

CC-211L

Object Oriented Programming

Laboratory 13

Exception Handling

Version: 1.0.0

Release Date: 27-04-2023

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
 - Exception
 - Exception Handling
- Activities
 - Pre-Lab Activity
 - Compile Time Errors
 - Run-Time Errors
 - Task 01
 - In-Lab Activity
 - Flow of Control
 - C++ try
 - C++ catch
 - C++ throw
 - Rethrowing an Exception
 - Constructor Destructor and Exception Handling
 - Task 01
 - Task 02
 - Task 03
 - Task 04
 - Post-Lab Activity
 - Task 01
 - Task 02
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

Learning Objectives:

- Exception
- Flow of Control
- Rethrowing an Exception
- Constructor, Destructor and Exception handling

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Lab Instructor	Azka Saddiqa	azka.saddiqa@pucit.edu.pk
Teacher Assistants	Saad Rahman	bsef19m021@pucit.edu.pk
	Zain Ali Shan	bcsf19a022@pucit.edu.pk

Background and Overview:

Exception:

An exception is a problem that arises during the execution of a program. It is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exception Handling:

Exception handling is the process of handling errors and exceptions in such a way that they do not hinder normal execution of the system. For example, User divides a number by zero, this will compile successfully but an exception or run time error will occur due to which our applications will be crashed.

Activities:

Pre-Lab Activities:

Compile Time Errors:

Compile Time Errors are those errors that are caught during compilation time. Some of the most common compile-time errors are syntax errors, library references, incorrect import of library functions and methods, uneven bracket pair(s), etc.

Example:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      cout << "Hello Scaler Topics!"
6      return 0;
7  }
```

Fig. 01 (Compile Time Error)

Output:

	Code	Description
abc	E0065	expected a ','
✖	C2143	syntax error: missing ';' before 'return'

Fig. 02 (Compile Time Error)

Run-Time Errors:

Run-Time Errors are those errors that cannot be caught during compilation time. As we cannot check these errors during compile time, we name them Exceptions. Exceptions can cause some serious issues so we should handle them effectively.

Example:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a = 5;
6      // Dividing the number a by zero, so the program will compile easily
7      // but run time error will be generated.
8      cout << a / 0;
9      return 0;
10 }
```

Fig. 03 (Run-Time Error)

Output:

```
(process 12740) exited with code -1073741676.
```

Fig. 04 (Run-Time Error)

Task 01: Run-Time Errors**[Estimated time 20 minutes / 15 marks]**

Write the following programs which should produce an exception:

- Write a program that declares an array of integers with a size of 5, and then tries to access an element at index 6.
- Write a program that declares a pointer to an integer, allocates memory for the integer using new, and then tries to delete the integer using delete twice.
- Write a program that declares a variable of type int, and then tries to divide it by zero.

In-Lab Activities:**Flow of Control:**

The exception handling is mainly performed using three keywords namely - try, catch, and throw using following syntax:

```
try
{
    // code
    throw exception;
}
catch(exception e)
{
    // code for handling exception
}
```

C++ try:

The try block is used to keep the code that is expected to throw some exception. Whenever our code leads to any exception or error, the exception or error gets caught in the catch block. In simple terms, we can say that the try block is used to define the block of code that needs to be tested for errors while it is being executed.

E.g., Suppose we are dealing with databases, we should put the code that is handling the database connection inside a try block as the database connection may raise some exceptions or errors.

C++ catch:

The catch block is used to catch and handle the error(s) thrown from the try block. If there are multiple exceptions thrown from the try block, then we can use multiple catch blocks after the try blocks for each exception. In this way, we can perform different actions for the various occurring exceptions. In simple terms, we can say that the catch block is used to define a block of code to be executed if an error occurs in the try block.

E.g., Let us take the same above example that we are dealing with the database. Now, if during the connection, there is an exception raised inside the try block, then there should be a catch block present to catch or accept the exception and handle the exception. The catch block ensures that the normal flow of the code is not halted.

C++ throw:

The throw block is used to throw exceptions to the exception handler which further communicates the error. The type of exception thrown should be same in the catch block. The throw keyword accepts one parameter which is passed to the exception handler. We can throw both pre-defined as well as custom exception(s) as per the requirements.

Whenever we want to explicitly throw an exception, we use the throw keyword. The throw keyword is also used to generate the custom exception.

Following example demonstrates the overall working and syntax of try, catch, and throw in exception handling.

