Tema: MKDIR

(Conocimiento Opcional).

El comando mkdir en C++ no existe de manera directa como en Bash o terminales de sistemas operativos, pero se puede crear un directorio utilizando funciones que proporciona la biblioteca estándar de C++ o bibliotecas específicas del sistema operativo.

Aquí te explicaré cómo se puede implementar y usar mkdir en C++, proporcionando ejemplos prácticos y detallando cuándo y dónde usarlo.

1. Librerías para crear directorios
Existen varias formas de crear un directorio en C++ dependiendo del sistema operativo y de las librerías que utilices:

a. Uso de filesystem (C++17 en adelante)
A partir de C++17, la biblioteca estándar incluye el módulo filesystem, que facilita muchas operaciones del sistema de archivos, incluyendo la creación de directorios con la función std::filesystem::create_directory.

b. Uso de mkdir **en sistemas** POSIX (Linux/Unix)
En sistemas operativos basados en Unix/Linux, también puedes usar la función mkdir() de la librería <sys/stat.h>.

c. Uso de _mkdir **en** Windows
En sistemas Windows, se usa la función _mkdir de la librería <direct.h>.

2. Implementación en diferentes plataformas
a. Usando std::filesystem::create_directory (C++17)
El método más moderno y portable para crear directorios en C++ es usando std::filesystem. Aquí tienes un ejemplo:

```
#include <iostream>
#include <filesystem> // Necesario para usar filesystem

namespace fs = std::filesystem;

int main() {
	// Especificamos el nombre o la ruta del directorio que queremos crear
	std::string path = "mi_directorio";

	// Intentamos crear el directorio
	if (fs::create_directory(path)) {
		std::cout << "Directorio creado: " << path << std::endl;
	} else {
		std::cout << "El directorio ya existe o no pudo crearse." << std::endl;
	}

	return 0;
}
```

Explicación del código:
#include <filesystem>: Importamos la biblioteca filesystem, que nos permite trabajar con archivos y directorios.
fs::create_directory(path): La función create_directory() intenta crear el directorio en la ruta especificada. Devuelve true si el directorio se creó correctamente y false si ya existe o no se pudo crear.
Uso de namespace fs = std::filesystem;: Esto simplemente hace que sea más corto escribir fs:: en lugar de std::filesystem::.
b. Usando mkdir() en sistemas Linux/Unix
Si estás en un sistema basado en Linux/Unix y quieres usar una función más cercana al sistema, puedes usar mkdir() de POSIX. Este método requiere incluir <sys/stat.h> y es muy común en aplicaciones C/C++ de bajo nivel.

Ejemplo:

```
#include <iostream>
#include <sys/stat.h> // Necesario para mkdir
#include <sys/types.h>

int main() {
	const char* path = "mi_directorio";

	// Intentamos crear el directorio
```

```
    if (mkdir(path, 0777) == 0) { // 0777 son los permisos del directorio
        std::cout << "Directorio creado exitosamente: " << path << std::endl;
    } else {
        std::cerr << "Error al crear el directorio o ya existe." << std::endl;
    }

    return 0;
}
```

Explicación del código:
`#include <sys/stat.h>` y `#include <sys/types.h>`: Estas bibliotecas proporcionan la función `mkdir()` y los tipos necesarios.
`mkdir(path, 0777)`: Crea un directorio en la ruta `path` con permisos `0777` (lectura, escritura y ejecución para todos). Si el directorio se crea correctamente, `mkdir()` devuelve `0`, de lo contrario, devuelve un valor distinto de cero (indicando que ya existe o que hubo un error).
c. Usando `_mkdir()` en Windows
Si estás en Windows, debes usar la función `_mkdir()` del archivo `<direct.h>`. Aquí un ejemplo:

```
#include <iostream>
#include <direct.h> // Necesario para _mkdir en Windows

int main() {
    const char* path = "mi_directorio";

    // Intentamos crear el directorio
    if (_mkdir(path) == 0) {
        std::cout << "Directorio creado exitosamente en Windows: " << path << std::endl;
    } else {
        std::cerr << "Error al crear el directorio o ya existe." << std::endl;
    }

    return 0;
}
```

Explicación del código:
`#include <direct.h>`: La librería `direct.h` incluye la función `_mkdir()` para crear directorios en Windows.
`_mkdir(path)`: Crea un directorio en la ruta especificada.

3. ¿Dónde y cuándo usar mkdir en C++?
Aplicaciones de gestión de archivos: Si tu programa necesita gestionar el sistema de archivos, como un gestor de proyectos que organiza carpetas automáticamente, usar `mkdir` es esencial.

Sistemas embebidos o software sin interfaz gráfica: En software donde se crean estructuras de archivos automáticamente para almacenar registros, logs o archivos de configuración, crear directorios es fundamental.

Desarrollo de sistemas o utilidades: Cuando estás creando herramientas de sistema que necesitan manipular directorios, como copias de seguridad, gestores de archivos o administradores de tareas.

4. Manejo de errores
Crear directorios puede fallar por varias razones: permisos insuficientes, rutas inválidas o si el directorio ya existe. Es importante manejar estos errores adecuadamente.

Por ejemplo, en el uso de `std::filesystem::create_directory`, podrías envolver la función en un bloque `try-catch` para capturar excepciones:

```
#include <iostream>
#include <filesystem>

namespace fs = std::filesystem;

int main() {
    std::string path = "mi_directorio";

    try {
        if (fs::create_directory(path)) {
            std::cout << "Directorio creado: " << path << std::endl;
        } else {
            std::cout << "El directorio ya existe." << std::endl;
        }
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }

    return 0;}
```

Resumen:
Para sistemas modernos con C++17, std::filesystem es la mejor opción porque es portable entre plataformas.
En Linux/Unix, mkdir() es una buena opción si prefieres trabajar a bajo nivel o no tienes soporte para C++17.
En Windows, _mkdir() es la alternativa específica para crear directorios.