Object-oriented programming CS10003

Lecturer: Do Nguyen Kha

Contents

Exception

Exceptions provide a way to react to exceptional circumstances (like runtime errors) in programs by transferring control to special functions called handlers.

An exception is thrown by using the throw keyword from inside the try block. Exception handlers are declared with the keyword catch, which must be placed immediately after the try block:

try-throw-catch

Exception handling in C++ consist of three keywords: `try`, `throw` and `catch`:

- The `try` statement allows you to define a block of code to be tested for errors while it is being executed.
- The `throw` keyword throws an exception when a problem is detected, which lets us create a custom error.
- The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

try-throw-catch

```
try {
 // Block of code to try
 throw exception; // Throw an exception when a problem
arise
}
catch () {
 // Block of code to handle errors
}
```

try-throw-catch

```
#include <iostream>
using namespace std;
int main () {
 try {
   throw 20;
 } catch (int e) {
   cout << "An exception occurred. Exception Nr. " << e << '\n';
 }
 return 0;
}
```

Multiple `catch`

Multiple handlers (catch) can be chained; each one with a different parameter type. Only the handler whose argument type matches the type of the exception specified in the throw statement is executed.

Handle any type of exceptions

If an ellipsis `(...)` is used as the parameter of catch, that handler will catch any exception no matter what the type of the exception thrown. This can be used as a default handler that catches all exceptions not caught by other handlers:

```
try {
 // code here
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

Exception specification

```
double myfunction (char param) throw (int);
```

This declares a function called myfunction, which takes one argument of type char and returns a value of type double. If this function throws an exception of some type other than int, the function calls `std::unexpected` instead of looking for a handler or calling `std::terminate`.

```
int myfunction (int param) throw(); // all exceptions call
unexpected
int myfunction (int param); // normal exception handling
```

Standard exceptions

The C++ Standard library provides a base class specifically designed to declare objects to be thrown as exceptions. It is called `std::exception` and is defined in the `<exception>` header. This class has a virtual member function called what that returns a null-terminated character sequence (of type `char *`) and that can be overwritten in derived classes to contain some sort of description of the exception.

Standard exceptions

```
#include <iostream>
#include <exception>
using namespace std;
class myexception: public exception {
 virtual const char* what() const throw() {
   return "My exception happened";
 }
} myex;
int main () {
 try {
   throw myex;
 } catch (exception& e) {
   cout << e.what() << '\n';
 }
 return 0;
}
```

Standard exceptions

exception	**description**
`bad_alloc`	thrown by `new` on allocation failure
`bad_cast`	thrown by `dynamic_cast` when it fails in a dynamic cast
`bad_exception`	thrown by certain dynamic exception specifiers
`bad_typeid`	thrown by `typeid`
`bad_function_call`	thrown by empty `function` objects
`bad_weak_ptr`	thrown by `shared_ptr` when passed a bad `weak_ptr`

Standard exceptions

exception	**description**
`logic_error`	error related to the internal logic of the program
`runtime_error`	error detected during runtime

Team Project: Extensions

- Adding new property
- Adding Path
- Group and Transformation