Example:

```

1  #include <iostream>
2  using namespace std;
3  int main(){
4      int x = 99;
5      cout << "Before the try block." << endl;
6      try{
7          cout << "Inside the try block." << endl;
8          // Throwing an exception if the
9          // value of x is smaller than 100.
10         if (x < 100){
11             // Throwing the value of x as exception as x is now less than 100.
12             throw x;
13             cout << "After throw the throw block." << endl;
14         }
15     }
16     // Catching the value of x thrown by the throw keyword from the try block.
17     catch (int x){
18         cout << "Exception caught in the catch block." << endl;
19     }
20     return 0;
21 }

```

Fig. 05 (Flow of Control)

Output:


```

Microsoft Visual Studio Debug Console
Before the try block.
Inside the try block.
Exception caught in the catch block.

```

Fig. 06 (Flow of Control)

If you do not know the type of throw used in the try block, we can always use the "three dots" syntax (...) inside the catch block, which will handle any type of exception.

Example:

```

1  #include <iostream>
2  using namespace std;
3  int main(){
4      try{
5          int age = 25;
6          if (age <= 18){
7              cout << "Access denied.";
8          }
9          else{
10             // throwing any random value as exception as age is less than 18.
11             throw 505;
12         }
13     }
14     // Catching the thrown exception and displaying access denied!
15     catch (...){
16         cout << "Access denied!" << endl;
17     }
18     return 0;
19 }

```

Fig. 07 (Flow of Control)

Output:

Fig. 08 (Flow of Control)

Rethrowing an Exception:

In the program execution, when an exception received by catch block is passed to another exception handler then such situation is referred to as rethrowing of exception.

This is done with the help of following statement,

throw;

The above statement does not contain any arguments. This statement throws the exception to next try catch block.

Example:

```

1  #include<iostream>
2  using namespace std;
3
4  void subtract(int p, int q){ //A function called subtract with two arguments is defined
5      cout << "The subtract() function contains\n";
6      try{ //try block1
7          if (p == 0) //This condition checks whether p is equal to zero or not.
8              //if yes throw an exception else perform subtraction
9              throw p; //This statement throws an exception
10         else
11             cout << "The result of subtraction is:" << p - q << "\n";
12     }
13     catch (int){ //This statement catches the exception throw it again to the next try block
14         cout << "NULL value is caught\n";
15         throw;
16     }
17     cout << "End of subtract()\n\n";
18 }
19 int main(){
20     cout << "The main() function contains\n";
21     try{
22         subtract(5, 3); //passing integer values to subtract
23         subtract(0, 2);
24     }
25     catch (int){ //This statement catches the rethrown exception
26         cout << "NULL value caught inside main()\n";
27     }
28     cout << "End of main() function \n";
29     return 0;
30 }

```

Fig. 09 (Rethrowing Exception)

Output:



```

Microsoft Visual Studio Debug Console

The main() function contains
The subtract() function contains
The result of subtraction is:2
End of subtract()

The subtract() function contains
NULL value is caught
NULL value caught inside main()
End of main() function

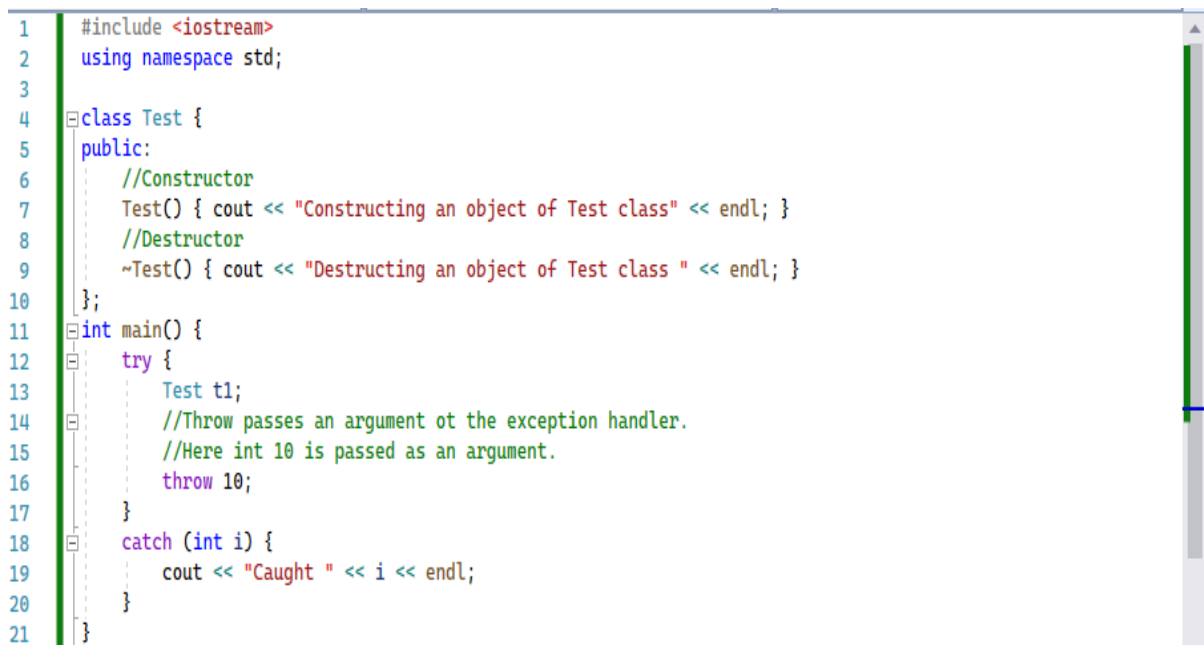
```

