UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

INGENÍERIA INFORMÁTICA

AC5: LA CASA DE PAPEL

GRUPO 08

PROFESOR

JORGE GARCÍA GONZÁLEZ

CURSO 2025-2026

TABLA DE CONTENIDOS

Capítulo 1. Objetivo

1.1 Descripción de la actividad a realizar

La Actividad Colaborativa 5 (AC5) propone diseñar e implementar un sistema de gestión y análisis de datos de seguridad usando técnicas de programación recursiva. El sistema se divide en cuatro fases: gestión de accesos mediante lista enlazada, control de seguridad con una pila de niveles, monitorización de actividades usando una cola, y análisis de patrones para detectar amenazas. La aplicación se ejecuta por consola y simula la operativa de un equipo de seguridad en un escenario tipo "La Casa de Papel".

1.2 Objetivos de la actividad

1.2.1 Objetivo general

Desarrollar una aplicación modular que gestione accesos y actividades de usuarios utilizando estructuras dinámicas y funciones recursivas, respetando los requisitos del enunciado.

1.2.2 Objetivos específicos:

- Implementar una lista enlazada para almacenar registros de acceso con usuario y timestamp.
- Gestionar una pila de niveles de seguridad (bajo, medio, alto) y actualizarla mediante recursión.
- Crear una cola de actividades con inserción y consulta recursiva por usuario, incluyendo timestamp.
- Implementar un analista que recorra estructuras para contar accesos y detectar actividades sospechosas dentro de una ventana temporal.
- Integrar las cuatro fases en un flujo de uso por consola con menús y datos de prueba.

Capítulo 2. Desarrollo de la actividad

2.1 Actividad 1 - Diseño del sistema

El proyecto se organiza por fases, cada una en su propia carpeta:

- Fase_1_GA: Access.h y LinkedList.h para la gestión de accesos.
- Fase_2_CS: Security.h para el control de seguridad por niveles.
- Fase_3_MA: ColaActividades.h para la monitorización de actividades.
- Fase_4_AS: Analista.h para el análisis de seguridad.

El punto de entrada es main.cpp, que presenta un menú y ejecuta demos por fase.
El build se gestiona con CMake (CMakeLists.txt) usando C++23.

2.2 Actividad 2 - Bases teóricas aplicadas

- TAD y encapsulamiento: clases Access, LinkedList, Security, colaActividades y Analista encapsulan datos y operaciones.
  Ejemplo:

```
1.  struct Access {
2.  string usuario;
3.  long long ts;
4.  string toString() const;
5.  };
```

- Estructuras dinámicas: lista enlazada para accesos, pila (stack) para niveles y cola enlazada para actividades.

- Recursión: insertSortedRec y searchRec en LinkedList; moveToTop/extract en Security; insertarRecursivo y mostrarRecursivo en colaActividades; detectarAmenazasRec y buscarRepeticionRec en Analista.

- Gestión de memoria: LinkedList libera nodos en su destructor, y colaActividades implementa clear y destructor para liberar la cola.

2.3 Actividad 3 - Fase 1: Gestión de accesos

- Se implementa Access como registro de acceso con usuario y timestamp.
- La lista enlazada LinkedList<Access> ofrece inserción ordenada recursiva por tiempo, búsqueda recursiva y eliminación de duplicados.
- En la aplicación, la fase 1 permite registrar accesos (login) con el timestamp actual del sistema, buscar por usuario y fecha (usuario + timestamp) y mostrar el registro ordenado.

2.4 Actividad 4 - Fase 2: Control de seguridad

- La clase Security mantiene una pila con tres niveles (BAJO, MEDIO, ALTO).
- El método restart carga los tres niveles, y changeLevel usa recursión para mover el nivel solicitado a la cima.
- En el menú de la fase 2 se puede cambiar de nivel y realizar la verificación de identidad, solicitando usuario para BAJO, usuario y contraseña para MEDIO, y usuario, contraseña y teléfono para ALTO.

2.5 Actividad 5 - Fase 3: Monitorización de actividades

Se crea la estructura Actividad (usuario, descripción, timestamp) y la cola enlazada colaActividades. La inserción en la cola se hace de forma recursiva y el listado de actividades por usuario recorre recursivamente los nodos para filtrar coincidencias.

Ejemplos:

```
1.  void insertarRecursivo(nodoCola* nodo, nodoCola* nuevo) { ... }
2.  void mostrarRecursivo(nodoCola* nodo, string usuario) { ... }
```

La fase 3 incluye un perfil supervisor que agrega actividades con el timestamp actual y un perfil usuario que consulta sus actividades.

2.6 Actividad 6 - Fase 4: Análisis de seguridad

La clase Analista recorre de forma recursiva la lista de accesos para contar el número de accesos por usuario. Para detectar amenazas, recorre la cola y busca actividades repetidas del mismo usuario en un periodo corto (ventana temporal configurable, por defecto 300 segundos), generando un informe con la diferencia de tiempo.

2.7 Actividad 7 - Manual de usuario

1. Fase 1: Gestión de Accesos
2. Fase 2: Control de Seguridad
3. Fase 3: Monitorización
4. Fase 4: Análisis de Seguridad
5. Salir

En la fase 1 se puede registrar accesos, buscar por usuario y fecha, mostrar registros y eliminar duplicados.

En la fase 2 se gestionan niveles y se verifica la identidad según el nivel.

La fase 3 incluye un submenú para agregar actividades (perfil supervisor) o listar actividades del usuario.

En la fase 4 se muestran los resultados del análisis de accesos y las alertas de actividades sospechosas.

2.8 Actividad 8 - Pruebas y resultados

Las pruebas se realizan de forma manual desde main.cpp con datos precargados:

- Fase 1: accesos precargados con duplicados y registro adicional mediante login, verificación de orden y búsqueda.
- Fase 2: cambios de nivel y verificación de requisitos solicitados.
- Fase 3: inserción de actividades con timestamps y consulta por usuario.
- Fase 4: conteo de accesos por usuario y detección de actividades repetidas dentro de la ventana temporal.

Los resultados observados son coherentes con el comportamiento esperado en cada fase.

Capítulo 3. Conclusiones

3.1 Conclusiones del trabajo

Se ha desarrollado una aplicación modular que integra lista, pila y cola con uso de recursión para operaciones clave. El sistema permite simular accesos, gestionar niveles de seguridad, monitorizar actividades y analizar patrones sospechosos. El proyecto cumple los objetivos del enunciado y queda preparado para ampliaciones (búsquedas por fecha, mejoras en reglas de detección y almacenamiento persistente).

Capítulo 4. Referencias

- Documento PEL EC5 v2.pdf
- Apuntes y ejemplos de clase de PEL
- Documentación C++: https://en.cppreference.com