【C++】Day14(3)

Class	C++
Date	@December 2, 2021
Material	
Series Number	
Summary	

【Ch5】Exception Handling

5.6 try Blocks and Exception Handling

Exceptions are run-time anomalies-such as losing a database connection or encountering unexpected input-that exist outside the normal functioning of a program. Dealing with anomalous behavior can be one of the most difficult part of designing any system.

Exception handling is generally used when one part of a program detects a problem that it cannot resolve and the problem is such that the detecting part of the program cannot continue.

In such cases, the detecting part needs a way to signal that something happened and that it cannot continue. Moreover, the detecting part needs a way to signal the problem without knowing what part of the program will deal with the exceptional condition.

Exception handling supports this cooperation between the detecting and handling parts of a program. In C++, exception handling involves:

- throw expressions: Which the detecting part uses to indicate that it encountered something it can't handle. We say that a throw raises an exception
- try blocks: Which the handling part uses to deal with an exception. A try block starts with the keyword try and end with one or more catch clauses.

- A set of exception classes that are used to pass information about what happened between a throw and an associated catch.

5.6.1 A throw Expression

The detecting part of a program uses a throw expression to raise an exception. The type of expression determines what kind of exception is thrown.

A throw expression is usually followed by a semicolon, making it into an expression statement.

Recall our `Sales_item` program, the part that adds the objects might be separated from the part that manages the interaction with a user. In this case, we might rewrite the test to throw an exception:

```
if (item1.isbn() != item2.isbn())
  throw runtime_error("Data must refer to the same ISBN");
//if we are here, the ISBN's the same
cout << item1 + item2 << endl
```

Throwing an exception terminates the current function and transfers control to a handler that will known how to handle this error.

5.6.2 The try Block

The general form of a try block is:

```
try {
  program-statements
} catch (exception-declaration) {
  handler-statements
} catch (exception-declaration) {
  handler-statements
}
```

A try block begins with the keyword try followed by a block, which is a sequence of statements enclosed in curly braces.

Following the try block is a list of one or more catch clauses. A catch consists of three parts: the keyword catch, the declaration of a (possibly unnamed) object within parentheses(referred to as an exception declaration), and a block.

When a catch is selected to handle an exception, the associated block is executed. Once the catch finishes, execution continues with the statement immediately following the last catch clause of the try block.

Note: *Variables declared inside a try block are inaccessible outsie the block-in particular, they are not accessible to the catch clause.*

Writing a Handler

```
int a, b;
cin >> a >> b;
try {
  if (a != b)
    throw runtime_error("Input doesn't match");
} catch (runtime_error err) {
  cout << err.what() << "\nWrong Input";
}
```

what is a member function of the runtime-error class. Each of the library exception class defines a member function named `what`.

These functions take no arguments are return a C-style character string(i.e a const char*). The what member of `runtime_error` returns a copy of the string used to initialize the particular object.

In complicated systems, the execution path of a program may pass through multiple try blocks before encountering code that throws an exception. The search for a handler reverses the call chain. It searches in the current function, if no matching catch is found, that function terminates. The function that called the one that threw is searched next.

If no appropriate catch is found, execution is transferred to a library function named terminate. The behaviour of that function is system dependent but is guaranteed to stop further execution of the program.