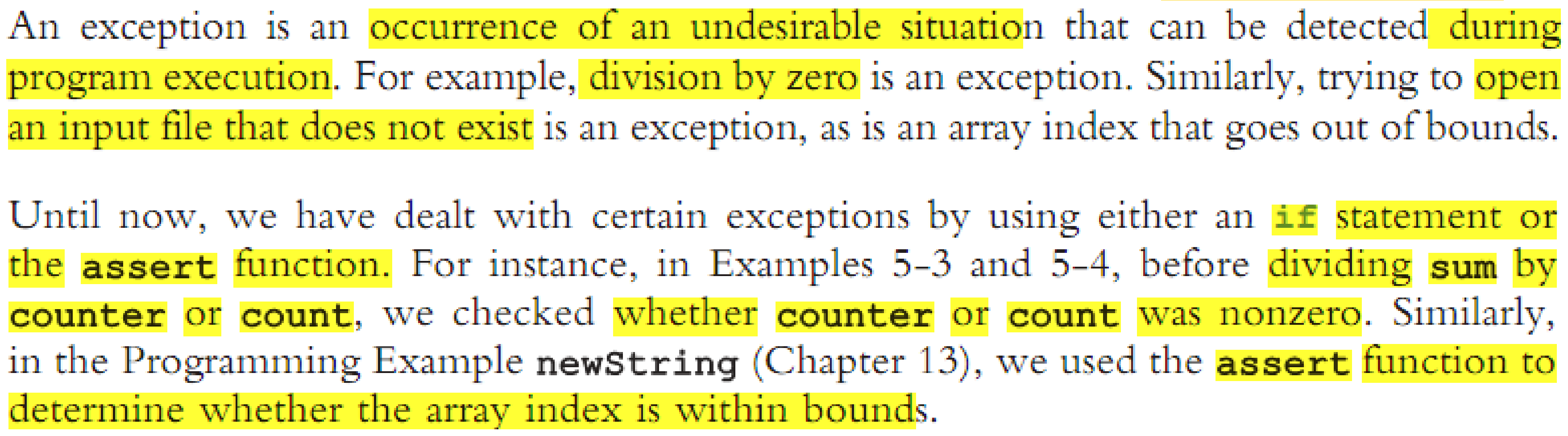
C++ Programming: From Problem Analysis to Program Design, Sixth Edition

Necessary Functions

Exception



An exception is an occurrence of an undesirable situation that can be detected during program execution. For example, division by zero is an exception. Similarly, trying to open an input file that does not exist is an exception, as is an array index that goes out of bounds.

Until now, we have dealt with certain exceptions by using either an `if` statement or the `assert` function. For instance, in Examples 5-3 and 5-4, before dividing `sum` by `counter` or `count`, we checked whether `counter` or `count` was nonzero. Similarly, in the Programming Example `newString` (Chapter 13), we used the `assert` function to determine whether the array index is within bounds.

Handling Exception

The program in Example 14-1 shows what happens when division by zero occurs and the problem is not addressed.

EXAMPLE 14-1

```cpp
// Division by zero.

#include <iostream>

using namespace std;

int main()
{
    int dividend, divisor, quotient;                  //Line 1

    cout << "Line 2: Enter the dividend: ";           //Line 2
    cin >> dividend;                                  //Line 3
    cout << endl;                                     //Line 4

    cout << "Line 5: Enter the divisor: ";            //Line 5
    cin >> divisor;                                   //Line 6
    cout << endl;                                     //Line 7

    quotient = dividend / divisor;                    //Line 8
    cout << "Line 9: Quotient = " << quotient
         << endl;                                     //Line 9

    return 0;                                         //Line 10
}
```

Sample Run 1:

```
Line 2: Enter the dividend: 12

Line 5: Enter the divisor: 5

Line 9: Quotient = 2
```

