CENTRO
DE COMPETENCIAS
TECMILENIO

Structs y clases

Structs y clases

Las **structs** y las **clases** son dos conceptos muy importantes en la programación orientada a objetos.

Las **clases (class)** y las **structs** son construcciones con las que define sus propios tipos, pueden contener miembros de datos y funciones miembro, permitiendo con esto describir el comportamiento y el estado del tipo.

Observemos sus definiciones, según Microsoft (2023):

Class. Declara un tipo de clase o define un objeto de un tipo de clase.

Sintaxis

```
[template-spec]
class [ms-decl-spec] [tag [:
base-list ]]
{
   member-list
} [declarators];
[ class ] tag declarators;
```

Struct. Define un tipo de estructura o variable de un tipo de estructura.

Sintaxis

```
[template-spec] struct [ms-
decl-spec] [tag [: base-list ]]
{
   member-list
} [declarators];
[struct] tag declarators;
```

La única diferencia que se puede observar entre estos elementos es que si no se especifica la visibilidad (privada, pública o protegida) de los miembros, en struct serán públicos y en class serán privados.

Tanto las **structs** como las **clases** pueden definir:

1. Propiedades para almacenar valores.
2. Métodos para proporcionar funcionalidad.
3. Inicializadores para configurar su estado inicial.
4. Subíndices para proporcionar acceso a sus valores utilizando la sintaxis de subíndices.

Además, alcanzan a extenderse para expandir su funcionalidad más allá de una implementación predeterminada. Se adaptan a los protocolos para proporcionar una funcionalidad estándar de cierto tipo, incluyen propiedades calculadas, según Baek (2022).

Las **clases** tienen capacidades adicionales a las **structs:**

1. *Herencia,* esta permite que una clase herede las características de otra.
2. *Typecasting* permitirá verificar y desarrollar el tipo de instancia de clase en tiempo de ejecución.
3. *Desinicializadores* autorizan que una instancia de una clase libere cualquier recurso que haya asignado.
4. *El conteo de referencias* permite más de una referencia a una instancia de clase.
5. *Los operadores de identidad* para verificar si dos constantes o variables se refieren a la misma instancia única: Idéntico a (===); No idéntico a (!==).

Las **structs** tienen capacidades adicionales que las clases no tienen, que son los inicializadores de miembros. Estos pueden usarlo para inicializar las propiedades de los miembros de la nueva instancia de estructura.

Las **structs** son una estructura de datos simple que se usa para agrupar datos relacionados, no admiten herencia ni métodos, se usan con frecuencia para estructuras de datos pequeñas y simples donde el rendimiento es una preocupación.

Las **clases** son muy complejas, debido a que permiten herencia, encapsulación y polimorfismo. Pueden contener miembros de datos como funciones de miembros (métodos), que se pueden usar para manejar los datos. Se usan frecuentemente para moldear objetos o sistemas complejos en el mundo real.

Las **structs** y las **clases** son muy similares en lenguaje C++, con la única diferencia de que, en forma preestablecida, los miembros de la estructura son públicos y los de las clases privados. En otros lenguajes como en JAVA, no hay distinción.

Para elegir entre **structs** y **clases** debemos revisar los requisitos específicos del programa. Si requerimos una estructura de datos simple sin comportamiento structs será el camino para seguir. Si, por otro lado, necesitamos un comportamiento más complejo o vamos a modelar un objeto de la vida real, una clase será la mejor opción.

Referencias bibliográficas

- Microsoft. (2023). *Clases y structs (C++)*. Recuperado de https://learn.microsoft.com/es-es/cpp/cpp/classes-and-structs-cpp?view=msvc-170

- Baek, S. (2022). *Struct vs. Class: Deciding between Structs and Classes*. Recuperado de https://www.shawnbaek.-com/posts/struct-vs-class-deci-de-how-to-store-data-and-model-behavior