Exception Handling

Exception in C++

- An exception is an error that occurs at run time.
- When exception handling is employed, your program automatically invokes an error-handling routine when an exception occurs.
- C++ exception handling is built upon three keywords: **try**, **catch**, and **throw**.
  - Program statements that you want to monitor for exceptions are contained in a **try** block
  - If an exception (i.e., an error) occurs within the **try** block, it is thrown (using **throw**).
  - The exception is caught, using **catch**, and processed.

General form of **try** and **catch**

The general
form of **try** and **catch** are shown here:

```
try {
// try block
}
catch (type1 arg) {
// catch block
}
catch (type2 arg) {
// catch block
}
catch (type3 arg) {
// catch block
}
// ...
catch (typeN arg) {
// catch block
}
```

- When an exception is thrown, it is caught by its corresponding **catch** statement, which then processes the exception.
- There can be more than one **catch** statement associate with a **try**. The type of the exception determines which **catch** statement is used.
- If the data type specified by a **catch** statement matches that of the exception, then that **catch** statement is executed (and all others are bypassed).

example that shows how C++ exception handling

Here is a very simple example that shows how C++ exception handling operates:

// 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 99; // 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;
}
```

- Within the **try** block, only two of the three statements will execute: the first **cout** statement and the **throw**. Once an exception has been thrown, control passes to the **catch** expression, and the **try** block is terminated.
- After the **catch** statement executes, program control continues with the statements following the **catch**.
- In cases where the error cannot be fixed, a **catch** block will usually end with a call to **exit( )**

This program displays the following output:

```
start
Inside try block
Caught an exception — value is: 99
end
```

Example

```
// 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 99; // 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;
}
```

This program produces the following output, because the integer exception will not be caught by the **catch(double i)** statement:

```
start
Inside try block
Abnormal program termination
```

An exception

An exception thrown by a function called from within a **try** block can be caught by that **try** block. For example, this is a valid program:

```
#include <iostream>
using namespace std;
void Xtest(int test)
{
cout << "Inside Xtest, test is: " << test <<
"\n";
if(test) throw test;
}
int main()
{
cout << "start\n";
try { // start a try block
cout << "Inside try block\n";
Xtest(0);
Xtest(1);
Xtest(2);
}
catch (int i) { // catch an error
cout << "Caught an exception -- value is: ";
cout << i << "\n";
}
cout << "end";
return 0;
```

This program produces the following output:

```
start
Inside try block
Inside Xtest, test is: 0
Inside Xtest, test is: 1
Caught an exception – value is: 1
end
```

A **try** block localized to a function.

- A **try** block can be localized to a function.
- When this is the case, each time the function is entered, the exception handling relative to that function is reset.

```
#include <iostream>
using namespace std;
// A try/catch is reset each time a
function is entered.
void Xhandler(int test)
{
try{
if(test) throw test;
}
catch(int i) {
cout << "Caught One! Ex. #: " << i <<
'\n';
}
}
int main()
{
cout << "start\n";
Xhandler(1);
Xhandler(2);
Xhandler(0);
Xhandler(3);
cout << "end";
return 0;
}
```

This program displays the following output:

```
start
Caught One! Ex. #: 1
Caught One! Ex. #: 2
Caught One! Ex. #: 3
end
```

Catching Class Types

- To create an object that describes the error that occurred.
- This information can be used by the exception handler to help it process the error.

```
// Use an exception class.
#include <iostream>
#include <cstring>
using namespace std;
class MyException {
public:
char str_what[80];
MyException() { *str_what = 0; }
MyException(char *s) {
strcpy(str_what, s);
}
};
int main()
{
int a, b;
try {
cout << "Enter numerator and denominator: ";
cin >> a >> b;
if(!b)
throw MyException("Cannot divide by zero!");
else
cout << "Quotient is " << a/b << "\n";
}
catch (MyException e) { // catch an error
cout << e.str_what << "\n";
}
return 0;
}
```

Here is a sample run:

```
Enter numerator and denominator: 10 0
Cannot divide by zero!
```

Using Multiple catch Statements

- You can associate more than one **catch** statement with a **try**.
- However, each **catch** must catch a different type of exception.

```
#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 One! Ex. #: " << i << '\n';
}
catch(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;
}
```

This program produces the following output:

```
start
Caught One! Ex. #: 1
Caught One! Ex. #: 2
Caught a string: Value is zero
Caught One! Ex. #: 3
end
```

Catching Base Class Exceptions

- A **catch** clause for a base class will also match any class derived from that
- base.
- Thus, if you want to catch exceptions of both a base class type and a derived
- class type, put the derived class first in the **catch** sequence.
- If you don’t, then the base class **catch** will also catch all derived classes.

```
// Catching derived classes.
#include <iostream>
using namespace std;
class B {
};
class D: public B {
};
int main()
{
D derived;
try {
throw derived;
}
catch(B b) {
cout << "Caught a base class.\n";
}
catch(D d) {
cout << "This won't execute.\n";
}
return 0;
}
```

Catching All Exceptions

- In some circumstances, you will want an exception handler to catch all exceptions,
- Instead of just a certain type.

```
// This example catches all exceptions.
#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(...) { // catch all exceptions
cout << "Caught One!\n";
}
}
int main()
{
cout << "start\n";
Xhandler(0);
Xhandler(1);
Xhandler(2);
cout << "end";
return 0;
}
```

This program displays the following output:

```
start
Caught One!
Caught One!
Caught One!
end
```

Using catch(...)

- One very good use for **catch(...)** is as the last **catch** of a cluster of catches.

```
// 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 " << i << '\n';
}
catch(...) { // catch all other exceptions
cout << "Caught One!\n";
}
}
int main()
{
cout << "start\n";
Xhandler(0);
Xhandler(1);
Xhandler(2);
cout << "end";
return 0;
}
```

The output produced by this program is shown here:

```
start
Caught 0
Caught One!
Caught One!
end
```

Restricting Exceptions Thrown by a Function

- You must add a **throw** clause to a function definition.
- The general form of this clause is

  ret-type func-name(*arg-list*)
  throw(*type-list*)
  {
  // ...
  }
- Here, only those data types contained in the comma-separated *type-list* can be thrown by the function.

```
// 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 int\n";
}
catch(char c) {
cout << "Caught char\n";
}
catch(double d) {
cout << "Caught double\n";
}
cout << "end";
return 0;
}
```