Fig. 10 (Rethrowing Exception)

Constructor Destructor and Exception Handling:

Consider the following example:

Example:



```

1  #include <iostream>
2  using namespace std;
3
4  class Test {
5  public:
6      //Constructor
7      Test() { cout << "Constructing an object of Test class" << endl; }
8      //Destructor
9      ~Test() { cout << "Destructing an object of Test class " << endl; }
10 };
11 int main() {
12     try {
13         Test t1;
14         //Throw passes an argument to the exception handler.
15         //Here int 10 is passed as an argument.
16         throw 10;
17     }
18     catch (int i) {
19         cout << "Caught " << i << endl;
20     }
21 }

```

Fig. 11 (Constructor Destructor and Exception Handling)

Output:



```

Microsoft Visual Studio Debug Console

Constructing an object of Test class
Destructing an object of Test class
Caught 10

```

Fig. 12 (Constructor Destructor and Exception Handling)

Explanation:

When an exception is thrown, the objects' destructors (whose scope ends with the try block) are automatically triggered before the catch block is performed. As a result, the preceding software outputs "Destructing an item of Test" before "Caught 10".

This happens when an exception is thrown from a Constructor:

Example:

```
1  #include <iostream>
2  using namespace std;
3  class Test1 {
4  public:
5      Test1() { cout << "Constructing an Object of Test1" << endl; }
6      ~Test1() { cout << "Destructing an Object of Test1" << endl; }
7  };
8  class Test2 {
9  public:
10     // Following constructor throws an integer exception
11     Test2() {
12         cout << "Constructing an Object of Test2" << endl;
13         throw 10;
14     }
15     ~Test2() { cout << "Destructing an Object of Test2" << endl; }
16 };
17 int main() {
18     try {
19         Test1 t1; // Constructed and destructed
20         Test2 t2; // Partially constructed
21         Test1 t3; // t3 is not constructed as this statement never gets executed
22     }
23     catch (int i) {
24         cout << "Caught " << i << endl;
25     }
26 }
```

Fig. 13 (Constructor Destructor and Exception Handling)

Output:

```
Microsoft Visual Studio Debug Console
Constructing an Object of Test1
Constructing an Object of Test2
Destructing an Object of Test1
Caught 10
```

Fig. 14 (Constructor Destructor and Exception Handling)

Explanation:

Destructors are only invoked for properly developed constructed objects. When a function object throws an exception, the object's destructor is not invoked.

Task 01: Calculate Average**[Estimated time 20 minutes / 20 marks]**

- Write a program that reads in a sequence of numbers from the user until the user enters a non-numeric value.
- Calculate the average of the numbers and print it to the console.
- If the user enters less than 5 numbers, throw an exception of type **"runtime_error"**.
- If the user enters at least 5 numbers but all the numbers are negative, throw an exception of type **"domain_error"**.
- Catch the exceptions and print appropriate error messages to the console.

Task 02: Banking System**[Estimated time 40 minutes / 30 marks]**

- Write a program that manages a simple banking system. The program should have the following functionality:
 - The user can create a new account with a name and a balance.
 - The user can deposit and withdraw funds from an account.
 - The user can transfer funds between two accounts.
 - The user can view the balance of an account.
- The program should use exception handling to handle errors that may arise during the execution of the program. Specifically, the program should throw an exception of type **"invalid_argument"** if the user attempts to create an account with a negative balance or a balance that exceeds \$1,000,000. The program should also throw an exception of type **"out_of_range"** if the user attempts to withdraw more funds than are available in an account or transfer more funds than are available in the source account.

