UTN Facultad Regional La Rioja

Trabajo Practico Nº 01

2024

Índice

Introducción

El presente trabajo práctico tiene como objetivo poner en práctica los conocimientos adquiridos en la materia hasta ahora.
Los ejercicios propuestos abarcan el uso de los conocimientos adquiridos en programación I (variables, el manejo de estructuras condicionales, estructuras cíclicas, vectores, matrices, funciones, punteros, estructura y ficheros) además de los conocimientos adquiridos en programación II (funciones cin, cout, clases).

Es fundamental comprender y aplicar adecuadamente estos conceptos para resolver eficientemente los problemas planteados. Además, se espera que los estudiantes adquieran habilidades en la organización del código, la lectura de entradas desde el teclado, y la presentación de resultados en la terminal.

Entrega

Se deberá entregar un archivo comprimido con wirar en formato .zip o .rar, este archivo deberá contener:

1. Un archivo PDF con los códigos realizados por cada ejercicio junto con la imagen de la salida de la Terminal (Aplicación Programada).
2. Un archivo de código fuente por cada ejercicio realizado, en este archivo deber estar el código funcional y comentado correctamente.

Nombres de los Archivos:
Los archivos deberán tener el siguiente formato de nombre:

- **Archivo Comprimido**: "TP 01 - Comisión X - Apellido Nombre"
  - *Ejemplo*:
    - TP 01 – Comisión B – Oviedo Carlos.rar
- **Archivo PDF**: “TP 01 - Comisión X - Apellido Nombre ".
  - *Ejemplo*:
    - TP 01 – Comisión B – Oviedo Carlos.pdf
- **Código Fuente:** " TP 01 - Comisión X - Apellido Nombre – Ejercicio xx"
  - *Ejemplo***:**
    - TP 01 – Comisión B – Oviedo Carlos – Ejercicio 01.cpp

Fecha de Entrega: *Martes 17de Septiembre del 2024*

Criterio de evaluación

Los trabajos prácticos de programación se calificarán según una rúbrica de 10 puntos por ejercicio, luego para la nota final del práctico se dividirá el puntaje total en 10. Criterios a evaluar:

- **Funcionamiento del programa**: Cada programa debería funcionar correctamente en todas las entradas. Además, si hay alguna especificación sobre cómo funciona el programa debería escribirse, o cómo debe aparecer el resultado, se deben seguir esas especificaciones.
- **Legibilidad**: Las variables y funciones deben tener nombres significativos. El código debe organizarse en funciones cuando corresponda. Debe haber una cantidad adecuada de espacio en blanco para que el código sea legible y la sangría debe ser coherente.
- **Documentación**: El código y funciones deben estar comentados apropiadamente. Sin embargo, no todas las líneas deben ser comentadas porque eso hace que el código esté demasiado ocupado. Pensar detenidamente dónde se necesitan comentarios.
- **Elegancia del código**: Hay muchas formas de escribir la misma funcionalidad en el código y algunas de ellas son innecesariamente lentas o complicadas. Por ejemplo, si está repitiendo el mismo código, debería estar dentro de una nueva función o bucle for.
- **Especificaciones y entrega del práctico**: Cada programa debe ser guardado con un determinado nombre de archivo u otras especificaciones similares y debe ser entregado en tiempo y forma. Ver especificaciones de entregas indicadas anteriormente.

Rubrica de Evaluación

Funcionamiento	*4*	*3*	*2*	*0*
	El programa siempre funciona correctamente y cumple con las especificaciones.	El programa no cumple con detalles menores de las especificaciones, o funciona incorrectamente en algunas entradas.	El programa no cumple con detalles significativos de las especificaciones, o exhibe un comportamiento incorrecto.	El programa no funciona, o no compila.
Legibilidad	*2*	*1*	*0,5*	*0*
	El código es limpio, entendible y bien organizado.	Problemas menores como sangría inconsistente, denominación de variables, organización general	Al menos un problema importante que dificulta la lectura.	Varios problemas importantes que dificultan su lectura.
Documentación	*2*	*1*	*0,5*	*0*
	El código está bien documentado.	Unos o dos lugares pueden beneficiarse de comentarios, o el código está comentado de más	La falta de comentarios dificulta la compresión del código.	El código no presenta comentarios.
Elegancia		*1*	*0,5*	*0*
		El código utiliza adecuadamente bucles for y métodos para código repetido, y la codificación es mínima.	El código utiliza un enfoque mal elegido al menos en un lugar, por ejemplo, codificar algo que podría implementarse a través de un bucle for.	En varias ocasiones en que el código podría haber utilizado un enfoque más fácil/rápido/mejor.
Especificaciones		*1*	*0,5*	*0*
		El ejercicio cumple con las especificaciones.	El ejercicio no cumple con especificaciones menores.	El ejercicio no cumple con especificaciones significativas.
Entrega			*0*	*-3*
			Se entregó dentro del plazo establecido.	Se entregó fuera del plazo establecido.

Enunciados

Ejercicio 1: Clase `Persona`

Crea una clase `Persona` que contenga atributos como nombre, edad, y género. Implementa métodos para establecer y obtener estos valores. Utiliza separación de archivos.
En el archivo principal, crea un programa que permita al usuario ingresar la información de tres personas. El programa debe verificar que la edad ingresada sea mayor que 0 y que el género sea "M" (masculino), "F" (femenino), o "O" (otro). Luego, muestra la información de cada persona.

