Exception Handling Fundamentals

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an **exception** (throw an error).

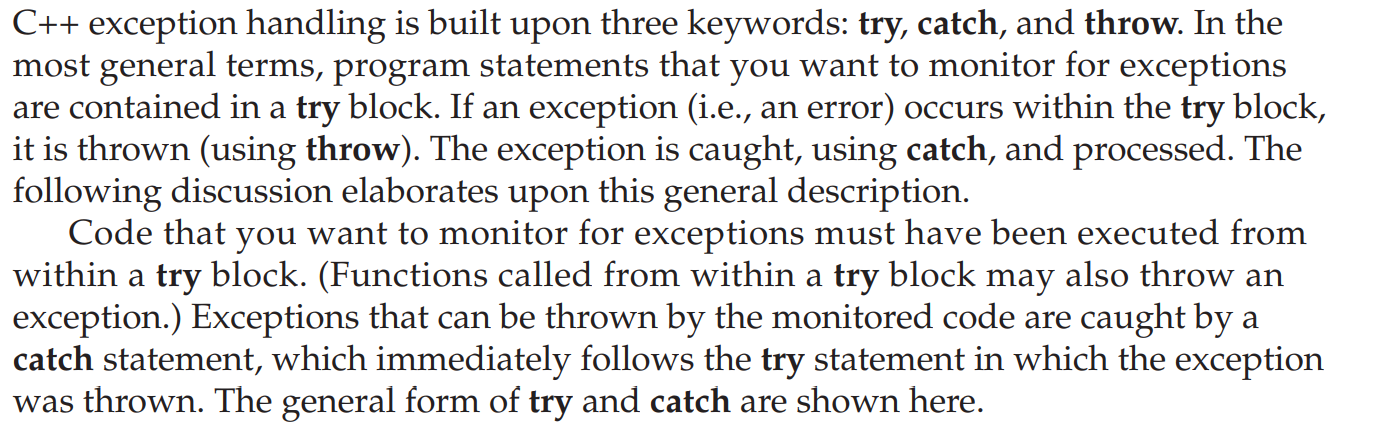
## C++ try and catch

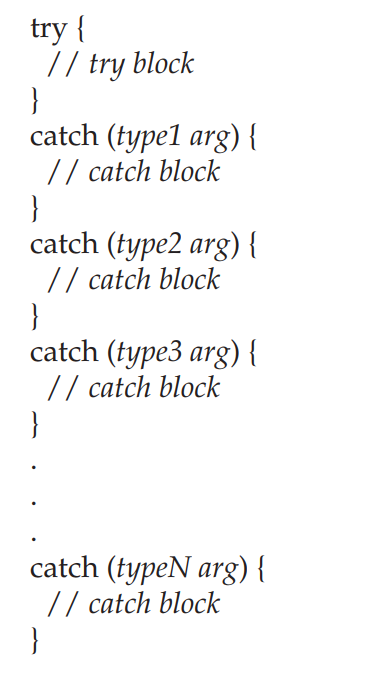
Exception handling in C++ consist of three keywords: try, throw and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed.

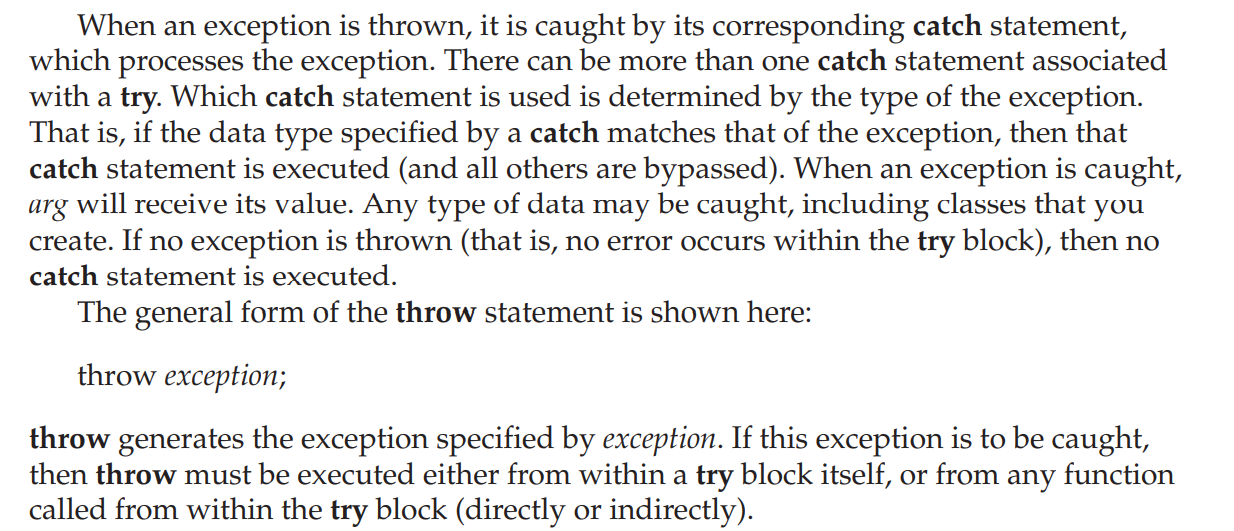
The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.