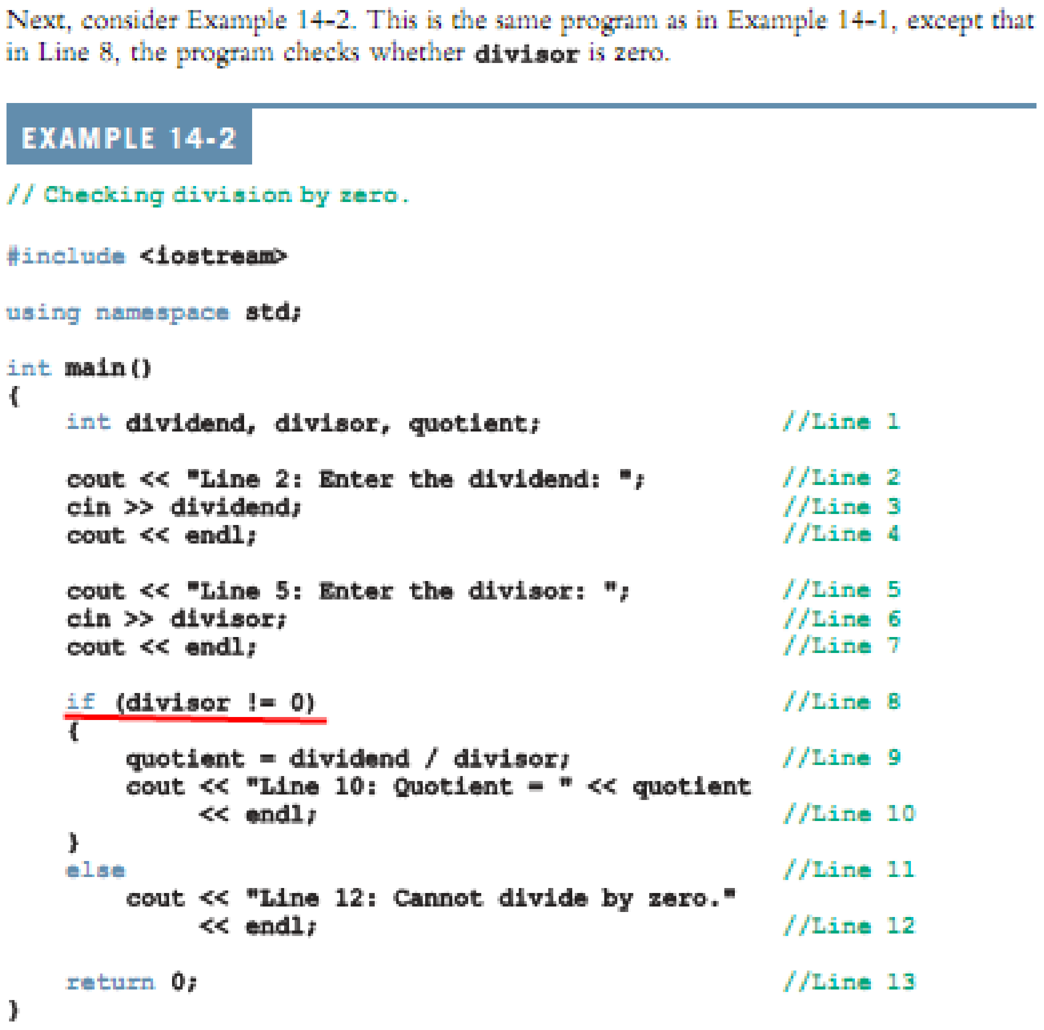
Sample Run 2:

```
Line 2: Enter the dividend: 24

Line 5: Enter the divisor: 0
```

```
CPP_Proj1.exe has encountered a problem and needs to close. We are
sorry for the inconvenience.
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

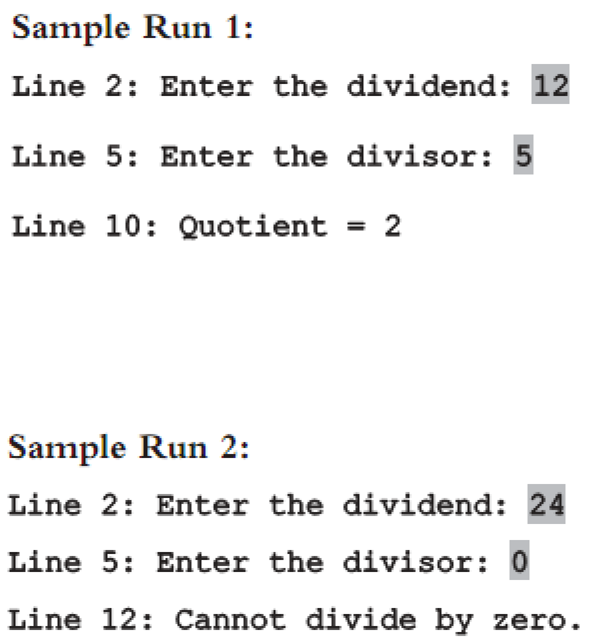
Handling Exception



Next, consider Example 14-2. This is the same program as in Example 14-1, except that in Line 8, the program checks whether **divisor** is zero.

EXAMPLE 14-2

```
// Checking division by zero.

#include <iostream>

using namespace std;

int main()
{
    int dividend, divisor, quotient;                    //Line 1

    cout << "Line 2: Enter the dividend: ";             //Line 2
    cin >> dividend;                                    //Line 3
    cout << endl;                                       //Line 4

    cout << "Line 5: Enter the divisor: ";              //Line 5
    cin >> divisor;                                     //Line 6
    cout << endl;                                       //Line 7

    if (divisor != 0)                                   //Line 8
    {
        quotient = dividend / divisor;                  //Line 9
        cout << "Line 10: Quotient = " << quotient
             << endl;                                   //Line 10
    }
    else                                                //Line 11
        cout << "Line 12: Cannot divide by zero."
             << endl;                                   //Line 12

