PRACTICA 1 – POO en C++

1. Completar la siguiente declaración con los métodos necesarios para implementar el objeto PUBLICACION de una biblioteca. Desarrollar el código respectivo.

```
#include <stdlib.h>
class publicacion{
  long codigo;
  unsigned int tema;
  char titulo[25];
  int prestado;    //1 indica si
public:
  void Muestra ();
  .........
};
```

2. Detectar errores en la siguiente declaración.

```
class punto {
   int x, y;
   void punto (int x1, int y1)  { x=x1; y=y1;}
   punto(){ x=0; y=0;}
public:
   void ponedatos(int x1, y1) {punto.x=x1; punto.y=y1;}
}
```

3. Definir una clase **Punto** identificada por un par de valores x e y reales. Debe contar con las funciones miembro necesarias para la inicialización e impresión de los objetos de la clase.

4. Definir la clase **ClienteBanco** con tres atributos: nombre, PIN, saldo. Implementar funciones para mantener los atributos; depositar y extraer dinero; y mostrar el estado de la cuenta. Para operar, el cliente debe ingresar un número, y éste debe coincidir con el PIN particular.

5. Reformular la clase definida en el ejercicio 3 de la siguiente manera:
- Si no lo hizo, que la inicialización se haga de forma automática.
- Agregar el método *resta* que permita restar un punto a otro en el cual debe quedar el resultado (un punto es parámetro).

6. Crear una clase **Hora** que contenga datos miembro de tipo entero para horas, minutos y segundos.
Definir dos constructores: un constructor inicializará los datos en 0 y otro a valores variables. Una función miembro mostrará la hora en formato hh:mm:ss. Otra función sumará dos objetos de tipo hora pasados como argumento.
Definir el *main()* que cree tres objetos, inicializando dos de ellos que deberán sumarse sobre el tercero. Mostrar el resultado de la suma.

7. Implementar la clase **Stack** que permita almacenar hasta 100 caracteres en un arreglo. Debe contar con constructor y las funciones necesarias para mantener la pila. (Recordar la necesidad de mantener la dimensión lógica del arreglo)

8. Realizar un programa que utilizando la clase **Stack** definida en el ejercicio 7, lea una cadena, la almacene en el Stack (carácter a carácter) y la imprima invertida.

9. Definir una clase **Complejo** que contenga dos partes: real e imaginaria y las operaciones de suma, resta, multiplicación entre dos complejos, multiplicación entre un

complejo y su conjugado (con un parámetro de salida) y multiplicación entre un complejo y un real.
Desarrollar el *main* invocando a algunas de las funciones definidas.
Observación: las funciones de multiplicación deben sobrecargarse para permitir las 3 posibilidades.

10. Definir una clase **Empleado** que contenga como atributos el nombre, el número de empleado, la fecha de ingreso y el área (1 a 5); como métodos: *asignadatos(), mostrar(), obt_area()* y otros.
Escribir el *main* que cree un arreglo de **Empleado**. Guarde un conjunto de N empleados y luego muestre los que tienen más de 10 años de antigüedad (no desarrollar la función para obtener la antigüedad)

11. A partir de la clase definida en el ejercicio 9, desarrollar una función amiga global de ésta que tome como argumento un puntero a un objeto de tipo **Complejo** y devuelva las componentes real e imaginaria.

12. A partir de la clase definida en el ejercicio 10, escribir el *main* que defina un arreglo de punteros a **Empleado**, guarde un conjunto dinámico de empleados y mediante un menú permita:
- Mostrar el arreglo
- Eliminar los que ingresaron antes 2004.

Al finalizar, destruir los objetos creados.

13. A partir de la clase definida en el ejercicio 10, escribir el *main* que defina y genere una lista en la que cada nodo contenga: puntero a **Empleado** y siguiente. La lista debe crearse ordenada por número de empleado. Se pide luego mostrar el año de ingreso de los empleados del área 3.
Observación: recordar retorna al finalizar la memoria solicitada dinámicamente.

