Segundo Certamen

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

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

1.- Responda brevemente y entregue en hoja con su nombre. Use buena caligrafía.

a) En el desarrollo de aplicaciones Android, ¿Qué es una activity? ¿Qué debe modificarse o agregarse en una para que ésta detecte eventos?

Son las componentes que manejan la interacción de cada pantalla de una aplicación Android.

Para que una activity sea capaz de detectar eventos, basta con que esta implemente la interfaz del tipo de evento que se desea detectar.

Ej:

```
public class MainActivity extends Activity implements View.OnClickListener{
```

b) ¿Qué diferencia hay entre declarar #include <algo.h> o declarar #include "algo.h"?

Al incluir un encabezado con <..> estamos indicando que ese archivo se encuentra en uno de los directorios estándares de encabezados definidos por el compilador.

Al indicar "..." el compilar entiende que se trata de un encabezado creado por el programador.

c) ¿Qué debería imprimir el código de la derecha?

```
class Abuelo {
public:
   salida() {
      cout << "Abuelo,";
   }
}
```

```
class Padre: public Abuelo {
public:
   virtual salida() {
      cout << "Padre,";
   }
}
```

```
class Hijo: public Padre  {
public:
   salida() {
      cout << "Hijo,";
   }
}
```

```
Abuelo &ra;
Padre p, *pp;
Hijo h;
ra=p;
ra.salida();
pp=&h;
pp->salida();
ra=h;
ra.salida();
```

Abuelo,Hijo,Abuelo,

d) Para cada una de las siguientes clases en C++, escriba SOLO la implementación del DESTRUCTOR de cada una de ellas. En caso de no necesitar implementación, justifique el por qué.

```
class A {
private:
   int a;
public:
   A();
   ~A();
}

~A() { //implementación vacía
       // porque no hay atributos
 }     // en memoria dinámica.
```

```
class B {
private:
   int *b;
public:
   B();
   ~B();
}
~B() {
    delete b;  //si verificó no null,
}              // es OK
```

```
class C {
private:
   int c1;
   int *c2;
public:
   C();
   ~C();
}
~C() {
   delete c2;
}
```

e) Complete la declaración de la clase MyVector para que el código indicado permita escalar el vector indicado. No se pide implementación.

```
class MyVector {
private:
    double x,y;
public:
     MyVector() { x=y=0;}
friend MyVector operator * (double f,MyVector v);
// ←?
     ...
}
```

```
MyVector v1, v2;
    v2 = 3.0*v1;
```

f) Implemente el constructor para la clase Chileno.

```
class Chileno {
private:
     const string RUT="";
public:
       Chileno () { RUT=""; }
       Chileno (string rut);
    :
}
```

```
Chileno::Chileno(string rut):RUT(rut) {
}
```

i) ¿Qué es la certificación CMM y para qué sirve? Escoja uno de los 5 niveles de certificación y explíquelo.

La certificación CMM (Capability Maturity Model) se utiliza para evaluar los procesos de desarrollo de software.

Los 5 niveles de certificación son (basta con explicar uno):

1. Inicial (caótico): Proceso sin control, impredecible, no necesariamente documentado, proceso inestable. Punto de partida o piso para una organización. El resultado varía según las personas. Puede generar buenos resultados, pero no hay evidencia.

2. Repetible: Algunos procesos son repetibles con resultados posiblemente consistentes. Hay disciplina pero no es rigurosa.

3. Definido: Proceso normado y consistente. Se ha establecido un conjunto de procesos estándares definidos y documentados, los cuales poseen un grado de mejoramiento en el tiempo.

4. Administrado: Proceso predecible. Usando alguna métrica de proceso, la administración puede efectivamente controlar el proceso. Aquí se introduce la medición para controlar calidad.

5. Optimizado: Proceso en mejora permanente. El foco es la mejora continua del desempeño del proceso.

j) Explique qué son los signals y slots y la relación entre ellos en el contexto de programación con bibliotecas Qt.

Signal: Son funciones de acceso público que no deben ser implementadas (solo se definen) y no deben retornar valores. Las señales son emitidas cuando hay un cambio de estado.

