Course Code: CSE-2104

Course Title: Object Oriented Programming Laboratory

Lab 9: Implementation of Exception Handling in Object Oriented Programming.

Date: 03/11/2024

Submission: 10/11/2024

C++ Exceptions

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things. When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an exception (throw an error).

C++ try and catch

- Exception handling in C++ consist of three keywords: `try`, `throw` and `catch`.
- The try statement allows to define a block of code to be tested for errors while it is being executed.
- The throw keyword throws an exception when a problem is detected, which lets us create a custom error.
- The catch statement allows to define a block of code to be executed, if an error occurs in the try block.

```
try
{
  // Block of code to try
  throw exception; // Throw an exception when a problem arises
}
catch ()
{
  // Block of code to handle errors
}
```

Example 1:

```
#include <iostream>
using namespace std;

int main()
{
  try

  {
    int age = 15;
    if (age >= 18)
    {
      cout << "Access granted - you are old enough.";
    } else
     {
      throw (age);
     }
    }
  catch (int myNum)
  {
    cout << "Access denied - You must be at least 18 years
    old.\n";
    cout << "Age is: " << myNum;
  }
  return 0;
}
```

Output:

```
Access denied - You must be at least 18 years old.
Age is: 15
```

Example explained

We use the try block to test some code: If the age variable is less than 18, we will throw an exception, and handle it in our catch block.

In the catch block, we catch the error and do something about it. The catch statement takes a parameter: in our example we use an int variable (myNum) (because we are throwing an exception of int type in the try block (age)), to output the value of age.

If no error occurs (e.g. if age is 20 instead of 15, meaning it will be be greater than 18), the catch block is skipped.

Example 2:

You can also use the throw keyword to output a reference number, like a custom error number/code for organizing purposes (505 in our example):

```
#include <iostream>
using namespace std;

int main() {
  try {
    int age = 15;
    if (age >= 18) {
      cout << "Access granted - you are old enough.";
    } else {
      throw 505;
    }
  }
  catch (int myNum) {
    cout << "Access denied - You must be at least 18 years old.\n";
    cout << "Error number: " << myNum;
  }
  return 0;
}
```

Output:

```
Access denied - You must be at least 18 years old.
Error number: 505
```

Example 3:

There is a special catch block called the ‘catch-all’ block, written as catch(...), that can be used to catch all types of exceptions. If you do not know the throw type used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception:

```
#include <iostream>
using namespace std;

int main() {
  try {
    int age = 15;
    if (age >= 18) {
      cout << "Access granted - you are old enough.";
    } else {
      throw 505;
    }
  }
  catch (...) {
    cout << "Access denied - You must be at least 18 years
old.\n";
  }
  return 0;
}
```

Output:

```
Access denied - You must be at least 18 years old.
```

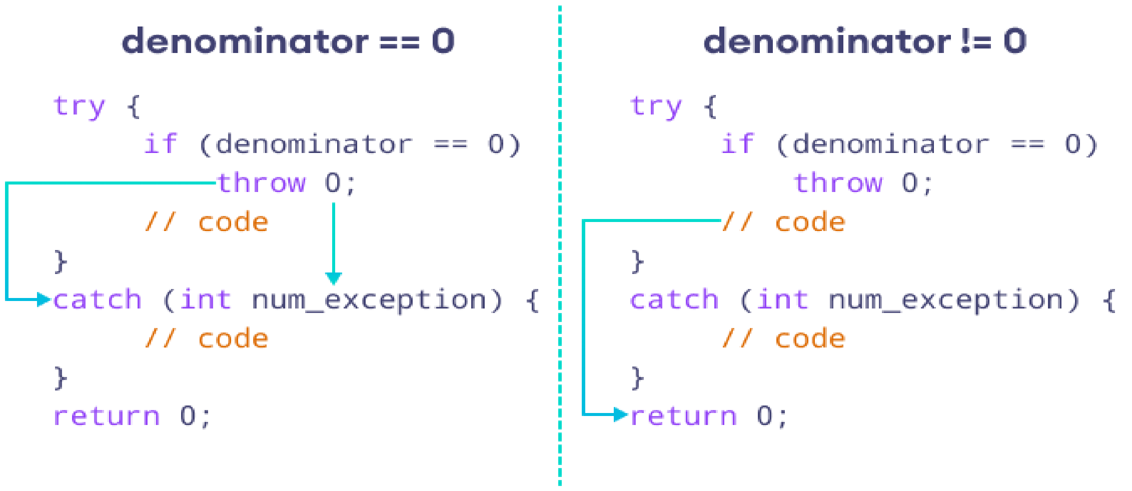
Example 4:

```
#include <iostream>
using namespace std;

int main()
{
    // try block
    try {

        // throw
        throw 10;
    }

    // catch block
    catch (char* excp) {
        cout << "Caught " << excp;
    }

    // catch all
    catch (...) {
        cout << "Default Exception\n";
    }
    return 0;
}
```

