Universidad Andrés Bello

Taxonimia de bloom

Samuel Angulo

Benjamín cofre

Keyly Molina

Genesis Moreno

Carlos Petit

Introduction

En el contexto de la asignatura Paradigmas de Programación, se desarrolló un sistema en C++ con el propósito de generar evaluaciones escritas categorizadas de acuerdo con los niveles de la Taxonomía de Bloom. Esta taxonomía es ampliamente utilizada en el ámbito educativo para clasificar los objetivos de aprendizaje, promoviendo así la construcción de preguntas que evalúan desde habilidades básicas de memorización hasta capacidades complejas de análisis y creación.

El presente informe tiene como objetivo principal documentar el proceso de diseño, implementación y decisiones técnicas adoptadas durante el desarrollo del sistema. Se busca explicar tanto la motivación detrás del proyecto como los elementos que lo componen, considerando especialmente las limitaciones impuestas por la consigna: evitar el uso de estructuras dinámicas como vectores (std::vector) y optar por alternativas más simples, como arreglos estáticos.

El documento está estructurado en tres secciones: la introducción, la descripción de la solución propuesta con sus componentes principales, y finalmente una conclusión con reflexiones personales sobre el trabajo realizado y el cumplimiento de los objetivos.

Descripción de la Solución

Objetivo del sistema

El sistema tiene como fin principal facilitar la creación de evaluaciones escritas que se ajusten a diferentes niveles cognitivos. Para ello, permite registrar preguntas de opción múltiple y preguntas de verdadero/falso, asignándoles un nivel específico de la taxonomía de Bloom, junto con otros atributos como el tiempo estimado de resolución, el año en que fue utilizada y la respuesta correcta.

Estructura general del sistemaEl diseño del sistema se basa en principios de programación orientada a objetos, donde se define una clase base abstracta (PreguntaBase) y dos clases derivadas que representan tipos concretos de preguntas: PreguntaMultiple y PreguntaVF. Esta arquitectura permite aplicar polimorfismo y facilitar la extensión del sistema con nuevos tipos de preguntas en el futuro.

Los principales componentes son los siguientes:

Clase PreguntaBase: clase abstracta que define la interfaz común para todas las preguntas. Contiene atributos generales como id, enunciado, nivel, tiempoEstimado, respuestaCorrecta y anioUso. Declara métodos virtuales puros que deben ser implementados por las clases hijas, tales como mostrar(), tipo(), crearDesdeConsola(), guardar() y cargar().

Clase PreguntaMultiple: Representa una pregunta de opción múltiple. Utiliza un arreglo estático de tamaño fijo (string opciones[10]) para almacenar las posibles respuestas. Esta clase permite ingresar preguntas con hasta 10 opciones, especificar cuál es la correcta e imprimir la pregunta con formato de letras (A, B, C, etc.).

Clase PreguntaVF: Modela preguntas del tipo verdadero/falso. Además de los atributos generales, incorpora un campo adicional para la justificación de la respuesta. Utiliza un valor booleano para indicar si la afirmación es verdadera o falsa.

Funciones auxiliares (nivelToStr y strToNivel): Se utilizan para convertir entre enteros y cadenas de texto correspondientes a los niveles de Bloom, lo cual facilita la interacción con el usuario y el almacenamiento en archivos.

Decisiones de diseño

Uno de los desafíos principales del proyecto fue prescindir del uso de std::vector o estructuras dinámicas, lo cual impuso ciertas restricciones en cuanto a flexibilidad y manejo de datos. Para suplir esta limitación, se optó por:

Uso de arreglos estáticos: se definió un máximo de 10 opciones por pregunta de opción múltiple, utilizando un arreglo de tamaño fijo. Esta decisión permite una implementación más sencilla sin uso de memoria dinámica, a costa de limitar el número de opciones posibles.

Diseño orientado a objetos: se aprovecharon conceptos como herencia y polimorfismo para construir una arquitectura extensible, manteniendo al mismo tiempo la cohesión del sistema. Esto permite que el tratamiento de preguntas de diferentes tipos sea uniforme y modular.

Persistencia mediante archivos de texto: se implementaron métodos para guardar y cargar preguntas en archivos CSV, lo que permite la reutilización de preguntas sin necesidad de volver a ingresarlas manualmente.

Entrada por consola: todas las interacciones con el usuario se realizan a través de la consola. Esto incluye la creación de preguntas, ingreso de datos y selección de opciones. Se privilegió una interfaz sencilla que prioriza la funcionalidad sobre la experiencia visual.

Validación mínima: para mantener la simplicidad del código y ajustarse a los objetivos pedagógicos, se decidió aplicar validaciones mínimas en la entrada del usuario (por ejemplo, que los índices de las opciones estén en el rango correcto).

Limitaciones conocidas

No se almacena una colección completa de preguntas en memoria (al menos no con estructuras dinámicas), por lo tanto, el sistema debe ser modificado o extendido si se desea generar una evaluación completa con múltiples preguntas cargadas al mismo tiempo.

No se implementaron mecanismos de ordenamiento o filtrado avanzado, aunque el diseño modular permite agregar estas funcionalidades en futuras versiones.

Conclusión

El desarrollo de este sistema permitió poner en práctica diversos conceptos del paradigma orientado a objetos en C++, tales como la herencia, el polimorfismo, el encapsulamiento y la reutilización de código. Asimismo, el desafío de evitar el uso de estructuras dinámicas impulsó el diseño de una solución eficiente y adaptada a restricciones reales de programación.

Se logró cumplir con los objetivos planteados inicialmente: construir una herramienta capaz de manejar preguntas clasificadas por niveles cognitivos, diferenciadas por tipo, y persistentes en archivos externos. A través de la implementación de una interfaz intuitiva desde consola, el sistema permite al usuario ingresar y visualizar preguntas de manera estructurada.

Finalmente, el trabajo sirvió para profundizar la comprensión de la programación en C++ y evidenció la importancia de planificar cuidadosamente el diseño de una aplicación antes de implementarla. En particular, el uso de una clase base abstracta permitió mantener una arquitectura limpia y extensible, que podría fácilmente ampliarse a otros tipos de preguntas en el futuro (como preguntas abiertas o con imágenes, por ejemplo).