Exception Handling in C++?

The following are mainly errors or bugs that occurred in any program:

1. **Logical error:**
   In this error the logic implemented is incorrect. This error occurs due to less concentration of programmer or poor knowledge of programmer regarding program.
2. **Compilation error:**
   This error is occurred due to use of wrong idiom, function or structure. This error is shown at compilation time of program.
3. **Run Time error:**
   This error occurred at the run time. This error occurs when program crashes during run time.
4. **Time limit error:**
   This error states that program takes more time than required time. This error occurs due to use of wrong logic or lengthy method in program.

.

Exception handling is the process of handling errors and exceptions in such a way that they do not harm normal execution of the system. and the only question is what you will do. Your choices are limited to these:

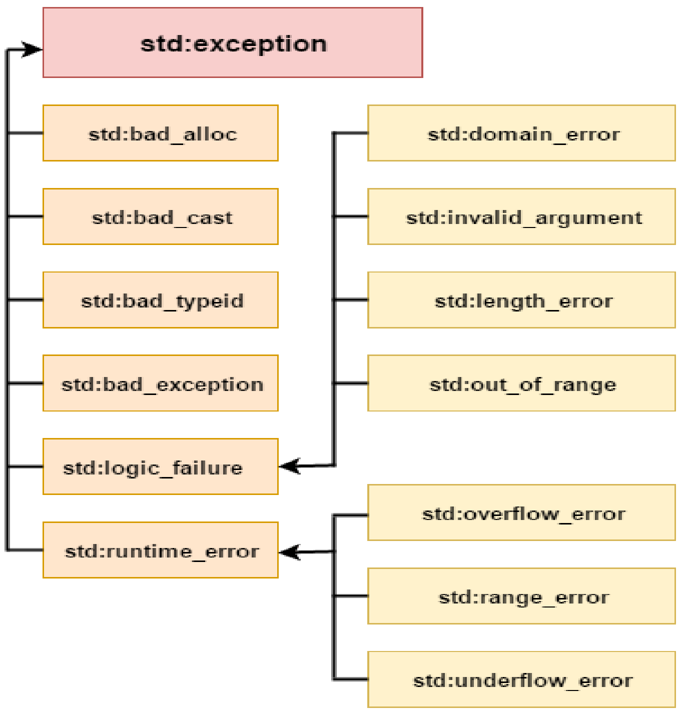
1. Crash the program.
2. Inform the user and exit gracefully.
3. Inform the user and allow the user to try to recover and continue.
4. Take corrective action and continue without disturbing the user.

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. In order to avoid this we'll introduce exception handling technics in our code..

All exceptions are derived from **std::exception** class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

C++ Exception Classes

In C++ standard exceptions are defined in <exception> class that we can use inside our programs. The arrangement of parent-child class hierarchy is shown below:



Standard Exceptions in C++

There are some standard exceptions in C++ under <exception> which we can use in our programs. They are arranged in a parent-child class hierarchy which is depicted below:

- **std::exception** - Parent class of all the standard C++ exceptions.
- **logic_error** - Exception happens in the internal logical of a program.
  - **domain_error** - Exception due to use of invalid domain.
  - **invalid argument** - Exception due to invalid argument.
  - **out_of_range** - Exception due to out of range i.e. size requirement exceeds allocation.
  - **length_error** - Exception due to length error.
- **runtime_error** - Exception happens during runtime.
  - **range_error** - Exception due to range errors in internal computations.
  - **overflow_error** - Exception due to arithmetic overflow errors.
  - **underflow_error** - Exception due to arithmetic underflow errors
- **bad_alloc** - Exception happens when memory allocation with new() fails.
- **bad_cast** - Exception happens when dynamic cast fails.

- **bad_exception** - Exception is specially designed to be listed in the dynamic-exception-specifier.
- **bad_typeid** - Exception thrown by typeid.

C++ Exception Handling Keywords

In C++, we use 3 keywords to perform exception handling:

Try

The code inside this block is like a trial code, which might throw an exception. This exception is caught inside the catch block.

Catch

The code in this block is executed when the code in the try blocks throws an exception.

Throw

This keyword is used to throw an exception when it is encountered. The exception is sent to the exception handler.

Syntax: The code inside the try block is executed. If there is an error generated, then the keyword throw throws the exception to the exception handler, that is, the catch block. The catch block then executed the code, which is inside its block, thus handling the exception.