    return 0;                                           //Line 13
}
```



Sample Run 1:

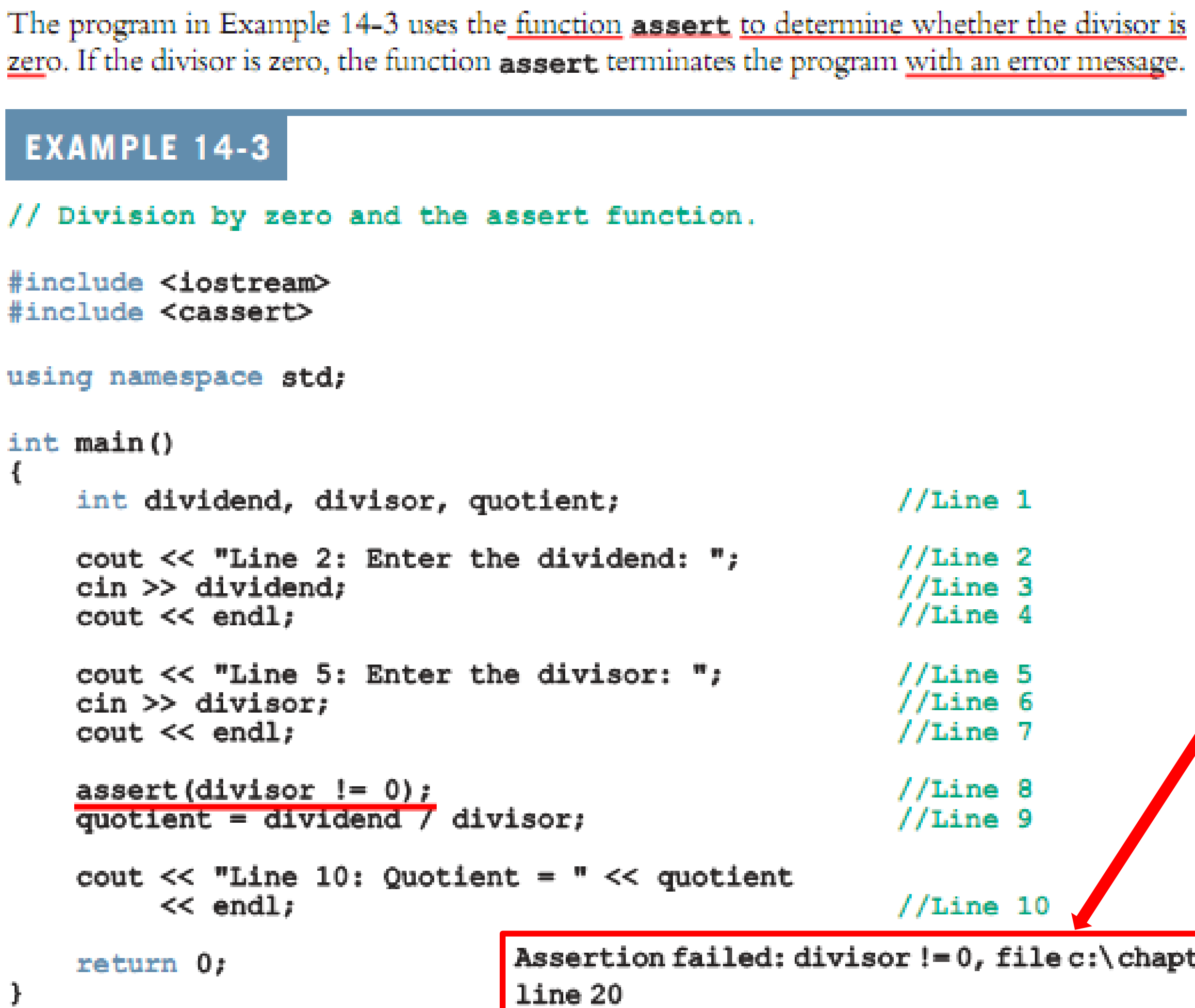
```
Line 2: Enter the dividend: 12

Line 5: Enter the divisor: 5

Line 10: Quotient = 2
```

Sample Run 2:

```
Line 2: Enter the dividend: 24

Line 5: Enter the divisor: 0

Line 12: Cannot divide by zero.
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Handling Exception



The program in Example 14-3 uses the function **assert** to determine whether the divisor is zero. If the divisor is zero, the function **assert** terminates the program with an error message.

EXAMPLE 14-3

```
// Division by zero and the assert function.

#include <iostream>
#include <cassert>

using namespace std;

int main()
{
    int dividend, divisor, quotient;                    //Line 1

    cout << "Line 2: Enter the dividend: ";             //Line 2
    cin >> dividend;                                    //Line 3
    cout << endl;                                       //Line 4

    cout << "Line 5: Enter the divisor: ";              //Line 5
    cin >> divisor;                                     //Line 6
    cout << endl;                                       //Line 7

    assert(divisor != 0);                               //Line 8
    quotient = dividend / divisor;                      //Line 9

    cout << "Line 10: Quotient = " << quotient
         << endl;                                       //Line 10

    return 0;
}
```

Sample Run 1:

```
Line 2: Enter the dividend: 26

Line 5: Enter the divisor: 7

Line 10: Quotient = 3
```

Sample Run 2:

```
Line 2: Enter the dividend: 24

Line 5: Enter the divisor: 0
```

```
Assertion failed: divisor != 0, file c:\chapter 14 source code\ch14_exp3.cpp,
line 20
```

