Universidad de Guadalajara.

Centro Universitario de Ciencias Exactas e Ingenierías.

Ingeniería en computación.

Sección: D05.

Computación Tolerante a Fallas.

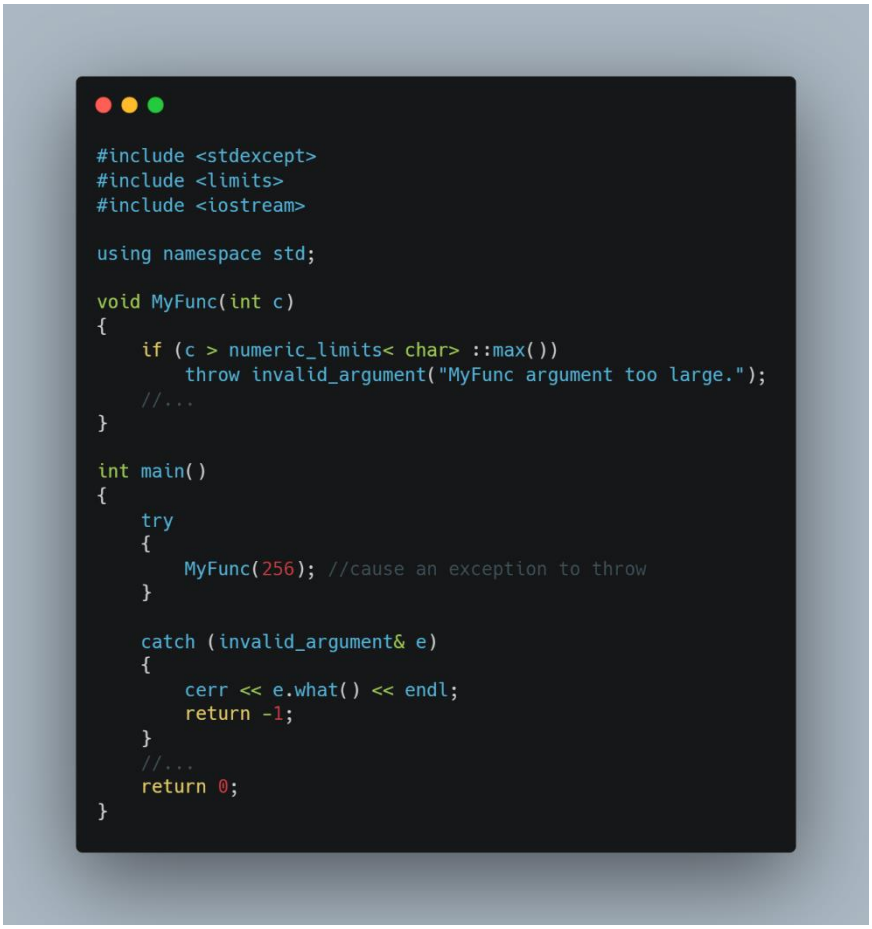
(Par. 1) Otras herramientas para el manejar errores

Prof. MICHEL EMANUEL LOPEZ FRANCO.

Montoya Vargas Roberto.

Otras herramientas para el manejo de errores.

Los errores en un programa se pueden dividir en dos categorías principales: errores lógicos, que surgen debido a errores en el código, como el "índice fuera del rango", y errores en tiempo de ejecución, que están fuera del control del programador, como el "servicio de red no disponible". En la programación de estilo C y COM, se manejan informes de errores devolviendo códigos de error o estados en las funciones, o estableciendo variables globales para que el autor de la llamada pueda verificar si se han producido errores. Por ejemplo, COM utiliza el valor devuelto HRESULT para comunicar errores, y la API Win32 ofrece la función GetLastError para obtener el último error. Depende del autor de la llamada reconocer y responder a estos códigos de error. Si no se manejan adecuadamente, el programa puede bloquearse sin previo aviso o continuar ejecutándose con datos incorrectos, lo que resulta en resultados incorrectos.



```cpp
#include <stdexcept>
#include <limits>
#include <iostream>

using namespace std;

void MyFunc(int c)
{
    if (c > numeric_limits< char> ::max())
        throw invalid_argument("MyFunc argument too large.");
    //...
}

int main()
{
    try
    {
        MyFunc(256); //cause an exception to throw
    }

    catch (invalid_argument& e)
    {
        cerr << e.what() << endl;
        return -1;
    }
    //...
    return 0;
}
```

En C++, las excepciones funcionan de manera similar a lenguajes como C# y Java. En un bloque try, si ocurre una excepción, se capturará en el primer bloque catch cuyo tipo coincida con la excepción. Esto lleva la ejecución desde la instrucción throw hasta el catch. Si no hay ningún bloque catch correspondiente, se invoca std::terminate y el programa se cierra. En C++, las excepciones pueden ser de cualquier tipo, pero es recomendable usar tipos que deriven de std::exception. Por ejemplo, invalid_argument es un tipo de excepción definido en la biblioteca estándar.

C++ no incluye un bloque finally para garantizar la liberación de recursos en caso de excepción. En su lugar, se utiliza el concepto de Inicialización Automática de Recursos (RAII), que emplea punteros inteligentes para administrar la limpieza de recursos. Estos mecanismos proporcionan la funcionalidad necesaria para manejar la liberación de recursos. Para obtener más información sobre seguridad en el diseño de excepciones, se puede consultar la documentación. Para detalles sobre el mecanismo de desenredo de la pila en C++, se puede consultar la sección de excepciones y desenredo de la pila.

Excepciones y rendimiento

El mecanismo de excepciones tiene un impacto mínimo en el rendimiento si no se produce ninguna excepción. En caso de que ocurra una excepción, el costo asociado con la navegación y el desenredo de la pila es aproximadamente equivalente al de una llamada a función. Se necesitan estructuras de datos adicionales para rastrear la pila de llamadas cuando se utiliza un bloque try, y se requieren instrucciones extra para desenredar la pila si se lanza una excepción. Sin embargo, en la mayoría de los casos, este costo en el rendimiento y la utilización de memoria es despreciable. Los efectos negativos en el rendimiento se notarían principalmente en sistemas con limitaciones de memoria.

Excepciones frente a aserciones

Las excepciones y las aserciones son dos métodos diferentes para identificar errores durante la ejecución de un programa. Las instrucciones assert se utilizan para verificar condiciones que nunca deberían ser verdaderas en un programa correctamente escrito

durante el desarrollo. Detectar estas condiciones es esencialmente encontrar errores en el código. Controlar estos errores con excepciones no tiene sentido, ya que no se espera que el programa se recupere durante la ejecución. Un assert detiene la ejecución para permitir la inspección en el depurador, mientras que una excepción permite la continuación desde un bloque catch.

Por otro lado, las excepciones se usan para manejar condiciones de error que podrían surgir en tiempo de ejecución, incluso si el código es correcto. En este caso, se trata de situaciones en las que el programa debe ser capaz de manejar errores y recuperarse.

Bibliografía:

Información extraída de:

https://learn.microsoft.com/es-es/cpp/cpp/errors-and-exception-handling-modern-cpp?view=msvc-170