Let us take a look at sample code for exception handling in c++

```
try {
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
}
```

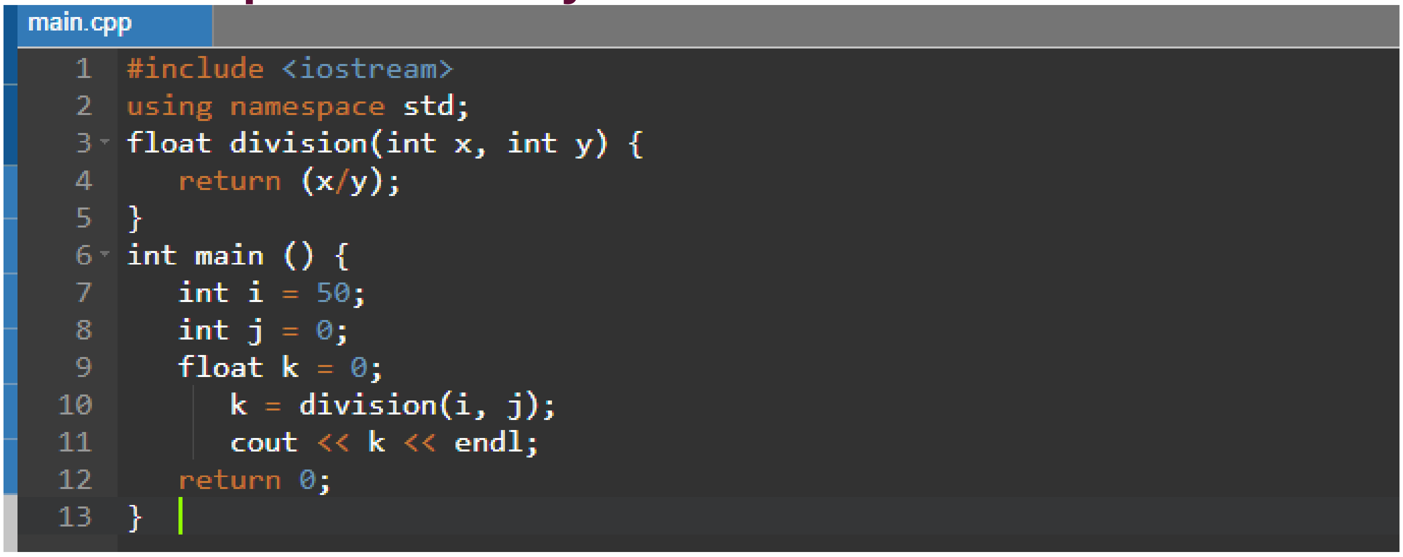
You can list down multiple **catch** statements to catch different type of exceptions in case your **try** block raises more than one exception in different situations.

