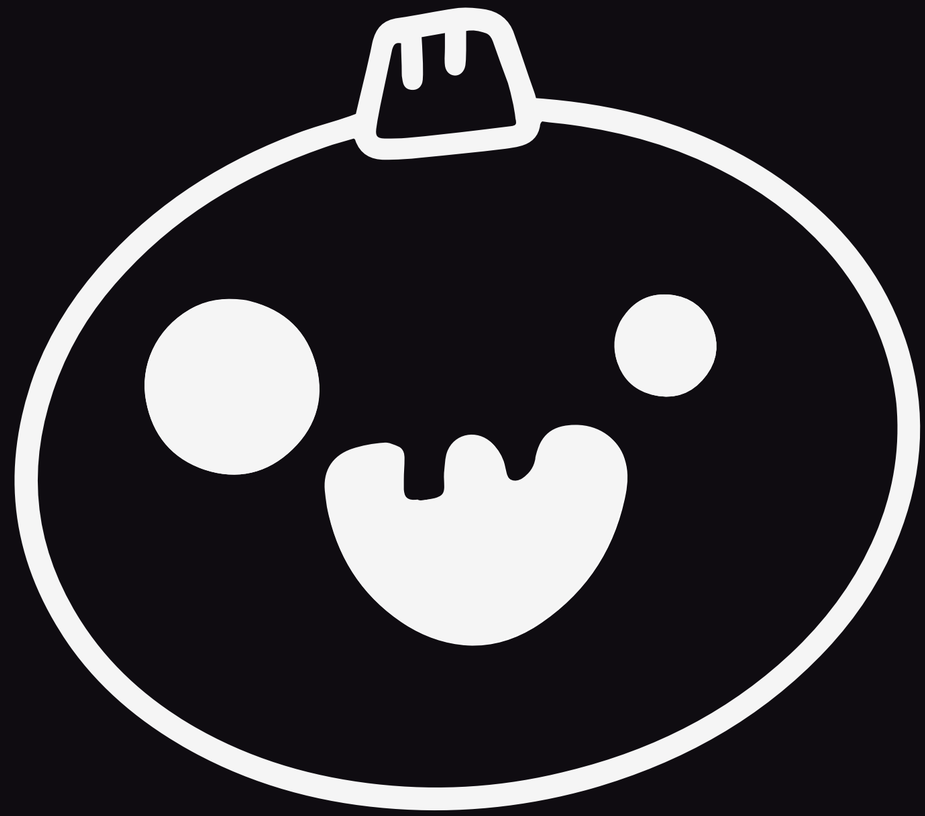

MANEJO DE
Excepciones

¿QUÉ SON LAS EXCEPCIONES?

Es un evento o situación anormal que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de instrucciones.



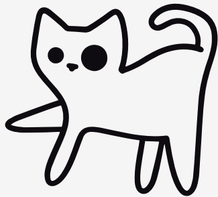
¿POR QUÉ EXISTEN LAS EXCEPCIONES?

- Mezcla lógica del programa con manejo de errores
- Fácil olvidar verificar códigos de error
- Difícil propagar errores a través de múltiples funciones
- Código difícil de leer y mantener

```
5   // Forma antigua: usando códigos de error
6   int dividir(int a, int b, double* resultado) {
7       if (b == 0) {
8           return -1;  // Código de error
9       }
10      *resultado = a / b;
11      return 0;  // Éxito
12  }
13
14  int main() {
15      double resultado;
16      int codigo = dividir(10, 0, &resultado);
```

CLASES DE EXCEPCIONES

Las excepciones en C++ son objetos que representan errores o situaciones anormales. Se pueden usar clases estándar o crear propias.



STD::RUNTIME_ERROR

Estas excepciones indican errores que solo pueden detectarse en tiempo de ejecución, no antes.
Por ejemplo, un fallo al leer un archivo, un desbordamiento, o una operación inválida por causas externas.

01

range_error
Cuando un cálculo produce un resultado fuera del rango permitido.

02

overflow_error
Cuando una operación produce un número demasiado grande.

03

underflow_error
Cuando una operación produce un número demasiado pequeño.

STD::LOGIC_ERROR

Estas excepciones indican errores de lógica en el código. Es decir, errores que deberían evitarse con una buena validación antes de ejecutar.
Por ejemplo, pasar argumentos inválidos o acceder a algo fuera de rango.

01

invalid_argument
Cuando una función recibe un argumento inválido.

02

domain_error
Cuando un valor está fuera del dominio válido matemático.

03

length_error
Cuando una operación excede una longitud máxima permitida.

04

out_of_range
Cuando se accede a un índice fuera de los límites

¿CÓMO USAR ESTAS CLASES?

TRY

Es un bloque protegido donde colocas el código que PODRÍA generar errores.

CATCH

Es el manejador de errores, donde decides qué hacer cuando algo falla.

THROW

Es la acción de "activar la alarma" cuando detectas que algo está mal.

¿CÓMO FUNCIONAN?

Cuando ocurre un throw dentro de un try:

01

El programa busca el catch que coincida con el tipo de excepción

02

Ejecuta el código dentro de ese catch

03

Después continúa con el programa normalmente (después del bloque try-catch)

IMPORTANTE

- Se puede tener múltiples catch para diferentes tipos de errores.
- El programa solo ejecuta un catch (el primero que coincida).
- Después del catch, el programa continúa (no se muere).

