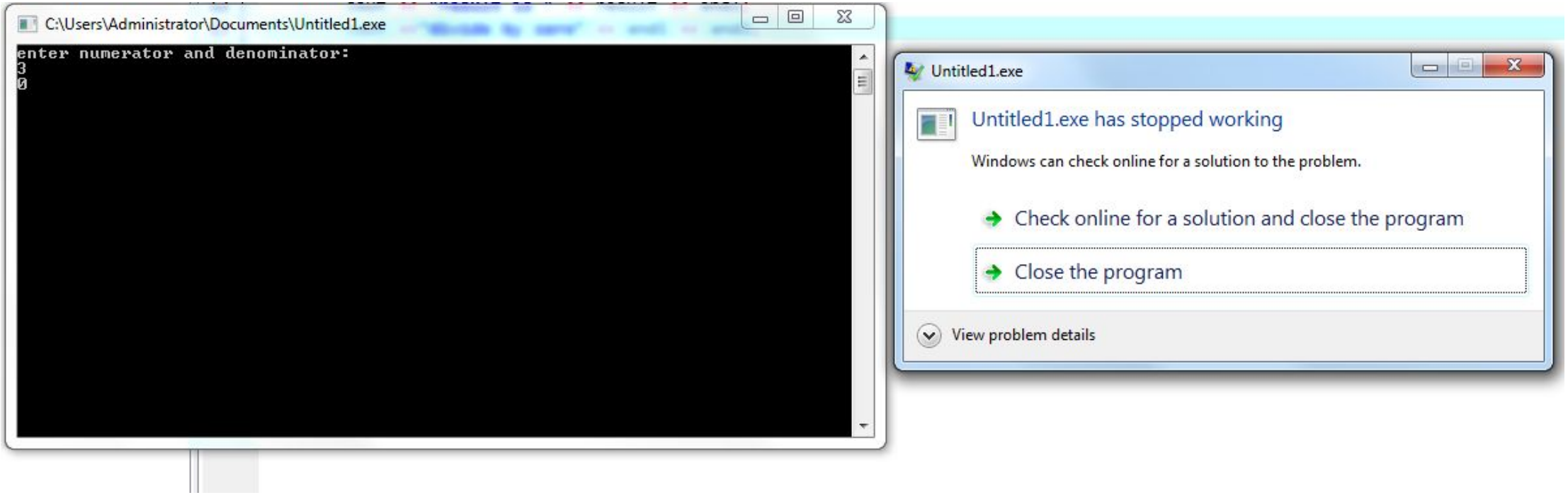
# Object-oriented Programming

Exception Handling

# Output?

```
#include <iostream>
using namespace std;
double zeroDivision(int x, int y) {
    return (x / y);}
int main() {
    int numerator;
    int denominator;
    double result;
    cout << "enter numerator and denominator: " << endl;
    cin>>numerator>>denominator; // 3 and 0
        result = zeroDivision(numerator, denominator);
        cout << "result is " << result << endl;
        cout <<"divide by zero" << endl << endl;
    return 0;}
```

# Output?


C:\Users\Administrator\Documents\Untitled1.exe
enter numerator and denominator:
3
0
Untitled1.exe
Untitled1.exe has stopped working
Windows can check online for a solution to the problem.
Check online for a solution and close the program
Close the program
View problem details

# Exceptions

- Exception handling in C++ provides you with a way of handling unexpected circumstances like runtime errors. So whenever an unexpected circumstance occurs, the program control is transferred to special functions known as handlers.
- An *exception is an abnormal* condition that arises in a code sequence at run time
- In other words, an exception is a run-time error

# Handlers

- 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**.

# Handlers

- **throw**- when a program encounters a problem, it throws an exception. The throw keyword helps the program perform the throw.
- **catch**- a program uses an exception handler to catch an exception. It is added to the section of a program where you need to handle the problem. It's done using the catch keyword.
- **try**- the try block identifies the code block for which certain exceptions will be activated. It should be followed by one/more catch blocks.

# Throwing Exceptions

- Exceptions can be thrown anywhere within a code block using **throw** statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.
- double division(int a, int b) {
- if( b == 0 ) {
- throw "Division by zero condition!"; //throw b;
- }
- return (a/b);
- }

# try

- A block of code which may cause an exception is typically placed inside the try block. It’s followed by one or more catch blocks. If an exception occurs, it is thrown from the try block.
- try {
- // protected code
- }

# Catching Exceptions

- The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.
- try {
- // protected code
- } catch( ExceptionName exception1 ) {
- // code to handle ExceptionName exception
- }