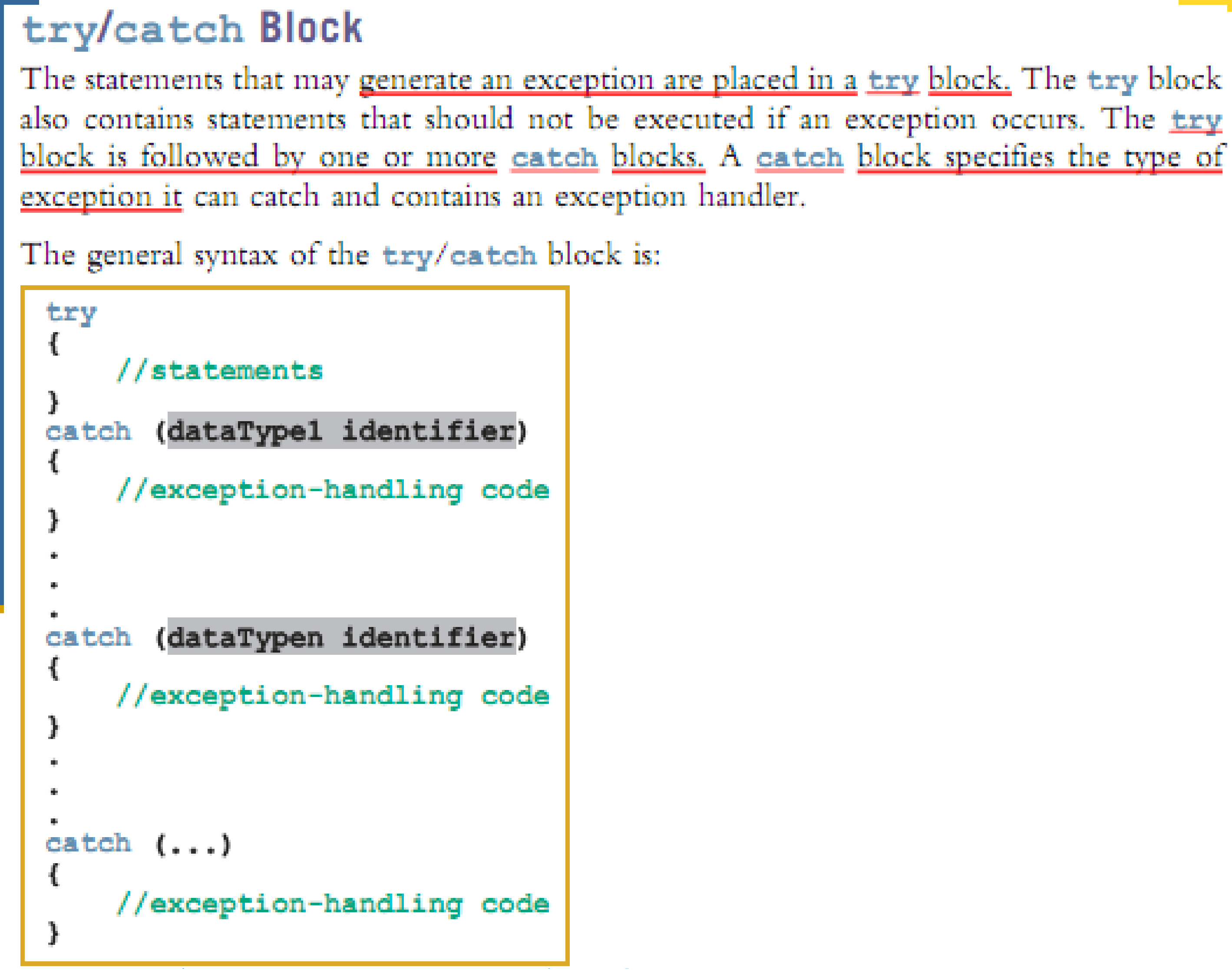
Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Mechanisms of Exception Handling in C++

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try**, **catch**, and **throw**.

1. **throw:** A program throws an exception when a problem shows up. This is done using a **throw** keyword.
2. **catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
3. **try:** A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Mechanisms of Exception Handling in C++



try/catch Block

The statements that may generate an exception are placed in a try block. The try block also contains statements that should not be executed if an exception occurs. The try block is followed by one or more catch blocks. A catch block specifies the type of exception it can catch and contains an exception handler.

The general syntax of the try/catch block is:

```
try
{
    //statements
}
catch (dataType1 identifier)
{
    //exception-handling code
}
.
.
.
catch (dataTypen identifier)
{
    //exception-handling code
}
.
.
.
catch (...)
{
    //exception-handling code
}
```

