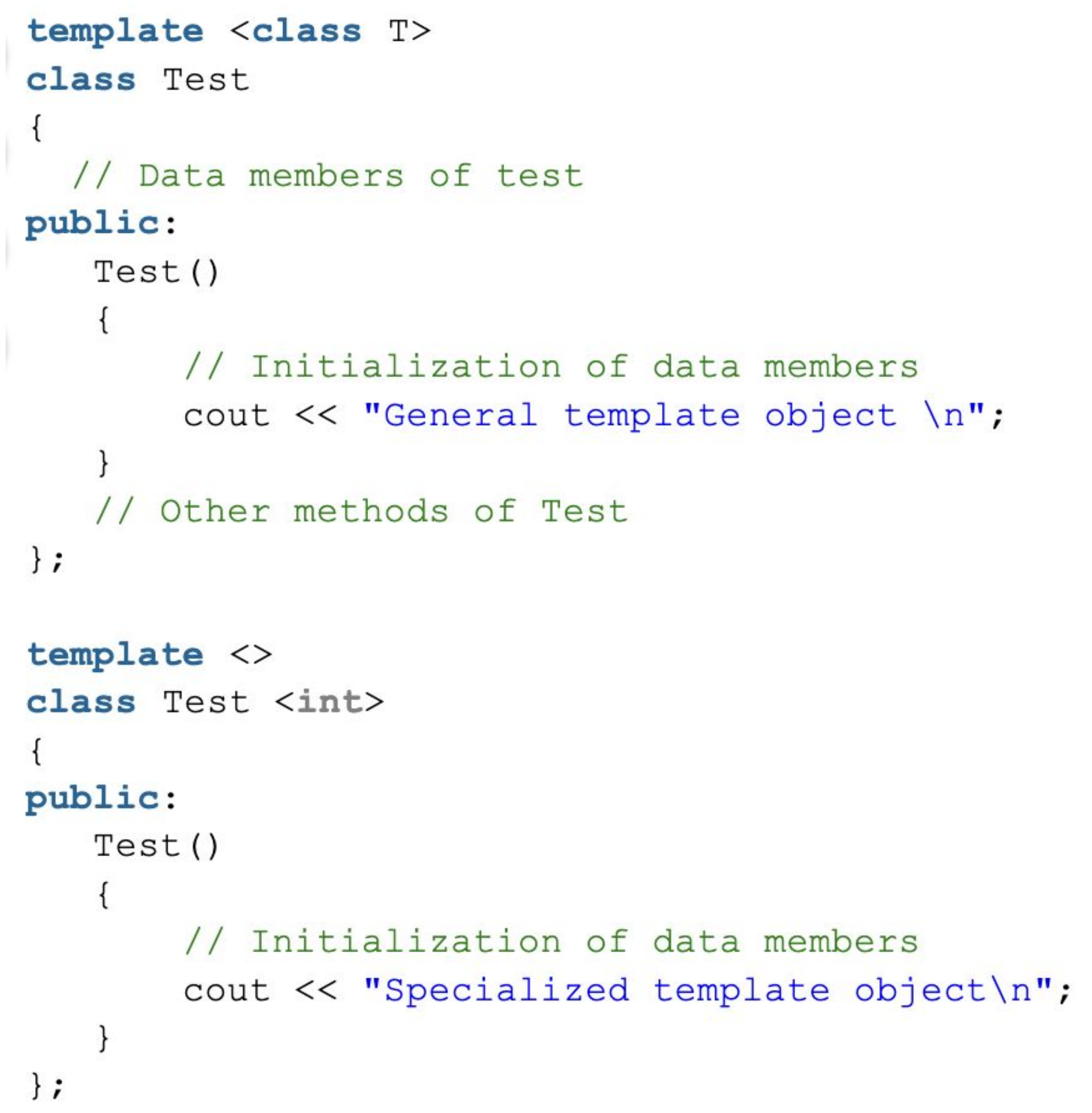
Templates
Exception Handling

Template Specialization

```
int main()
{
    Test<int> a;
    Test<char> b;
    Test<float> c;
    return 0;
}
```

Output:

```
Specialized template object
General template object
General template object
```



```
template <class T>
class Test
{
  // Data members of test
public:
   Test()
   {
       // Initialization of data members
       cout << "General template object \n";
   }
   // Other methods of Test
};

template <>
class Test <int>
{
public:
   Test()
   {
       // Initialization of data members
       cout << "Specialized template object\n";
   }
};
```

