**basic\_try\_catch.cpp**

#include <iostream>

#include <stdexcept>

#include <cmath> // for sqrt

using namespace std;

// Function that throws if input is negative

double safeSqrt(double x) {

    if (x < 0) {

        throw runtime\_error("Cannot compute square root of a negative number!");

    }

    return sqrt(x);

}

int main() {

    double num;

    cout << "Enter a number to find its square root: ";

    cin >> num;

    try {

        double result = safeSqrt(num);

        cout << "Square root: " << result << endl;

    }

    catch (const runtime\_error &e) {

        cout << "Exception caught: " << e.what() << endl;

    }

    cout << "Program continues..." << endl;

    return 0;

}

// g++ -std=c++2b basic\_try\_catch.cpp -o basicTC

**basic\_try\_catch\_2.cpp**

#include <iostream>

#include <stdexcept>

using namespace std;

// A function that may thro an exception

double divide(int a,int b);

int main(){

    int x,y;

    cout << "Enter two numners  (Enter two exclamation marks (!) to quit: )>";

    cin >> x >> y;

    try{

        double result = divide(x,y); // Call the function

        cout << "Result: " << result << endl;

    }

    catch(const exception &e){

        cout << "Exception caught: " << e.what() << endl;

        //cerr << "Exception caught: " << e.what() << endl;

    }

    cout << "Program continues..." << endl;

    return 0;

}

double divide(int a,int b){

    if(b==0){

        throw logic\_error("Denominator should not be zero.");

    }

    return static\_cast<double>(a)/b;

}

**basic\_try\_catch\_3.cpp**

#include <iostream>

#include <stdexcept>

#include <string>

using namespace std;

// A function that may thro an exception

double divide(int a,int b);

int main(){

    int x,y;

    bool cont = true;

    cout << "Enter two numners  (Enter two exclamation marks (!) to quit: )>";

    while(cont){

        try{

            cin >> x >> y;

            if (cin.fail()){

                string maybeQuit;

                cin.clear(); // clear error flags

                cin >> maybeQuit; // read the remaining invalid part

                if (maybeQuit == "!!") {

                    cout << "Quitting...\n";

                    cont = false;

                    break;

                }

                cout << "\nInvalid input. Enter two numners  (Enter two exclamation marks (!) to quit: )>";

                cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // discard rest of line

                continue;

                throw invalid\_argument("Invalid argument passed.");

            }

        }catch(const exception &e){

            cout << "Exception caught: " << e.what() << endl;

        }

        try{

            double result = divide(x,y); // Call the function

            cout << "Result: " << result << endl;

        }

        catch(const exception &e){

            cout << "Exception caught: " << e.what() << endl;

            //cerr << "Exception caught: " << e.what() << endl;

        }

        cout << "Program continues..." << endl;

        cout << "Enter two numners  (Enter two exclamation marks (!) to quit: )>";

    }

    return 0;

}

double divide(int a,int b){

    if(b==0){

        throw logic\_error("Denominator should not be zero.");

    }

    return static\_cast<double>(a)/b;

}

// g++ -std=c++2b basic\_try\_catch\_3.cpp -o basicTC3

**activity\_custom\_exception.cpp**

#include <iostream>

#include <string>

#include <exception>

using namespace std;

// TODO: Define class NegativeAgeException here

class NegativeAgeException:public exception{

    private:

        string message;

    public:

        NegativeAgeException(const string &msg):message(msg){}

        virtual const char\* what() const \_NOEXCEPT override { //virtual const char\* what() const override noexcept {

            return message.c\_str();

        }

};

void checkAge(int age) {

    if (age < 0) {

        throw NegativeAgeException("Cannot backwards in time from when you are born!");

    }

    cout << "You entered a valid age: " << age << endl;

}

int main() {

    int inputAge;

    cout << "Enter your age: ";

    cin >> inputAge;

    try {

        checkAge(inputAge);

    }

    catch (const NegativeAgeException &e) {

        cerr << "Custom Exception caught: " << e.what() << endl;

    }

    catch (const exception& e) {

        cerr << "Standard Exception: " << e.what() << endl;

    }

    return 0;

}

// g++ -std=c++2b activity\_custom\_exception.cpp -o ace

**bank\_account.cpp**

#include <iostream>                             /\*(1)\*/

#include <stdexcept>                            /\*(2)\*/

#include <string>                               /\*(3)\*/

// Custom exception class                       /\*(4)\*/

class InsufficientFundsException : public std::exception { /\*(5)\*/

private:                                        /\*(6)\*/

    std::string message;                        /\*(7)\*/

public:                                         /\*(8)\*/

    InsufficientFundsException(const std::string& msg)     /\*(9)\*/

        : message(msg) {}                       /\*(10)\*/

    // Override what() to return the custom message         /\*(11)\*/

    virtual const char\* what() const noexcept override {   /\*(12)\*/

        return message.c\_str();                 /\*(13)\*/

    }                                           /\*(14)\*/

};                                              /\*(15)\*/

class BankAccount {                             /\*(16)\*/

private:                                        /\*(17)\*/

    double balance;                             /\*(18)\*/

public:                                         /\*(19)\*/

    BankAccount(double initialBalance)          /\*(20)\*/

        : balance(initialBalance) {}            /\*(21)\*/

    void withdraw(double amount) {              /\*(22)\*/

        if (amount > balance) {                 /\*(23)\*/

            throw InsufficientFundsException(   /\*(24)\*/

                "Withdrawal amount exceeds balance."       /\*(25)\*/

            );                                  /\*(26)\*/

        }                                       /\*(27)\*/

        balance -= amount;                      /\*(28)\*/

        std::cout << "Withdrawal successful. New balance: $"  /\*(29)\*/

                  << balance << std::endl;      /\*(30)\*/

    }                                           /\*(31)\*/

};

