# LAB QUESTIONS

Q1. Write a C++ program to handle divide-by-zero exception using try-catch block. Input two numbers. If denominator is zero, throw and catch an exception.

#include<iostream>

using namespace std;

int main(){

double numerator,denominator,result;

cout<<"enter numerator and denominator"<<endl;

cin>>numerator>>denominator;

try{

if(denominator==0){

throw("division by zero not possible");

}

result= numerator/denominator;

cout<<"result= "<<result<<endl;

}

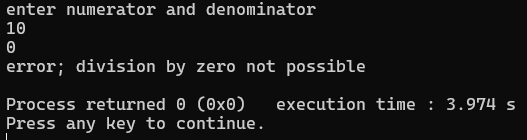
catch(const char\* msg){

cout<<"error; "<<msg<<endl;

}

return 0;

}



QN2. Write a C++ program to demonstrate multiple catch blocks handling different data types. Throw and handle int, char, and string type exceptions in separate catch blocks.

#include<iostream>

using namespace std;

int main(){

int choice;

cout<<"enter 1 for int, 2 for char, and 3 for string exception"<<endl;

cin>>choice;

try{

if(choice==1){

throw 36; }

else if(choice== 2){

throw 'S';

}

else if(choice==3){

throw string("string exception");}

else{

cout<<"no exceptiont thrown"<<endl;

}}

catch(int e){

cout<<"caught integer exception "<<e<<endl;}

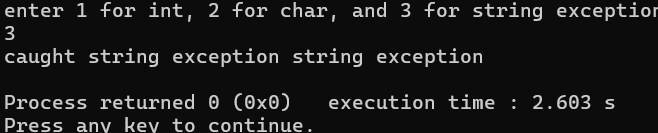
catch (char e){

cout<<"caught character exception "<<e<<endl;}

catch (string e){

cout<<"caught string exception "<<e<<endl;}

return 0;



Q3. Write a program using catch-all handler (catch(...)) to handle any kind of exception. Illustrate a case where an unexpected data type is thrown and caught generically.

#include <iostream>

using namespace std;

int main() {

int choice;

cout << "Enter a 1 for double, 2 for bool, and any other for any exception): "; cin >> choice;

try {

if (choice == 1) {

throw 3.14;

}

else if (choice == 2) {

throw true;

}

else {

throw nullptr;

}

}

catch (double e) {

cout << "C double exception: " << e << endl;

}

catch (bool e) {

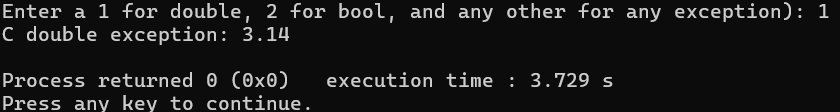
cout << "boolean exception: " << (e ? "true" : "false") << endl; }

catch (...) {

cout << " unknown exception " << endl; }

return 0;

}



Q4. Write a C++ program that rethrows an exception after catching it once.Use a nested try-catch where the inner catch block rethrows the exception to be handled by the outer block

#include<iostream>

using namespace std;

int main(){

int value;

cout<<"enter number for value and 0 for exception: "<<endl;

cin>>value;

try{

try{

if(value==0){

throw("zero detected");

}

cout<<"no exception found, the value is: "<<value<<endl;

}

catch(const char\* msg){

cout<<"inner catched"<<msg<<endl;

throw;

}

}

catch(const char\* msg){

cout<<"another catched"<<msg<<endl;

}

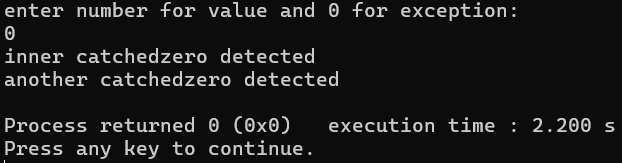
catch (...) {

cout <<"Unknown exception caught" << endl;

}

return 0;

}



Q5. Write a program to demonstrate throwing and catching a user-defined exception class with message argument.Define a custom class MyException and pass an error message to its constructor

#include<iostream>

#include<string>

using namespace std;

class myclass{

private:

string message;

public:

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

string getMessage() const { return message; }

};

int main() {

int value;

cout << "Enter a number and negative to throw exception:";

cin >> value;

try {

if (value < 0) {

throw myclass("Negative value is not permitted!");

}

cout << "Valid value entered: " << value << endl;

}

catch (const myclass& e) {

cout << "Caught myclass: " << e.getMessage() << endl;

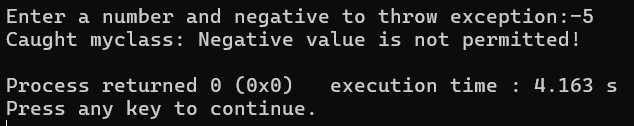
}

catch (...) {

cout << "Caught unknown exception" << endl;

}

return 0;

}

Q6. Write a program to use the STL algorithm functions: sort(), reverse(), find(), and count() on a vector.