Task 03: Create Object**[Estimated time 30 minutes / 30 marks]**

- You are given a class Person with the following private members:
 - name (string)
 - age (int)
 - isStudent (bool)
- Implement a program that creates a Person object from user input.
- The program should prompt the user for the name, age, and student status of the person.
 - If the user enters an age that is negative, the program should throw an exception of type **invalid_argument** with the message "Age cannot be negative".
 - If the user enters an invalid input for the student status (i.e. anything other than "yes" or "no"), the program should throw an exception of type **invalid_argument** with the message "Invalid input for student status".
- If an exception is thrown during the creation of the Person object, the program should catch the exception, print the error message to the console, and then rethrow the exception.
- If the Person object is created successfully, the program should print the object's details to the console.

Task 04: Standard Deviation**[Estimated time 30 minutes / 20 marks]**

- Write a program that reads in a list of integers from a file and calculates the mean and standard deviation of the numbers.

- The program should be able to handle errors that may arise during the input and calculation process. Specifically, the program should use exception handling to handle the following types of errors:
 - If the file containing the list of integers cannot be opened, the program should throw an exception of type **“runtime_error”**.
 - If the file does not contain any integers, the program should throw an exception of type **“runtime_error”**.
 - If the mean or standard deviation cannot be calculated due to division by 0 or other errors, the program should throw an exception of type **“runtime_error”**.
- To calculate the mean and standard deviation, you can use the following formulas:
 - **Mean:** $\text{mean} = (\text{sum of numbers}) / (\text{number of numbers})$
 - **Standard deviation:** $\text{sd} = \sqrt{(\text{sum of } (\text{number} - \text{mean})^2) / (\text{number of numbers})}$

Post-Lab Activities:**Task 01: File Search****[Estimated time 40 minutes / 30 marks]**

- Write a program that reads in a file containing a list of words, and then prompts the user to enter a search term.
- The program should then search the list of words for the search term and print out all occurrences of the search term in the file.
- The program should be able to handle errors that may arise during the input and search process. Specifically, the program should use exception handling to handle the following types of errors:
 - If the file cannot be opened, the program should throw an exception of type **"runtime_error"**.
 - If the search term is not found in the file, the program should throw an exception of type **"logic_error"**.
 - If any other type of error occurs during the search process, the program should throw an exception of type **"exception"**.

Task 02: 2D Matrix**[Estimated time 60 minutes / 40 marks]**

- You are given a class Matrix that represents a 2-dimensional matrix of integers. The class has the following private members:
 - data (int **)
 - rows (int)
 - cols (int)
- Implement the constructor, destructor, and set method of the Matrix class.
- The constructor should take two integers rows and cols and initialize the matrix with those dimensions.
- The set method should take two integers i and j and an integer value, and set the value of the matrix at position (i, j) to value.
- If the user tries to set a value at a position that is out of bounds, the program should throw an exception of type out_of_range with the message **"Index out of range"**.
- If there is not enough memory to allocate the matrix, the program should throw an exception of type bad_alloc with the message **"Failed to allocate memory for matrix"**.

Submissions:

- For In-Lab Activity:
 - Save the files on your PC.
 - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
 - Submit the .cpp file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** [15 marks]
 - Task 01: Run-Time Errors [15 marks]
- **Division of In-Lab marks:** [100 marks]
 - Task 01: Calculate Average [20 marks]
 - Task 02: Banking System [30 marks]
 - Task 03: Create Object [30 marks]
 - Task 04: Standard Deviation [20 marks]
- **Division of Post-Lab marks:** [70 marks]
 - Task 01: File Search [30 marks]
 - Task 02: 2D Matrix [40 marks]

References and Additional Material:

- **Exception Handling**
https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm
- **Rethrowing Exceptions**
<https://www.learncpp.com/cpp-tutorial/rethrowing-exceptions/>
- **Exception Handling and Object Destruction**
<https://www.geeksforgeeks.org/exception-handling-and-object-destruction-in-cpp/>

Lab Time Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:40: In-Lab Task
- Slot – 03 – 00:40 – 01:20: In-Lab Task
- Slot – 04 – 01:20 – 02:20: In-Lab Task
- Slot – 05 – 02:20 – 02:45: Evaluation of Lab Tasks
- Slot – 06 – 02:45 – 03:00: Discussion on Post-Lab Task