Exceptions

- Indicate that something unexpected has occurred or been detected
- Allow program to deal with the problem in a controlled manner
- Can be as simple or complex as program design requires

Exceptions - Terminology

- Exception: object or value that signals an error
- Throw an exception: send a signal that an error has occurred
- Catch/Handle an exception: process the exception; interpret the signal

Exceptions – Key Words

- `throw` – followed by an argument, is used to throw an exception
- `try` – followed by a block `{ }`, is used to invoke code that throws an exception
- `catch` – followed by a block `{ }`, is used to detect and process exceptions thrown in preceding `try` block. Takes a parameter that matches the type thrown.

Exceptions – Flow of Control

1) A function that throws an exception is called from within a try block
2) If the function throws an exception, the function terminates and the try block is immediately exited. A catch block to process the exception is searched for in the source code immediately following the try block.
3) If a catch block is found that matches the exception thrown, it is executed. If no catch block that matches the exception is found, the program terminates.

Example

```
try {
  // Block of code to try
  throw exception; // Throw an exception when a
problem arise
}
catch () {
  // Block of code to handle errors
}
```

Example

```
try {
  int age = 15;
  if (age >= 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw (age);
  }
}
catch (int myNum) {
  cout << "Access denied - You must be at least 18 years 
old.\n";
  cout << "Age is: " << myNum;
}
```

Example (Error No.)

```
try {
  int age = 15;
  if (age >= 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw 505;
  }
}
catch (int myNum) {
  cout << "Access denied - You must be at least 18 years
old.\n";
  cout << "Error number: " << myNum;
}
```

Handle Any Type of Exceptions (...)

- If you do not know the throw **type** used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception:

```
try {
  int age = 15;
  if (age >= 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw 505;
  }
}
catch (...) {
  cout << "Access denied - You must be at least 18 years
old.\n";
}
```

Multiple Catch Blocks

```
try {
// protected code
}
catch( ExceptionName e1 ) {// catch block}
catch( ExceptionName e2 ) {// catch block}
catch( ExceptionName eN ) {// catch block}
```

Exceptions – Example (1)

```
// function that throws an exception
int totalDays(int days, int weeks)
{
 if ((days < 0) || (days > 7))
   throw "invalid number of days";
// the argument to throw is the
// character string
  else
   return (7 * weeks + days);
}
```

Exceptions – Example (2)

```
try // block that calls function
{
   totDays = totalDays(days, weeks);
     cout << "Total days: " << days;
 }
 catch (char *msg) // interpret
     //  exception
 {
     cout << "Error: " << msg;
 }
```

Exceptions – What Happens

1) `try` block is entered. `totalDays` function is called
2) If 1st parameter is between 0 and 7, total number of days is returned and `catch` block is skipped over (no exception thrown)
3) If exception is thrown, function and `try` block are exited, `catch` blocks are scanned for 1st one that matches the data type of the thrown exception. `catch` block executes

```
int main()
{
   int num1, num2;  // To hold two numbers
   double quotient; // To hold the quotient of the numbers

   // Get two numbers.
   cout << "Enter two numbers: ";
   cin >> num1 >> num2;

   // Divide num1 by num2 and catch any
   // potential exceptions.
   try
   {
      quotient = divide(num1, num2);
      cout << "The quotient is " << quotient << endl;
   }
   catch (char *exceptionString)
   {
      cout << exceptionString;
   }

   cout << "End of the program.\n";
   return 0;
}
```

