C++ 10_1

10.1 Exception handling
Exceptions provide a way to react to exceptional circumstances (like runtime errors) in our program by transferring control to special functions called handlers.
C++ exception handling is built upon three keywords: try, catch, and throw.

throw: A program throws an exception when a problem shows up. This is done using a throw keyword.
catch: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
try: A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
// protected code
}catch( ExceptionName e1 )
{
// catch block
}catch( ExceptionName e2 )
{
// catch block
}catch( ExceptionName eN )
{
// 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 statements. The operand of the throw statements 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.

Following is an example of throwing an exception when dividing by zero condition occurs:

```
double division(int a, int b)
{
if( b == 0 )
{
throw "Division by zero condition!";
```

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

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 e )
{
// code to handle ExceptionName exception
}
```

C++ Standard Exceptions
std::exception - An exception and parent class of all the standard C++ exceptions.
std::bad_alloc - This can be thrown by new.
std::bad_cast - This can be thrown by dynamic_cast.
std::bad_exception - This is useful device to handle unexpected exceptions in a C++ program
std::bad_typeid - This can be thrown by typeid.
std::logic_error - An exception that theoretically can be detected by reading the code.
std::domain_error - This is an exception thrown when a mathematically invalid domain is used
std::invalid_argument - This is thrown due to invalid arguments.
std::length_error - This is thrown when a too big std::string is created
std::out_of_range - This can be thrown by the at method from for example a std::vector and std::bitset < >::operator[]().
std::runtime_error - An exception that theoretically can not be detected by reading the code.
std::overflow_error - This is thrown if a mathematical overflow occurs.
std::range_error - This is occured when you try to store a value which is out of range.
std::underflow_error - This is thrown if a mathematical underflow occurs.

Define your own exceptions
You can define your own exceptions by inheriting and overriding exception class functionality.
eg.

```
struct MyException : public exception
{
const char * display() const throw ()
{
return "This is Exception";
}
};
```

Here, what() is a public method provided by exception class and it has been overridden by all the child exception classes. This returns the cause of an exception.