ORDER OF catch BLOCKS

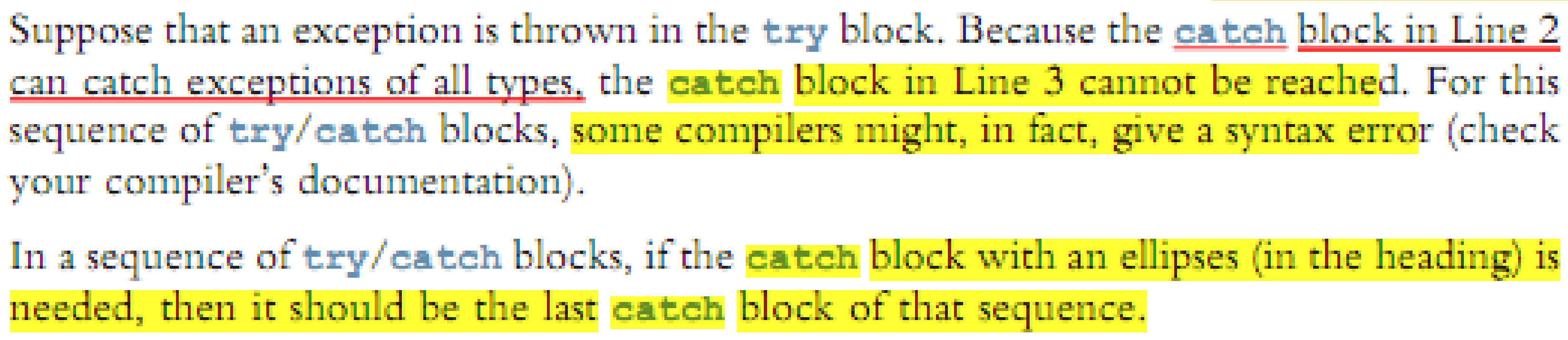
A `catch` block can catch either all exceptions of a specific type or all types of exceptions. The heading of a `catch` block specifies the type of exception it handles. As noted previously, the `catch` block that has an ellipses (three dots) is designed to catch any type of exception. Therefore, if we put this `catch` block first, then this `catch` block can catch all types of exceptions.

Suppose that an exception occurs in a `try` block and is caught by a `catch` block. The remaining `catch` blocks associated with that `try` block are then ignored. Therefore, you should be careful about the order in which you list `catch` blocks following a `try` block. For example, consider the following sequence of `try/catch` blocks:

```
try                       //Line 1
{
    //statements
}
catch (...)               //Line 2
{
    //statements
}
catch (int x)             //Line 3
{
    //statements
}
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

ORDER OF catch BLOCKS



Suppose that an exception is thrown in the `try` block. Because the `catch` block in Line 2 can catch exceptions of all types, the `catch` block in Line 3 cannot be reached. For this sequence of `try/catch` blocks, some compilers might, in fact, give a syntax error (check your compiler's documentation).

In a sequence of `try/catch` blocks, if the `catch` block with an ellipses (in the heading) is needed, then it should be the last `catch` block of that sequence.

ORDER OF catch BLOCKS

This example illustrates how to catch and handle division by zero exceptions. It also shows how a try/catch block might appear in a program.

```
// Handling division by zero exception.

#include <iostream>

using namespace std;

int main()
{
    int dividend, divisor, quotient;                     //Line 1

    try                                                  //Line 2
    {
        cout << "Line 3: Enter the dividend: ";          //Line 3
        cin >> dividend;                                 //Line 4
        cout << endl;                                    //Line 5

        cout << "Line 6: Enter the divisor: ";           //Line 6
        cin >> divisor;                                  //Line 7
        cout << endl;                                    //Line 8

        if (divisor == 0)                                //Line 9
            throw 0;                                     //Line 10

        quotient = dividend / divisor;                   //Line 11

        cout << "Line 12: Quotient = " << quotient
             << endl;                                    //Line 12
    }
    catch (int)                                          //Line 13
    {
       cout << "Line 14: Division by 0." << endl;  //Line 14
    }

    return 0;                                      //Line 15
}
```

Sample Run 1: In this sample run, the user input is shaded.

```
Line 3: Enter the dividend: 17

Line 6: Enter the divisor: 8

Line 12: Quotient = 2
```

Sample Run 2: In this sample run, the user input is shaded.

```
Line 3: Enter the dividend: 34

Line 6: Enter the divisor: 0

Line 14: Division by 0.
```

EXAMPLE 14-7

```
// Handle division by zero, division by a negative integer,
// and input failure exceptions.

#include <iostream>
#include <string>

using namespace std;

int main()
{
    int dividend, divisor = 1, quotient;            //Line 1

    string inpStr
       = "The input stream is in the fail state.";  //Line 2

    try                                             //Line 3
    {
        cout << "Line 4: Enter the dividend: ";     //Line 4
        cin >> dividend;                            //Line 5
        cout << endl;                               //Line 6

        cout << "Line 7: Enter the divisor: ";      //Line 7
        cin >> divisor;                             //Line 8
        cout << endl;                               //Line 9

        if (divisor == 0)                           //Line 10
            throw divisor;                          //Line 11
        else if (divisor < 0)                       //Line 12
            throw string("Negative divisor.");      //Line 13
        else if (!cin)                              //Line 14
            throw inpStr;                           //Line 15

        quotient = dividend / divisor;              //Line 16

        cout << "Line 17: Quotient = " << quotient
             << endl;                               //Line 17
    }
    catch (int x)                                   //Line 18
    {
        cout << "Line 19: Division by " << x
             << endl;                               //Line 19
    }
    catch (string s)                                //Line 20
    {
        cout << "Line 21: " << s << endl;           //Line 21
    }

    return 0;                                       //Line 22
}
```

Sample Run 1: In this sample run, the user input is shaded.

```
Line 4: Enter the dividend: 23

Line 7: Enter the divisor: 6

Line 17: Quotient = 3
```

Sample Run 2: In this sample run, the user input is shaded.

```
Line 4: Enter the dividend: 34

Line 7: Enter the divisor: -6

Line 21: Negative divisor.
```

Sample Run 3: In this sample run, the user input is shaded.

