Trabajo Practico Nº 03

TRABAJO FINAL

2024

Índice

Introducción

El presente trabajo práctico tiene como objetivo poner en práctica todos los conocimientos los conocimientos adquiridos de programación en C y C++.
Para ello el trabajo se dividirá en 3 partes.

- **Frontend**, utilizando QT Designer y una hoja de estilo (CSS).
- **Backend**, usando todos los conceptos vistos en las dos materias (programación I y programación II).
- **Almacenamientos de datos**, para la parte de resguardar la información una vez cerrado el programa. Se utilizará archivos para almacenar la información.

Este trabajo práctico es en grupo de 4 o 5 integrantes. Los Temas son:

- Transporte Colectivo
- Alquiler de canchas
- Librería (Libros)
- Farmacia
- Minimarket
- Peaje
- Restaurant
- Cine
- Banco
- Estacionamiento

Esta lista de temas se sorteara entre los diferentes grupos formados en la cátedra.

Entrega

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

1. El programa compilado y listo para ejecutar, con todos los archivos DLL (*biblioteca de enlaces dinámicos*) necesarios para su funcionamiento en un equipo diferente al desarrollado (*programado*).

2. Un informe en formato PDF (*debe incluir caratula, con el nombre del equipo, integrantes, etc*) con los diagramas de clases de UML, y los códigos explicando detalladamente cada una de las funciones y clases desarrolladas, capturas de pantalla del front diseñado y un diagrama de Gantt, de los tiempos dedicados a cada parte del proyecto (*Frontend, Backend, Informe*).

3. Todo el código fuente, en estos archivos deberán estar el código funcional y comentado correctamente.

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

- **Archivo Comprimido**: "Comisión X - Nombre del Equipo". *Ejemplo*:
  - Comisión B - LosProgramadores.rar
- **Archivo PDF**: “Comisión X - Nombre del Equipo ". *Ejemplo*:
  - Comisión B - LosProgramadores.pdf

Fecha de Entrega: *24 de Noviembre 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:

1. **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.

2. **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.

3. **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.

4. **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.

5. **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.

Informe

El informe entregado en formato PDF debe tener lo siguiente:

- **Caratula**:
  Una caratula con el nombre del equipo, los integrantes, y el tema. (Puede incluir logo del equipo).

- **Descripción del Proyecto:**
  Una breve descripción del proyecto y los objetivos del sistema según el enunciado, detallando el problema que resuelve y las funcionalidades principales.

- **Explicación de la Estructura de Archivos y Almacenamiento:**
  Una sección que describa cómo se almacenan los datos en archivos, incluyendo la estructura del archivo y ejemplos de datos. Esto es útil para comprender la persistencia de datos en sistemas sin bases de datos.

- **Descripción de las Clases y Funcionalidades:**
  Un listado de las clases principales y sus métodos más importantes, detallando las funcionalidades de cada uno y la lógica detrás de los métodos relevantes.

- **Capturas de Pantalla de la Interfaz de Usuario:**
  Imágenes de la interfaz desarrollada, mostrando cada pantalla principal y explicando brevemente su funcionalidad. Esto da contexto visual al funcionamiento del sistema.

- **Comentarios sobre Validaciones y Manejador de Errores:**
  Una descripción de las validaciones implementadas y cómo se manejan los errores. Puedes incluir ejemplos de cómo se muestran los QMessageBox para advertencias y mensajes de error.

- **Código Fuente Explicado:**
  Incluir fragmentos clave del código, especialmente en secciones de lógica compleja o en partes relacionadas con los requisitos, con explicaciones para mostrar el entendimiento del estudiante sobre la programación aplicada.

- **Diagrama de Gantt:**
  Incluir un diagrama de Gantt como el siguiente donde muestre los tiempos tomados para cada parte del desarrollo. *Ejemplo de Diagrama*

Integrantes/Fechas	Fecha 01	Fecha 02	Fecha 03	Fecha 04	Fecha 05	Fecha 06
Integrante 01	Planteo de Clases	Diseño de Ventana Secundaria				
Integrante 02	Diseño de Ventana Principal					
Integrante 03	Programacion de Backend					
Integrante 04	Planteo de Clases			Desarrollo de Informe		
Integrante 05	Diseño de Ventana Principal					

- **Conclusiones y Aprendizajes:**
  Un espacio final donde los estudiantes compartan sus conclusiones, retos enfrentados, y aprendizajes adquiridos a lo largo del desarrollo del proyecto.

