Exception Handling

Faculty : SREEDIVYA
Branch : S6ECE

Nested try statements

- In C++, a nested try block refers to a try-block nested inside another try or catch block.
- It is used to handle exceptions in cases where different exceptions occur in different parts of the code.
- When try blocks are nested and a throw occurs in a function called by an inner try block, control is transferred outward through the nested try blocks until the first catch block is found whose argument matches the argument of the throw expression.

Syntax of nested try

```
try
{
    // Code..... throw e2
    try
    {
        // code..... throw e1
    }
    catch (Exception e1)
    {
        // handling exception
    }
}
catch (Exception e2)
{
    // handling exception
}
```

Here,
e1: Exception thrown in inner block.
e2: Exception thrown in outer block.

Example of Nested Try Blocks

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

void func (int n)
{
    if (n < 10) {
        throw 22;
    }
    else {
        throw 'c';
    }
}
```

```
int main()
{
    try {
        try {
            cout << "Throwing exception from inner try "
                    "block\n";
            func(2);
        }
        catch (int n) {
            cout << "Inner Catch Block caught the exception"
                 << endl;
        }
    }
    catch (char c) {
        cout << "Outer catch block caught the exception"
             << endl;
    }

    cout << "Out of the block";

    return 0;
}
```

OUTPUT

Throwing exception from inner try block

Inner Catch Block caught the exception

Out of the block

Explanation

Here, we used func() function to throw two exceptions of int and char type. We used an inner try block to catch integer exceptions. Now, whenever the try blocks throw an exception, the control moves outwards from the nested block till the matching catch block is found. In this case, it was the inner catch block that caught the exception.

What happens if we throw a character exception that the outer catch block is programmed to handle?

```
#include <iostream>
using namespace std;
void func(int n)
{
    if (n < 10) {
        throw 22;
    }
    else {
        throw 'c';
    }
}
```

```
int main()
{
    try {
        try {
            cout << "Throwing exception from inner try "
                 "block\n";
            func(12);
        }
        catch (int n) {
            cout << "Inner Catch Block caught the exception"
                 << endl;
        }
    }
    catch (char c) {
        cout << "Outer catch block caught the exception"
             << endl;
    }

    cout << "Out of the block";

    return 0;
}
```

OUTPUT

Throwing exception from inner try block

Outer catch block caught the exception

Out of the block

Rethrowing the exception

- If a catch block cannot handle the particular exception it has caught, you can rethrow the exception.
- The rethrow expression (throw without assignment_expression) causes the originally thrown object to be rethrown.
- Rethrowing an exception in C++ involves catching an exception within a try block and instead of dealing with it locally throwing it again to be caught by an outer catch block.
- By doing this, we preserve the type and details of the exception ensuring that it can be handled at the appropriate level within our program.
- Because the exception has already been caught at the scope in which the rethrow expression occurs, it is rethrown out to the next dynamically enclosing try block. Therefore, it cannot be handled by catch blocks at the scope in which the rethrow expression occurred. Any catch blocks for the dynamically enclosing try block have an opportunity to catch the exception.

Example for rethrowing exceptions

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

int divide(int numerator, int denominator)
{
    try {
        if (denominator == 0) {
            throw runtime_error("Division by zero!");
        }
        return numerator / denominator;
    }
    catch (const exception& e) {
        cout << "Caught exception in divide(): " << e.what()
             << endl;
        throw;                                  // Rethrow the caught exception to handle it at a higher level
    }
}
```

```
int calculateSum(int a, int b)
{
   try {
      if (a < 0 || b < 0) {
        throw invalid_argument("Negative numbers not allowed!");  // Throw an invalid_argument exception for negative numbers
      }
      return a + b;
   }
   catch (const exception& e) {
      cout << "Caught exception in calculateSum(): "  << e.what() << endl;
      throw;                    // Rethrow the caught exception to handle it at a higher level

   }
}
```

```
int main()
{
    try {
            int result = calculateSum(10, divide(20, 2));
            cout << "Result: " << result << endl;

            int invalidResult = calculateSum(5, divide(10, 0));
           cout << "Invalid Result: " << invalidResult << endl;
    }
    catch (const exception& e) {
        cout << "Caught exception in main: " << e.what()  << endl;
            }

    return 0;
}
```

OUTPUT

Result: 20

Caught exception in divide(): Division by zero!

Caught exception in main: Division by zero!

Explanation:

- The program first calculates the sum of 10 and the result of dividing 20 by 2, which is 20. This result is printed and there are no exceptions raised in this part.

- Next, the program attempts to divide by zero when calculating the sum of 5 and the result of dividing 10 by 0. This triggers a “Division by zero!” exception which is caught within the divide() function and rethrown. The rethrown exception is then caught in the main() function and is printed as “Division by zero!” along with the appropriate exception handling messages.