The try can be as short as a few statements within one function or as all, encompassing as enclosing the main( ) function code within a try block (which effectively causes the entire program to be monitored).



If you throw an exception for which there is no applicable catch statement, an abnormal program termination may occur. Throwing an unhandled exception causes the standard library function terminate( ) to be invoked. By default, terminate( ) calls abort( ) to stop your program, but you can specify your own termination handler,

// A simple exception handling example.

#include <iostream>

using namespace std;

int main()

{

cout << "Start\n";

try { // start a try block

cout << "Inside try block\n";

throw 100; // throw an error

cout << "This will not execute";

}

catch (int i) { // catch an error

cout << "Caught an exception -- value is: ";

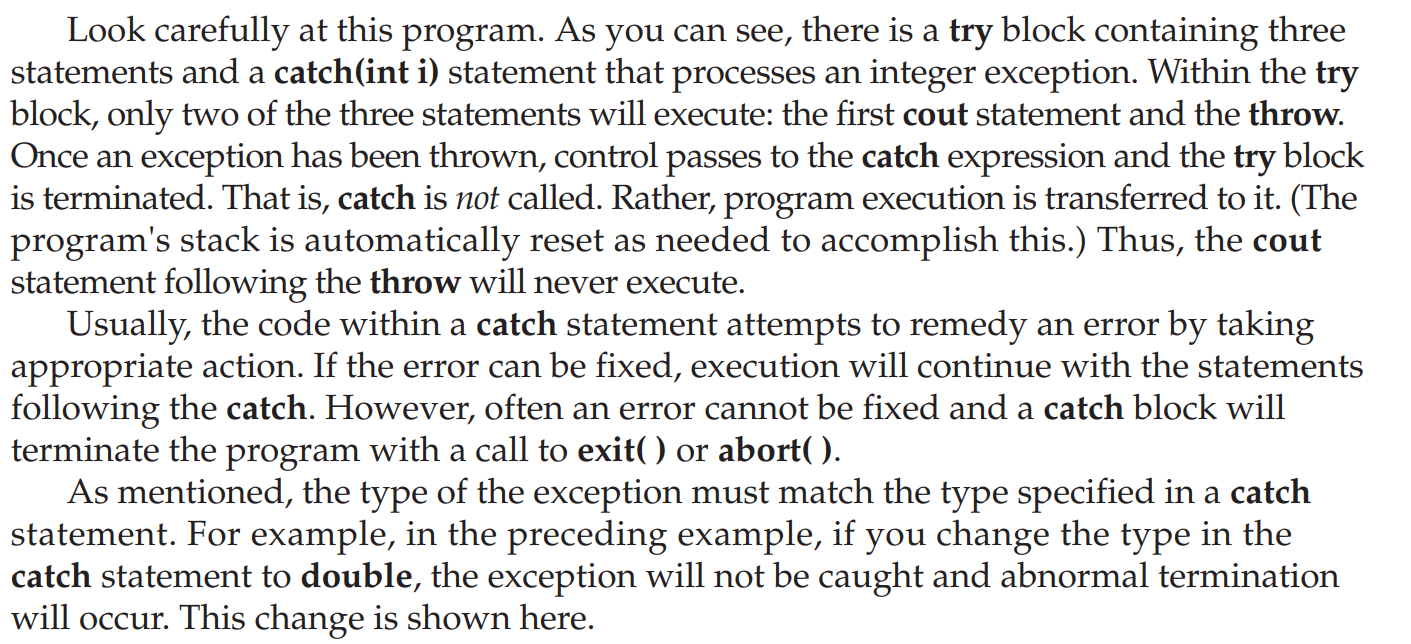
cout << i << "\n";

}

cout << "End";

return 0;

}



// This example will not work.