14. Se tiene una jerarquía de clases como la que sigue

```
class PERSONA {
  int edad;
  char nya [25];
public:
  void Muestra(); {cout edad << nya;}
  void Pone_Nom(char *cad){nya=cad;}
}
class PROFESOR:public PERSONA {
  int CantMaterias;
  char nom [20], ape[20];
public:
  void Muestra(){cout nom << ape << CantMaterias << PERSONA::Muestra(););
  void Pone_Mate(int m){CantMaterias = m;}
}
class ALUMNO:public PERSONA {
  longint Matricula;
  char carrera [20];
public:
  void Muestra(){Muestra();}
}
```

a. Detectar y corregir los errores encontrados
b. Agregar constructores en las clases
c. Escribir el *main* declarando un objeto de cada tipo, inicializarlos y mostrar sus datos.

d. Escribir el *main* declarando un puntero a PERSONA, utilizarlo para apuntar sucesivamente a objetos dinámicos de los otros tipos, inicializarlos, mostrar sus datos y luego destruirlos.

15. A continuación se muestran las diferencias entre una llamada a una función virtual y a una función estática. ¿Cuál es el resultado de ejecutar cada uno de los *main*? Justificar su respuesta.

```
#include <iostream.h>
class base {
  public:
    virtual void f(){cout<<"f() clase base \n";}
    void g(){cout<<"g() clase base \n";}
};
class derivada1:public base {
  public:
    void f(){cout<<"f() clase derivada1 \n";}
    void g(){cout<<"g() clase derivada1 \n";}
};

class derivada2:public derivada1 {
  public:
    void f(){cout<<"f() clase derivada2 \n";}
    void g(){cout<<"g() clase derivada2 \n";}
};
```

```
main(){
  base b;
  derivada1 d1;
  derivada2 d2;
  base *p=&b;   p->f();   p->g();
  p=&d1;   p->f();   p->g();
  p=&d2;   p->f();   p->g();
}
```

```
main(){
  base b, *pb;
  derivada1 d1, *pd1;
  derivada2 d2, *pd2;
  pb = &b;   pb->f();   pb->g();
  pd1=&d1;   pd1->f();   pd1->g();
  pd2=&d2;   pd2->f();   pd2->g();
}
```

16. Definir la jerarquía que considere adecuada con las siguientes clases: Figura, Rectángulo y Triángulo Rectángulo. Implementar funciones para obtener el perímetro y el área de cada una de ellas.

17. Analizar la jerarquía del ejercicio 16: ¿alguna clase es abstracta? Justificar la respuesta. Si la misma fue afirmativa, definirla como tal.

18. Si en la jerarquía definida en el ejercicio 16 quisiera agregarse la clase **Circulo**, ¿cómo debería hacerse?

19. Considerar las definiciones del ejercicio 16. Crear una clase **Geometrica** que contenga un atributo que sea un arreglo de punteros a **Figura**. Desarrollar funciones para agregar, eliminar y mostrar los elementos de dicha estructura, que deberá ser manejada como una Pila. Desarrollar un *main* que permita agregar una cantidad de figuras (de tipos diversos, pueden ser estáticas o dinámicas) no definida a priori y luego muestre el perímetro de las últimas K figuras agregadas. Al finalizar recordar eliminar los elementos que hayan sido creados, si fuera necesario.

20. Considerar las definiciones del ejercicio 16. Crear una clase **Geometrica** que contenga un único atributo que sea una lista dinámica de **Figura** (cada nodo de la lista tendrá dos campos: un puntero a Figura y la dirección del siguiente elemento). Desarrollar métodos para agregar, eliminar y mostrar elementos de dicha estructura,

que deberá ser manejada como una Pila. Desarrollar un *main* que permita agregar una cantidad de figuras (de tipos diversos, dinámicas) no definida a priori y luego muestre la superficie de las últimas K figuras agregadas. Al finalizar recordar eliminar los elementos que hayan sido creados, si fuera necesario.

21. Considerar las definiciones del ejercicio 16. Crear una clase **Geometrica** que contenga un único atributo que sea una lista dinámica de **Figura** (cada nodo de la lista tendrá dos campos: un puntero a Figura y la dirección del siguiente elemento). Desarrollar métodos para agregar (de forma ordenada por perímetro), eliminar y mostrar elementos de dicha estructura, y los siguientes:

a. mostrar los lados de la figura con mayor perímetro
b. mostrar el perímetro de las figuras cuya superficie sea mayor a 4.
c. eliminar las figuras que tengan una superficie de al menos 10

22. Sea la siguiente jerarquía de objetos que representa el manejo de las cuentas en una sucursal de algún Banco:

CLASE BASE: CUENTA (abstracta)
Atributos: CODCLIENTE, NROCUENTA, FECHAAPERTURA, SALDO, MOVIMIENTOS (últimos 20: tipo, monto, fecha), INTERES
Métodos: cierre, depósito (puede ser cheque o efectivo), extracción, resumen, descuentos por gastos (una vez al mes $12,10) y los que considere necesarios.