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!";
   }
   return (a/b);
}
```

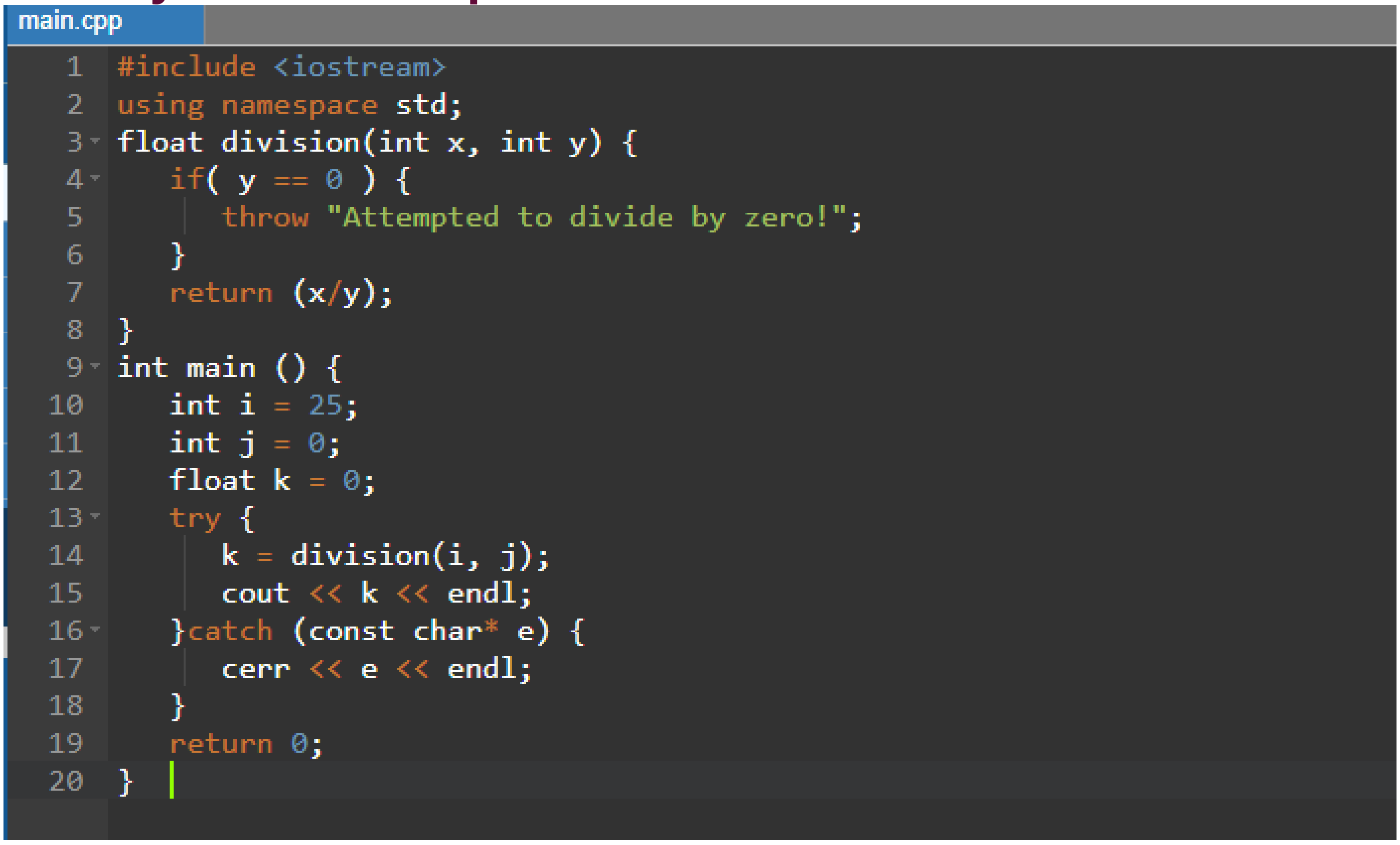
C++ example without try/catch



main.cpp

```
#include <iostream>
using namespace std;
float division(int x, int y) {
   return (x/y);
}
int main () {
   int i = 50;
   int j = 0;
   float k = 0;
      k = division(i, j);
      cout << k << endl;
   return 0;
}
```

C++ try/catch example



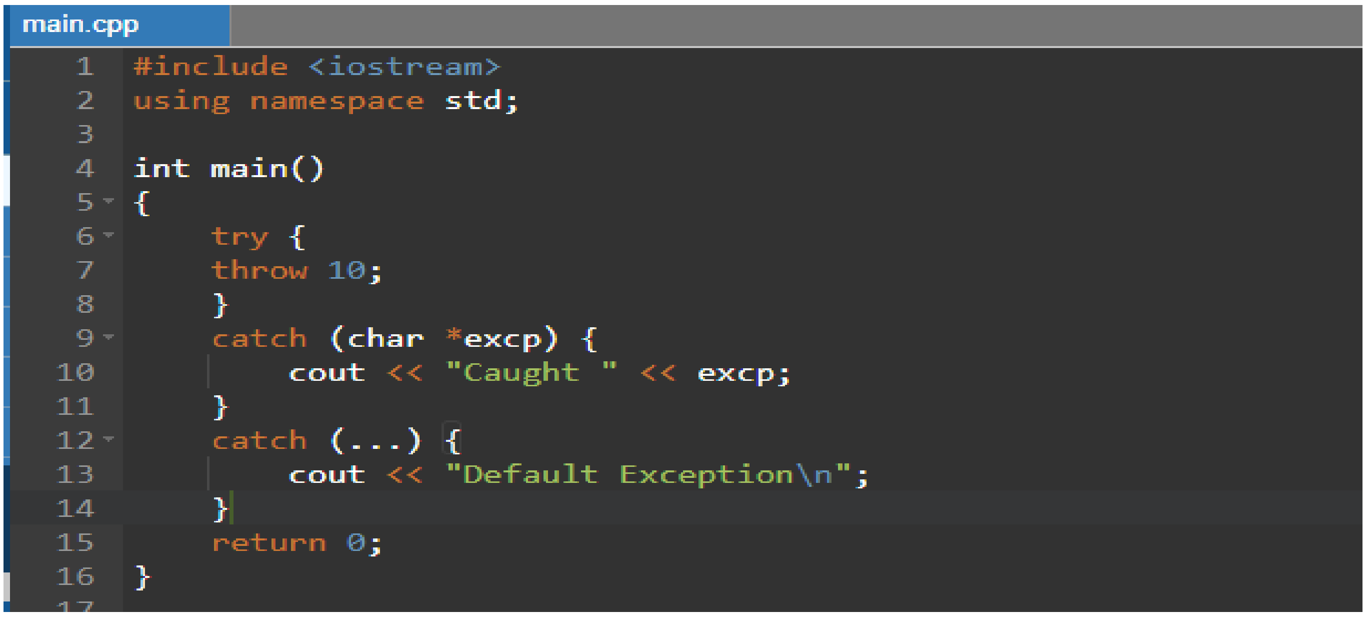
main.cpp