# Catching Exceptions

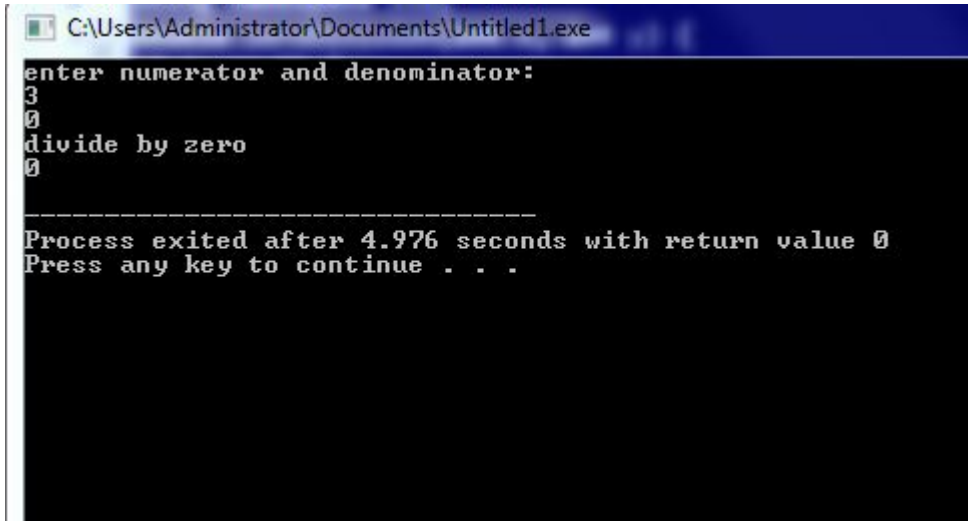
- The ExceptionName is the name of the exception to be caught.
- The exception1 are your defined names for referring to the exceptions.

# Example

```
#include <iostream>
using namespace std;
double zeroDivision(int x, int y) {
if (y == 0) {
                throw y;}
        return (x / y);}
int main() {
        int numerator;
        int denominator;
        double result;
        cout << "enter numerator and denominator: " << endl;
        cin>>numerator>>denominator;

        try {
                result = zeroDivision(numerator, denominator);
                cout << "result is " << result << endl;     }
        catch (int ex) {
                cout <<"divide by zero" << endl << ex << endl;     }
        return 0;}
```

# Example


C:\Users\Administrator\Documents\Untitled1.exe
enter numerator and denominator:
3
0
divide by zero
0
---
Process exited after 4.976 seconds with return value 0
Press any key to continue . . .

# flow of execution of try/catch blocks.

```
#include <iostream>
#include <exception>
using namespace std;
double zeroDivision(int x, int y) {
    if (y == 0) {
            throw y;
cout << "After throw (Never executed) \n";}
    return (x / y);}
int main() {
    int numerator;
    int denominator;
    double result;
    cout << "enter numerator and denominator: " << endl;
    cin>>numerator>>denominator;
    cout << "Before try \n";
```

# flow of execution of try/catch blocks.

```
try {
        result = zeroDivision(numerator, denominator);
        cout << "result is " << result << endl;   }
        catch (int e) {
        cout <<"divide by zero"  << endl;    }
        cout << "After catch (Will be executed) \n";
return 0;}
```

# flow of execution of try/catch blocks.

```
C:\Users\Administrator\Documents\Untitled2.exe
enter numerator and denominator:
2
0
Before try
divide by zero
After catch (Will be executed)

--------------------------------
Process exited after 5.921 seconds with return value 0
Press any key to continue . . .
```

# Without catch

```
#include <iostream>
#include <exception>
using namespace std;
double zeroDivision(int x, int y) {
if (y == 0) {
                throw y;}
        return (x / y);}
int main() {
        int numerator;
        int denominator;
        double result;
        cout << "enter numerator and denominator: " << endl;
        cin>>numerator>>denominator;

        try {
                result = zeroDivision(numerator, denominator);
                cout << "result is " << result << endl;     }

        return 0;}
```

# Without catch

```
In function 'int main()':
[Error] expected 'catch' before numeric constant
```

# Multiple catch Clauses

- In some cases, more than one exception could be raised by a single piece of code
- To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception
- After one catch statement executes, the others are bypassed

# Multiple catch Clauses

- You can list down multiple **catch** statements to catch different type of exceptions in case your **try** block raises more than one exception in different situations.
- try { // protected code }
- catch( ExceptionName e1 )
- { // catch block }
- catch( ExceptionName e2 ) { // catch block }
- catch( ExceptionName eN ) { // catch block }

# Multiple catch Clauses

```
#include <iostream>
#include <exception>
using namespace std;
double zeroDivision(int x, int y) {
if (y == 0) {
                throw y;}
        return (x / y);}
int main() {
        int numerator;
        int denominator;
        double result;
        cout << "enter numerator and denominator: " << endl;
        cin>>numerator>>denominator;

        try {
                result = zeroDivision(numerator, denominator);
                cout << "result is " << result << endl;      }
```