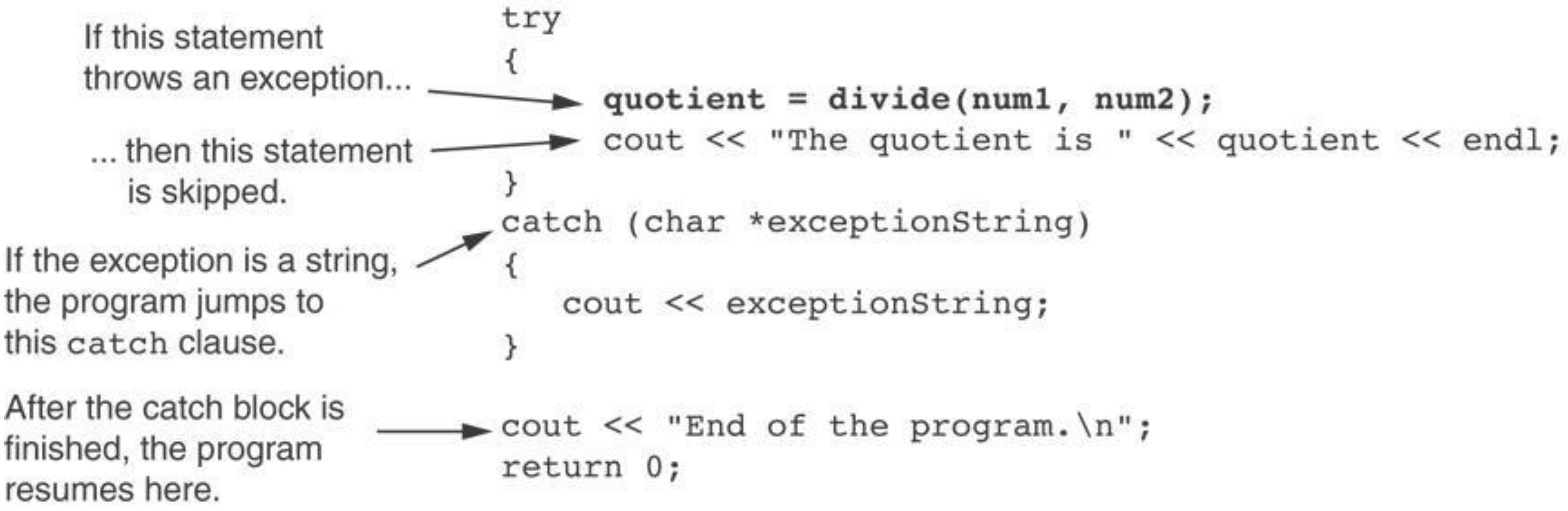
```cpp
//**********************************************
// The divide function divides numerator by   *
// denominator. If denominator is zero, the   *
// function throws an exception.              *
//**********************************************

double divide(int numerator, int denominator)
{
   if (denominator == 0)
      throw "ERROR: Cannot divide by zero.\n";

   return static_cast<double>(numerator) / denominator;
}
```

Program Output with Example Input Shown in Bold

```
Enter two numbers: 12 2 [Enter]
The quotient is 6
End of the program.
```

Program Output with Example Input Shown in Bold

```
Enter two numbers: 12 0 [Enter]
ERROR: Cannot divide by zero.
End of the program.
```



```
try
{
    quotient = divide(num1, num2);
    cout << "The quotient is " << quotient << endl;
}
catch (char *exceptionString)
{
   cout << exceptionString;
}

cout << "End of the program.\n";
return 0;
```

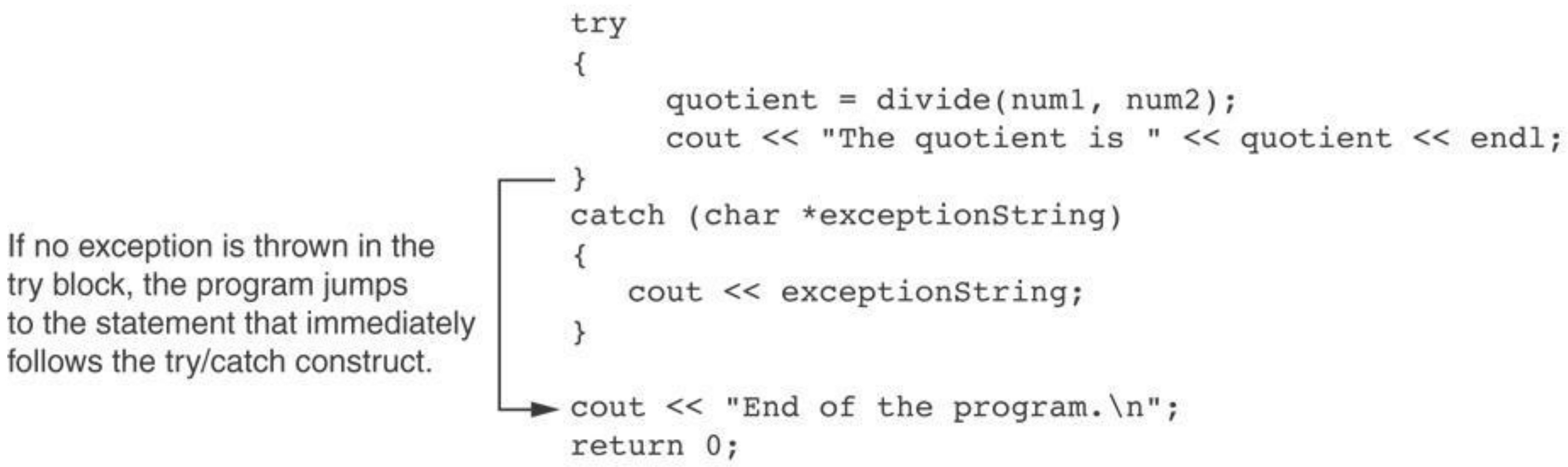
If this statement throws an exception...

