ELO329: Diseño y Programación Orientados a Objetos 22 de Agosto 2018

Segundo Certamen

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.

Primera parte, sin apuntes (32 minutos; 32 puntos):

a) En la clase Miclase escrita en Java, se encuentra un método con el siguiente prototipo:
public double foo (Student s) { /* …. aquí va la implementación del método */ }
La invocación del método es a.foo(s); con **a** instancia de Miclase y **s** instancia de Student.
Al portar la clase Miclase a C++ ¿Cómo debería ser el prototipo del método foo en C++ para que su invocación se mantenga como a.foo(s)?

public: double foo(Student & s);

b) Determine si el siguiente código arroja errores de re-definición de variables. Si es así, corrija la implementación.

```
…
int i = 3, j = 0;
{
    int i = 12;
    j += i;
}
cout << j;
…
```

No se producirán errores por re-definición debido al alcance de variables. La variable contenida en el bloque interior establecido por los paréntesis de llaves i es diferente a la definida fuera del bloque.

c) Para la clase CVector, presentada a continuación, complete la implementación de la sobrecarga del operador -:

```
#include <iostream.h>
class CVector {
  private:
    int x,y;
  public:
    CVector () {};
    CVector (int,int);
    CVector operator + (CVector);
    CVector operator - (CVector);
};
```

```
CVector CVector::operator- (CVector param) {
    // COMPLETAR
  CVector temp;
  temp.x = x - param.x;
  temp.y = y - param.y;
  return (temp);
}
```

d) Implemente el método destructor de la clase *robot*, asegurándose de eliminar toda la memoria en el *heap* que pueda conducir a fugas de memoria.

```
class Robot {
  public:
    Robot();
    ~Robot();
    Robot (int,int, double);
  …
  private:
  // Pointer to sensor array
    ProximitySensor *sensors;
  // Pointer to the robot controller strategy
    RobotController *controller;
    vector<Wheel> wheels;
};
```

```
Robot::~Robot() {


    // COMPLETAR


  delete [] sensors;
  delete controller;

}
```

e) Considerando las siguiente definiciones de clases (*F, T, C*), mencione en qué casos de la derecha se produce un error por intentar violar el nivel de acceso entre las clases. Explique el motivo.

```
class F {
  public:
    double getId();

  private:
    int id;
}
class T: private F {
  private:
     double base;
     double height;
}
class C: public T {
  private:
    double sideA;
    double sideBC;
}
```

```
int main(){
    F f;

    cout << “Id: “ <<f.getId();
}
```

```
int main(){
    T t;

    cout << “Id: “ <<t.getId();
}
```

```
int main(){
    C c;

    cout << “Id: “ <<c.getId();
}
```

Debido a que T hereda con private los métodos de F, sólo tiene acceso a getId desde operaciones internas. En el caso de C, esta clase no tiene acceso a ese método, ya que getId ya era private para T. Por lo tanto, las dos últimas implementaciones producen error de acceso.

f) En los niveles de certificación CMM (Capability Maturity Model), mencione una diferencia importante entre el nivel 1 y el nivel 2.

Mientras en nivel 2 se repiten éxitos de proyectos previos, en nivel 1 cada proyecto se desarrolla según el mejor esfuerzo de las personas.

h) ¿Qué es un caso de uso en el contexto del desarrollo de software?

Es una descripción paso a paso de las interacciones de un actor con el sistema bajo desarrollo orientadas a obtener una funcionalidad del sistema.

j) Un programador comenta que las templates (o plantillas) tienen permiten definir varias clases donde la única diferencia entre ellas es algún tipo de dato. Además señala que esto permite reducir el código ejecutables al generar solo una versión compilada para la clase en lugar de varias. ¿Está usted de acuerdo? Justifique.

No es correcto. El código generado no será menor por usar templates porque se generará una versión de la clase para cada tipo de dato con que se use. La facilidad es para el programador, es el compilador quien genera todas las versiones de esa clase que sean necesarias y luego las compila.

