try-block

Associates one or more exception handlers (catch-clauses) with a compound statement.

Syntax

try *compound-statement handler-sequence*

where *handler-sequence* is a sequence of one or more *handler*s, which have the following syntax:

catch (*attr*(optional) *type-specifier-seq declarator* **)** *compound-statement*	(1)
catch (*attr*(optional) *type-specifier-seq abstract-declarator*(optional) **)** *compound-statement*	(2)
catch (...) *compound-statement*	(3)

compound-statement	-	brace-enclosed sequence of statements
***attr*(C++11)**	-	optional list of attributes, applies to the formal parameter
type-specifier-seq	-	part of a formal parameter declaration, same as in a function parameter list
declarator	-	part of a formal parameter declaration, same as in a function parameter list
abstract-declarator	-	part of an unnamed formal parameter declaration, same as in function parameter list

1) Catch-clause that declares a named formal parameter

```
try { /* */ } catch (const std::exception& e) { /* */ }
```

2) Catch-clause that declares an unnamed parameter

```
try { /* */ } catch (const std::exception&) { /* */ }
```

3) Catch-all handler, which is activated for any exception

```
try { /* */ } catch (...) { /* */ }
```

Explanation

See throw exceptions for more information about throw-expressions

A try-block is a statement, and as such, can appear anywhere a statement can appear (that is, as one of the statements in a compound statement, including the function body compound statement). See function-try-block for the try blocks around function bodies. The following description applies to both try-blocks and function-try-blocks.

The formal parameter of the catch clause (*type-specifier-seq* and *declarator* or *type-specifier-seq* and *abstract-declarator*) determines which types of exceptions cause this catch clause to be entered. It cannot be an rvalue reference type, abstract class, incomplete type, or pointer to incomplete type (except that pointers to (possibly cv-qualified) `void` are allowed). If the type of the formal parameter is array type or function type, it is treated as the corresponding pointer type (similar to a function declaration).

When an exception of type E is thrown by any statement in *compound-statement*, it is matched against the types of the formal parameters T of each catch-clause in *handler-seq*, in the order in which the catch clauses are listed. The exception is a match if any of the following is true:

- E and T are the same type (ignoring top-level cv-qualifiers on T)
- T is an lvalue-reference to (possibly cv-qualified) E
- T is an unambiguous public base class of E
- T is a reference to an unambiguous public base class of E
- T is (possibly cv-qualified) U or `const U&` (since C++14), and U is a pointer or pointer to member (since C++17) type, and E is also a pointer or pointer to member (since C++17) type that is implicitly convertible to U by one or more of
  - a standard pointer conversion other than one to a private, protected, or ambiguous base class
  - a qualification conversion
  - a function pointer conversion (since C++17)
- T is a pointer or a pointer to member or a reference to a const pointer (since C++14), while E is `std::nullptr_t`.

```
try {
    f();
} catch (const std::overflow_error& e) {
    // this executes if f() throws std::overflow_error (same type rule)
} catch (const std::runtime_error& e) {
    // this executes if f() throws std::underflow_error (base class rule)
} catch (const std::exception& e) {
    // this executes if f() throws std::logic_error (base class rule)
} catch (...) {
    // this executes if f() throws std::string or int or any other unrelated type
}
```

The catch-all clause `catch (...)` matches exceptions of any type. If present, it has to be the last catch clause in the *handler-seq*. Catch-all block may be used to ensure that no uncaught exceptions can possibly escape from a function that offers nothrow exception guarantee.

If no matches are found after all catch-clauses were examined, the exception propagation continues to the containing try-block, as described in throw-expression. If there are no containing try-blocks left, `std::terminate` is executed (in this case, it is implementation-defined whether any stack unwinding occurs at all: throwing an uncaught exception is permitted to terminate the program without invoking any destructors).

When entering a catch clause, if its formal parameter is a base class of the exception type, it is copy-initialized from the base class subobject of the exception object. Otherwise, it is copy-initialized from the exception object (this copy is subject to copy elision).

```
try {
   std::string("abc").substr(10); // throws std::length_error
// } catch (std::exception e) { // copy-initialization from the std::exception base
//     std::cout << e.what(); // information from length_error is lost
// }
```

```
} catch (const std::exception& e) { // reference to the base of a polymorphic object
     std::cout << e.what(); // information from length_error printed
}
```

If the parameter of the catch-clause is a reference type, any changes made to it are reflected in the exception object, and can be observed by another handler if the exception is rethrown with `throw;`. If the parameter is not a reference, any changes made to it are are local and its lifetime ends when the handler exits.

Within a catch-clause, `std::current_exception` can be used to capture the exception in an `std::exception_ptr`, and `std::throw_with_nested` may be used to build nested exceptions. (since C++11)

Other than by throwing or rethrowing the exception, the catch-clause after a regular try block (not function-try-block) may be exited with a return, continue, break, goto, or by reaching the end of its *compound-statement*. In any case, this destroys the exception object (unless an instance of `std::exception_ptr` exists that refers to it).

Notes

The throw-expression `throw NULL;` is not guaranteed to be matched by a pointer catch clause, because the exception object type may be `int`, but `throw nullptr;` is assuredly matched by any pointer or pointer-to-member catch clause.

If a catch-clause for a derived class is placed after the catch-clause for a base class, the derived catch-clause will never be executed.

```
try {
    f();
} catch (const std::exception& e) {
    // will be executed if f() throws std::runtime_error
} catch (const std::runtime_error& e) {
    // dead code!
}
```

If goto is used to exit a try-block and if any of the destructors of block-scoped automatic variables that are executed by the `goto` throw exceptions, those exceptions are caught by the try blocks in which the variables are defined:

```
label:
    try {
        T1 t1;
        try {
            T2 t2;
            if(condition)
                goto label; // destroys t2, then destroys t1, then jumps to label
        } catch (...) {  } // catches the exception from the destructor of t2
    } catch (...) {  } // catches the exception from the destructor of t1
```

Keywords

`try`, `catch`, `throw`

Example

The following example demonstrates several usage cases of the `try-catch` block

Run this code

```cpp
#include <iostream>
#include <vector>

int main() {
    try {
        std::cout << "Throwing an integer exception...\n";
        throw 42;
    } catch (int i) {
        std::cout << " the integer exception was caught, with value: " << i << '\n';
    }

    try {
        std::cout << "Creating a vector of size 5... \n";
        std::vector<int> v(5);
        std::cout << "Accessing the 11th element of the vector...\n";
        std::cout << v.at(10); // vector::at() throws std::out_of_range
    } catch (const std::exception& e) { // caught by reference to base
        std::cout << " a standard exception was caught, with message '"
                  << e.what() << "'\n";
    }

}
```

Possible output:

```
Throwing an integer exception...
 the integer exception was caught, with value: 42
Creating a vector of size 5...
Accessing the 11th element of the vector...
 a standard exception was caught, with message 'out_of_range'
```

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=cpp/language/try_catch&oldid=110472"