```
Line 4: Enter the dividend: 34

Line 7: Enter the divisor: g

Line 21: The input stream is in the fail state.
```

Handling Exception

EXAMPLE 14-9

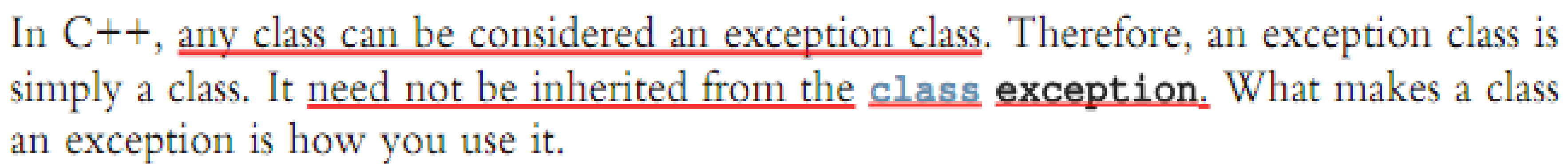
```
// Handling bad_alloc exception thrown by the operator new.

#include <iostream>

using namespace std;

int main()
{
    int *list[100];                                    //Line 1

    try                                                //Line 2
    {
        for (int i = 0; i < 100; i++)                  //Line 3
        {
            list[i] = new int[50000000];               //Line 4
            cout << "Line 4: Created list[" << i
                 << "] of 50000000 components."
                 << endl;                              //Line 5
        }
    }
    catch (bad_alloc be)
    {
        cout << "Line 7: In the bad_alloc catch "
              << "block: " << be.what() << "."
              << endl;
    }

    return 0;
}
```

Sample Run:

```
Line 4: Created list[0] of 50000000 components.
Line 4: Created list[1] of 50000000 components.
Line 4: Created list[2] of 50000000 components.
Line 4: Created list[3] of 50000000 components.
Line 4: Created list[4] of 50000000 components.
Line 4: Created list[5] of 50000000 components.
Line 4: Created list[6] of 50000000 components.
Line 4: Created list[7] of 50000000 components.
Line 7: In the bad_alloc catch block: bad allocation.
```

Creating Your Own Exception

C++ enables programmers to create their own exception classes to handle both the exceptions not covered by C++'s exception classes and their own exceptions. You must throw your own exceptions using the throw statement.


In C++, any class can be considered an exception class. Therefore, an exception class is simply a class. It need not be inherited from the `class exception`. What makes a class an exception is how you use it.

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Creating Your Own Exception

The program in Example 14-10 uses a user-defined class (with no members) to throw an exception.

EXAMPLE 14-10

```cpp
// Using a user-defined exception class.

#include <iostream>

using namespace std;

class divByZero
{};


int main()
{
    int dividend, divisor, quotient;                       //Line 1
    try                                                    //Line 2
    {
        cout << "Line 3: Enter the dividend: ";            //Line 3
        cin >> dividend;                                   //Line 4
        cout << endl;                                      //Line 5

        cout << "Line 6: Enter the divisor: ";             //Line 6
        cin >> divisor;                                    //Line 7
        cout << endl;                                      //Line 8

        if (divisor == 0)
            throw divByZero();

        quotient = dividend / divisor;
        cout << "Line 12: Quotient = " << quotient
             << endl;
    }
    catch (divByZero)
    {
        cout << "Line 14: Division by zero!"
             << endl;
    }

    return 0;
}
```

Sample Run 1: In this sample run, the user input is shaded.

```
Line 3: Enter the dividend: 34

Line 6: Enter the divisor: 5

Line 12: Quotient = 6
```

Sample Run 2: In this sample run, the user input is shaded.

```
Line 3: Enter the dividend: 56

Line 6: Enter the divisor: 0

Line 14: Division by zero!
```

User-defined exception class

```cpp
// User-defined exception class.

#include <iostream>
#include <string>

using namespace std;

class divisionByZero                                  //Line 1
{                                                     //Line 2
public:                                               //Line 3
    divisionByZero()                                  //Line 4
    {
        message = "Division by zero";                 //Line 5
    }                                                 //Line 6

    divisionByZero(string str)                        //Line 7
    {                                                 //Line 8
        message = str;                                //Line 9
    }                                                 //Line 10

    string what()                                     //Line 11
    {                                                 //Line 12
        return message;                               //Line 13
    }                                                 //Line 14

private:                                              //Line 15
    string message;                                   //Line 16
};                                                    //Line 17
```

User-defined exception class

EXAMPLE 14-11

```
// Using user-defined exception class divisionByZero with
// default error message.

#include <iostream>
#include "divisionByZero.h"

using namespace std;

int main()
{
    int dividend, divisor, quotient;                   //Line 1

    try                                                //Line 2
    {
        cout << "Line 3: Enter the dividend: ";        //Line 3
        cin >> dividend;                               //Line 4
        cout << endl;                                  //Line 5

        cout << "Line 6: Enter the divisor: ";         //Line 6
        cin >> divisor;                                //Line 7
        cout << endl;                                  //Line 8

        if (divisor == 0)                              //Line 9
            throw divisionByZero();                    //Line 10

        quotient = dividend / divisor;                 //Line 11
        cout << "Line 12: Quotient = " << quotient
             << endl;                                  //Line 12
    }
    catch (divisionByZero divByZeroObj)                //Line 13
    {
        cout << "Line 14: In the divisionByZero "
             << "catch block: "
             << divByZeroObj.what() << endl;           //Line 14
    }

    return 0;                                          //Line 15
}
```

Sample Run 1: In this sample run, the user input is shaded.