# Multiple catch Clauses

```
• catch (int e) {
•         cout <<"divide by zero" << endl << e << endl;     }
•         catch (int e) {
•         cout <<"another exception" << endl << e << endl;   }
•         catch (int e) {
•         cout <<"another exception" << endl << e << endl;   }
•   return 0;}
```

# catch all block

- If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the exception declaration as follows −
- try { // protected code }
- catch(...)
- { // code to handle any exception }

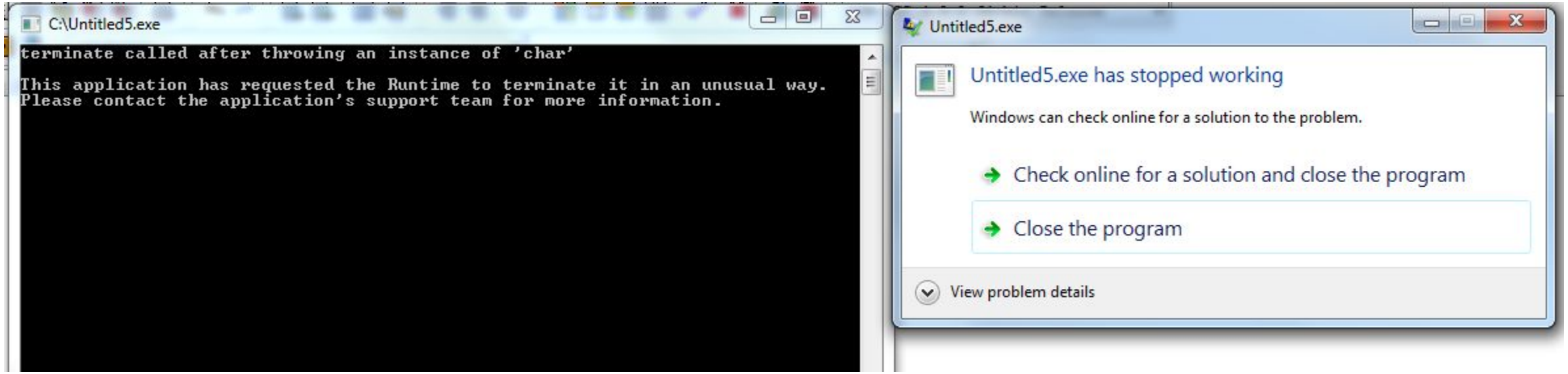
# catch all block

```
#include <iostream>
using namespace std;
double zeroDivision(int x, int y) {
if (y == 0) {
                throw y;}
        return (x / y);}
int main() {
        int numerator;
        int denominator;
        double result;
        cout << "enter numerator and denominator: " << endl;
        cin>>numerator>>denominator;

        try {
                result = zeroDivision(numerator, denominator);
                cout << "result is " << result << endl;     }
        catch(...){
                cout <<"divide by zero" << endl << endl;              }
        return 0;}
```

- If an exception is thrown and not caught anywhere, the program terminates abnormally.
- #include <iostream>
- using namespace std;
-
- int main(){
-  try  {
-  throw 'a';  }
-  catch (int x)  {
-  cout << "Caught ";  }
-  return 0;}

- Implicit type conversion doesn't happen for primitive types. For example, in the following program 'a' is not implicitly converted to int

