Instituto Superior de Formación Técnica N° 151

Carrera: Analista de Sistemas

Algoritmos y Estructuras de Datos I

1 Año. FINAL

Marco Teórico:

1. ¿Qué son los paradigmas?, comparar Estructurado VS Objetos.

Los paradigmas de programación son formas de abordar el desarrollo de software con diferentes enfoques y principios.

El paradigma imperativo o estructurado	El paradigma orientado a objetos
Se centra en describir la programación en términos del estado del programa y las instrucciones que lo modifican, es como una receta donde cada paso es una instrucción, y hay una clara separación entre datos y métodos. Tiene presente la separación de datos y métodos.	Define programas en términos de comunidades de objetos que combinan estado y comportamiento, agrupa objetos con características comunes en clases y permite que los objetos se comuniquen entre sí para realizar tareas. Facilitar la escritura, mantenimiento y reutilización del código. Los principios fundamentales del paradigma orientado a objetos incluyen encapsulación, herencia, polimorfismo y abstracción

2. ¿Qué es la Abstracción, dar ejemplos?

La abstracción es un concepto fundamental que permite a los programadores representar elementos del espacio del problema mediante herramientas que no limitan a un tipo específico de problema.

La complejidad de los problemas que se pueden resolver está relacionada con el tipo y calidad de la abstracción. Los lenguajes de programación ofrecen diferentes niveles de abstracción, desde el lenguaje ensamblador hasta lenguajes más abstractos.

La programación orientada a objetos proporciona una abstracción más flexible y potente, permitiendo describir problemas en sus propios términos en lugar de términos computacionales. Cada objeto se asemeja a una pequeña computadora con estado y operaciones, facilitando la representación de características y comportamientos en el programa.

Un ejemplo sería un sistema de gestión de bibliotecas donde se define una clase llamada "Libro" que encapsula atributos como título, autor y año de publicación, y métodos para realizar acciones como prestar o devolver un libro. La abstracción en programación orientada a objetos está estrechamente vinculada a clases y objetos, donde cada objeto es una instancia de una clase que define un tipo con características y comportamientos comunes.

Los objetos almacenan datos y pueden realizar operaciones sobre sí mismos, representando componentes conceptuales del problema. La creación de nuevos tipos de objetos permite ocultar la complejidad, y cada objeto tiene su propia memoria compuesta de otros objetos.

3. ¿Qué es el Isomorfismo Estructuras y el GAP semántico?

El Gap Semántico es un modelo teórico de un problema se podría decir que es la distancia entre realidad y código, el cual, se debe minimizar creando abstracciones y estructura de datos que se ajusten lo más posible para facilitar la comprensión. **El isomorfismo** estructural es la correspondencia entre dos estructuras diferentes que tienen la misma forma y funcionalidad, haciendo que el programa trabaje de manera coherente con las distintas partes. Es parte de la solución, hace que el código tenga la misma forma que la realidad para reducir esa distancia.

4. ¿Qué es un Puntero? Dar un ejemplo de creación Objeto en Memoria dinámica

Los punteros son variables que almacenan la dirección de memoria de otros datos, permitiendo así el acceso indirecto a dichos datos. Son útiles para manejar arreglos, memoria dinámica y mejorar el rendimiento al acceder directamente a la memoria. Los punteros permiten guardar la dirección de memoria de una variable en lugar de su valor, y son cruciales para la manipulación de arreglos y la memoria dinámica. Los datos dinámicos creados en el montón o heap solo pueden ser accedidos a través de punteros, ya que no están asociados a un identificador de variable estática.

En C++ existen dos operadores clave para trabajar con punteros. **El operador ampersand (&)** te permite obtener la dirección de memoria donde está guardada una variable, mientras que **el operador asterisco (*)** te permite acceder al valor que está almacenado en la dirección a la que apunta el puntero. La memoria dinámica se crea en una zona llamada "heap" o montón, y a diferencia de las variables normales que se crean automáticamente en el stack, los datos en el heap solo pueden ser accedidos mediante punteros porque no tienen un nombre de variable asociado

Para crear objetos en memoria dinámica usamos el operador "new". Este operador solicita al sistema un espacio de memoria del tamaño necesario para el tipo de

dato que querés crear, y te devuelve la dirección de memoria donde se guardó ese dato. Por ejemplo, si escribís "p = new int", estás pidiendo memoria para guardar un entero, y la dirección de esa memoria se guarda en el puntero p. También podés inicializar ese valor al crearlo escribiendo "p = new int(12)", lo cual crea el espacio para un entero y le pone el valor 12 desde el principio.

Un detalle importante es que en C++ vos sos responsable de liberar la memoria que pedís con "new". Cuando ya no necesitás más ese dato, tenés que usar el operador "delete" para devolver esa memoria al sistema, sino vas a tener fugas de memoria y tu programa va a consumir cada vez más recursos. Si trabajás con estructuras mediante punteros, en vez de usar el punto (.) para acceder a los campos, tenés que usar el operador flecha (->). Por ejemplo, si tenés una estructura Registro y un puntero "puntero" que apunta a esa estructura, para acceder a un campo escribirías "puntero->campo1" en lugar de "puntero.campo1".

Ejemplo de creación de objeto en memoria dinámica:

```
// Declaración de puntero

int *p;

// Asignación de memoria dinámica (básica)

p = new int;

// Asignación de memoria dinámica con inicialización

p = new int(12);  // Asigna memoria para un entero y lo inicializa en 12
```