Try_cach

El manejo de excepciones en C++ se realiza utilizando las estructuras try, catch, y throw. Estas estructuras te permiten manejar errores de manera controlada en lugar de dejar que el programa termine abruptamente. Aquí te explico cómo funciona cada parte y cómo usarlas de manera efectiva.

Conceptos Básicos

try: Bloque de código donde se pueden lanzar excepciones.
throw: Utilizado para lanzar una excepción.
catch: Bloque de código que captura y maneja las excepciones lanzadas.

Sintaxis General

```
try {
     // Código que puede lanzar una excepción
} catch (const TipoDeExcepcion& e) {
     // Manejo de la excepción
}
```

try contiene el código que podría causar una excepción.
catch maneja la excepción si ocurre. Puede haber múltiples bloques catch para diferentes tipos de excepciones.

Ejemplo Básico

```
#include <iostream>
#include <stdexcept> // Para std::runtime_error

int division(int a, int b) {
     if (b == 0) {
          throw std::runtime_error("División por cero");
     }
     return a / b;
}

int main() {
     try {
          int resultado = division(10, 0);
          std::cout << "Resultado: " << resultado << std::endl;
     } catch (const std::runtime_error& e) {
          std::cerr << "Error: " << e.what() << std::endl;
     }

     return 0;
}
```

Explicación:
La función division lanza una excepción de tipo std::runtime_error si el divisor es cero.
El bloque try en main llama a division, y si ocurre una excepción, el bloque catch maneja el error imprimiendo un mensaje.

Múltiples Bloques catch
Puedes manejar diferentes tipos de excepciones con varios bloques catch.

```
#include <iostream>
#include <stdexcept>

void puedeFallir(int x) {
     if (x == 0) {
          throw std::runtime_error("Error de tiempo de ejecución");
     } else if (x == 1) {
          throw std::invalid_argument("Argumento inválido");
     }
}

int main() {
     try {
          puedeFallir(0);
     } catch (const std::invalid_argument& e) {
          std::cerr << "Argumento inválido: " << e.what() << std::endl;
     } catch (const std::runtime_error& e) {
          std::cerr << "Error de tiempo de ejecución: " << e.what() << std::endl;
```

```
    }

    return 0; }
```

Explicación:
Se manejan dos tipos de excepciones diferentes (std::invalid_argument y std::runtime_error) en bloques catch separados.

Excepciones y Punteros

Si lanzas y capturas excepciones usando punteros, asegúrate de usar referencias o punteros inteligentes para evitar fugas de memoria.

```
#include <iostream>
#include <memory>

void funcionQueLanza() {
    throw std::make_unique<std::runtime_error>("Error con puntero inteligente");
}

int main() {
    try {
        funcionQueLanza();
    } catch (const std::unique_ptr<std::runtime_error>& e) {
        std::cerr << "Capturado: " << e->what() << std::endl;
    }

    return 0;
}
```

Explicación:
Se usa std::unique_ptr para manejar la excepción, lo que ayuda a evitar fugas de memoria.

Re-throws

Puedes volver a lanzar la excepción capturada para manejarla en niveles superiores.

```
#include <iostream>
#include <stdexcept>

void funcion() {
    try {
        throw std::runtime_error("Error en función");
    } catch (const std::runtime_error& e) {
        std::cerr << "Capturado en función: " << e.what() << std::endl;
        throw; // Vuelve a lanzar la excepción
    }
}

int main() {
    try {
        funcion();
    } catch (const std::runtime_error& e) {
        std::cerr << "Capturado en main: " << e.what() << std::endl;
    }

    return 0;
}
```

Explicación:
throw; vuelve a lanzar la misma excepción después de manejarla en el bloque catch.

Captura de Excepciones por Valor o Referencia

Generalmente, se recomienda capturar excepciones por referencia constante para evitar copias innecesarias y para manejar el objeto de excepción correctamente.

```
try {
    // Código que lanza una excepción
} catch (const std::exception& e) {
    std::cerr << "Error: " << e.what() << std::endl;
}
```

Explicación:
const std::exception& e captura la excepción por referencia constante, evitando la copia del objeto de excepción.

Resumen

try: Contiene el código que puede lanzar excepciones.
throw: Lanza una excepción que puede ser capturada por un bloque catch.
catch: Maneja excepciones específicas o generales.
Múltiples catch: Permiten manejar diferentes tipos de excepciones.
Re-throws: Permiten volver a lanzar excepciones para manejarlas en niveles superiores.

Captura por referencia: Es la práctica recomendada para evitar copias innecesarias y manejar excepciones eficientemente.
El manejo adecuado de excepciones en C++ te permite escribir código más robusto y menos propenso a fallos inesperados.