```
#include <iostream>
using namespace std;
float division(int x, int y) {
   if( y == 0 ) {
      throw "Attempted to divide by zero!";
   }
   return (x/y);
}
int main () {
   int i = 25;
   int j = 0;
   float k = 0;
   try {
      k = division(i, j);
      cout << k << endl;
   }catch (const char* e) {
      cerr << e << endl;
   }
   return 0;
}
```

Because we are raising an exception of type **const char***, so while catching this exception, we have to use const char* in catch block. If we compile and run above code, this would produce the following result −

```
Attempted to divide by zero!
```

There is a special catch block called 'catch all' catch(…) that can be used to catch all types of exceptions. For example, in the following program, an int is thrown as an exception, but there is no catch block for int, so catch(…) block will be executed.

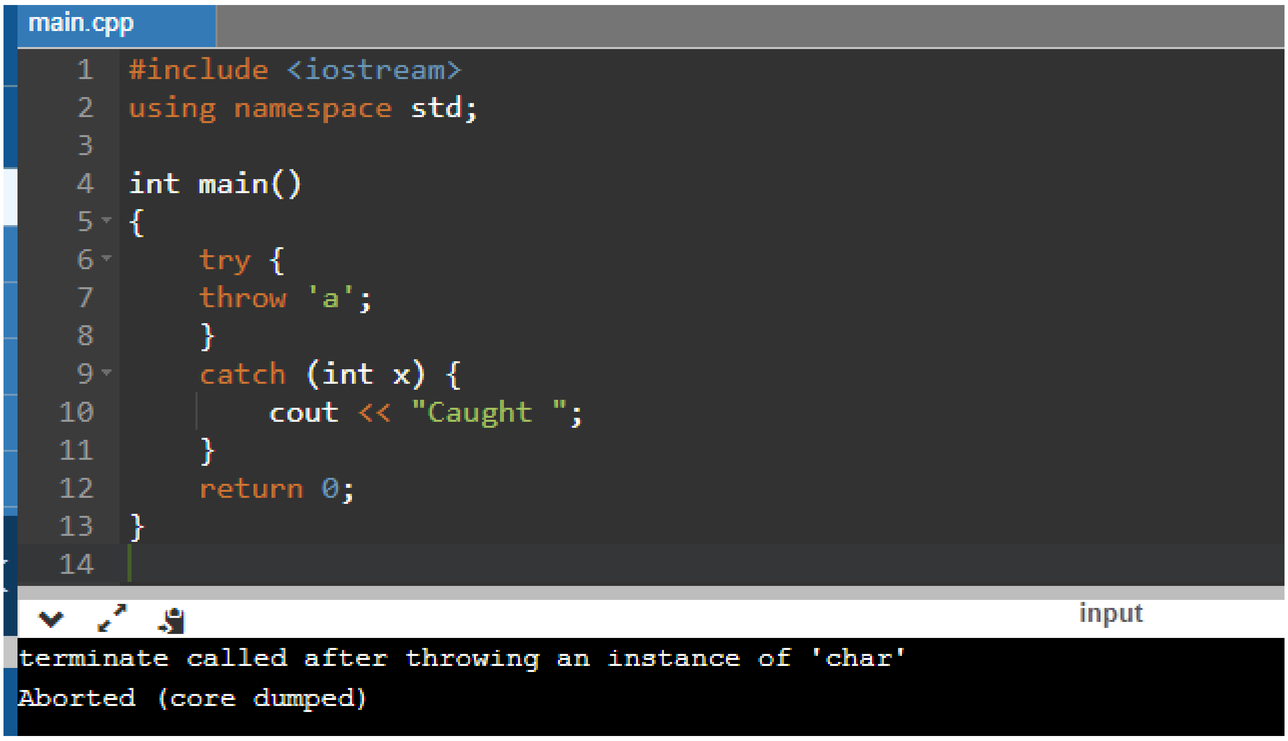


main.cpp

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

int main()
{
    try {
    throw 10;
    }
    catch (char *excp) {
        cout << "Caught " << excp;
    }
    catch (...) {
        cout << "Default Exception\n";
    }
    return 0;
}
```

