Enumeración (enum)

Las enumeraciones en C++ (también conocidas como enum) son una forma de definir un conjunto de constantes con nombre. Se utilizan para asignar nombres a valores enteros, lo que mejora la legibilidad y el mantenimiento del código.

1. ¿Qué es una Enumeración (enum) en C++?

Una **enumeración** es un tipo de datos que consiste en un conjunto de **valores constantes**. Cada valor de una enumeración es un identificador que representa un número entero.

2. Creación de una Enumeración

Para declarar una enumeración, se utiliza la palabra clave enum seguida por el nombre de la enumeración y el conjunto de identificadores entre llaves {}.

Ejemplo Básico:

```
enum DiaDeLaSemana {
      Lunes,
      Martes,
      Miercoles,
      Jueves,
      Viernes,
      Sabado,
      Domingo
};
```

En este ejemplo, DiaDeLaSemana es un tipo de enumeración que tiene siete valores:
Lunes, Martes, Miercoles, etc. Por defecto:

Lunes tiene el valor 0.
Martes tiene el valor 1.

Y así sucesivamente.

3. Uso de Enumeraciones

Una vez que has definido una enumeración, puedes declarar variables de ese tipo y asignarles cualquiera de los valores definidos.

Declaración de una Variable:

```
DiaDeLaSemana hoy = Lunes;
```

Impresión del Valor de una Enumeración:
Si deseas imprimir el valor de una enumeración, lo verás como un número entero.

```
std::cout << hoy << std::endl; // Salida: 0
```

4. Asignación de Valores Específicos a Enumeradores
Puedes asignar valores específicos a los identificadores en una enumeración. Esto es útil si deseas que los valores comiencen desde un número distinto o si deseas saltarte números.

Ejemplo con Valores Específicos:

```
enum Mes {
      Enero = 1,
      Febrero,
      Marzo,
      Abril = 10,
      Mayo,
      Junio
};
```

Aquí:
Enero tiene el valor 1.
Febrero tiene el valor 2.
Marzo tiene el valor 3.
Abril tiene el valor 10.
Mayo tendrá el valor 11.
Junio tendrá el valor 12.

5. Modificación de una Variable enum
Una vez que has declarado una variable de tipo enum, puedes cambiar su valor a cualquiera de los valores definidos en la enumeración.

Ejemplo:

```
hoy = Viernes; // Cambia el valor de hoy a Viernes (4)
```

6. Enumeraciones enum class en C++11 y Posteriores
A partir de C++11, se introdujo enum class para mejorar la seguridad de tipo. A diferencia de los enums tradicionales, los valores de enum class no se convierten implícitamente a enteros y los identificadores están en un espacio de nombres.

Ejemplo de enum class:

```
enum class Color {
     Rojo,
     Verde,
     Azul
};
```

Uso de enum class:

```
Color colorFavorito = Color::Verde; // Uso de "Color::" para referenciar los valores
```

Acceso al Valor Subyacente:
Si necesitas obtener el valor entero subyacente, puedes hacerlo con una conversión explícita:

```
int valor = static_cast<int>(Color::Verde); // valor será 1
```

7. Restricciones y Consideraciones
No se pueden eliminar valores de una enumeración una vez que han sido definidos.
No se pueden añadir nuevos valores a una enumeración ya declarada dentro del mismo bloque de código.
No se puede cambiar el valor de un identificador una vez que la enumeración ha sido compilada.
Resumen
Creación: enum NombreEnum { Ident1, Ident2, ... };
Asignación de valores específicos: enum NombreEnum { Ident1 = 5, Ident2, ... };
Uso: NombreEnum var = Ident1;
Enumeraciones de clase (enum class): Ofrecen un mejor control y seguridad de tipos.

En C++, un enum (enumeración) se utiliza para declarar un *conjunto de constantes* bajo un nombre de tipo. Aquí te explico cómo trabajar con enumeraciones y cómo mostrar sus valores en un ciclo for.

Ejemplo básico de enum
Primero, veamos un ejemplo simple donde definimos colores y luego los mostramos:

```
#include <iostream>

enum Colores {
    blanco,   // valor 0
    negro,    // valor 1
    azul,     // valor 2
    amarillo  // valor 3
};

int main() {
    system("clear");
    std::cout << "\n\n";

    for (int i = blanco; i <= amarillo; i++) {
        if (i == 0) {
            std::cout << "\nEs el primer color: blanco";
        } else if (i == 1) {
            std::cout << "\nEs el segundo color: negro";
        } else if (i == 2) {
            std::cout << "\nEs el tercer color: azul";
        } else if (i == 3) {
            std::cout << "\nEs el cuarto color: amarillo";
        }
    }

    std::cout << "\n\n";
    return 0;
```

```
}
```

Explicación:

El enum Colores contiene tres elementos: Rojo, Verde y Azul, que por defecto tienen valores enteros consecutivos empezando en 0.
En el ciclo for, iteramos sobre los valores de Rojo a Azul y *usamos condiciones para mostrar* los nombres correspondientes.

