**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Spring 23, Lab Manual - 13**

|  |  |
| --- | --- |
| **Course Code: CL-1004** | **Course : Object Oriented Programming Lab** |
| **Instructor(s) :** | **Nida Munawar** |

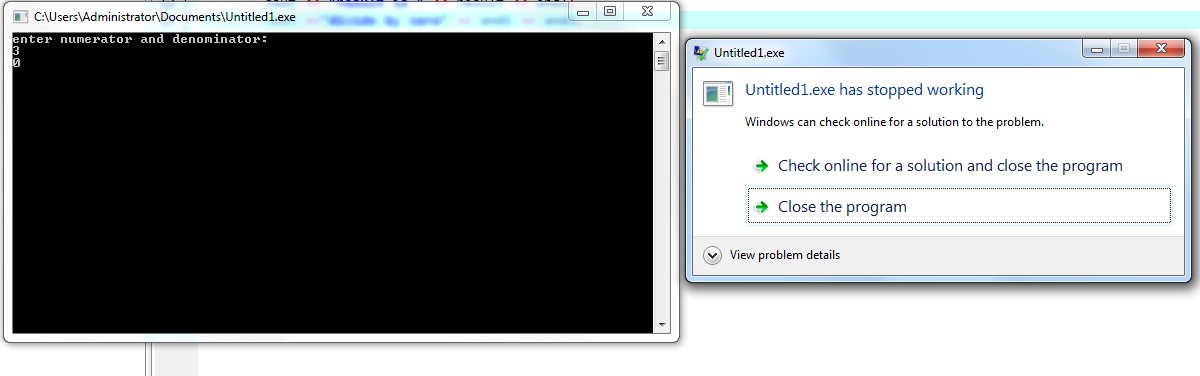
# Outline:Exception handling

**Lab # 14**

Output?

* #include <iostream>
* using namespace std;
* double zeroDivision(int x, int y) {
* return (x / y);}
* int main() {
* int numerator;
* int denominator;
* double result;
* cout << "enter numerator and denominator: " << endl;
* cin>>numerator>>denominator; // 3 and 0
* result = zeroDivision(numerator, denominator);
* cout << "result is " << result << endl;
* cout <<"divide by zero" << endl << endl;
* return 0;}

Output?



# Exceptions

* Exception handling in C++ provides you with a way of handling unexpected circumstances like runtime errors. So whenever an unexpected circumstance occurs, the program control is transferred to special functions known as handlers.
* An *exception is an abnormal* condition that arises in a code sequence at run time • In other words, an exception is a run-time error

# Handlers

* Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords:
* **try,**
* **catch,** and
* **throw**.

# Handlers

* **throw**- when a program encounters a problem, it throws an exception. The throw keyword helps the program perform the throw.
* **catch**- a program uses an exception handler to catch an exception. It is added to the section of a program where you need to handle the problem. It's done using the catch keyword.
* **try**- the try block identifies the code block for which certain exceptions will be activated. It should be followed by one/more catch blocks.

# Throwing Exceptions

* Exceptions can be thrown anywhere within a code block using **throw** statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.
* double division(int a, int b) {
* if( b == 0 ) {
* throw "Division by zero condition!"; //throw b;
* }
* return (a/b);
* }

# try

* A block of code which may cause an exception is typically placed inside the try block. It’s followed by one or more catch blocks. If an exception occurs, it is thrown from the try block.
* try { • // protected code • }

# Catching Exceptions

* The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.
* try { • // protected code • } catch( ExceptionName exception1 ) { • // code to handle ExceptionName exception • }

# Catching Exceptions

* The ExceptionName is the name of the exception to be caught.
* The exception1 are your defined names for referring to the exceptions.

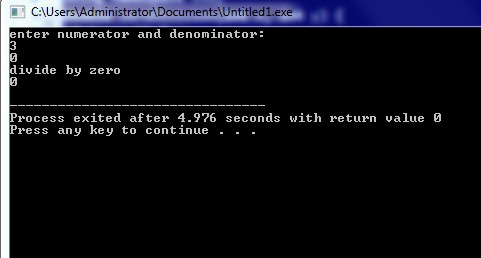
# Example

* #include <iostream>
* using namespace std;
* double zeroDivision(int x, int y) {
* if (y == 0) {
* throw y;}
* return (x / y);}
* int main() {
* int numerator;
* int denominator;
* double result;
* cout << "enter numerator and denominator: " << endl;
* cin>>numerator>>denominator;