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



main.cpp

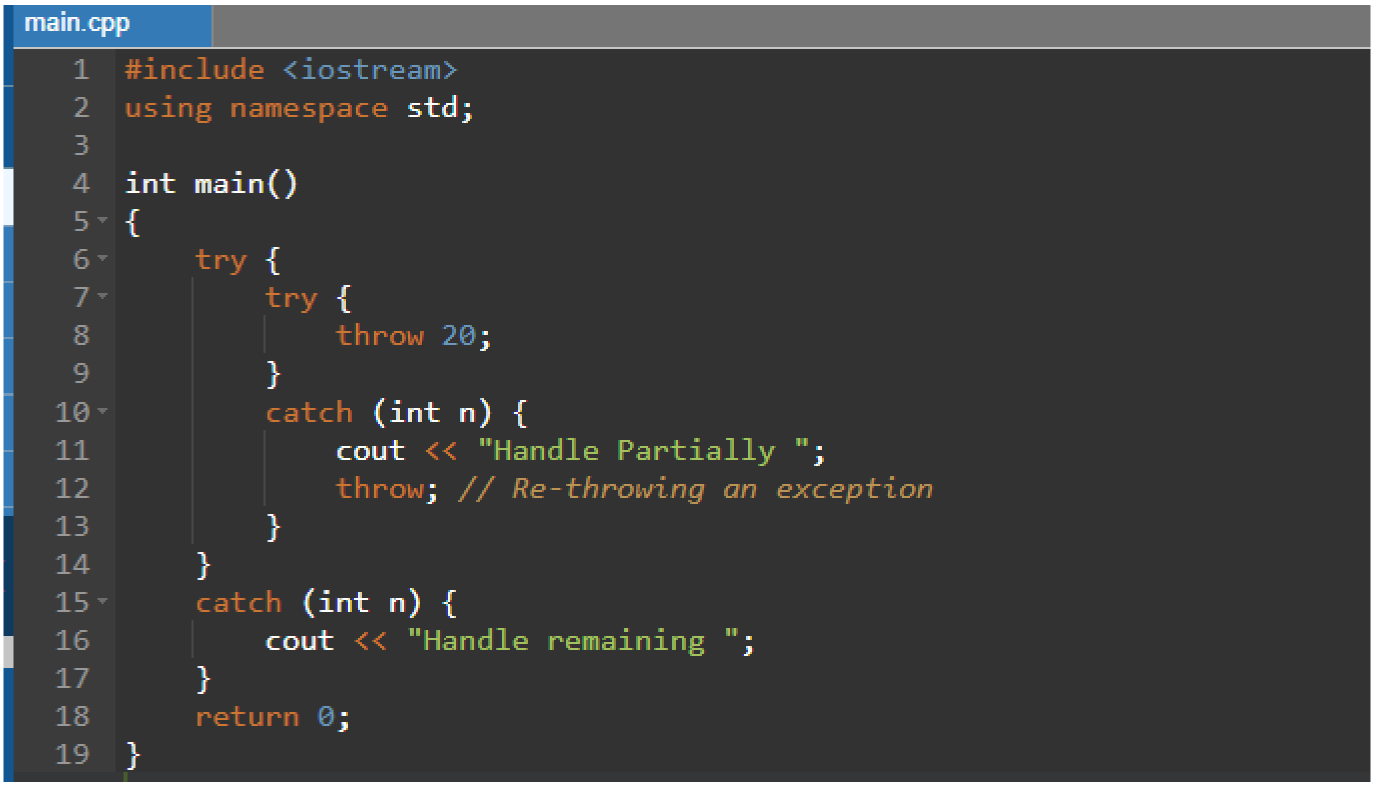
```cpp
#include <iostream>
using namespace std;

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