EJEMPLOS:

RESUMEN

Tipo	Ejemplo típico
LOGIC_ERROR	
`invalid_argument`	Edad negativa, temperatura imposible
`out_of_range`	Acceder índice 10 de array con 5 elementos
`domain_error`	Raíz cuadrada de número negativo
`length_error`	String demasiado largo
RUNTIME_ERROR	
`runtime_error`	Archivo no encontrado, conexión fallida
`overflow_error`	Número excede el máximo representable
`underflow_error`	Número menor al mínimo representable
`range_error`	Resultado fuera de rango válido

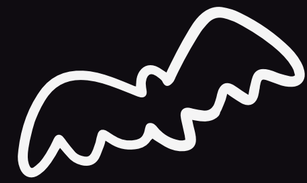
PRÁCTICA

Objetivo: Este programa tiene múltiples problemas que lo harán crashear. Su trabajo es:

1. Identificar dónde pueden ocurrir errores
2. Agregar bloques try-catch apropiados
3. Capturar las excepciones correctas
4. Hacer que el programa termine correctamente (sin crashear)

Reglas:

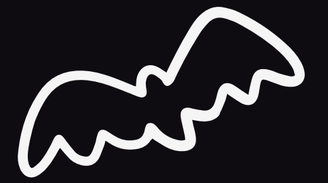
- NO puede cambiar la lógica del código (las operaciones deben intentarse)
- SOLO puede agregar try-catch y mensajes de error
- El programa debe continuar ejecutándose después de cada error
- Debe capturar el tipo correcto de excepción



PARA CUANDO ESTUDIEN

En C++, todas las excepciones estándar derivan de la clase base std::exception, que provee el método what() para obtener una descripción del error. A partir de esta clase se dividen dos grandes categorías: std::logic_error y std::runtime_error.

- Las excepciones de tipo std::logic_error representan errores de programación que podrían haberse evitado con una validación adecuada antes de ejecutar el programa. Dentro de esta categoría se incluyen: std::invalid_argument, que se lanza cuando una función recibe un argumento inválido; std::domain_error, usada cuando un valor está fuera del dominio matemático permitido; std::length_error, cuando una operación excede la longitud máxima de un contenedor o cadena; y std::out_of_range, cuando se accede a una posición inexistente en un arreglo o string.
- Las excepciones de tipo std::runtime_error se refieren a errores que solo pueden detectarse durante la ejecución, como fallos en operaciones externas o numéricas. Sus subclases incluyen: std::range_error, usada cuando un cálculo produce un resultado fuera de un rango válido; std::overflow_error, cuando ocurre un desbordamiento numérico; y std::underflow_error, cuando un resultado es demasiado pequeño para representarse con precisión.



¿QUÉ PASA CUANDO TENEMOS ERRORES ESPECÍFICOS?

CREAR NUESTRAS PROPIAS EXCEPCIONES

- Excepciones específicas (lógica de negocio)
- Información adicional además del mensaje
- Jerarquía de errores organizada
- Mayor claridad en tu código

PASOS

01 Crear una clase de excepción personalizada

02 Lanzar (throw) la excepción

03 Capturar la excepción

01 Crear una clase de excepción personalizada

Todas las excepciones deberían heredar de la clase base **std::exception**, para poder usar el método what() y mantener compatibilidad con el sistema estándar.

```cpp
#include <iostream>
#include <exception>
#include <string>
using namespace std;
class SaldoInsuficienteException : public exception {
private:
    string mensaje;

public:
    // Constructor
    SaldoInsuficienteException(string msg) {
        mensaje = msg;
    }
    // Método what() que devuelve el mensaje de error
    const char* what() const noexcept override {
        return mensaje.c_str();
    }
};
```

La clase **SaldoInsuficienteException** solo define el tipo de error (es decir, qué representa y qué mensaje tiene)

02 Lanzar (throw) la excepción

Se puede lanzar una instancia de la clase cuando ocurre una condición de error.

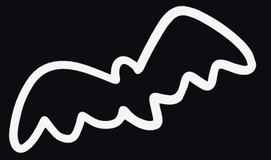
```
20  void retirarDinero(double saldo, double monto) {
21      if (monto > saldo)
22          throw SaldoInsuficienteException("Error: saldo insuficiente para realizar la transaccion.");
23      cout << "Retiro exitoso. Saldo restante: " << saldo - monto << endl;
24  }
```

03 Capturar la excepción

El bloque try-catch nos permite manejar el error adecuadamente:

```
27    int main() {
28        try {
29            double saldo = 500.0;
30            double monto = 800.0;
31            retirarDinero(saldo, monto);
32        }
33        catch (const SaldoInsuficienteException& e) {
34            cout << e.what() << endl;
35        }
36
37        cout << "Programa finalizado correctamente." << endl;
38    }
```

“LA CLASE DEFINE QUÉ TIPO DE ERROR EXISTE,
LA FUNCIÓN DECIDE CUÁNDO LANZARLO,
Y EL BLOQUE CATCH DECIDE CÓMO MANEJARLO”



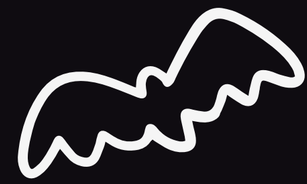
PRÁCTICA

Objetivo: Este programa tiene múltiples problemas que lo harán crashear. Su trabajo es:

1. Crear los .hpp de las excepciones, cada uno así.
2. Modificar main.cpp para agregar try-catch en cada función.

Reglas:

- NO puede cambiar la lógica del código (las operaciones deben intentarse)
- SOLO puede agregar try-catch y mensajes de error
- El programa debe continuar ejecutándose después de cada error
- Debe capturar el tipo correcto de excepción



Muchas gracias