Algoritmos y Estructuras de Datos I

Trabajo Práctico N°8	Unidad 8
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo grupal
Metodología de Desarrollo: acordar	**Metodología de Corrección:** acordar docente
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

MARCO TEÓRICO

1. ¿Qué es abstracción en programación?

La abstracción en programación es la simplificación de problemas complejos eliminando detalles irrelevantes para centrarse en los aspectos esenciales. Además, el concepto de abstracción de datos permite crear tipos de datos abstractos, separando las propiedades lógicas de su implementación, lo que facilita la reutilización del software en distintos programas. En C++, la abstracción se implementa mediante clases, las cuales permiten definir tipos de datos abstractos.

2. ¿Qué relación hay entre Modelo y Abstracción?

La relación entre modelo y abstracción se centra en el uso de diagramas como los de entidad-relación, clases y casos de uso para visualizar componentes de un sistema. Una clase es un modelo que abstrae un concepto y permite la creación de objetos basados en esa abstracción, facilitando la representación y organización de las entidades y relaciones relevantes del problema.

3. ¿Por qué usaría un “Proceso de Abstracción” para resolver un Problema?

El proceso de abstracción en el desarrollo de software implica identificar el dominio del problema y los requisitos del software. Al centrarse en los aspectos relevantes y eliminar los detalles innecesarios, se facilita la modelación del problema en términos de objetos y relaciones, lo cual es clave para una solución más clara y eficiente.

4. ¿Qué es la descomposición en el proceso de Abstracción?

La descomposición en módulos facilita la modificación y el mantenimiento del software, ya que la mayoría de los cambios afectan solo a uno o pocos módulos. Esta técnica reduce la complejidad al ocultar detalles irrelevantes para el usuario del módulo, lo que también minimiza la interdependencia entre módulos, simplificando el proceso de abstracción.

5. ¿Asocie abstracción con Descomposición?

La abstracción y la descomposición son conceptos interrelacionados en el desarrollo de software. La descomposición en módulos facilita la modificación y el mantenimiento del software, ya que permite que la mayoría de los cambios se realicen en uno o pocos módulos, lo que reduce la complejidad del sistema. Al ocultar detalles irrelevantes para el usuario del módulo, se minimiza la interdependencia entre módulos, lo que simplifica el proceso de abstracción.

Por su parte, la abstracción en programación implica simplificar problemas complejos al eliminar detalles innecesarios y centrarse en los aspectos esenciales. Este proceso de abstracción permite la creación de tipos de datos abstractos, separando las propiedades lógicas de su implementación. Esta separación no solo facilita la reutilización del software en distintos programas, sino que también es fundamental en lenguajes como C++, donde se implementa a través de clases que permiten definir y gestionar estos tipos de datos abstractos.
Así, la descomposición contribuye a una mejor práctica de la abstracción, al organizar el software en módulos manejables, lo que a su vez mejora la claridad y la eficacia en la resolución de problemas complejos.

6. ¿Explicar los Mecanismos de Abstracción?

Estos mecanismos de abstracción permiten a los programadores diseñar y manejar estructuras de datos más complejas y flexibles, simplificando el desarrollo y el mantenimiento de software. Podemos encontrar:
Tipos de Enumeración: Permite definir un tipo de enumeración listando los identificadores que constituyen su dominio. Este tipo de datos es distinto a los tipos integrados, proporcionando una forma de representar datos de manera más específica para el contexto del programa.
Typedef: Aunque la sentencia typedef no crea un nuevo tipo de datos, permite renombrar un tipo existente. Esto puede facilitar la legibilidad del código al dar nombres más descriptivos a tipos que ya existen.
Clases: La clase es uno de los mecanismos más poderosos de abstracción en C++. Permite la creación de tipos de datos abstractos, encapsulando datos y funciones dentro de una sola entidad. A través de las clases, los programadores pueden modelar estructuras de datos complejas, proporcionando tanto la declaración como la manipulación de datos de manera eficiente.
Constructores: Los constructores son funciones miembro especiales que garantizan la correcta inicialización de los objetos de una clase. C++ permite definir múltiples constructores con diferentes listas de parámetros, lo que proporciona flexibilidad al momento de crear objetos. Si no se especifica un constructor por defecto y se intenta crear un objeto sin argumentos, el compilador arroja un error de sintaxis.
Destructores: Complementando a los constructores, los destructores son invocados automáticamente cuando un objeto de clase es destruido, por ejemplo, al salir del bloque donde fue declarado. Esto asegura que los recursos utilizados por el objeto se liberen adecuadamente.

7. ¿Explicar los Tipos de Abstracción?