# #include<iostream>

# #include <string>

# using namespace std;

# void customTerminate() {

# cout << "Custom terminate handler called: Uncaught exception detected!" << endl; exit(1);

# }

# int main() {

# set\_terminate(customTerminate);

# int value;

# cout << "Enter a number (non-zero to throw uncaught exception): ";

# cin >> value;

# try {

# if (value != 0) {

# throw string("Uncaught string exception thrown");

# }

# cout << "No exception, value is: " << value << endl;

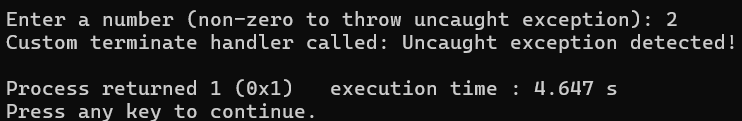
# }

# catch (int e) {

# cout << "This catch block won't be used: " << e << endl;

# }

# return 0;



Q7. /\*Write a C++ program with a function that violates its exception specification and handles it using set\_unexpected().Use throw(double) in the function declaration but throw an in.\*/

#include <iostream>

#include <exception>

using namespace std;

void customUnexpected() {

cout << "Custom unexpected handler called" << endl; exit(1);

}

void riskyFunction(int value) throw(double) {

if (value < 0) {

throw 38;

}

cout << "Function executed successfully with value: " << value << endl;

}

int main() {

set\_unexpected(customUnexpected);

int value;

cout << "Enter a number (negative to throw int exception): ";

cin >> value;

try {

riskyFunction(value);

}

catch (double e) {

cout << "Caught double exception: " << e << endl;

}

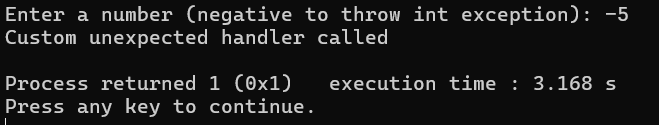
catch (...) {

cout << "Caught unexpected exception in main" << endl;

}

return 0;

}



Q8. /\* Write a program to show exception handling inside a class constructor and destructor. Handle constructor exceptions properly; avoid throwing from destructor\*/

#include<iostream>

using namespace std;

using namespace std;

class testclass {

private:

int value;

public:

testclass (int val){

try{

if(val<0){

throw string("negative value not allowed");

}

value=val;

cout<<"constructor. object created with value "<<value<<endl;

}

catch (const string& e) {

cout << "Constructor exception: " << e << endl;

throw;

}

}

~testclass() {

cout << "Destructor. Cleaning up object with value " << value << endl;

}

void display() const {

cout << "Current value: " << value << endl;

}

};

int main() {

try {

int input;

cout << "Enter a number for object creation: ";

cin >> input;

testclass obj(input);

obj.display();

}

catch (const string& e) {

cout << "Main caught exception: " << e << endl;

}

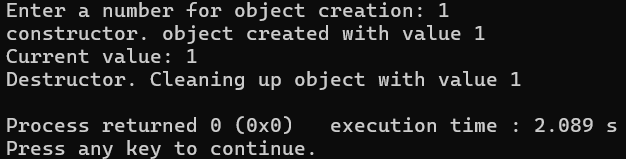
catch (...) {

cout << "Main caught unknown exception" << endl;

}

return 0;

}



# DISCUSSIONS

# In this lab session, we studied exception handling in C++, which allows programs to detect and manage runtime errors gracefully. We learned how to use try, catch, and throw to handle different types of exceptions like divide by zero errors, invalid inputs, and user-defined exceptions. The lab also covered topics like multiple catch blocks, catch-all handlers catch(...), nested try-catch with rethrowing, and the use of set\_terminate() and set\_unexpected() for handling uncaught or unexpected exceptions. These exercises demonstrated how exception handling improves program robustness, reliability, and maintainability.

# CONCLUSIONS

In conclusion, this lab session on exception handling helped us understand how the exception handling works. It separates error-handling logic from normal code flow, making programs easier to debug and maintain. By mastering exception handling techniques, programmers can ensure that unexpected runtime errors do not cause immediate program termination and can provide meaningful feedback to users.