•

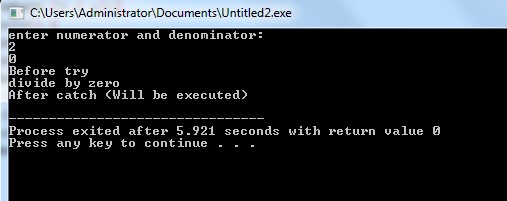
* try {
* result = zeroDivision(numerator, denominator);
* cout << "result is " << result << endl; }
* catch (int ex) {
* cout <<"divide by zero" << endl << ex << endl; }
* return 0;}

# Example



* #include <iostream>
* #include <exception>
* using namespace std;
* double zeroDivision(int x, int y) {
* if (y == 0) {
* throw y;
* cout << "After throw (Never executed) \n";}
* return (x / y);}
* int main() {
* int numerator;
* int denominator;
* double result;
* cout << "enter numerator and denominator: " << endl;
* cin>>numerator>>denominator;
* cout << "Before try \n";

## • try { • result = zeroDivision(numerator, denominator); • cout << "result is " << result << endl; } • catch (int e) { • cout <<"divide by zero" << endl; } • cout << "After catch (Will be executed) \n"; • return 0;}



# Without catch

* #include <iostream>
* #include <exception>
* using namespace std;
* double zeroDivision(int x, int y) {
* if (y == 0) {
* throw y;}
* return (x / y);}
* int main() {
* int numerator;
* int denominator;
* double result;
* cout << "enter numerator and denominator: " << endl;
* cin>>numerator>>denominator;

•

* try {
* result = zeroDivision(numerator, denominator);
* cout << "result is " << result << endl; }

•

* return 0;}

# Without catch



* In some cases, more than one exception could be raised by a single piece of code
* To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception

## • After one catch statement executes, the others are bypassed

• You can list down multiple **catch** statements to catch different type of exceptions in case your **try** block raises more than one exception in different situations.

## • try { // protected code } • catch( ExceptionName e1 ) • { // catch block } • catch( ExceptionName e2 ) { // catch block } • catch( ExceptionName eN ) { // catch block }

* #include <iostream>
* #include <exception>
* using namespace std;
* double zeroDivision(int x, int y) {
* if (y == 0) {
* throw y;}
* return (x / y);}
* int main() {
* int numerator;
* int denominator;
* double result;
* cout << "enter numerator and denominator: " << endl;
* cin>>numerator>>denominator;

•

* try {
* result = zeroDivision(numerator, denominator);
* cout << "result is " << result << endl; }
* catch (int e) { • cout <<"divide by zero" << endl << e << endl; } • catch (int e) { • cout <<"another exception" << endl << e << endl; } • catch (int e) { • cout <<"another exception" << endl << e << endl; } • return 0;}

# catch all block

• If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the exception declaration as follows − • try { // protected code } • catch(...) • { // code to handle any exception }

# catch all block

* #include <iostream>
* using namespace std;
* double zeroDivision(int x, int y) {
* if (y == 0) {
* throw y;}
* return (x / y);}
* int main() {
* int numerator;
* int denominator;
* double result;
* cout << "enter numerator and denominator: " << endl;
* cin>>numerator>>denominator;

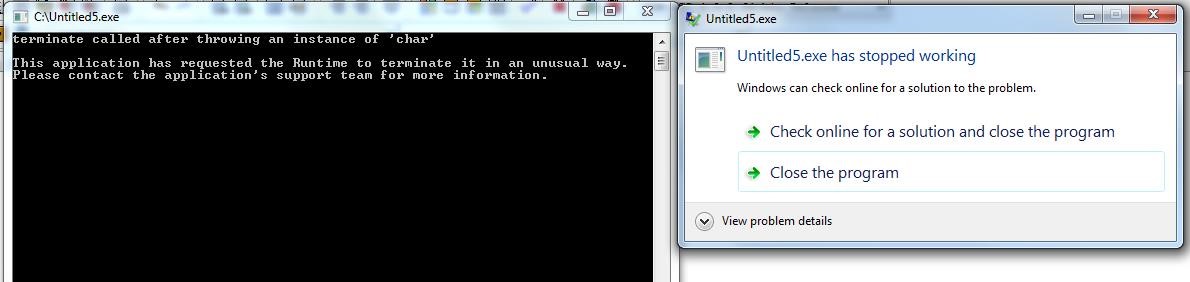
•

* try {
* result = zeroDivision(numerator, denominator);
* cout << "result is " << result << endl; }
* catch(...){
* cout <<"divide by zero" << endl << endl; }
* return 0;}
* If an exception is thrown and not caught anywhere, the program terminates abnormally.
* #include <iostream>
* using namespace std;