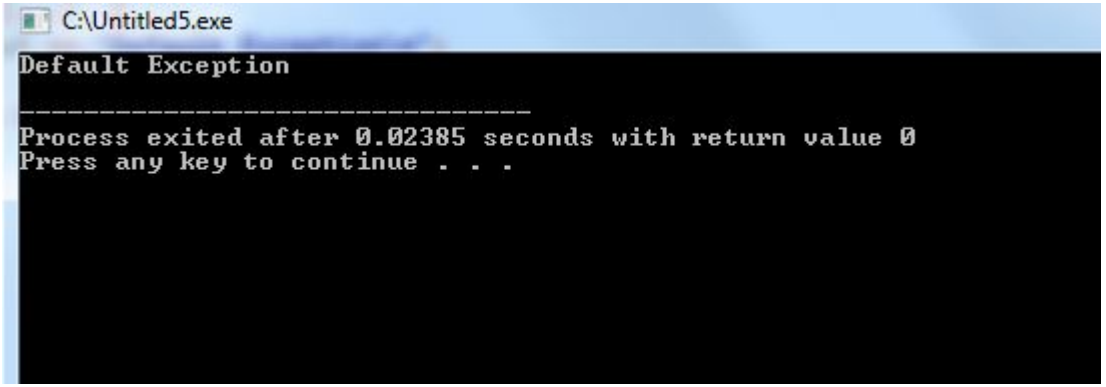


# Correct version

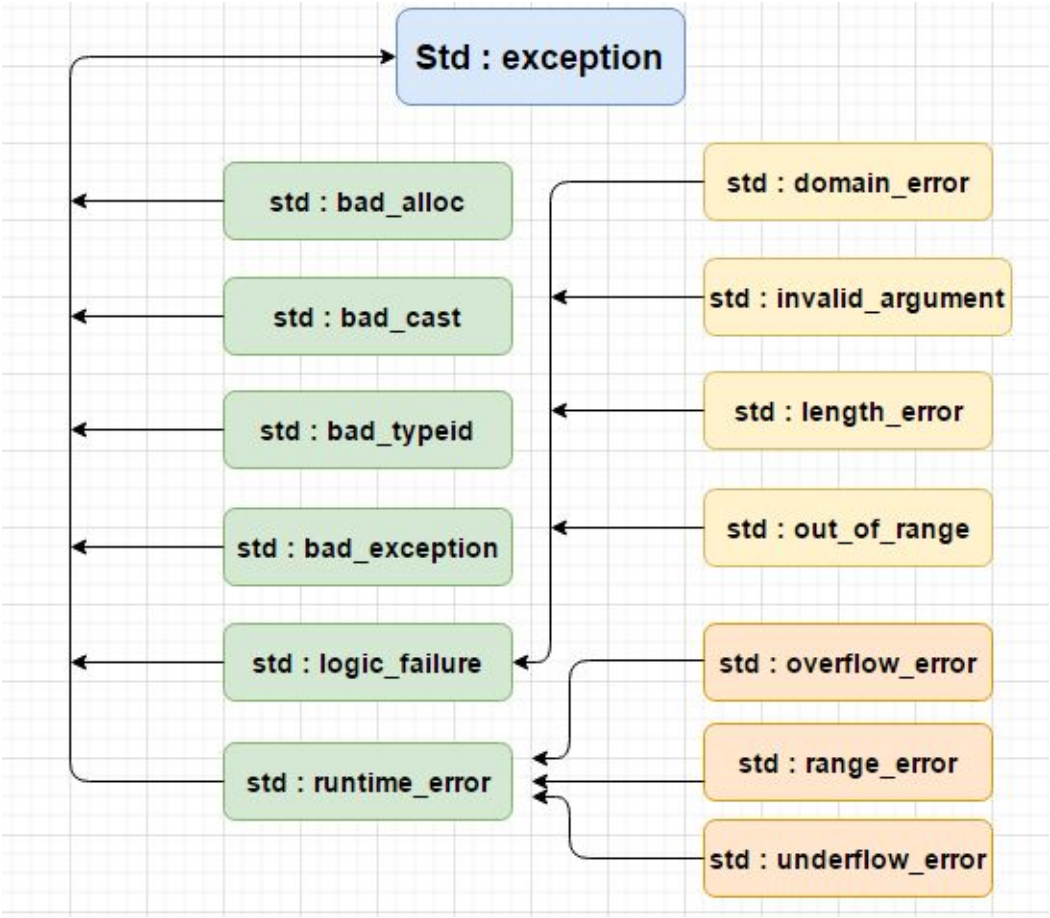
```
#include <iostream>
using namespace std;

int main()
{
    try  {
      throw 'a';
    }
    catch (int x)  {
        cout << "Caught " << x;
    }
    catch (...)  {
        cout << "Default Exception\n";
    }
    return 0;
}
```

# output


C:\Untitled5.exe
Default Exception
--------------------------------
Process exited after 0.02385 seconds with return value 0
Press any key to continue . . .

# C++ Standard Exceptions



# C++ Standard Exceptions

- C++ provides a list of standard exceptions defined in <exception> which we can use in our programs

# C++ Standard Exceptions

| Exception | Description |
|---|---|
| std::exception | An exception and parent class of all the standard C++ exceptions. |
| std::bad_alloc | This can be thrown by new. |
| std::bad_cast | This can be thrown by dynamic_cast. |
| std::bad_exception | This is useful device to handle unexpected exceptions in a C++ program |
| std::bad_typeid | This can be thrown by typeid. |
| std::logic_error | An exception that theoretically can be detected by reading the code. |
| std::domain_error | This is an exception thrown when a mathematically invalid domain is used |
| std::invalid_argument | This is thrown due to invalid arguments. |
| std::length_error | This is thrown when a too big std::string is created |
| std::out_of_range | This can be thrown by the at method from for example a std::vector and std::bitset<>::operator[]() |
| std::runtime_error | An exception that theoretically can not be detected by reading the code. |
| std::overflow_error | This is thrown if a mathematical overflow occurs. |
| std::range_error | This is ocurred when you try to store a value which is out of range. |
| std::underflow_error | This is thrown if a mathematical underflow occurs. |