... then this statement is skipped.

If the exception is a string, the program jumps to this `catch` clause.

After the catch block is finished, the program resumes here.

What if no exception is thrown?



If no exception is thrown in the try block, the program jumps to the statement that immediately follows the try/catch construct.

```
try
{
     quotient = divide(num1, num2);
     cout << "The quotient is " << quotient << endl;
}
catch (char *exceptionString)
{
   cout << exceptionString;
}

cout << "End of the program.\n";
return 0;
```

Exceptions - Notes

- Predefined functions such as `new` may throw exceptions
- The value that is thrown does not need to be used in `catch` block.
  - in this case, no name is needed in catch parameter definition
  - `catch` block parameter definition *does* need the type of exception being caught

Exception Not Caught?

- An exception will not be caught if
  - it is thrown from outside of a `try` block
  - there is no `catch` block that matches the data type of the thrown exception
- If an exception is not caught, the program will terminate

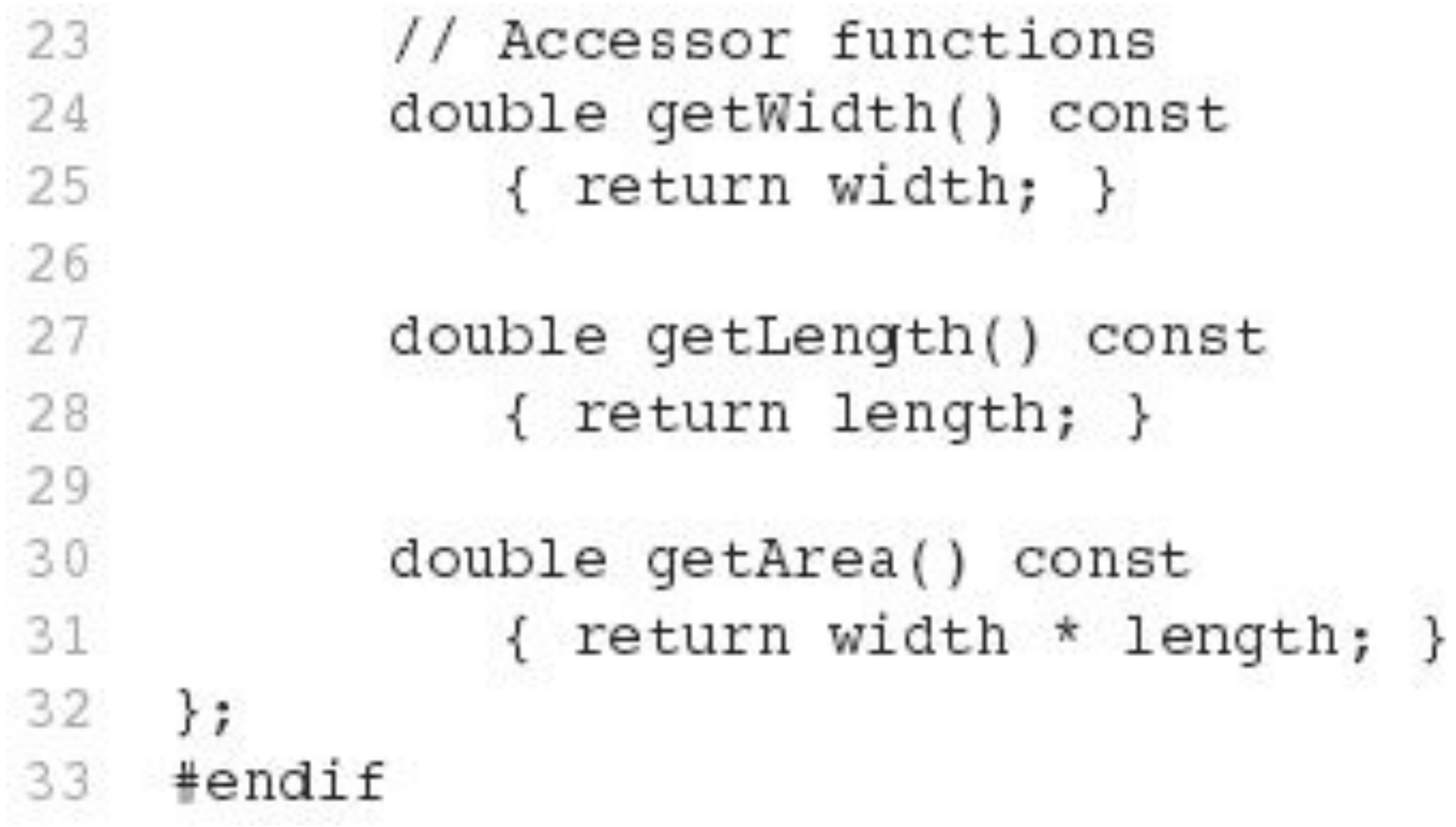
Exceptions and Objects

- An <u>exception class</u> can be defined in a class and thrown as an exception by a member function
- An exception class may have:
  - no members: used only to signal an error
  - members: pass error data to `catch` block
- A class can have more than one exception class

Contents of `Rectangle.h` (Version 1)

```
// Specification file for the Rectangle class
#ifndef RECTANGLE_H
#define RECTANGLE_H

class Rectangle
{
   private:
      double width;     // The rectangle's width
      double length;    // The rectangle's length
   public:
      // Exception class
      class NegativeSize
         { };            // Empty class declaration

      // Default constructor
      Rectangle()
         { width = 0.0; length = 0.0; }

      // Mutator functions, defined in Rectangle.cpp
      void setWidth(double);
      void setLength(double);

```

Contents of Rectangle.h (Version1) (Continued)



```
      // Accessor functions
      double getWidth() const
         { return width; }

      double getLength() const
         { return length; }

      double getArea() const
         { return width * length; }
};
#endif
```

Contents of `Rectangle.cpp` (Version 1)

```
// Implementation file for the Rectangle class.
#include "Rectangle.h"

//************************************************************
