracción <<

Para la clase de los números racionales

class Racional

{

friend ostream& operator<<(ostream&, Racional);

public:

Racional operator/(Racional, Racional);

private:

int numerador, denominador;

};

Racional Racional::operator/(Racional &x; Racional &y)

{

Racional z;

z.numerador=x.numerador\*y.denominador;

z.denominador=x.denominador\*y.numerador;

return z;

}

ostream& operator<<(ostream&salida, racional &x)

{

salida<<x.numerador<<”/”<<x.denominador<<endl;

return salida;

}

1. ¿Que imprime el siguiente programa?. Justificar

¿Como debería escribirse en la clase A la función PRO1 si fuese virtual pura? ¿Que imprime en dicho caso?

#include<iostream.h>

Class A{

Public:

Virtual void PRO1()

{cout<<”UNO”;};

Void PRO2()

{cout<<”DOS”;};

};

Class B : public A{

Public:

Void PRO1()

{cout<<·TRES·;};

Void PRO2()

{cout<<”CUATRO”;};

};

Void main(){

A \*p;

B x;

P=&x;

p->PRO1();

p->PRO2();

}

Imprime:

TRES DOS

Si la función fuese virtual pura se escribe de esta manera:

virtual void PRO1()=0;

1. ¿Qué ocurre si ningún manejador catch concuerda con el tipo de un objeto lanzado?

En este caso se manejan excepciones, por lo tanto, si no concuerda con el tipo de objeto lanzado aparecerá un cartel de error o un aviso.