SUBCLASE: AHORRO (de CUENTA)
Agrega: Atributos: CANT EXTRACCIONES (máxima permitida), DEPOSITA CHEQUE (S/N)
Métodos: para el mantenimiento de los atributos, y la autorización o no para depositar cheques.
(Una Cuenta de Ahorro no tiene descubierto permitido)

SUBCLASE: CORRIENTE (de CUENTA)
Agrega: Atributos: MONTO DESCUBIERTO MAXIMO
Métodos: para el mantenimiento de los atributos, y el cierre por descubierto superior al permitido.

CLASE: BANCO
Atributos: LISTA de CUENTAS (puede suponerse que no excederán las 1000 y ordenada por Número de Cuenta), LISTA de CLIENTES (puede suponerse que no excederán los 500 y ordenada por Código de Cliente y además de éste dato contiene otro como DNI, teléfono, etc) y otros (de ser necesario)
Métodos: apertura y cierre de nuevas cuentas, resumen de cuentas, depósitos y extracciones en las cuentas, listados, ingreso y egreso de clientes.

Se pide:

a. Definir las clases de objetos (atributos y cabeceras de los métodos) para los objetos descriptos y para el **Banco** que los controla.
b. Desarrollar el método *Extraer* de **Banco** que recibe un Número de Cuenta y un monto y debe informar si la extracción pudo concretarse y el saldo actual de la cuenta.
c. Desarrollar el método *ListadoporSaldo* de **Banco** que debe mostrar todos los datos (incluidos los datos del titular de la misma) de aquellas cuentas que tengan saldo deudor.
d. Desarrollar el método *Elimina* de **Banco** que dado un código de Cuenta, se dé de baja de la lista. Si dicha cuenta era la única del cliente, informar la situación.
e. Desarrollar todos los métodos que se hayan invocado en **b.-**, **c.-** y **d.-**.
f. Si no lo hizo, redefinir el tipo de la lista de clientes, utilizando una clase CLIENTE que permita que los métodos de Banco accedan a sus atributos y rehacer el inciso **c.**

23. Una Agencia de vehículos de alquiler desea llevar control de los mismos y para ello los ha clasificado de acuerdo a su tipo. Todos tienen licencia, nombre chofer, nombre dueño, modelo (año patentamiento) y cantidad de pasajeros que están transportando. Las características que los diferencian son:

- Los TAXIS cobran cada viaje $13,12 por km recorrido más $14,20 por la bajada de bandera.
- Los REMISES cobran $13,14 por km recorrido y no cobran bajada de bandera.
- Las COMBIS agregan su capacidad y una lista dinámica con los DNI de los pasajeros que transportan (está vacía si no está en viaje). Cobran el viaje así: por viajes de
  - hasta 100km $100 por persona si la combi transporta hasta un 50% de su capacidad y $90 por persona en cualquier otro caso.
  - entre 100 y hasta 200km; $170 por persona
  - más de 200km; $1000 independientemente de la cantidad de pasajeros

Los autos de más de un año de antigüedad realizan: del 10% si tienen más de 1 año y del 20% si tienen más de 4 años.

Se pide:

a. Implementar las clases de objetos para los tipos de autos descriptos con los atributos y métodos que considere necesarios y la clase **Agencia** que los controla.

b. Desarrollar los siguientes métodos de **Agencia**
   - *MuestraViaje*(): que recibe tres argumentos: K, P y L y debe mostrar todos los datos del vehículo con licencia L, incluyendo el precio de un viaje de K kilómetros para P pasajeros.
   - *GenteEnViaje*(): que determina y muestra la cantidad de gente que está transportando la empresa en un determinado momento.
   - *ViajaDNI()*: que recibe un DNI y verifica alguna persona con ese DNI está viajando está registrado en algún viaje en combi.

c. Desarrollar todos los métodos que se hayan invocado en **b.-** y los destructores.

d. En la función *main()*, almacenar 4 vehículos en la agencia, hacer que 2 de ellos estén en viaje. Invocar las funciones desarrolladas en **b.-**.