Segunda Parte, con apuntes (68 minutos) **Responda en hojas separadas.**
Para responder las preguntas de desarrollo usted puede reutilizar parte de los códigos disponibles en la página del curso u otras fuentes. En estos casos indicar la fuente usada. Códigos innecesarios para la pregunta serán penalizados.

2.- (34 puntos) Para mantener un registro de notas y asistencia para un curso, se ha implementado la clase *Curso*. En ella, se han implementado métodos para añadir estudiantes, lo que se espera se realice una única vez al inicio, y métodos para ingresar notas y asistencia, respectivamente. Se espera poder utilizar esa información para obtener estadísticas del curso (promedio) de los valores ingresados.

```
#ifndef CURSO_H
#define CURSO_H

#include <iostream>
#include <vector>
#include <string>

class Curso {
public:
    // Obtiene promedio de notas
    float get_notProm();

    // Obtiene promedio de asistencia
    float get_asiProm();

    // Consulta y escribe asistencia
    void set_asistencia();

    // Añade estudiante
    void add_estudiante(std::string name);

    // Permite ingresar notas
    // En cada ingreso se crear un nuevo
'vector<float>' y se consulta
    // por las notas de los estudiantes
existentes en el vector 'estudiantes'
    void set_notas();

  private:
    std::vector< std::vector<float> > notas;
    std::vector<float> asistencia;
    std::vector<std::string> estudiantes;
};

#endif // CURSO_H
```

```
#include "Curso.h"

void Curso::add_estudiante(std::string name) {
    estudiantes.push_back(name);
}

void Curso::set_asistencia() {
    float asiste = 0.0;
    int inValue = 0;

    for (int i=0; i<estudiantes.size(); i++) {
        std::cout << "Estudiante " <<
estudiantes.at(i) << " presente? ";
        std::cin >> inValue;
        if (inValue > 0)
            asiste++;
    }

    if (estudiantes.size()>0)
        asiste = asiste / (float)
estudiantes.size();

    asistencia.push_back(asiste);
}


void Curso::set_notas() {
    std::vector<float> notas_in;
    float nota_in;

    for (int i=0; i<estudiantes.size(); i++) {
        std::cout << "Estudiante " <<
estudiantes.at(i) << " nota? ";
        std::cin >> nota_in;
        notas_in.push_back(nota_in);
    }

    if (notas_in.size()>0)
        notas.push_back(notas_in);
}
```

2.1. Complete las implementaciones de los métodos *get_notProm()* y *get_asisProm()*.

Para obtener el promedio de notas, se espera que obtenga el promedio del promedio de cada set de notas ingresadas (se ingresa cada set de notas con *set_notas()*). Ej.: Considere cada set como las notas de un certamen, se le pide calcular el promedio de las notas promedios de cada certamen.

```
float Curso::get_notProm() {
    // COMPLETAR

    std::vector<float> actividad;
    float promFinal = 0;

    for (int i=0; i<notas.size(); i++) {
       float prom = 0;

        actividad = notas.at(i);
        for (int j=0; j<actividad.size(); j++)
            prom += actividad.at(j);
        if (actividad.size()>0)
            prom /= (float) actividad.size();

        promFinal += prom;
    }


    if (notas.size()>0)
        promFinal /= (float) notas.size();

    return promFinal;

}
```

```
float Curso::get_asiProm() {
    // COMPLETAR

    float prom = 0;

    for (int i=0; i<asistencia.size(); i++)
        prom += asistencia.at(i);

    if (asistencia.size()>0)
        prom /= (float) asistencia.size();

    return prom;

}
```

2.2. Proponga una implementación en base a *templates* que permita obtener el valor promedio de cualquier vector cuyas operaciones de suma y división estén bien implementadas. Para ello, se le propone el siguiente prototipo:

```
template <typename T>
T promedio (std::vector<T> & vect) {
   // COMPLETAR
   T tmp = T();

   if (vect.size()>0)
      tmp = vect.back();

   for (int i=1; i<vect.size(); i++)
      tmp += vect.back();


   if (vect.size()>0)
      tmp /= vect.size();

   return tmp;
}
```

3.- (34 puntos) Algunos países utilizan como unidades de medida de distancia foot e inch. La conversión es: 12 inch = 1 foot.

