Exception Handling

Dr. Alekha Kumar Mishra

What is exception handling?

- It provides a mechanism to decouple handling of errors or other exceptional circumstances from the typical control flow of our code.
- Provides more freedom to handle errors.
- Enable returning codes cause whenever required.

Basics of exception handling

- Using three keywords that works in conjunction with each other: **throw, try, and catch**.
- In C++, a **throw** statement is used to signal that an exception or error case has occurred.
  - It is also commonly called raising an exception.
- Using a **throw** statement
  - throw *val;*
- Typically, this value will be an error code, a description of the problem, or a custom exception class.

throw Examples

- // throw a literal integer value
  - throw -1;
- // throw an enum value
  - throw ENUM_INVALID_INDEX;
- // throw a literal char* string
  - throw "Can not take square root of negative number";
- // throw a double variable that was previously defined
  - throw dX;
- // Throw an object of class MyException
  - throw MyException("Fatal Error");
- Each of these statements acts a signal that some kind of problem that needs to be handled has occurred.

Looking for exception

- Try keyword is used to define a block of statements where exception might get raised.
- The try block acts as an observer, looking for any exceptions that are thrown by statements within the try block.
- Note: try block doesn't define HOW we're going to handle the exception.

```
try {
/* Statements that may throw exceptions you want to handle now go here */
    throw -1;
}
```

Handling exceptions

- Handling exceptions is actually the job of the catch block(s).
- The **catch** keyword is used to define a block of code (called a catch block) that handles exceptions for a single data type.
- Try blocks and catch blocks work together
  - A try block detects any exceptions that are thrown by statements within the try block, and routes them to the appropriate catch block for handling.
- A try block must have at least one catch block attached to it, but may have multiple catch blocks.

```
catch (int) {
    // Handle an exception of type int here
    cerr << "We caught an exception of type int" << endl;
}
```

An example

```
int main(){
    try {
        // Statements that may throw
      //exceptions you want to handle now go here
        throw -1;
    }
    catch (int){
    // Any exceptions of type int thrown
   //within the above try block get sent here
        cerr << "We caught an exception of type int" << endl;
    }
    catch (double){
      // Any exceptions of type double thrown
      //within the above try block get sent here
        cerr << "We caught an exception of type double" << endl;
    }
     return 0;
}
```

Output:

We caught an exception of type int

How exception works?

- When an exception is raised (using throw), execution of the program immediately jumps to the nearest enclosing try block.
- If any of the catch handlers attached to the try block handle that type of exception, that handler is executed.
- If no appropriate catch handlers exist, execution of the program propagates to the next enclosing try block.
- If no appropriate catch handlers can be found before the end of the program, the program will fail with an exception error.

Example

```
int main() {
    try {
        throw 4.5; // throw exception of type double
        cout << "This never prints" << endl;
    }
    catch(double dX) // handle exception of type double {
        cerr << "We caught a double of value: " << dX << endl;
    }
}
```

Output:

We caught a double of value: 4.5

Example(2)

```
int main() {
      cout << "Enter a number: ";
      double dX;
      cin >> dX;
      try  {
        // If the user entered a negative number
        //, this is an error condition
           if (dX < 0.0)
                throw "Can not take sqrt of negative number";
                cout << "The sqrt of " << dX << " is " << sqrt(dX) << endl;
    }
    catch (char* strException) {
        cerr << "Error: " << strException << endl;
    }
}
```

Nested try blocks

```
int main ()
{
    try
    {
        try
        {
            throw 1.0;
        }
        catch (int x)
        {
            cout << "Exception int type";
        }
    }
    catch (double x)
    {
        cout << "Exception double type";
    }
}// end of main
```

Exception inside a function

```
double MySqrt(double dX) {
    if (dX < 0.0)
        throw "Can not take sqrt of negative number";
     // throw exception of type char*
     return sqrt(dX);
}
int main()  {
    cout << "Enter a number: ";
    double dX;
    cin >> dX;
     try {
        cout << "The sqrt of " << dX << " is " << MySqrt(dX) << endl;
     }
    catch (const char* strException) // catch exceptions of type char* {
        cerr << "Error: " << strException << endl;
    }
}
```

Uncaught exception

```
double MySqrt(double dX){
        if (dX < 0.0)
        throw "Can not take sqrt of negative number";
        return sqrt(dX);
}
 int main() {
    cout << "Enter a number: ";
    double dX;
    cin >> dX;
     // Look, no exception handler!
    cout << "The sqrt of " << dX << " is " << MySqrt(dX) << endl;
}
```

Explanation

- When a function doesn't handle the exception, so the program stack unwinds and control returns to the function where it is called.
- If there's no exception handler here either, it returns to previous function and this continues till it reaches main function().
- If main() either dont have a handler, it terminates.
- When main() terminates with an unhandled exception, the operating system will generally notify us that an unhandled exception error has occurred.
- It depends on the operating system how to handle this error
- Possibilities
  - printing an error message
  - popping up an error dialog
  - Simply crashing. (avoid this case)

catch all handler

- C++ provides us with a mechanism to catch all types of exceptions.
- This is known as a catch-all handler.
- A catch-all handler works just like a normal catch block, except that instead of using a specific type to catch, it uses the ellipses operator (…) as the type to catch.

```
try {
    throw 5.5; // throw an int exception
}
catch (double dX){
    cout << "We caught an exception of type double: " << dX << endl;
}
catch (...) // catch-all handler {
    cout << "We caught an exception of an undetermined type" << endl;
}
```

catch all handler

- The catch-all handler should be placed last in the catch block chain.
- This is to ensure that exceptions can be caught by exception handlers tailored to specific data types if those handlers exist.
- Often, the catch-all handler block is left empty:
  - catch(...) {} // ignore any unanticipated exceptions

Restricting Exceptions

- We can restrict the type of exceptions that a function can throw outside of itself
- ret-type func-name(arg-list) throw(type-list){

  // function body

  }

```
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
}
```

terminate() & unexpected()

- Throwing an unhandled exception causes the standard library function **terminate()** to be invoked.
- Attempting to throw an exception that is not supported by a function will cause the standard library function **unexpected( )** to be called.
- By default, terminate( ) and unexpected() calls **abort()** to stop your program, but you can specify your own termination handler

A terminate handler

```
// Set a new terminate handler.
#include <iostream>
#include <cstdlib>
#include <exception>
using namespace std;
void my_Thandler() {
cout << "Inside new terminate handler\n";
abort();
}
int main() {
      // set a new terminate handler
      set_terminate(my_Thandler);
      try {
            cout << "Inside try block\n";
            throw 100; // throw an error
      }
      catch (double i) { // won't catch an int exception
            // ...
      }
return 0;
}
```

Rethrowing an exception

```
int main ()
{
    try
    {
        try
        {
            throw 1;
        }
        catch (int x)
        {
            //line1 //line2
            throw x;
        }
    }
    catch (...)
    {
        cout << "Exception occurred";
    }
}// end of main
```

End of exception