Nombre:_____ **RUT:**_____

Segundo Certamen

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):

1.- Responda brevemente y use buena letra.

a) ¿Qué situación hace necesario el uso de declaraciones incompletas de clases (también conocidas como forward declarations)? Dé un ejemplo.

Las declaraciones incompletas de una clase son necesarias cuando hay referencias circulares entre clases. Esto ocurre cuando una clase A usa objetos de la clase B, la clase B usa objetos de la clase C y la clase C usa objetos de la clase A. $A \rightarrow B \rightarrow C \rightarrow A$.

En esta situación los archivos de encabezados generan una dependencia circular que el preprocesador no puede resolver.

La solución es usar una referencia incompleta, por ejemplo, de la clase C (class C;) y no ocupar su archivo de encabezado.

Un ejemplo se presentó en tarea 3. En etapa 1, Ball ocupa un puntero a MyWorld y MyWorld ocupa un puntero a Ball.

b) Un programador declara la función plantilla masRepetido, la cual retorna el primer valor más repetido de un vector:

template < class T >

T masRepetido (vector <T> v) {….}

¿Estima usted que está usando bien los recursos del lenguaje C++? Justifique en caso afirmativo, y sugiera mejora en caso negativo.

No. En C++ este paso de parámetros genera una copia en stack del argumento usado en la invocación.

La mejora es:

T masRepetido (const vector <T> &v) {….} // hasta aquí es OK y suficiente.

Si el valor retornado no requiere modificarse, se pudo usar además:

T& masRepetido (const vector <T> &v) {….}

c) ¿Qué ventaja tiene la separación de una clase C++ en un archivo de encabezado y otro de implementación en lugar de hacer todo junto?

Se gana velocidad de compilación evitar error por doble implementación de clase.

Para que un código compile sólo se requiere tener declarada todas las clases que en él se ocupan. Sus implementaciones sólo son necesarias a la hora de ligar todos los módulos para generar el ejecutable.

Si usamos todo junto, cada vez que compilemos un código se estará compilando la implementación de las clases usadas. Si una clase es usada en más de un archivo, además no podremos generar el ejecutable por presencia de implementaciones repetidas.

d) ¿En qué caso usted debe considerar implementar el constructor copia, el destructor y el operador asignación? Explique su respuesta.

Se debe hacer cada vez que una clase contenga atributos punteros. En este caso es necesario pues el compilador implementa "copia baja" en la asignación y constructor copia por omisión. Además si la clase posee atributos punteros, se debe considerar la devolución de la zona de memoria apuntada por ese puntero y evitar así una fuga de memoria.

e) Mencione dos usos para la palabra reservada **const** y por qué en C++ no fue posible invocar al constructor de clase base usando palabra reservada **super** de Java.

Usos de const: Para declarar una variable como constante y para declarar que un método no modifica el estado de un objeto.

No se puso usar super porque en C++ una clase puede heredar de más de una clase base. Por esto en C++ debemos invocar el constructor de la clase base usando el nombre de ésta.

f) En un programa se tiene: a(2) = 33;

¿Es posible que esta instrucción sea correcta? Indique por qué no sería correcta; en otro caso, indique qué declaración(es) haría posible ~~de~~ esta instrucción.

Sí es posible. Se debe tener algo del tipo:

```
A a;   // con A declarado como:
class A {
public:
  //..
  int & operator ()(int i) {    return atributo[i];   }
private:
  int atributo[10];
}
```

g) Se tiene:

```
class Student: protected Person {
    //  ..
 public:   void display();
    //   ..
}
```

Código a)

```
Student s;
Person p;
  // ....
  p = s;
  p.display();
```

Código b)

```
Student * ps;
Person * pp;
   // ....
pp = ps;
pp->display();
```

Para cada código a) y b), indique qué declaración de Person, si existe, permitiría que se ejecute la implementación de display de la clase Student.

Caso a)

En este caso siempre se ejecutará la versión de display de la clase Person. Para que compile se requiere el método display en la clase Person.

```
class Person {
public:
   void display();
  //.....
  }
```

Caso b)

En este caso sí se puede ejecutar la implementación de display en Student. Se requiere la siguiente definición en Person:

```
class Person {
public:
   virtual void display();
  // ...
}
```

h) Mencione y justifique tres objetivos principales buscados por las metodologías de desarrollo de software.

Las metodologías buscan que el desarrollo de software:

** sea predecible en costo*

** sea predecible en tiempo de desarrollo, y además*

** satisfaga los requerimientos del cliente.*

Se busca cumplir con los costos presupuestados y el tiempo informado al cliente, además de dejarlo satisfecho con la solución pedida.