```
Line 3: Enter the dividend: 34

Line 6: Enter the divisor: 5

Line 12: Quotient = 6
```

Sample Run 2: In this sample run, the user input is shaded.

```
Line 3: Enter the dividend: 56

Line 6: Enter the divisor: 0

Line 14: In the divisionByZero catch block: Division by zero
```

User-defined exception class

EXAMPLE 14-13

```cpp
// Handling an exception thrown by a function.

#include <iostream>
#include "divisionByZero.h"

using namespace std;

void doDivision();

int main()
{
    doDivision();

    return 0;
}
void doDivision()
{
    int dividend, divisor, quotient;

    try
    {
        cout << "Line 4: Enter the dividend: ";
        cin >> dividend;
        cout << endl;

        cout << "Line 7: Enter the divisor: ";
        cin >> divisor;
        cout << endl;

        if (divisor == 0)
            throw divisionByZero();

        quotient = dividend / divisor;
        cout << "Line 13: Quotient = " << quotient
             << endl;
    }
    catch (divisionByZero divByZeroObj)
    {
        cout << "Line 15: In the function "
             << "doDivision: "
             << divByZeroObj.what() << endl;
    }
}
```

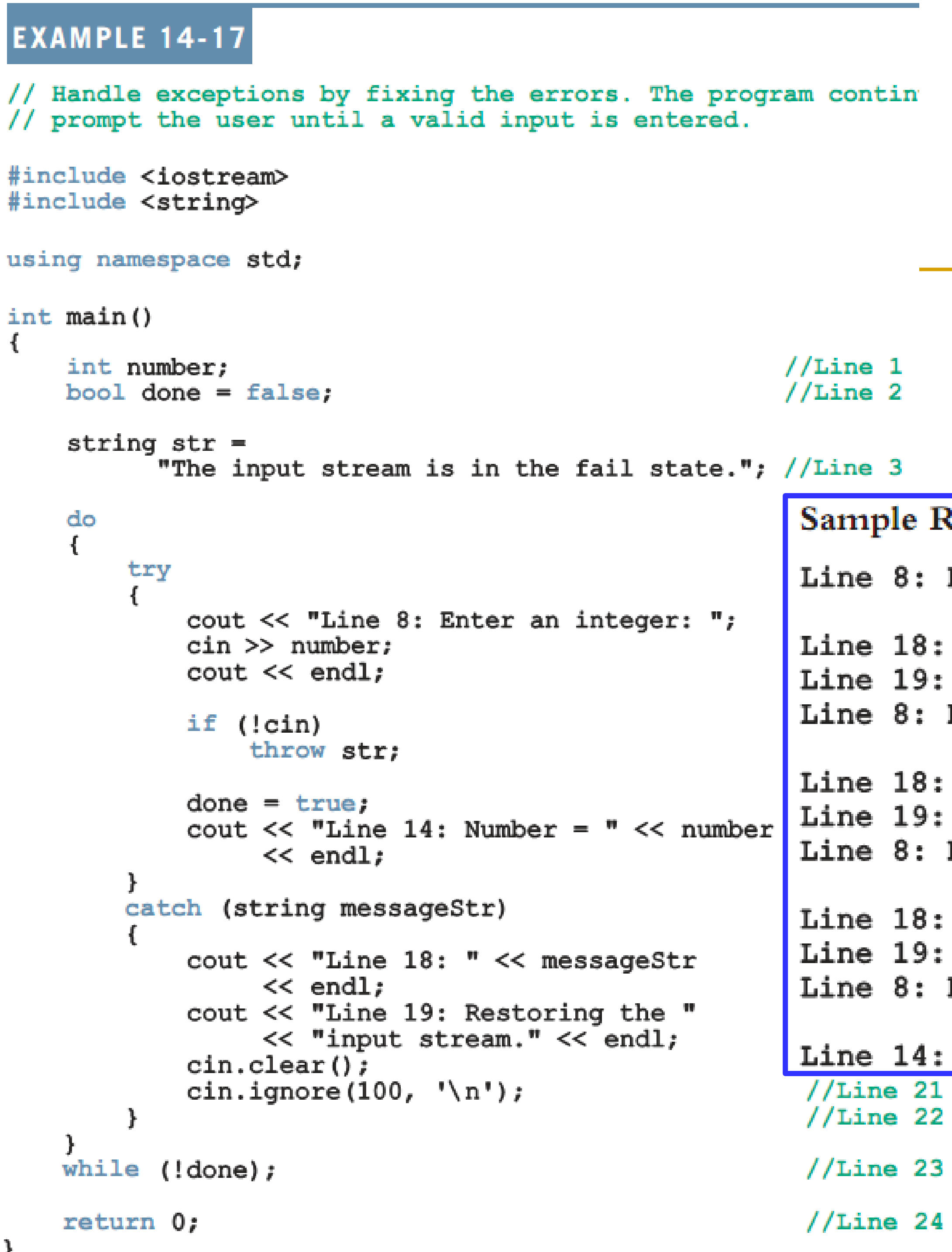
Sample Run 1: In this sample run, the user input is shaded.