```
terminate called after throwing an instance of 'char'
Aborted (core dumped)
```

NOTE : In C++, try-catch blocks can be nested. Also, an exception can be re-thrown using "throw; "

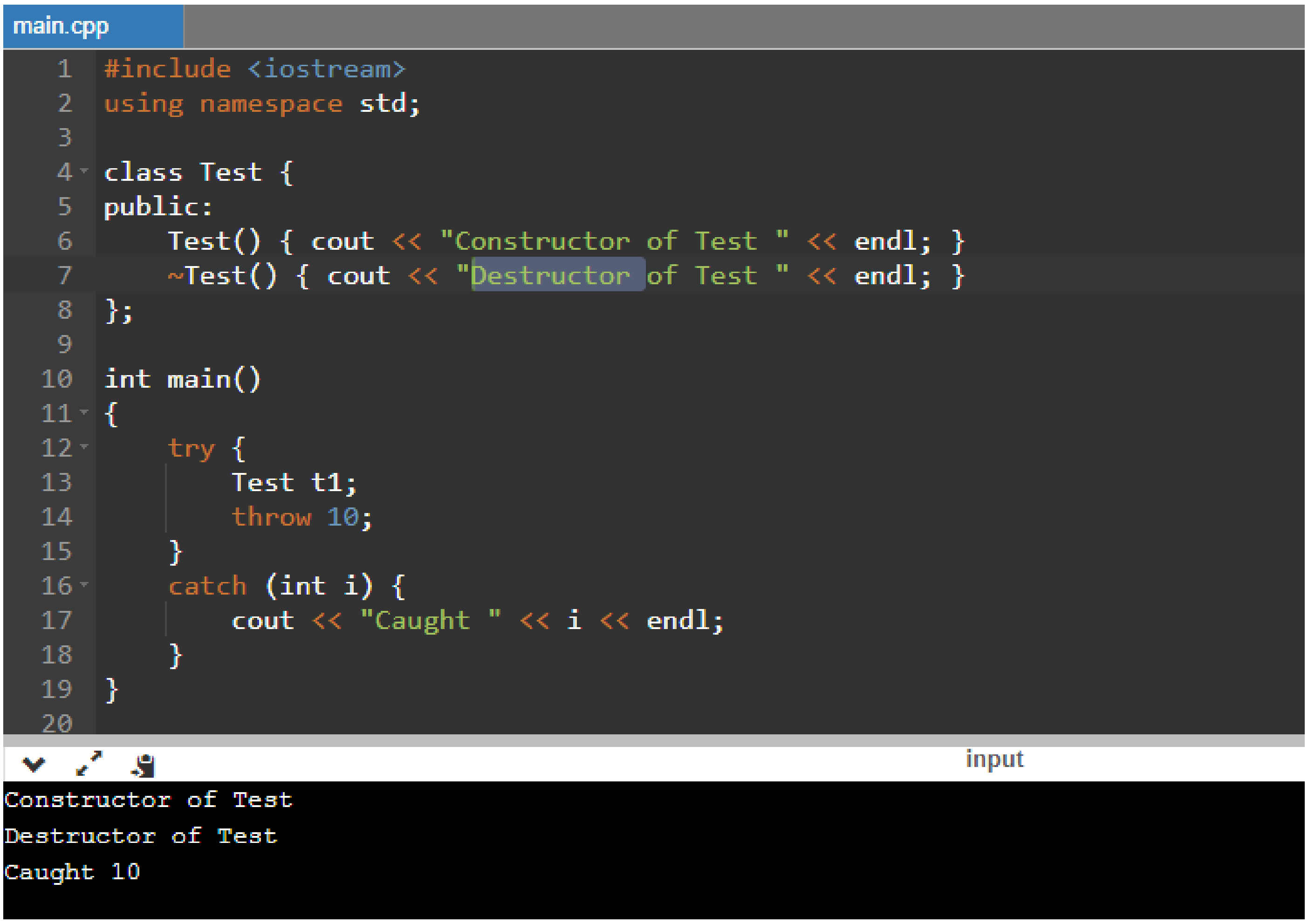
Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz: https://t.me/ccatpreparations **Visit:** https://ccatpreparation.com


main.cpp

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

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

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



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

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

int main()
{
    try {
        Test t1;
        throw 10;
    }
    catch (int i) {
        cout << "Caught " << i << endl;
    }
}
```

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

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.

Summary:

- With exception handling in C++, you can handle runtime errors.
- Runtime errors are the errors that occur during program execution.
- Exception handling helps you handle any unexpected circumstances in your program.
- When the unexpected circumstance occurs, program control is transferred to handlers.
- To catch an exception, you place a section of code under the try-catch block.
- The throw keyword helps the program throw exceptions, helping the program to handle the problem.

- The try keyword helps identify the code block for which certain exceptions will be activated.
- We can overwrite the what() function of the exception header file to define our exceptions.