Un tipo de datos abstracto (TDA) es una estructura cuyas propiedades (dominio y operaciones) se especifican de forma independiente de cualquier implementación específica. Para manejar los TDA, se definen tres categorías principales de operaciones:

Constructores: Estas operaciones crean una nueva instancia de un TDA. Por ejemplo, crear una nueva lista es una operación de constructor.
Transformadores: Modifican un TDA a partir de uno o más valores previos. Por ejemplo, insertar o eliminar un elemento de una lista son operaciones transformadoras.
Observadores: Permiten examinar el estado de un TDA sin modificarlo. Por ejemplo, una función booleana que verifica si una lista está vacía o si contiene un valor específico.
Iteradores (opcional): Estas operaciones permiten procesar, uno a uno, los componentes de un TDA. Un ejemplo es una operación que devuelve el primer elemento de una lista y, sucesivamente, los siguientes elementos en cada llamada.

8. ¿Qué entiende por TDA?

Un Tipo de Dato Abstracto (TDA) es un concepto fundamental en programación que define una estructura de datos junto con un conjunto de operaciones que pueden realizarse sobre esa estructura. El TDA proporciona una interfaz para interactuar con la estructura de datos, ocultando su implementación interna. Esto significa que el usuario puede trabajar con los datos sin conocer los detalles de cómo están implementados.
Un TDA como una Lista podría definir una colección de elementos y ofrecer operaciones como "insertar", "eliminar" o "obtener" elementos, sin que el usuario sepa si la lista está implementada mediante un arreglo, una lista enlazada u otra estructura.
El uso de un TDA facilita la reutilización de código y la abstracción de la complejidad de la implementación, permitiendo que existan múltiples implementaciones del mismo TDA sin afectar a los usuarios finales.

9. ¿Qué mejoras aportan los TDA?

Los Tipos de Datos Abstractos (TDA) mejoran el desarrollo de software al encapsular datos, ocultando su implementación interna y permitiendo que las operaciones sean claras y seguras. Facilitan la reutilización de código, hacen que los programas sean más fáciles de mantener y escalar, y promueven la claridad y legibilidad. Además, al controlar el acceso a los datos, los TDA aumentan la seguridad y robustez del software, reduciendo errores y garantizando consistencia.

10. ¿Explicar la Visión Externa y la Visión Interna de los TDA?

La visión externa de un Tipo de Dato Abstracto (TDA) se refiere a la interfaz pública que expone los métodos y operaciones que pueden realizarse sobre el TDA. Desde esta perspectiva, los usuarios interactúan con el TDA sin conocer ni necesitar entender cómo están implementados internamente los datos o las operaciones. La visión externa solo muestra el comportamiento accesible, es decir, lo que el TDA puede hacer, pero no cómo lo hace.
Por otro lado, la visión interna se refiere a la implementación oculta del TDA, que incluye la representación de los datos (estado) y la forma en que se llevan a cabo

las operaciones. Esta parte interna no es accesible para los usuarios y define cómo se manipulan los datos dentro del TDA. En programación orientada a objetos, los TDAs suelen implementarse a través de clases, donde los atributos (estado) y los métodos (comportamiento) están encapsulados.

11. ¿Qué es Encapsulamiento?

El encapsulamiento es un principio fundamental de la programación orientada a objetos que consiste en agrupar tanto el estado como el comportamiento de un objeto en una unidad, generalmente una clase. Este concepto permite ocultar los detalles internos de la implementación, creando una "barrera de abstracción" que sólo puede ser atravesada a través de la interfaz pública del objeto, es decir, los métodos accesibles por el usuario. Al encapsular los datos y sus operaciones, se impide que el usuario dependa de los detalles internos o los manipula incorrectamente, lo que facilita la seguridad y la integridad del software. Además, el encapsulamiento facilita la realización de cambios en la implementación sin afectar a quienes utilizan el objeto.

12. ¿Qué entiende por Interfaces?

Una interfaz en programación es un conjunto de definiciones que establece cómo interactúan los diferentes componentes del software entre sí.
Se enfoca en la modularidad y el ocultamiento de la complejidad interna, permitiendo que los módulos interactúen sin conocer los detalles de implementación del otro.
Las interfaces pueden estar representadas por clases abstractas, interfaces de implementación o patrones de diseño, y sirven para definir cómo los componentes o TDA interactúan entre sí. Además, cuando se diseñan funciones, se deben establecer claramente los parámetros y su comportamiento, por ejemplo, usando “const” para arrays que no deben modificarse. Para un desarrollo exitoso en equipo, es esencial definir explícitamente las interfaces entre módulos y asegurarse de que el código cumpla con las especificaciones acordadas, evitando el uso de variables globales que puedan causar conflictos entre los programadores.

Lic. Oemig José Luis.