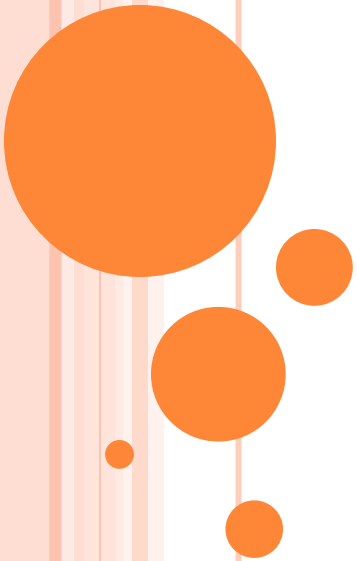

OOP/Computer Programming

Instructor: Dr. Danish Shehzad

Exceptions

- Exception: undesirable event detectable during program execution
- An error that occurs when a program is running
- Abnormal behavior- runtime error
- Can add exception-handling code at the point where an error may occur

```
#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
```

```
#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.
```

ASSERT()

- Assertions are statements used to test assumptions made by programmer
  - Checks if an expression meets certain condition(s)
  - If conditions are not met, it terminates the program
- Example: division by `0`
  - If divisor is zero, `assert` terminates the program with an error message

```
#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;                                           //Line 11
}
```

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 16 source code\ch16_exp3.cpp,
line 19
```

Exception Handling

- Exception: undesirable event detectable during program execution

- Can add exception-handling code at the point where an error may occur

Exception Handling Techniques

- When an exception occurs, the programmer usually has three choices:
  - Terminate the program
  - Include code to recover from the exception
  - Log the error and continue

Terminate the Program

- In some cases, it is best to let the program terminate when an exception occurs
- For example, if the input file does not exist when the program executes
  - There is no point in continuing with the program
- The program can output an appropriate error message and terminate

Fix the Error and Continue

- In some cases, you would like to handle the exception and let the program continue
- For example, if a user inputs a letter in place of a number
  - The input stream will enter the fail state
- You can include the necessary code to keep prompting the user to input a number until the entry is valid

Log the Error and Continue

- For example, a program that monitors a patient's heartbeat cannot be terminated if the blood pressure goes very high
- Or if your program is designed to run a nuclear reactor or continuously monitor a satellite
  - It cannot be terminated if an exception occurs
- When an exception occurs
  - The program should write the exception into a file and continue to run

C++ Mechanisms of Exception Handling

- The `try`/`catch` block handles exceptions

- Exception must be *thrown in a* `try` *block* and caught by a *catch block*

TRY/CATCH BLOCK

Try Block:

- 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

Catch Block:

- Specifies the type of exception it can catch
- Contains an exception handler

General syntax of the try/catch block:

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

Using try/catch Blocks in a Program:

```
#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.
```

try/catch Block (continued)

- If no exception is thrown in a `try` block

  All `catch` blocks for that `try` block are ignored
  Execution resumes after the last `catch` block

- If an exception is thrown in a `try` block

  Remaining statements in that `try` block are ignored

TRY/CATCH BLOCK (CONTINUED)

- The program searches `catch` blocks in order, looking for an appropriate exception handler
- If the type of thrown exception matches the parameter type in one of the `catch` blocks:
  - Code of that `catch` block executes
  - Remaining `catch` blocks are ignored

```
catch (int x)
{
    //exception handling code
}
```

In this `catch` block:

- The identifier `x` acts as a parameter. In fact, it is called a `catch` block parameter.
- The data type `int` specifies that this `catch` block can catch an exception of type `int`.
- A `catch` block can have *at most* one `catch` block parameter.

Throwing an Exception

- For `try`/`catch` to work, the exception must be thrown in the `try` block
- General syntax to throw an exception is:

```
throw expression;
```

where `expression` is a constant value, variable, or object

Throwing an Exception (continued)

- The object being thrown can be:
  - Specific object
  - Anonymous object
- In C++
  - An exception is a value
  - `throw` is a reserved word

EXAMPLE 15-4

Suppose we have the following declaration:

```
int num = 5;
string str = "Something is wrong!!!";
```

`throw` expression	Effect
`throw 4;`	The constant value `4` is thrown.
`throw x;`	The value of the variable `x` is thrown.
`throw str;`	The object `str` is thrown.
`throw string("Exception found!");`	An anonymous `string` object with the string `"Exception found!"` is thrown.

Order of catch Blocks

- Catch block can catch
  - All exceptions of a specific type
  - All types of exceptions
- A `catch` block with an ellipses (three dots) catches any type of exception
- In a sequence of `try`/`catch` blocks, if the `catch` block with an ellipses is needed
  - It should be the last `catch` block of that sequence

Example 15-7

```
#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.
```

Summary

- Exception: an undesirable event detectable during program execution
- `assert` checks whether an expression meets a specified condition and terminates if not met
- `try`/`catch` block handles exceptions
- Statements that may generate an exception are placed in a `try` block
- Catch block specifies the type of exception it can catch and contains an exception handler

Summary (continued)

- If no exceptions are thrown in a `try` block, all `catch` blocks for that `try` block are ignored and execution resumes after the last `catch` block

- Data type of `catch` block parameter specifies type of exception that `catch` block can catch
- Catch block can have at most one parameter
- exception is base class for exception classes