// setWidth sets the value of the member variable width.    *
//************************************************************

void Rectangle::setWidth(double w)
{
   if (w >= 0)
      width = w;
   else
      throw NegativeSize();
}

//************************************************************
// setLength sets the value of the member variable length.  *
//************************************************************

void Rectangle::setLength(double len)
{
   if (len >= 0)
      length = len;
   else
      throw NegativeSize();
}
```

Program 16-2

```
// This program demonstrates Rectangle class exceptions.
#include <iostream>
#include "Rectangle.h"
using namespace std;

int main()
{
   int width;
   int length;

   // Create a Rectangle object.
   Rectangle myRectangle;

```

Program 16-2 *(continued)*

```
    // Get the width and length.
    cout << "Enter the rectangle's width: ";
    cin >> width;
    cout << "Enter the rectangle's length: ";
    cin >> length;

    // Store these values in the Rectangle object.
    try
    {
       myRectangle.setWidth(width);
       myRectangle.setLength(length);
       cout << "The area of the rectangle is "
            << myRectangle.getArea() << endl;
    }
    catch (Rectangle::NegativeSize)
    {
       cout << "Error: A negative value was entered.\n";
    }
    cout << "End of the program.\n";

    return 0;
 }
```

Program 16-2 (Continued)

Program Output with Example Input Shown in Bold

```
Enter the rectangle's width: 10 [Enter]
Enter the rectangle's length: 20 [Enter]
The area of the rectangle is 200
End of the program.
```

Program Output with Example Input Shown in Bold

```
Enter the rectangle's width: 5 [Enter]
Enter the rectangle's length: -5 [Enter]
Error: A negative value was entered.
End of the program.
```

What Happens After `catch` Block?

- Once an exception is thrown, the program cannot return to throw point. The function executing `throw` terminates (does not return), other calling functions in `try` block terminate, resulting in <u>unwinding the stack</u>
- If objects were created in the `try` block and an exception is thrown, they are destroyed.

Nested `try` Blocks

- `try/catch` blocks can occur within an enclosing `try` block
- Exceptions caught at an inner level can be passed up to a `catch` block at an outer level:

```
catch ( )
{
    ...
    throw;  // pass exception up
}          // to next level
```