Slot: Son funciones definidas normalmente y que pueden ser llamadas como cualquier otra cuya única diferencia es que se pueden conectar con una función signal.
Un slot es llamado cuando la señal o señales (signals) a la que se encuentra conectado son emitidas.

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.- Se desea simular en C++ un semáforo simple para un cruce donde pasan personas preferentes. Éste tiene sólo luz verde y roja. El semáforo normalmente muestra rojo al peatón. Cuando éste solicita cruzar, el semáforo le da luz verde inmediatamente y por un tiempo fijo. Pasado este tiempo, y si no hay nuevos requerimientos, el semáforo pasa a rojo. Las solicitudes de cruce son ingresadas a través de un archivo de texto. A la izquierda se muestra el código del main, a la derecha el archivo de entrada y la salida generada para ese caso. Entregue el contenido de los archivos Semaforo.h (15 pts.) y Semaforo.cpp (19 pts).

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

int main(int argc, char* argv[]) {
    float t=0, tiempoPeticionCruce;
    const float delta=1;
    Semaforo sem(5);    //Duración de luz verde

    ifstream file(argv[1]);
    while(!file.eof()){
        file >> tiempoPeticionCruce;
        while (t<tiempoPeticionCruce) {
            t+=delta;
            sem.avanceTiempo(delta);
            cout << t << '\t' << sem << endl;
        }
        sem.teSolicitoCruzar();
    }
    while(!sem.estaRojo()){
       t+=delta;
       sem.avanceTiempo(delta);
       cout << t << '\t' << sem << endl;
    }
    return 0;
}
```

Ejemplo de archivo de entrada “data.txt”:

```
2
12
```

Ejemplo de salida:

```
1       rojo
2       rojo
3       verde
4       verde
5       verde
6       verde
7       verde
8       rojo
9       rojo
10      rojo
11      rojo
12      rojo
13      verde
14      verde
15      verde
16      verde
17      verde
18      rojo
```

Semaforo.h (15 pts)

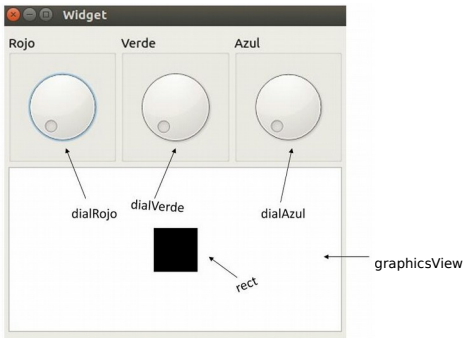
```
#ifndef SEMAPHORE_H_
#define SEMAPHORE_H_
#include <ostream>
using namespace std;

enum Color {rojo, verde};
class Semaforo {
private:              // 3 pts.
    float t;
    Color light;
    const float greenTime;
public:
      Semaforo(float gt);              // 2 pts
      void avanceTiempo(float delta); // 2 pts
      void teSolicitoCruzar();         // 2 pts
      bool estaRojo() const;           // 2 pts
      friend ostream& operator<<(ostream &os, const Semaforo &sem);  // 4 pts
};
#endif /* SEMAPHORE_H_ */
```

Semaforo.cpp (19 pts.)

```
#include "Semaforo.h"
Semaforo::Semaforo(float gt):greenTime(gt){  //  5 pts
      t=0;
      light=rojo;
}
void Semaforo::avanceTiempo(float delta) {   //  4 pts
     if (t >= 0 ) {
       t-=delta;
       if (t<0)
            light=rojo;
     }
}
void Semaforo::teSolicitoCruzar(){           // 3 pts
      t=greenTime;
      light=verde;
}
bool Semaforo::estaRojo() const {            // 3 pts
      return light==rojo;
}
ostream& operator<<(ostream &os, const Semaforo &sem){  // 4 pts
      os << (sem.light==rojo?"rojo":"verde");
      return os;
}
```

3.- Desarrolle el programa **DialColor** en C++ utilizando las bibliotecas Qt. Este programa cuenta con 3 diales y un cuadrado de dimensiones 50px por 50px y de color inicialmente negro dentro de una QGraphicsView. La disposición de los elementos así como los nombres de las variables asociadas a cada uno de estos se indican en la siguiente imagen:



Cada uno de los diales representa los colores rojo, verde y azul respectivamente y cuyos valores van del 0 al 255. El color del cuadrado viene dado por la combinación de los valores de la posición de cada uno de los diales en codificación RGB. Si uno de los diales se mueve, el color del cuadrado se actualiza con los nuevos valores presentes.

a) Considere la clase RectangleColor que hereda de QGraphicsRectItem y QObject, usada para la creación del cuadrado y su implementación:

```
#ifndef RECTANGLECOLOR_H
#define RECTANGLECOLOR_H

#include <QGraphicsRectItem>
#include <QObject>
#include <QBrush>
#include <QColor>

class RectangleColor:public QObject,public QGraphicsRectItem{
   Q_OBJECT
public:
   RectangleColor(int x, int y, int w, int h);

public slots:
   void changeRed(int);
   void changeGreen(int);
   void changeBlue(int);

private:
   int rojo = 0;
   int verde = 0;
   int azul = 0;
   void updateColor(); // Actualiza el color del cuadrado
};

#endif // RECTANGLECOLOR_H
```

```
#include "rectanglecolor.h"

RectangleColor::RectangleColor(int x, int y, int w, int h) :
   QGraphicsRectItem(x,y,w,h)
{
   updateColor();
}
// Completar métodos
void RectangleColor::changeRed(int rojo){



}
void RectangleColor::changeGreen(int verde){



}
void RectangleColor::changeBlue(int azul){



}
void RectangleColor::updateColor(){



}
```

Complete las implementaciones de los métodos **changeRed**, **changeGreen**, **changeBlue** y **updateColor**. Para el método **updateColor**, se recomienda utilizar las clases QColor y QBrush (17 pts.).

```
#include "rectanglecolor.h"

RectangleColor::RectangleColor(int x, int y, int w, int h) :
   QGraphicsRectItem(x,y,w,h)
{
   updateColor();
}

void RectangleColor::changeRed(int rojo){          // 3 pts
   this->rojo = rojo;
   updateColor();
}

void RectangleColor::changeGreen(int verde){       // 3 pts
   this->verde = verde;
   updateColor();
}

void RectangleColor::changeBlue(int azul){         // 3 pts
   this->azul = azul;
   updateColor();
}

void RectangleColor::updateColor(){                // 8 pts
   QColor color(this->rojo,this->verde,this->azul);
   this->setBrush(QBrush(color));
}
```

b) Considere la siguiente definición de la clase principal Widget

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QGraphicsScene>
#include "rectanglecolor.h"

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private:
    Ui::Widget *ui;
    QGraphicsScene *scene;
    RectangleColor *rect;
};

#endif // WIDGET_H
```

```
#include "widget.h"
#include "ui_widget.h"

// Completar métodos
Widget::Widget(QWidget *parent) :
QWidget(parent),ui(new Ui::Widget){
    ui->setupUi(this);

}

Widget::~Widget()
{

}
```

Complete la implementación del constructor y del destructor de la clase (17 pts.).
La implementación del constructor debe incluir:

- Creación de la QGraphicsScene **scene** y asociarla con la variable QGraphicsView **graphicsView**
- Creación del cuadrado RectangleColor **rect** y añadirla a la variable **scene.**
- Conectar cada uno de los QDial **dialRojo**, **dialVerde** y **dialAzul** a través de su signal **valueChanged** con el slot que correspondiente de **rect**.

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui→setupUi(this);
    // 4 pts
    scene = new QGraphicsScene(this);
    ui→graphicsView→setScene(scene);
    // 4 pts
    rect = new RectangleColor(0,0,50,50);
    scene→addItem(rect);
    // 4 pts
    connect(ui->dialRojo,SIGNAL(valueChanged(int)),rect,SLOT(changeRed(int)));
    connect(ui->dialVerde,SIGNAL(valueChanged(int)),rect,SLOT(changeGreen(int)));
    connect(ui->dialAzul,SIGNAL(valueChanged(int)),rect,SLOT(changeBlue(int)));
}

// 5 pts
Widget::~Widget()
{
    delete ui;
    delete scene;
    delete rect;
}
```