•

* int main(){
* try {
* throw 'a'; }
* catch (int x) {
* cout << "Caught "; }
* return 0;}

## • Implicit type conversion doesn’t happen for primitive types. For example, in the following program ‘a’ is not implicitly converted to int



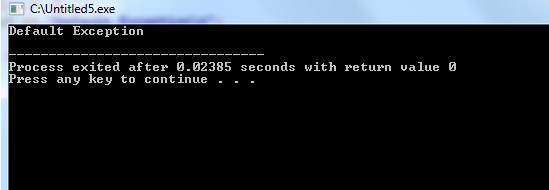
# Correct version

* #include <iostream>
* using namespace std;

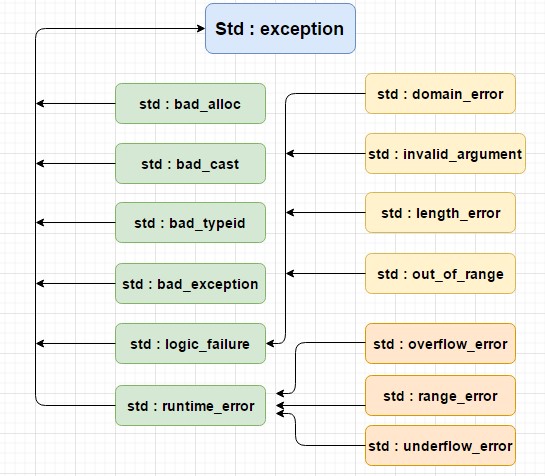
•

* int main()
* {
* try {
* throw 'a';
* }
* catch (int x) {
* cout << "Caught " << x;
* }
* catch (...) {
* cout << "Default Exception\n";
* }
* return 0;
* }

# output



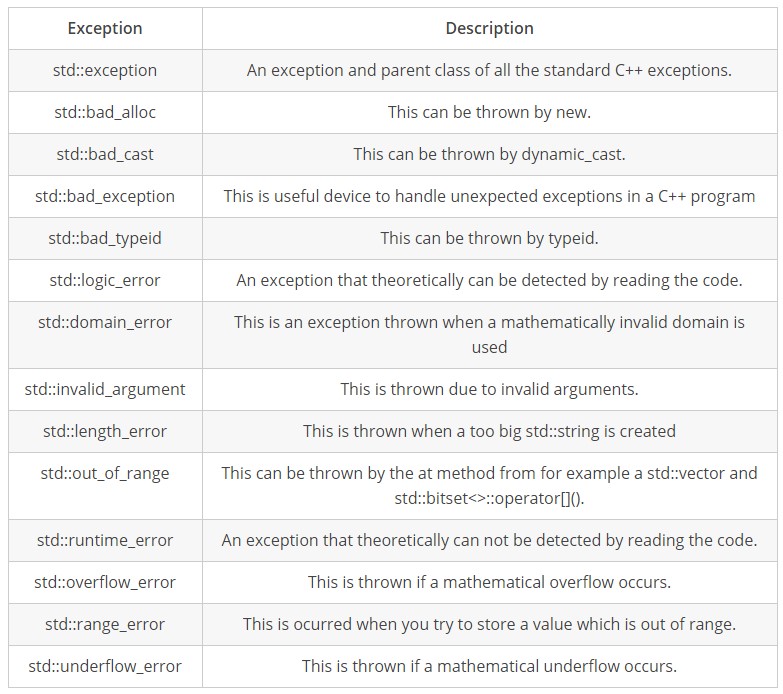
# C++ Standard Exceptions



# C++ Standard Exceptions

• C++ provides a list of standard exceptions defined in <exception> which we can use in our programs

# C++ Standard Exceptions



# User-Defined Exceptions

* The C++ std::exception class allows us to define objects that can be thrown as exceptions. This class has been defined in the <exception> header. The class provides us with a virtual member function named what.
* This function returns a null-terminated character sequence of type char \*. We can overwrite it in derived classes to have an exception description.

# User-Defined Exceptions

* #include <iostream>
* #include <exception> • using namespace std; class newException : public exception{
* virtual const char\* what() const throw()//We can overwrite the what() function of the exception header file to define our exceptions.
* {
* return "newException occurred"; }
* } newex; int main() { try {
* throw newex; }
* catch (exception& ex) {
* cout << ex.what() << '\n'; }
* return 0; }