Output:

```
Default Exception
```

Example 5:



denominator == 0

```
try {
    if (denominator == 0)
        throw 0;
    // code
}
catch (int num_exception) {
    // code
}
return 0;
```

denominator != 0

```
try {
    if (denominator == 0)
        throw 0;
    // code
}
catch (int num_exception) {
    // code
}
return 0;
```

```
#include <iostream>
using namespace std;

int main() {

    double numerator, denominator, divide;

    cout << "Enter numerator: ";
    cin >> numerator;

    cout << "Enter denominator: ";
    cin >> denominator;

    try {

        // throw an exception if denominator is 0
        if (denominator == 0)
            throw 0;

        // not executed if denominator is 0
        divide = numerator / denominator;
        cout << numerator << " / " << denominator << " = " <<
divide << endl;
    }

    catch (int num_exception) {
        cout << "Error: Cannot divide by " << num_exception <<
endl;
    }

    return 0;
}
```

Output1:

```
Enter numerator: 72
Enter denominator: 0
Error: Cannot divide by 0
```

Output2:

```
Enter numerator: 72
Enter denominator: 3
72 / 3 = 24
```

Example 6:

```
#include <iostream>
#include <stdexcept> //Defines several standard classes used
for reporting exceptions.
using namespace std;

int main()
{

    // try block
    try {
        int numerator = 10;
        int denominator = 0;
        int res;

        // check if denominator is 0 then throw runtime
        // error.
        if (denominator == 0) {
            throw runtime_error(
                "Division by zero not allowed!");
        }

        // calculate result if no exception occurs
        res = numerator / denominator;
        //[printing result after division
        cout << "Result after division: " << res << endl;
    }
    // catch block to catch the thrown exception
    catch (const exception& e) {
        // print the exception
        cout << "Exception " << e.what() << endl;
    }

    return 0;
}
```

Output:

```
Exception Division by zero not allowed!
```

Example 7:

```
#include <iostream>
  #include <exception>
  using namespace std;
  class MyException : public exception{
      public:
          const char * what() const throw()
          {
              return "Attempted to divide by zero!\n";
          }
  };
  int main()
  {
      try
      {
          int x, y;
        cout << "Enter the two numbers : \n";
          cin >> x >> y;
          if (y == 0)
          {
              MyException z;
              throw z;
          }
          else
          {
              cout << "x / y = " << x/y << endl;
          }
      }
      catch(exception& e)
      {
          cout << e.what();
      }
  }
```

Output1:

```
Enter the two numbers :
10
2
x / y = 5
```

Output2:

```
Enter the two numbers :
10
0
Attempted to divide by zero!
```

Example 8:

```
#include <iostream>
using namespace std;

int main()
{
    int x = -1;

    // Some code
    cout << "Before try \n";

    // try block
    try {
        cout << "Inside try \n";
        if (x < 0) {
            // throwing an exception
            throw x;
            cout << "After throw (Never executed) \n";
        }
    }

    // catch block
    catch (int x) {
        cout << "Exception Caught \n";
    }

    cout << "After catch (Will be executed) \n";
    return 0;
}
```