#include <iostream>

using namespace std;

int main()

{

cout << "Start\n";

try { // start a try block

cout << "Inside try block\n";

throw 100; // throw an error

cout << "This will not execute";

}

catch (double i) { // won't work for an int exception

cout << "Caught an exception -- value is: ";

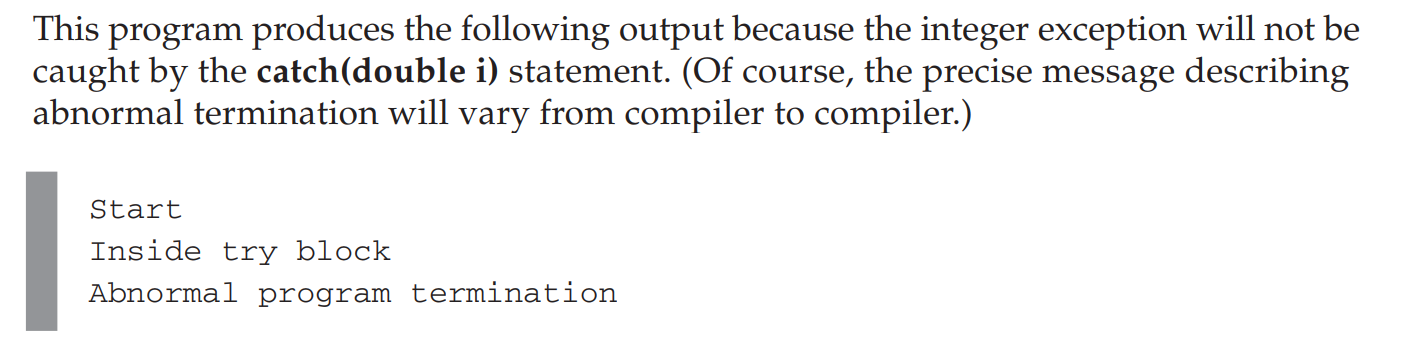
cout << i << "\n";

}

cout << "End";

return 0;

}



Using Multiple catch Statements

As stated, you can have more than one catch associated with a try. In fact, it is common to do so. However, each catch must catch a different type of exception. For example, this program catches both integers and strings.

#include <iostream>

using namespace std;

// Different types of exceptions can be caught.

void Xhandler(int test)

{

try{

if(test) throw test;

else throw "Value is zero";

}

catch(int i) {

cout << "Caught Exception #: " << i << '\n';

}

catch(const char \*str) {

cout << "Caught a string: ";

cout << str << '\n';

}

}

int main()

{

cout << "Start\n";

Xhandler(1);

Xhandler(2);

Xhandler(0);

Xhandler(3);

cout << "End";

return 0;

}

Practice programs

// This example uses catch(...) as a default.

#include <iostream>

using namespace std;

void Xhandler(int test)

{

try{

if(test==0) throw test; // throw int

if(test==1) throw 'a'; // throw char

if(test==2) throw 123.23; // throw double

}

catch(int i) { // catch an int exception

cout << "Caught an integer\n";

}

catch(...) { // catch all other exceptions

cout << "Caught One!\n";

}

}

int main()

{

cout << "Start\n";

Chapter 19: Exception Handling 499 C++Xhandler(0);

Xhandler(1);

Xhandler(2);

cout << "End";

return 0;

}

// Restricting function throw types.

#include <iostream>

using namespace std;

// This function can only throw ints, chars, and doubles.

void Xhandler(int test) throw(int, char, double)

{

if(test==0) throw test; // throw int

if(test==1) throw 'a'; // throw char

if(test==2) throw 123.23; // throw double

}

int main()

{

cout << "start\n";

try{

Xhandler(0); // also, try passing 1 and 2 to Xhandler()

}

catch(int i) {

cout << "Caught an integer\n";

}

catch(char c) {

cout << "Caught char\n";

}

catch(double d) {

cout << "Caught double\n";

}

cout << "end";

return 0;

}