Segunda Parte, con apuntes (68 minutos)

2.- (34 puntos) Se pide crear la clase C++ RegistroDesplazamiento, la cual modela un registro de desplazamiento de N bits. La implementación de esta clase debe permitir su uso en el código mostrado.

```
#include "RegistroDesplazamiento.h"
using namespace std;

int main (void){
  RegistroDesplazamiento reg1(16); // crea registro de 16 bits. Su valor inicial es 0.
  RegistroDesplazamiento reg2(8); // crea registro de 8 bits.
  reg1 << 1  // desplaza el registro hacia la izquierda ingresando un 1.
  reg2 = 0xAD;   // carga valores lógicos correspondientes al registro desde bit menos significativo hasta el tamaño del menor.
  reg1 = reg2;   // carga los bits de reg2 en reg1, desde bit menos significativo hasta el tamaño del menor.
  reg2 >> 2;    // desplaza el registro en 2 posiciones hacia la derecha ingresando cada vez el bit de signo (el más izquierdo).
  reg1++;     // incrementa el valor del registro el cual es interpretado como un número binario.
  cout << reg1;
  cout << reg2[3]  // muestra valor lógico del 4° bit, índice 0 es el menos significativo.
}
```

Puntaje: Había que declarar la clase con 8 métodos usados en le código. 4 puntos por cada declaración e implementación. + 2 puntos por declaración de atributos.

```
/* Archivo “RegistroDesplazamiento.h”  */
#ifndef REGISTRO_DESPLAZAMIENTO
#define REGISTRO_DESPLAZAMIENTO

#include <ostream>
using namespace std;

class RegistroDesplazamiento {
public:
   RegistroDesplazamiento(int n);
   RegistroDesplazamiento(const RegistroDesplazamiento &r);
   const RegistroDesplazamiento& operator << (int b);
   const RegistroDesplazamiento& operator = (int i);
   const RegistroDesplazamiento& operator = (const RegistroDesplazamiento& r);
   const RegistroDesplazamiento& operator >> (int i);
   const RegistroDesplazamiento& operator ++ (int);
   int& operator [] (int i);
   friend ostream& operator << (ostream &os, const RegistroDesplazamiento &r);
   ~RegistroDesplazamiento();
private:
   int size;
   int * pr;  // no es eficiente en almacenamiento, pero OK en este caso.
};
#endif
/* Archivo “RegistroDesplazamiento.cpp”  */
#include "RegistroDesplazamiento.h"
using namespace std;

RegistroDesplazamiento::RegistroDesplazamiento(int i): size(i){
   pr = new int[i];
}
RegistroDesplazamiento::~RegistroDesplazamiento(){
   delete [] pr;
}
RegistroDesplazamiento::RegistroDesplazamiento(const RegistroDesplazamiento& r):size(r.size){
   pr = new int[size];
   for (int i=0; i<size; i++)
      pr[i] = r.pr[i];
}
const RegistroDesplazamiento& RegistroDesplazamiento::operator<<(int b){
   for (int i=0; i<size-1; i++)
      pr[i+1] = pr[i];
```

```
   pr[0]=b;
   return *this;
}
const RegistroDesplazamiento & RegistroDesplazamiento::operator=(int v){
  for (int i=0; i<size; i++){
      pr[i] = v%2;
      v/=2;
   }
   return *this;
}
const RegistroDesplazamiento & RegistroDesplazamiento::operator=(const RegistroDesplazamiento & r){
   int min = size < r.size? size:r.size;
   for (int i=0; i<min; i++)
      pr[i] = r.pr[i];
   return *this;
}
const RegistroDesplazamiento & RegistroDesplazamiento::operator>>(int j){
   for (int i=0; i<size-j; i++)
      pr[i] = pr[i+j];
   for (int i=size-j; i<size-1; i++)
      pr[i]=pr[size-1];
   return *this;
}
const RegistroDesplazamiento & RegistroDesplazamiento::operator++(int j){
   int c=1,s;
   for (int i=0; i<size-1; i++){
      s = pr[i] || c ? 1:0;
      c = pr[i] && c ? 1:0;
      pr[i]=s;
   }
   return *this;
}
int& RegistroDesplazamiento::operator[](int i){
   return pr[i];
}
ostream& operator << (ostream &os, const RegistroDesplazamiento &r){
   for (int i=r.size-1; i>=0; i--)
      os << r.pr[i];
   return os;
}
```

3.- (34 puntos) Un elástico puede ser modelado en forma similar a un resorte, con fuerza proporcional a su estiramiento por sobre su largo natural y con fuerza cero cuando no está estirado.
Considere que en su tarea 3 se desea enlazar bolas con elásticos o resortes (springs). Indique e implemente las clases nuevas que usted debe crear, y las partes de la implementación de las clases Ball y Spring de su tarea que deben ser modificadas.