```
Line 4: Enter the dividend: 34

Line 7: Enter the divisor: 5

Line 13: Quotient = 6
```

Sample Run 2: In this sample run, the user input is shaded.

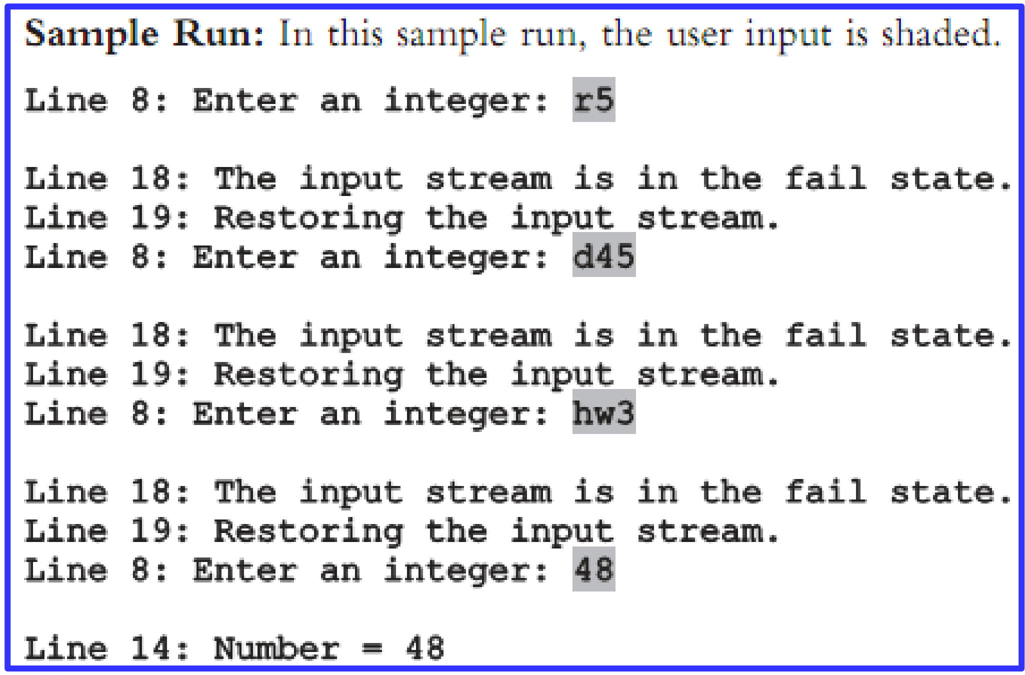
```
Line 4: Enter the dividend: 56

Line 7: Enter the divisor: 0

Line 15: In the function doDivision: Division by zero
```

User-defined exception class



EXAMPLE 14-17

```
// Handle exceptions by fixing the errors. The program contin
// prompt the user until a valid input is entered.

#include <iostream>
#include <string>

using namespace std;

int main()
{
    int number;                                          //Line 1
    bool done = false;                                   //Line 2

    string str =
          "The input stream is in the fail state."; //Line 3

    do
    {
        try
        {
            cout << "Line 8: Enter an integer: ";
            cin >> number;
            cout << endl;

            if (!cin)
                throw str;

            done = true;
            cout << "Line 14: Number = " << number
                 << endl;
        }
        catch (string messageStr)
        {
            cout << "Line 18: " << messageStr
                 << endl;
            cout << "Line 19: Restoring the "
                 << "input stream." << endl;
            cin.clear();
            cin.ignore(100, '\n');                          //Line 21
        }                                                   //Line 22
    }
    while (!done);                                          //Line 23

    return 0;                                               //Line 24
}
```



Sample Run: In this sample run, the user input is shaded.

```
Line 8: Enter an integer: r5

Line 18: The input stream is in the fail state.
Line 19: Restoring the input stream.
Line 8: Enter an integer: d45

Line 18: The input stream is in the fail state.
Line 19: Restoring the input stream.
Line 8: Enter an integer: hw3

Line 18: The input stream is in the fail state.
Line 19: Restoring the input stream.
Line 8: Enter an integer: 48

Line 14: Number = 48
```

Assignment

1. Write a program that prompts the user to enter a length in feet and inches and outputs the equivalent length in centimeters. If the user enters a negative number or a nondigit number, throw and handle an appropriate exception and prompt the user to enter another set of numbers.
2. Write a program that prompts the user to enter time in 12-hour notation. The program then outputs the time in 24-hour notation. Your program must contain three exception classes: invalidHr, invalidMin, and invalidSec. If the user enters an invalid value for hours, then the program should throw and catch an invalidHr object. Similar conventions for the invalid values of minutes and seconds.

Assignment

3. Write a program that prompts the user to enter a person's date of birth in numeric form such as 8-27-1980. The program then outputs the date of birth in the form: August 27, 1980. Your program must contain at least two exception classes: invalidDay and invalidMonth. If the user enters an invalid value for day, then the program should throw and catch an invalidDay object. Similar conventions for the invalid values of month and year. (Note that your program must handle a leap year.)