DAY 05

Exceptions

Nested classes :

- A nested class is a class that is defined inside another class. A nested class can have its own members, including data members and member functions, and can also access the private members of the enclosing class.

```
class Cat
{
public:
    class Leg
    {
        //[...]
    };
};

int main()
{
 Cat somecat;                    // instantiate the Cat class
 Cat::Leg somecatsleg;           // instantiate the Leg class
}
```

Exceptions :

- Exception handling is a mechanism that allows a program to deal with unexpected or exceptional situations, such as runtime errors, by transferring control to a designated exception handler. The basic idea is that when an exception occurs, the program can "throw" an object of any type, which is then caught by an exception handler that can handle the exception appropriately.

```
try {
    // code that may throw an exception
}
catch (exceptionType1& e1) {
    // exception handling code for exceptionType1
}
catch (exceptionType2& e2) {
    // exception handling code for exceptionType2
}
// ...
catch (...) {
    // exception handling code for any other exceptions
}
```

```
int main() {
    int x, y, z;
    try {
        cout << "Enter two numbers: ";
        cin >> x >> y;
        if (y == 0) {
            throw runtime_error("Division by zero error.");
        }
        z = x / y;
        cout << "Result: " << z << endl;
    }
    catch (exception& e) {
        cout << "Exception: " << e.what() << endl;
    }
    return 0;
}
```

Catch exception by reference :

- Exceptions can be caught either by value or by reference. Catching exceptions by reference is useful because it avoids creating a copy of the exception object, which can improve performance and reduce memory usage.

```
try {
    // code that may throw an exception
}
catch (ExceptionType& e) {
    // code to handle the exception
}
```

Virtual const char* what() const throw();

- std::exception is a base class for exceptions that can be thrown. The what() function is a virtual function defined in std::exception that returns a null-terminated string describing the exception .

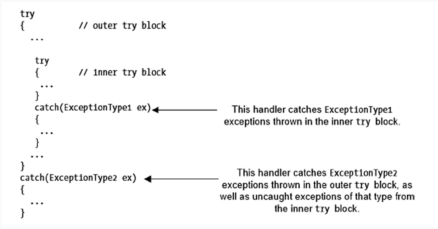
- The const qualifier at the end of the function declaration means that the what() function does not modify the state of the object it is called on. This is important because what() is often called from within a catch block that takes its argument as a const reference, so it is required that the function does not modify the exception object.

- The throw() clause at the end of the function declaration is a deprecated exception specification that was used to indicate that the function does not throw any exceptions. In modern C++, this specification is replaced by the noexcept keyword, which has the same meaning. So the function signature could be rewritten as: **virtual const char* what() const noexcept;**

- The what() function is commonly overridden in derived exception classes to provide a more specific error message .

Nested try blocks :

- You can nest a try block inside another try block. Each try block has its own set of catch blocks to handle exceptions that may be thrown within it, and the catch blocks for a try block are only invoked for exceptions thrown within that try block.



Add custom exception class :

- you can create custom exception classes by deriving from std::exception or one of its derived classes .

```
class myexception: public std::exception
{
public:
  virtual const char* what() const throw()
  {
    return "My exception happened";
  }
};
```

- virtual: It adds nothing, as the method being overridden is already virtual. It can be omitted.
- const char* what(): A member function named what() that takes no arguments and returns a pointer to const char .
- const: The member function can be called via a const pointer or reference to an instance of this class or a derived class .
- throw(): throws no exceptions .