Varias soluciones son posibles. Éstas dependen también se su solución para para la tarea 3. En cualquier caso, los elementos a tomar en cuenta son:
** Se debe crear una clase abstracta para representar elásticos y resortes para los cuales la diferencia está básicamente en el método para calcular la fuerza. (8 puntos)*
** Nueva clase Elastico. Archivo .h y .cpp (3 puntos)*
** Las clases Spring y Elastico debe derivar de la clase abstracta nueva y cambiar acorde la implementación del sus constructores. (8 puntos)*
** La clase Ball debe tener como atributo un vector (o similar) de punteros a esa clase abstracta y el método attach debe ser modificado conforme a ello. (5 puntos)*
** Para reutilizar código, el nuevo diseño debería incluir en esa clase abstracta la funcionalidad común entre objetos tipo spring o elásticos. (5 puntos)*
** Método para el cálculo de la fuerza de un elástico. (5 puntos).*

Una opción para estos archivos es (versión completa de tarea con elásticos se puede ver aquí):

```
/* Attachable.h */
#ifndef ATTACHABLE_H
#define ATTACHABLE_H
class Ball;
#include <string>
#include "CVector.h"
#include "PhysicsElement.h"

class Attachable:public PhysicsElement {
protected:
  const double restLength;
   const double stiffness;
   Ball * a_end, *b_end;
   Attachable(int id, double restLength, double stiffness);

public:
   void attachEnd (Ball * pb);
   CVector getAendPosition() const;
   CVector getBendPosition() const;
   virtual CVector getForce(const Ball * sa) const=0;
   virtual string getDescription() const;
   string getState() const;
   virtual const PhysicsElement * collideWith (const Ball * pb) const;
};
#endif

/*Spring.h */
#ifndef SPRING_H
#define SPRING_H
class Ball;    // Forward declaration
#include <string>
#include "CVector.h"
#include "Attachable.h"

class Spring: public Attachable {
private:
  static int id;  // this class identification number
   Spring();
public:
   Spring(double restLength, double stiffness);
   CVector getForce(const Ball * sa) const;
   string getDescription() const;
};
#endif

/* Elastico.h */
#ifndef ELASTICO_H
#define ELASTICO_H
class Ball;    // Forward declaration
#include <string>
#include "CVector.h"
#include "Attachable.h"

class Elastico: public Attachable {
private:
   static int id;  // thia class identification number
   Elastico();
public:
   Elastico(double restLength, double stiffness);
   virtual CVector getForce(const Ball * sa) const;
   string getDescription() const;
};
#endif

/* Ball.h */
#ifndef BALL_H
#define BALL_H
class PhysicsElement; // incomplete class declaration to deal with circular class definition
class Attachable;   //change
```

```
#include <string>
#include <vector>
#include "CVector.h"
#include "InertialElement.h"
#include "MyWorld.h"
using namespace std;

class Ball: public InertialElement {
private:
   //...
   vector<Attachable *> attachables;  //change
   Ball();
public:
  //..
   void attach(Attachable *s);   // Change
  //...
};
#endif

* Attachable.cpp */
#include "Attachable.h"
#include "Ball.h"

Attachable::Attachable(int id, double restLength, double stiffness):PhysicsElement(id),
              restLength(restLength), stiffness(stiffness) {
   a_end = b_end = NULL;
}
void Attachable::attachEnd (Ball * pb) {  // note: we attach an attachable to a ball,
   if(a_end==NULL)                              //       not the other way around.
       a_end = pb;
   else
       b_end = pb;
   pb->attach(this);
}
CVector Attachable::getAendPosition() const {
   CVector l(restLength,0);
   if (a_end != NULL)
      return a_end->getPosition();
   if (b_end != NULL)
      return b_end->getPosition()-l;
}
CVector Attachable::getBendPosition() const {
   CVector l(restLength,0);
   if (b_end != NULL)
      return b_end->getPosition();
   if (a_end != NULL)
      return a_end->getPosition()+l;
}

string Attachable::getDescription() const {
    return ":a_end,\tb_end\t";
}
string Attachable::getState() const {
    return a_end->getState()+"\t"+b_end->getState();
}
const PhysicsElement * Attachable::collideWith(const Ball * pb) const {
  return NULL;  // nothing collides with a spring in this model
}

/* Elastico.cpp */
#include "Elastico.h"
#include "Ball.h"

int Elastico::id=0;  // Elastico identification
Elastico::Elastico():Elastico(0,0){
}
Elastico::Elastico(double restLength, double stiffness):
              Attachable(id++, restLength, stiffness) {
}
CVector Elastico::getForce(const Ball * pball) const {
   CVector force;
   if ((a_end == NULL) || (b_end == NULL))
```

```
      return force;
   if ((pball != a_end) && (pball != b_end))
      return force;
    CVector a_pos = getAendPosition();
    CVector b_pos = getBendPosition();
    CVector extension = b_pos-a_pos;
   if (extension.module() > restLength) {
       force = -stiffness*(1-restLength/extension.module())*extension;
       if (pball==b_end) return force;
       return (-1)*force;
    } return force; // zero
}

string Elastico::getDescription() const {
    return "Elastico_"+to_string(getId())+ Attachable::getDescription();
}


/* Spring.cpp */
#include "Spring.h"
#include "Ball.h"

//....
Spring::Spring(double restLength, double stiffness): Attachable(id++,restLength, stiffness) {
}
//...

/* Ball.cpp */
#include "Ball.h"
#include "Wall.h"
#include "Attachable.h"  // change
#include "PhysicsElement.h"

//....
void Ball::attach(Attachable *s){   //change
    attachables.push_back(s);
}
//......
}
```