Instituto Tecnológico de Las Américas

Tema

Introduccion a clases y objectos

Participante

Robert Adolfo Santana

Facilitador

Francis Ramirez

Fecha

29 abril de 2025

¿Qué son las clases y los objetos?

◆ Concepto de clase

Una **clase** es una plantilla o modelo que define un tipo de dato personalizado. Incluye **atributos** (datos) y **funciones** (comportamientos) que actúan sobre esos datos. En C++, el lenguaje ofrece soporte directo para clases. Sin embargo, en **C puro**, este concepto se simula mediante:

- struct para definir los datos (atributos)
- Funciones asociadas para representar los comportamientos

◆ Concepto de objeto

Un **objeto** es una **instancia de una clase**. En C, cuando se crea una variable de tipo struct, se está creando un "objeto" de esa estructura.

Ejemplo:

c
CopiarEditar
```
typedef struct {
    int dia;
    int mes;
    int anio;
} Fecha;

Fecha hoy = {29, 4, 2025};  // objeto de tipo Fecha
```

Encapsulamiento: Ocultamiento de la información

El **encapsulamiento** es una característica clave de la POO que busca ocultar la implementación interna de un tipo de dato, exponiendo solo lo necesario al usuario.

Aunque C no tiene modificadores de acceso como private o public, se puede simular el encapsulamiento colocando solo la declaración de la estructura en el archivo .h y definiendo las funciones en el .c y no esponiendo los detalles internos

Esto mejora la **modularidad** y permite cambios sin afectar a los programas que usan el módulo.

Definición y uso de estructuras (struct)

Una estructura permite agrupar distintos tipos de datos bajo un solo nombre.

Ejemplo:

c
CopiarEditar
```
typedef struct {
    char nombre[50];
    int edad;
    float promedio;
} Estudiante;
```

Aquí Estudiante actúa como una "clase" que tiene tres atributos. Para manejar el comportamiento, se escriben funciones como:

c
CopiarEditar
```
void imprimirEstudiante(Estudiante e) {
    printf("Nombre: %s\nEdad: %d\nPromedio: %.2f\n", e.nombre, e.edad, e.promedio);
}
```

Métodos o funciones asociadas a estructuras

En la POO, los métodos operan sobre los objetos. En C, esto se logra escribiendo funciones que reciben un puntero o copia de la estructura como argumento.

Ejemplo con puntero:

CopiarEditar
```
void actualizarEdad(Estudiante *e, int nuevaEdad) {
    e->edad = nuevaEdad;
}
```

Esta forma imita el comportamiento de los métodos de una clase y se puede llamar así:

CopiarEditar
```
actualizarEdad(&alumno, 21);
```

Inicialización y uso de objetos

En C, puedes inicializar estructuras de diferentes maneras:

Inicialización directa:

CopiarEditar
Estudiante e1 = {"Laura", 20, 8.5};

Asignación por campos:

CopiarEditar
Estudiante e2;
strcpy(e2.nombre, "Juan");
e2.edad = 22;
e2.promedio = 7.9;

Estas instancias representan objetos concretos con valores particulares.

Abstracción

La **abstracción** permite centrarse en lo que hace un objeto en lugar de cómo lo hace. Se logra definiendo **interfaces claras** (funciones públicas) y ocultando la implementación.

Esto se traduce en ofrecer funciones bien nombradas y documentadas que permiten al usuario manipular los objetos sin conocer su implementación interna.

7. Modularidad y separación en archivos

Una buena práctica en programación estructurada (y orientada a objetos simulada en C) es **dividir el código**:

- Archivo .h: contiene la definición de estructuras y funciones públicas (interfaz)
- Archivo .c: contiene la implementación de las funciones

Ventajas:

- Separación de responsabilidades
- Mejor mantenimiento
- Reutilización del código

Ventajas de aplicar la POO en C

A pesar de no tener clases formales, **emular la orientación a objetos en C** trae beneficios como:

Principio	Beneficio en C	Cómo se logra
Encapsulamiento	Protege datos internos	Uso de punteros y archivos `.h`
Modularidad	Código limpio y organizado	Separación de `.c` y `.h`
Reutilización	Componentes usables en múltiples programas	Diseño genérico de structs y funciones
Abstracción	Oculta detalles complejos	Interfaz clara (funciones públicas)
Mantenibilidad	Fácil modificación sin romper código	Interfaz estable, implementación separada

---