Ejemplo con valores específicos en enum
Ahora veamos un caso en el que asignamos valores específicos a cada constante en el enum:

```
#include <iostream>
using namespace std;

// Definición de un enum con valores específicos
enum Mensajes { Hola = 1, Buenos = 2, Dias = 3 };

int main() {
	// Mostrar los valores del enum
	for (int i = Hola; i <= Dias; i++) {
		switch (i) {
			case Hola:
				cout << "Hola = " << i << endl;
				break;
			case Buenos:
				cout << "Buenos = " << i << endl;
				break;
			case Dias:
				cout << "Dias = " << i << endl;
				break;
		}
	}
	return 0;
}
```

Explicación:

Aquí definimos el enum Mensajes, donde *Hola* tiene un valor de 1, *Buenos* tiene un valor de 2, y Dias tiene un valor de 3.
Usamos un ciclo for que va desde *Hola* hasta *Dias*, y un switch para mostrar el nombre y valor correspondiente de cada elemento.

Notas importantes:
Los elementos del enum se pueden recorrer usando un ciclo for siempre que los valores sean consecutivos.
Si no son consecutivos o hay saltos en los valores, es necesario utilizar un switch o alguna lógica adicional para imprimir cada valor con su respectivo nombre.

Nota:

En C++, los enum son tipos definidos en tiempo de compilación, lo que significa que no se pueden modificar en tiempo de ejecución como si fueran una lista o arreglo. Sin embargo, podemos simular este comportamiento utilizando otras estructuras de datos, como std::map, para almacenar nombres y valores ingresados por el usuario de forma dinámica.

Voy a mostrarte cómo podrías implementar una simulación de un enum cargado dinámicamente utilizando std::map, donde el programa pedirá al usuario los nombres de los elementos y sus valores:

Ejemplo de simulación de un enum dinámico

```
#include <iostream>
#include <map> //punto -SECCION- 27
#include <string>
using namespace std;

int main() {
	int cantidad;
	map<string, int> miEnum;

	// Pedimos al usuario cuántos elementos quiere agregar
	cout << "¿Cuántos elementos deseas ingresar en el enum simulado?: ";
	cin >> cantidad;

	// Ciclo para pedir al usuario los nombres y valores de los elementos
	for (int i = 0; i < cantidad; ++i) {
```

```
            string nombre;
            int valor;
            cout << "Ingresa el nombre del elemento " << i + 1 << ": ";
            cin >> nombre;
            cout << "Ingresa el valor para " << nombre << ": ";
            cin >> valor;

            // Almacenamos en el mapa
            miEnum[nombre] = valor;
      }

      // Mostramos los elementos del "enum"
      cout << "\nElementos del enum simulado:\n";
      for (const auto& elem : miEnum) {
            cout << elem.first << " = " << elem.second << endl;
      }

      return 0;
}
```

Explicación:
Entrada dinámica de elementos:
El programa pregunta cuántos elementos se desean ingresar en el "*enum simulado*".
En cada iteración, el usuario ingresa un nombre (como si fuera un elemento de enum) y un valor asociado (como los valores que podrías asignar a los elementos de un enum tradicional).
Almacenamiento dinámico:
Usamos un std::map<string, int> para almacenar los pares de nombres y valores. Aquí, la clave es el nombre del ***"elemento"*** y el valor es el número asociado.

Salida de los elementos:
El programa recorre el map y muestra cada par *nombre-valor*, simulando el comportamiento de un enum.
Ejemplo de ejecución:

```
¿Cuántos elementos deseas ingresar en el enum simulado?: 3
Ingresa el nombre del elemento 1: Hola
Ingresa el valor para Hola: 1
Ingresa el nombre del elemento 2: Buenos
Ingresa el valor para Buenos: 2
Ingresa el nombre del elemento 3: Dias
Ingresa el valor para Dias: 3

Elementos del enum simulado:
Hola = 1
Buenos = 2
Días = 3
```

Limitaciones:
Esto es una simulación porque los enum reales son constantes que no pueden ser modificados en tiempo de ejecución. Sin embargo, esta solución permite cargar nombres y valores de manera dinámica, lo que no es posible con un enum normal.

