1 Exceptions

One of the language features that attempts to add a degree of safety is *exceptions*. An exception is an unexcepted situation. When a piece of code detects an exceptional situation, it *throws* an exception. Another piece of code *catches* the exception and takes appropriate action.

```
#include <stdexcept>
double divideNumbers(double numerator, double denominator)
{
   if (denominator == 0) {
       throw std::invalid_argument("Denominator cannot be 0.");
   }
   return numerator / denominator;
}
```

When the `throw` line is executed, the function immediately ends without returning a value. If the caller surrounds the function call with a `try/catch` block, it receives the exception and is able to handle it.

```
int main()
{
   try {
       std::cout << divideNumbers(2.5,0.5) << std::endl;
       std::cout << divideNumbers(2.3,0) << std::endl;
       std::cout << divideNumbers(4.5,2.5) << std::endl;
   } catch (const std::exception& exception) {
       std::cout << "Exception caught: " << exception.what() << std::endl;
   }
   return 0;
}
```

The first call to `divideNumbers()` executes successfully, and the result is output to the user. The second call throws an execption. No value is returned, and the only output is the error message that is printed when the exception is caught. The third cal is never executed. The output is then

```
5
An exception was caught: Denominator cannot be 0.
```

The preceding example used the built-in std::invalid_argument type, but it is preferable to write your own exception types that are more specific to the error being thrown.