Output:

```
Before try
Inside try
Exception Caught
After catch (Will be executed)
```

C++ Multiple catch Statements

In C++, we can use multiple catch statements for different kinds of exceptions that can result from a single block of code such as:

```
try {
    // code
}
catch (exception1) {
    // code
}
catch (exception2) {
    // code
}
catch (...) {
    // code
}
```

Here, our program catches exception1 if that exception occurs. If not, it will catch exception2 if it occurs. If there is an error that is neither exception1 nor exception2, then the code inside of catch (...) { } is executed.

Example 9:

This program divides two numbers and stores the result in an array element. There are two possible exceptions that can occur in this program:

- If the array is out of bounds i.e. if the index of the array is greater than the size of the array
- If a number is divided by 0

These exceptions are caught in multiple catch statements.

```
#include <iostream>
using namespace std;

int main() {

    double numerator, denominator, arr[4] = {0.0, 0.0, 0.0, 
0.0};
    int index;

    cout << "Enter array index: ";
    cin >> index;

    try {

        // throw exception if array out of bounds
        if (index >= 4)
            throw "Error: Array out of bounds!";


        // not executed if array is out of bounds
        cout << "Enter numerator: ";
        cin >> numerator;

        cout << "Enter denominator: ";
        cin >> denominator;

        // throw exception if denominator is 0
        if (denominator == 0)
            throw 0;


        // not executed if denominator is 0
        arr[index] = numerator / denominator;
        cout << arr[index] << endl;
    }

    // catch "Array out of bounds" exception
    catch (const char* msg) {   //Notice the catch parameter
const char* msg. This indicates that the catch statement takes
a string literal as an argument.
        cout << msg << endl;
    }

    // catch "Divide by 0" exception
    catch (int num) {
        cout << "Error: Cannot divide by " << num << endl;
    }

    // catch any other exception
    catch (...) {
        cout << "Unexpected exception!" << endl;
    }


    return 0;
}
```

Output1:

```
Enter array index: 5
Error: Array out of bounds!
```

Output2:

```
Enter array index: 2
Enter numerator: 5
Enter denominator: 0
Error: Cannot divide by 0
```

Output3:

```
Enter array index: 2
Enter numerator: 5
Enter denominator: 2
2.5
```

Example 10:

In C++, try/catch blocks can be nested. Also, an exception can be re-thrown using “throw; “. The following program shows try/catch blocks nesting.

```
#include <iostream>
using namespace std;

int main()
{

    // nesting of try/catch
    try {
        try {
            throw 20;
        }
        catch (int n) {
            cout << "Handle Partially ";
            throw; // Re-throwing an exception
        }
    }
    catch (int n) {
        cout << "Handle remaining ";
    }
    return 0;
}
```

Output:

```
Handle Partially Handle remaining
```

Example 11:

If an exception is thrown and not caught anywhere, the program terminates abnormally. In the following program, a char is thrown, but there is no catch block to catch the char.

```
#include <iostream>
using namespace std;

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

Output:

```
terminate called after throwing an instance of 'char'
```

Example 12:

When an exception is thrown, all objects created inside the enclosing try block are destroyed before the control is transferred to the catch block.

```
#include <iostream>
using namespace std;

// Define a class named Test
class Test {
public:
    // Constructor of Test
    Test() { cout << "Constructor of Test " << endl; }
    // Destructor of Test
    ~Test() { cout << "Destructor of Test " << endl; }
};

int main()
{
    try {
        // Create an object of class Test
        Test t1;

        // Throw an integer exception with value 10
        throw 10;
    }
    catch (int i) {
        // Catch and handle the integer exception
        cout << "Caught " << i << endl;
    }
}
```

Output:

```
Constructor of Test
Destructor of Test
Caught 10
```