Uso Practico:

El uso de enum en C++ es común *cuando se necesita manejar un conjunto de valores constantes* que representan estados o categorías. Aquí te doy algunos ejemplos prácticos de cómo y cuándo se utiliza enum en C++:

1. Uso de enum para representar estados
Un uso muy común es representar estados en un programa. Por ejemplo, en una máquina de café, podrías usar enum para representar los diferentes estados:

```
#include <iostream>
using namespace std;

enum EstadoMaquina { APAGADA, ENCENDIDA, HACIENDO_CAFE, FUERA_DE_SERVICIO };

int main() {
      EstadoMaquina estado = ENCENDIDA;

      if (estado == ENCENDIDA) {
            cout << "La máquina está encendida y lista para hacer café." << endl;
            estado = HACIENDO_CAFE;
      }

      if (estado == HACIENDO_CAFE) {
```

```
        cout << "La máquina está haciendo café." << endl;
    }

    return 0;
}
```

Usos prácticos:
Este tipo de enum *facilita el manejo de múltiples estados con nombres claros en lugar de números o cadenas*, mejorando la legibilidad y el mantenimiento del código.

2. Uso de enum para manejar opciones o categorías
Los **enum** también se pueden utilizar para representar un conjunto de opciones o categorías en un menú, como en una aplicación para gestionar tareas.

```
#include <iostream>
using namespace std;

enum OpcionesMenu { AGREGAR_TAREA = 1, BORRAR_TAREA, MOSTRAR_TAREAS, SALIR };

int main() {
    int opcion;

    cout << "Menú de opciones:\n";
    cout << "1. Agregar Tarea\n";
    cout << "2. Borrar Tarea\n";
    cout << "3. Mostrar Tareas\n";
    cout << "4. Salir\n";

    cout << "Seleccione una opción: ";
    cin >> opcion;

    switch (opcion) {
        case AGREGAR_TAREA:
            cout << "Opción: Agregar Tarea" << endl;
            break;
        case BORRAR_TAREA:
            cout << "Opción: Borrar Tarea" << endl;
            break;
        case MOSTRAR_TAREAS:
            cout << "Opción: Mostrar Tareas" << endl;
            break;
        case SALIR:
            cout << "Opción: Salir" << endl;
            break;
        default:
            cout << "Opción no válida." << endl;
    }

    return 0;
}
```

Usos prácticos:

Se utiliza enum para *asociar opciones con valores numéricos*, haciendo más legible y fácil de mantener un menú interactivo.

3. Uso de enum para representar días de la semana
Otra aplicación práctica de enum es para representar los días de la semana en un programa de gestión de eventos o recordatorios:

```
#include <iostream>
using namespace std;

enum DiaSemana { LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO };

int main() {
    DiaSemana diaActual = MIERCOLES;

    if (diaActual == MIERCOLES) {
        cout << "Hoy es miércoles, día de reunión." << endl;
    }

    return 0;
```

```
}
```

Usos prácticos:

Este tipo de enum es ideal *para manejar operaciones basadas en días de la semana*, como recordatorios automáticos o planificación de eventos.

4. Uso de enum para representar niveles de acceso en un sistema
Los enum pueden ser muy útiles para representar niveles de acceso en un sistema de seguridad o gestión de usuarios:

```
#include <iostream>
using namespace std;

enum NivelAcceso { INVITADO = 1, USUARIO, ADMINISTRADOR };

int main() {
	NivelAcceso nivelUsuario = ADMINISTRADOR;

	if (nivelUsuario == ADMINISTRADOR) {
		cout << "Tienes acceso total al sistema." << endl;
	} else if (nivelUsuario == USUARIO) {
		cout << "Tienes acceso limitado." << endl;
	} else if (nivelUsuario == INVITADO) {
		cout << "Tienes acceso solo a funciones básicas." << endl;
	}

	return 0;
}
```

Usos prácticos:

Ayuda a *manejar los niveles de permiso* de manera clara y estructurada *en lugar de usar números* o *cadenas dispersas en el código*.

5. Uso de enum class para mayor seguridad
En C++11, enum class se introdujo para evitar conflictos de nombres y proporcionar un mayor control de tipo. Es útil cuando se manejan varios enum en el mismo espacio de nombres:

```
#include <iostream>
using namespace std;

enum class Dia { LUNES, MARTES, MIERCOLES };
enum class Mes { ENERO, FEBRERO, MARZO };

int main() {
	Dia diaActual = Dia::MARTES;
	Mes mesActual = Mes::FEBRERO;

	if (diaActual == Dia::MARTES) {
		cout << "Hoy es martes." << endl;
	}

	if (mesActual == Mes::FEBRERO) {
		cout << "Estamos en febrero." << endl;
	}

	return 0;
}
```

Usos prácticos:

enum class proporciona *un espacio de nombres propio para evitar conflictos entre diferentes enumeraciones*, lo que es útil en programas grandes.

Resumen de Usos Prácticos:
Gestión de estados: Facilita el control de diferentes fases o modos de un programa (máquinas, videojuegos, etc.).
Opciones de menú: Proporciona una manera sencilla de manejar múltiples opciones en menús.
Planificación y control: Se utiliza para representar días de la semana, meses o cualquier otro conjunto finito de valores.
Gestión de acceso: Facilita la gestión de permisos y roles de usuarios.
Espacios de nombres (enum class): Evita conflictos de nombres en proyectos más grandes.