                                         /\*(32)\*/

int main() {

    BankAccount account(100.00); // Initialize account with $100

    try {

        std::cout << "Attempting to withdraw $150..." << std::endl;

        account.withdraw(150.00); // Attempt to withdraw $150

    } catch (const InsufficientFundsException& e) {

        std::cerr << "An error occurred: " << e.what() << std::endl;

    }

    return 0;

}

// g++ -std=c++2b bank\_account.cpp -o bankAccount

**handling\_multipleException2.cpp**

// handling\_multipleException2.cpp

#include <iostream>

#include <stdexcept>

#include <string>

using namespace std; // <- Added for simplicity

void processInput(const string &input){

    int number = 0;

    try {

        number = stoi(input);

        if (number < 0 || number > 100){

            throw out\_of\_range("Number is out of acceptable range (0-100)");

        }

        cout << "Processed number: " << number << endl;

    }

    catch (const invalid\_argument &e){

        cout << "invalid\_argument";

        cerr << "Exception: Invalid argument. Please enter a valid number." << endl;

    }

    catch (const out\_of\_range &e){

        cout << "out\_of\_range";

        cerr << "Exception: " << e.what() << endl;

    }

    catch (const exception &e){

        cout << "exception";

        cerr << "Unexpected error: " << e.what() << endl;

    }

}

int main(){

    string userInput = "105"; // Can change to "abc", "50", etc. to test different cases

    cout << "Enter a number>";

    cin >> userInput;

    processInput(userInput);

    return 0;

}

// g++ -std=c++2b handling\_multipleException2.cpp -o mException2

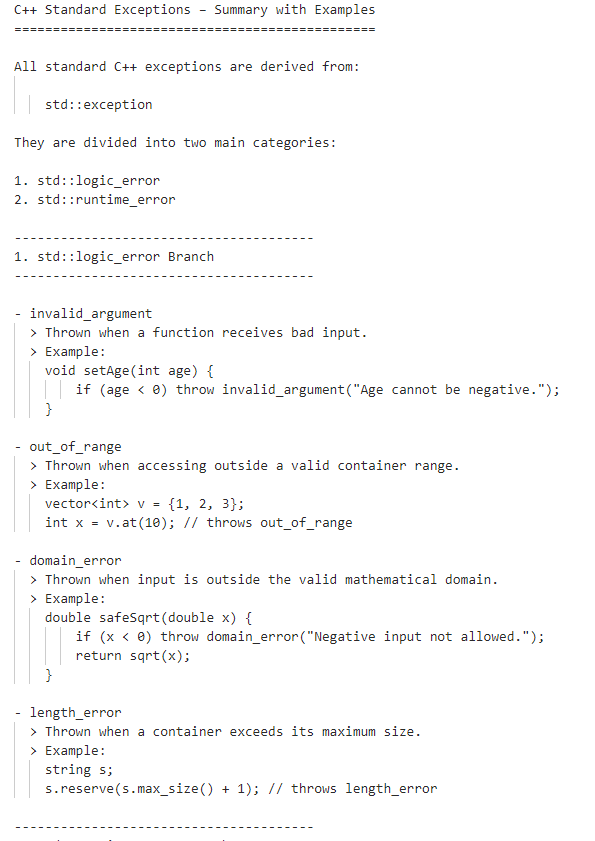
// "abc"

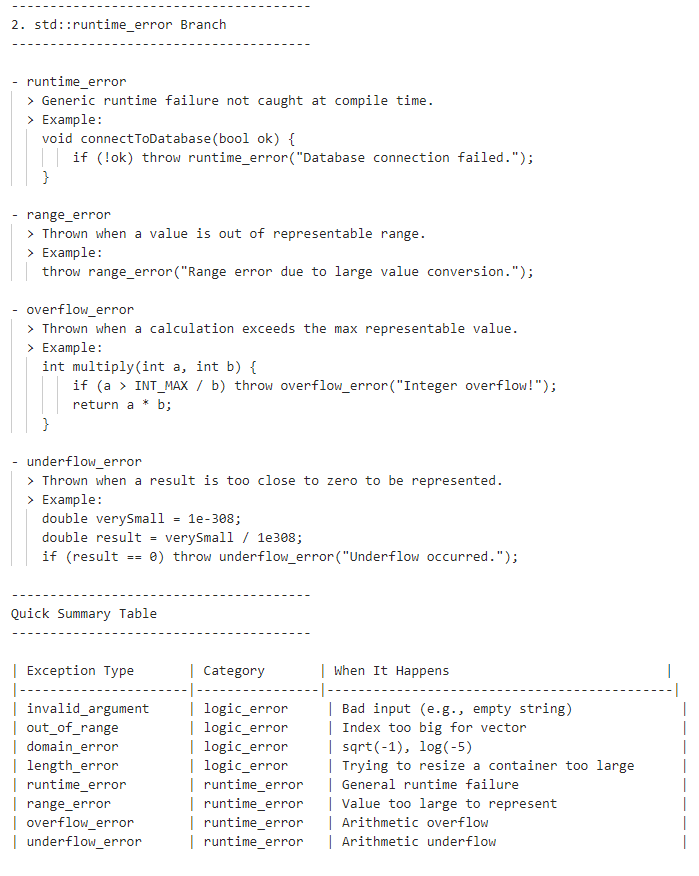
// "3.14"

// "!"

// "ten"

// "" (empty string)

****

****