Other *catch* Blocks

If your function may throw more than one exception, add cascading `catch` blocks following the **`try`** block, each designed to handle a different type of exception, like this:

```
try {
    int n = stoi(str);    // may throw an invalid_argument exception
    int x = str.at(5);    // may throw an out_of_range exception
    // ... other statements that may throw exceptions
}
catch (const invalid_argument& e) {
    // handle errors from stoi
}
case (const out_of_range& e) {
    // handle errors from at()
}
case (...) {
    // handle any exceptions not previously caught
}
```

The last block, with the **`...`** in the argument list is the **catch all** handler. It catches**any exceptions** thrown in the **`try`** block, **not previously caught.** The **catch all** hander **only** catches thrown exceptions, not other errors like segmentation faults or **operating system traps or signals** such as those caused by dividing by zero. Code jumps to **only one** of the **`catch`** blocks shown here. If no exceptions are thrown, then no **`catch`** blocks are entered.

Finish the Sample

After adding **`try-catch`** to **`main()`**, print an error message inside the **`catch`** block. Use **`cerr`**, print the word *"Error: "* and then call **`e.what()`** like this:

```
catch (const invalid_argument& e) {
    cerr << "Error: " << e.what() << endl;
}
cout << "--program done--" << endl;
```

Now your program should work the same whether compiled with C++17 or C++98 (even if the error messages differ between versions.)