Enunciados

Proyecto: Transporte Colectivo

"Somos una empresa de transporte colectivo y necesitamos un sistema para gestionar nuestras rutas, conductores y colectivos. Cada ruta debe tener un número de identificación, destino, horario, el nombre del chofer a cargo y el colectivo asignado. Queremos ver una lista de todas las rutas y poder agregar o modificar información fácilmente. Necesitamos que el sistema nos alerte si los datos ingresados son incorrectos o incompletos y que mantenga un historial de viajes realizados cada día. Almacenar el empleado que genero el viaje"

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Alquiler de canchas

"Tenemos un complejo de canchas de fútbol, tenis y pádel, y necesitamos una aplicación para gestionar las reservas, considerando qué empleado las generó. Los usuarios deben poder elegir el tipo de cancha, la fecha y el horario, y recibir confirmación de su reserva. El sistema debe validar los horarios para evitar reservas duplicadas y mostrar una lista de todas las reservas del día, además de guardar un historial diario."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Librería (Libros)

"Soy dueño de una librería y necesito un sistema de inventario y ventas para registrar los libros disponibles. Cada libro debe incluir título, autor, editorial y cantidad en stock. El sistema debe emitir alertas cuando un libro esté fuera de stock, permitir búsquedas por título o autor, y registrar cada venta, indicando qué empleado la generó. Además, debe guardar un historial de ventas diarias."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Farmacia

"Manejamos una farmacia y necesitamos una aplicación para controlar nuestro inventario y las ventas, registrando el empleado que las genera. Cada medicamento debe tener un código, nombre, dosis y precio. Necesitamos una función de búsqueda para verificar la disponibilidad de medicamentos y alertas para productos que se estén agotando. El sistema debe guardar un historial de ventas diarias."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Minimarket

"Tenemos un minimarket y necesitamos un sistema para registrar las ventas en caja. El sistema debe registrar cada producto comprado, calcular el total de la compra y aplicar descuentos cuando corresponda. También queremos un historial de ventas diarias para seguimiento y control y ventas por empleados."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Peaje

"Administramos un peaje y buscamos una aplicación para registrar el paso de vehículos. El sistema debe diferenciar entre tipos de vehículos (automóviles, camiones y motos) y calcular la tarifa correspondiente. Queremos ver el total recaudado al día y recibir alertas en caso de error. Además, debe registrar el empleado que cobra el peaje y almacenar un historial diario de tickets generados."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Restaurante

"Tenemos un restaurante y queremos informatizar nuestras reservas de mesas, registrando el nombre del cliente, número de personas, horario y el empleado que generó la reserva. Nos gustaría que el sistema alerte si no hay mesas disponibles en el horario solicitado y que podamos revisar el historial de reservas del día."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Cine

"Somos un cine y necesitamos un sistema para gestionar la venta de entradas. Los clientes deben poder elegir una película, un horario y seleccionar los asientos disponibles. El sistema debe emitir alertas si un horario está lleno y almacenar el historial de ventas diarias y los empleados que realizaron cada venta."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Banco

"Somos un banco y queremos una aplicación para gestionar las cuentas de nuestros clientes. Cada cuenta debe mostrar el saldo y permitir operaciones de depósito, retiro y transferencia, notificando al usuario cada vez que se realice una operación. El sistema debe mantener un historial de movimientos diarios en cada cuenta."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.

Proyecto: Estacionamiento

"Tenemos un estacionamiento y necesitamos una aplicación para controlar la entrada y salida de vehículos. Cada vehículo debe registrar la hora de entrada y calcular el tiempo total al salir. Queremos alertas para avisar si el estacionamiento está lleno, ver la lista de vehículos actuales y almacenar un historial diario de tickets generados y el empleado responsable de cada registro."

Validaciones: Usar **QMessageBox** para advertir al usuario del error en caso de no cumplir la validación.