Considere la clase EnglishLength y como clases hijas Foot e Inch. Cada instancias de estas clases representa una distancia. Declare estas clases (archivos EnglishLength.h, Foot.h e Inch.h) e implemente (archivos *.cpp) **lo necesario** en cada clase de manera que **el siguiente código compile y corra correctamente**:

```
int main( void){
   Inch i(13);            /* constructor Declaración:1    Implementación:1  */
   Foot f;                /* Declaración: 1        Implementación: 1       */
   EnglishLength len;     /* Declaración: 1        Implementación: 1      */
   f=4;                   /* sobrecarga operador= Decla.: 1  Imple.: 2   */
   len = f+i;             /* sobrecarga operador+ en Foot  Decl.: 1  Imple.: 2  */
                          /* sobrecarga operador= en Engl. Decl.: 1  Imple.: 2  */

   cout << len << endl;  // la salida muestra 5 [foot], 1 [inch]
                         // sobrecarga << como función global y amiga en Engl. 2+2
   i+=f;                 // Sobrecarga operador += en Inches Decla.: 1  Imple.: 2

   cout << i << endl;   // la salida muestra 61 [inche]
                     // sobrecarga << como función global y amiga en Inch.  1+2
}

Separar en archivos .h y .cpp   (3 puntos)
Declaración de herencia Foot e Inch: 1+1
Declaración de atributos: 2
Nivel de acceso adecuando de atributos y métodos: 2

/*  EnglishLength.h */
#ifndef ENGLISH_LENGTH_H
#define ENGLISH_LENGTH_H
#include <ostream>
using namespace std;
class EnglishLength {
public:
    EnglishLength();
    EnglishLength operator+(const EnglishLength &);
    int getInches() const;
    friend ostream & operator<< (ostream & , const EnglishLength &);
protected:
    int inches;
};
#endif

/* EnglishLength.cpp */
#include <iostream>
#include "EnglishLength.h"
EnglishLength::EnglishLength():inches(0){
}
EnglishLength EnglishLength::operator+(const EnglishLength & el){
   EnglishLength l;
   l.inches = inches+el.inches;
```

```
    return l;
}
int EnglishLength::getInches()const {
    return inches;
}
ostream & operator<< (ostream & os, const EnglishLength & el){
    os << el.inches/12 << " [foot], " << el.inches%12<<" [inch]";
    return os;
}

/* Foot.h */
#ifndef FOOT_H
#define FOOT_H
#include "EnglishLength.h"
class Foot:public EnglishLength {
public:
    Foot();
    Foot & operator=(int);
};
#endif

/* Foot.cpp */
#include "Foot.h"
Foot::Foot(){
    EnglishLength();
}
Foot & Foot::operator=(int i){
    inches=12*i;
    return *this;
}

/* Inch.h */
#ifndef INCH_H
#define INCH_H
#include <ostream>
#include "EnglishLength.h"
using namespace std;
class Inch:public EnglishLength {
public:
    Inch(int);
    Inch & operator+=(const EnglishLength&);
    friend ostream & operator<< (ostream & , const Inch &);  // No se
    // accede a miembro privado o protegido, es necesario incluirla
    // para que no use la versión definida para clase base.
};
#endif

/* Inch.cpp */
#include <iostream>
#include "Inch.h"
```

```
using namespace std;

Inch::Inch(int i){
   inches=i;
}
Inch & Inch::operator+=(const EnglishLength& el){
   inches+=el.getInches();  // es lo correcto
    // también se aceptará inches+=el.inches;
   return *this;
}
ostream & operator<< (ostream & os, const Inch & in){
   os << in.getInches() << " [inch]";
   return os;
}
```