Certamen Parcial

1.- Suponga que se desea crear una clase para manipular vectores de dos dimensiones. Las operaciones que son de interés son la de sumar un vector a otro, otra que retorne verdadero si el vector en cuestión es de mayor magnitud que otro, y una tercera función que indique si el vector posee mayor ángulo que otro. Asuma y exponga cualquier otra consideración que usted estime necesaria.

a) Muestre cual sería la declaración de la clase vector (el .h correspondiente).
b) Haga la implementación de las operaciones indicadas (el .cpp correspondiente).

En vector.h:

```
Class Vector {
public:
        Vector() { r=0; theta=0;}
        Void crecerEn (const Vector & v);
        Bool mayorMagQue(const Vector & v) const;
        Bool mayorAngQue(const Vector & v) const;
        // posiblemente otros métodos...
private:
        double r;  // magnitud del vector
        double theta;  /* vectores en el tercer y cuarto cuadrante se representan
                                con ángulos negativos. */
        static const double PI=3.1415;
};
```

En vector.cpp

```
#include <math.h>

void Vector::crecerEn(const Vector & v) {
        Double x = r*cos(theta)+v.r*cos(v.theta);
        Double y = r*sin(theta)+v.r*sin(v.theta);
        Double tr = sqrt(x*x + y*y);

        If (x==0) if (y > 0) theta = PI/2; else theta=3*PI/2;
        Else if (x > 0) theta = arctg(y/x); else theta = PI-arctg(y/x);
};

bool Vector::mayorMagQue(const Vector &v) const {
        return (r >v.r);
}

bool Vector::mayorAngQue(const Vector &v) const {
        return (theta > v.theta);
}
```

Si usáramos representación cartesiana:

En vector.h:

```
Class Vector {
public:
```

```
        Vector() { r=0; theta=0;}
        Void crecerEn (const Vector & v);
        Bool mayorMagQue(const Vector & v) const;
        Bool mayorAngQue(const Vector & v) const;
        // posiblemente otros métodos...
private:
        double x, y;
};
```

En vector.cpp

```
#include <math.h>

void Vector::crecerEn(const Vector & v) {
        x += v.x;
        y+=v.y
};

bool Vector::mayorMagQue(const Vector &v) const {
        return (x*x + y*y  > v.x*v.x +v.y*v.y);
}

bool Vector::mayorAngQue(const Vector &v) const {  // sólo para 1er cuadrante
        return (y/x > v.y/v.x);
}
```

2.- Considere la siguiente declaración para una clase:

```
class prueba {
public:
        prueba ();
private:
        int * dato;
}
```

a) ¿Qué aspectos no han sido considerados en esta declaración?
b) Complete la declaración para incluir los elementos mencionados en su respuesta a).

a) Como la clase posee un miembro que se ubica en el heap, algunas operaciones "por defecto" no generan el resultado esperado. Se debería incluir el **constructor de copia**, **sobre-escribir el operador de asignación**, e incluir del **destructor**.

b)

```
class prueba {
public:
        prueba ();
        prueba( prueba p); // constructor de copia
        ~prueba(); // destructor para asegurarse que no se genera una fuga de memoria.
        Const prueba & operador = (const prueba & p); // operador asignación.
private:
        int * dato;
}
```

3.- ¿Cuál es el error de compilación que indicaría la siguiente implementación?:

```
class tres {
public:
        tres () { dato=8;}
        int getDato(){ return  dato;}
        void muestre() const ;
        int * mitad ();
private:
        int dato;
};
void tres::muestre() const
{ cout << getDato();
}
int * tres::mitad()
{ int m=getDato()/2;
  return &m;
}
int amplifique (tres x) {
        return x.dato * 8;
}
```

Errores de compilación presentes:

a) El método muestre es constante y está llamando a un método que no está declarado como tal.
b) La función amplifique accede a un miembro dato privado. Esto no es permitido para funciones generales.

OJO es un error de ejecución el siguiente y por ente no debe ser listado como error de compilación: El método mitad retorna un puntero a una variable automática cuyo alcance se pierde con el término de la función.

4.- Indique y describa brevemente cuatro fases del desarrollo de software.

Estas fases pueden tener nombres distintos y en ocasiones aparecen agrupados, en general respuestas posibles son cualquiera de los cuatro siguienttes:

Especificación de requerimientos: Fase en que se debe tomar nota de cual es el problema a resolver y cuales son los requerimientos del usuario a tomar en cuenta en la solución.

Análisis: Fase en que se estudian distintos escenarios para reconocer los entes participantes en el problema. Se identifica el entorno del sistema y se modela el mundo real considerando los elementos relevantes del problema.

Diseño: Fase en que se toman decisiones sobre la forma que adoptará la solución. Se identifican los módulos de software que dan origen a la solución, sus funciones e interacción con los elementos internos y externos del sistema.

Implementación: Aquí se codifica y se lleva la solución a un lenguaje de programación y los datos son representados en el computador.

Pruebas: Etapa en que se hacen las pruebas de cada módulos y la integración entre ellos.

Mantención: Fase en que el código en adaptado a nuevos requerimientos del usuario, se hace también la depuración de errores no identificados en la fase de prueba..

5.- Uno de los principios de David Parnas dice: “Se debe proveer al implementador toda la información para completar un módulo, y nada mas.”
¿Qué situación puede generarse cuando no se cumple este principio?

La idea es no dar espacio a posibles errores humanos de comunicación o interpretación. No debemos compartir más información que la justa y necesaria con el implementador. Así evitamos que él o ella nos mal interprete y asuma o entienda que los datos extras son parte de la especificación del módulo. Se puede generar código más complejo que el necesario al tratar de cubrir requerimientos inexistentes o propios de otro módulo del sistema.

6.- ¿En qué consiste el ligado dinámico? Dé un ejemplo en que exista ligado dinámico.
Consiste en que la asociación del código de la función llamada desde un punto invocación se hace en tiempo de ejecución.
En otras palabras, en forma dinámica y durante la ejecución se decide en base al objeto el código a ser llamado.

Ejemplo:

```
Class CBase
{
.....
public:
        virtual void foo(void);
...
};

class Subclase1: public CBase
{
public:
        virtual void foo(void);  // no estrictamente necesario que sea virtual
....
};

class Subclase2: public CBase
{
public:
        virtual void foo(void);  // no estrictamente necesario que sea virtual
....
};
```

...
Ahora, en el código siguiente, se ejecutará el código de foo de la Subclase1 o Subclase2 dependiendo del objeto específico que sea pasado a la función.

```
Void funcion(CBase & a) {
  a.foo();
...
}
```