- **Validaciones**: La edad debe ser mayor que 0, y el género debe ser uno de los valores permitidos.

Ejercicio 2: Clase `CuentaBancaria`

Diseña una clase `CuentaBancaria` con atributos como el número de cuenta, saldo y nombre del titular. Implementa métodos para depositar y retirar dinero.
En el archivo principal, crea un programa que permita crear dos cuentas bancarias, realizar depósitos y retiros. El programa debe verificar que no se intente retirar más dinero del que hay disponible en la cuenta. Muestra el saldo de cada cuenta al final.

- **Validaciones**: El saldo nunca debe ser negativo. El número de cuenta debe tener al menos 6 dígitos.

Ejercicio 3: Clase `Rectangulo` con sobrecarga de operadores

Crea una clase Rectangulo que tenga atributos como largo y ancho. Implementa un método que permita calcular el área de un rectángulo.
En el archivo principal, permite que el usuario ingrese las dimensiones de dos rectángulos. Calcula las áreas de ambos rectángulos utilizando el método correspondiente y luego muestra la suma de sus áreas..

- **Validaciones**: El largo y el ancho deben ser valores positivos.

Ejercicio 4: Clase `Vehiculo` y Herencia

Crea una clase `Vehiculo` que tenga atributos como marca, modelo, y velocidad máxima. Luego crea dos clases derivadas: `Coche` y `Moto`. Cada clase debe tener un método `mostrarDetalles()` que sobrescriba el método base y muestre información específica.
En el archivo principal, permite al usuario crear un coche y una moto, ingresando sus atributos. Luego, utiliza polimorfismo para llamar al método mostrarDetalles() desde un puntero a Vehiculo. Valida que la velocidad máxima sea un valor positivo.

- **Validaciones**: La velocidad máxima debe ser positiva y menor a 300 km/h

Ejercicio 5: Polimorfismo con Animales

Crea una clase base `Animal` con un método virtual `hacerSonido()`. Luego, crea clases derivadas como `Perro` y `Gato`, que sobrescriban este método para hacer sonidos específicos. Usa punteros de tipo `Animal` para demostrar el polimorfismo.

En el archivo principal, permite al usuario crear varios animales (perros y gatos) e invoca el método hacerSonido() para cada uno utilizando un puntero a la clase Animal. Asegúrate de que el usuario no ingrese un nombre vacío al crear cada animal.

- **Validaciones**: El nombre del animal no debe estar vacío.

Ejercicio 6: Sistema de Estudiantes con Polimorfismo

Crea una clase `Estudiante` con un método virtual `calcularPromedio()`. Luego crea clases derivadas como `EstudianteUniversitario` y `EstudianteSecundario` que sobrescriban el para calcular el promedio según diferentes criterios (por ejemplo, con ponderación de materias en la universidad).
El archivo principal, crea un sistema que permita ingresar notas de varios estudiantes de ambos tipos y calcular sus promedios. Valida que las notas ingresadas estén en un rango entre 0 y 10.

- **Validaciones**: Las notas deben estar entre 0 y 10. El promedio debe calcularse solo si se han ingresado notas válidas.

Ejercicio 7: Clase `Empleado` con archivo cabecera y fuente

Implementa una clase `Empleado` que tenga atributos como nombre, salario y puesto. Se deben separar los archivos `.h` y `.cpp`. Crea métodos para calcular el salario con bonificaciones y descuentos.
En el archivo principal, permite al usuario crear varios empleados, ingresar sus datos y calcular el salario final después de aplicar las bonificaciones y descuentos. Valida que el salario inicial sea mayor a 0.

- **Validaciones**: El salario debe ser mayor a 0.

Ejercicio 8: Sistema de Inventario con Archivos

Crea una clase `Producto` que tenga atributos como código, nombre, precio, y cantidad. Implementa métodos para guardar y cargar los productos desde un archivo de texto.
En el archivo principal, permite al usuario agregar varios productos y guardar la información en un archivo de texto. El programa también debe cargar la información de los productos desde el archivo al iniciar.

- **Validaciones**: El precio y la cantidad deben ser positivos. El código del producto debe tener al menos 5 caracteres.

Ejercicio 9: Corregir

Corregir los errores presentes en la definición de la clase y el uso de la misma en el archivo principal. Deben justificar las correcciones realizadas.

```
class Coche {

    string marca

public:
    Coche(string m, int v) {
        marca = m;
        velocidadMax = v;
    }
    void mostrarInfo() const {
        cout << "Marca: " << marca << " - Velocidad Máxima: " << velocidadMax
    }
};
```

```
#include <iostream>


int main() {
    Coche miCoche("Toyota", 180);
    miCoche.mostrarInfo();
    return 0;
}
```

Ejercicio 10: Simulación de un Juego con Polimorfismo

Crea una clase base `Personaje` y clases derivadas como `Guerrero` y `Mago`. Cada clase derivada debe sobrescribir un método `atacar()` que realice una acción distinta. Utiliza punteros a la clase base y demuestra el uso de polimorfismo.
En el archivo principal, crea un juego simple donde el usuario pueda elegir entre varios personajes y realizar ataques utilizando polimorfismo. Valida que los puntos de ataque sean siempre mayores a 0.

- **Validaciones**: Los puntos de ataque deben ser mayores a 0.