Exception Handling

Errors

Compile time Errors

- Compile time errors are syntactic errors which occurs during writing of the program.
- Common examples are missing semicolon, missing comma, etc.
- They occurs due to *poor understanding of language*.

Logical Errors

- They occur due to *improper understanding of the program logic*.
- Logical errors cause the unexpected behavior and output of program.

Run time Errors or Exceptions

- They *occurs accidentally* which may result in abnormal termination of the program.
- Common examples are division by zero, opening file to read which does not exist, insufficient memory, violating array bounds, etc.

Exception Handling

- Exception handling is the process to handle the exception *if generated* by the program at runtime.
  - The aim is to write code, which passes exception to a routine.
  - This routine can handle the exception and can take suitable action.
- Exception Handling Steps are:

  Step 1 : Writing try block.

  Step 2 : Throwing an exception.

  Step 3 : Catching and handling the exception thrown.

- C++ provides exception handling mechanism
  - To trap different exceptions in programs.
  - To make programs running smoothly after catching the exception.

Exception Handling

- One of the advantages of C++ over C is Exception Handling. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. There are two types of exceptions:
  - Synchronous,
  - Asynchronous (Ex:which are beyond the program's control, Disc failure etc).
- C++ provides following specialized keywords for this purpose.
  ***try*:** represents a block of code that can throw an exception.
  ***catch*:** represents a block of code that is executed when a particular exception is thrown.
  ***throw*:** Used to throw an exception. Also used to list the exceptions that a function throws, but doesn't handle itself.

Exception Handling

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

int main()
{
int x = -1;

cout << "Before try \n";
try {
        cout << "Inside try \n";
        if (x < 0)
        {
                throw x;
                cout << "After throw (Never executed) \n";
        }
}
catch (int x ) {
        cout << "Exception Caught \n";
}

cout << "After catch (Will be executed) \n";
return 0;
}
```

```
Before try
Inside try
Exception Caught
After catch (Will be executed)
```

Exception Handling **Step 3 : Catching the exception**

- A try block
  - Must have **at least one matching** catch block.
  - Can have more than one catch blocks for catching different types of exceptions.
- A catch block must have a **try block prior written** which will throw an exception.
- A catch block should have only **one argument**.
- An exception thrown by try block is caught and handled by the catch block.
  - If exception thrown **matches with the argument** in catch block,
    - Then exception will be caught successfully by the catch block.
    - After successful execution of the catch block any code written after the catch block will be executed.
  - If argument **does not match** with the exception thrown,
    - Catch block cannot handle it and this may results in abnormal program termination.

Exception Handling **Example Division by zero**

```
void main(){
      float x, y;
      cout << "Enter two numbers“ <<endl;
      cin >> x >> y;
      try { // start a try block
            if (y != 0)
                  cout << "Div = " << x / y << endl;
            else
                  throw y;
        }
      catch (float y) { // catch an error
            cout << "Caught Division by " << y << endl;
      }
      cout << "Out of try catch block " << endl;
}
```

```
Enter two numbers
55
23
Div = 2.3913
Out of try catch block
```

```
Enter two numbers
55
0
Caught Division by 0
Out of try catch block
```

Exception Handling

There is a special catch block called ‘catch all’ catch(…) that can be used to catch all types of exceptions. For example, in the following program, an int is thrown as an exception, but there is no catch block for int, so catch(…) block will be executed.

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

int main()
{
        try {
        throw 10;
        }
        catch (char excp) {
                cout << "Caught " << excp;
        }
        catch (...) {
                cout << "Default Exception\n";
        }
        return 0;
}
```

Exception Handling

If an exception is thrown and not caught anywhere, the program terminates abnormally. For example, in the following program, a char is thrown, but there is no catch block to catch a char.

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

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

Output:

terminate called after throwing an instance of 'char’.

This application has requested the Runtime to terminate it in an unusual way. Please contact the application's support team for more information.

Exception Handling **Functions**

- A function can handle the exception by adding local try catch blocks.

```
float Divide(float x, float y){
      try {
         if (y != 0)
             return x / y;
          else
             throw y;
    }
      catch (float y) { // catch an error
         cout << " Inside Divide" << endl;
         cout << "Caught Division by "
          << y << endl;
      }
}
```

```
void main(){
      float x, y;
      cout << "Enter two numbers" <<endl
      cin >> x >> y;
      cout << "Div = " << Divide(x, y);
}
```

```
Enter two numbers
55
8
Div = 6.875
```

```
Enter two numbers
55
0
 Inside Divide
Caught Division by 0
Div = -nan(ind)
```

Exception Handling **Functions**

- The function can simply throw the exception.

```
float Divide(float x, float y){
      if (y != 0)
            return x / y;
         else
            throw y;
}
```

```
Enter two numbers
88
0
Inside Caller
Caught Division by 0
```

The caller will be responsible for catching and handling the thrown exception.

```
void main(){
      float x, y;
      cout << "Enter two numbers" <<endl;
      cin >> x >> y;
      try {
       cout << "Div = " << Divide(x, y);
      }
      catch (float y) { // catch an error
         cout << "Inside Caller" << endl;
         cout << "Caught Division by "
          << y << endl;
      }
}
```

```
void fun(int *ptr, int x)
{
   if (ptr == NULL)
      throw ptr;
   if (x == 0)
      throw x;
   /* Some functionality */
}

int main()
{
   try {
     fun(NULL, 0);
   }
   catch(...) {
      cout << "Caught exception from fun()";
   }
   return 0;
}
```

Exception in classes

When an exception is thrown, all objects created inside the enclosing try block are destructed before the control is transferred to catch block.

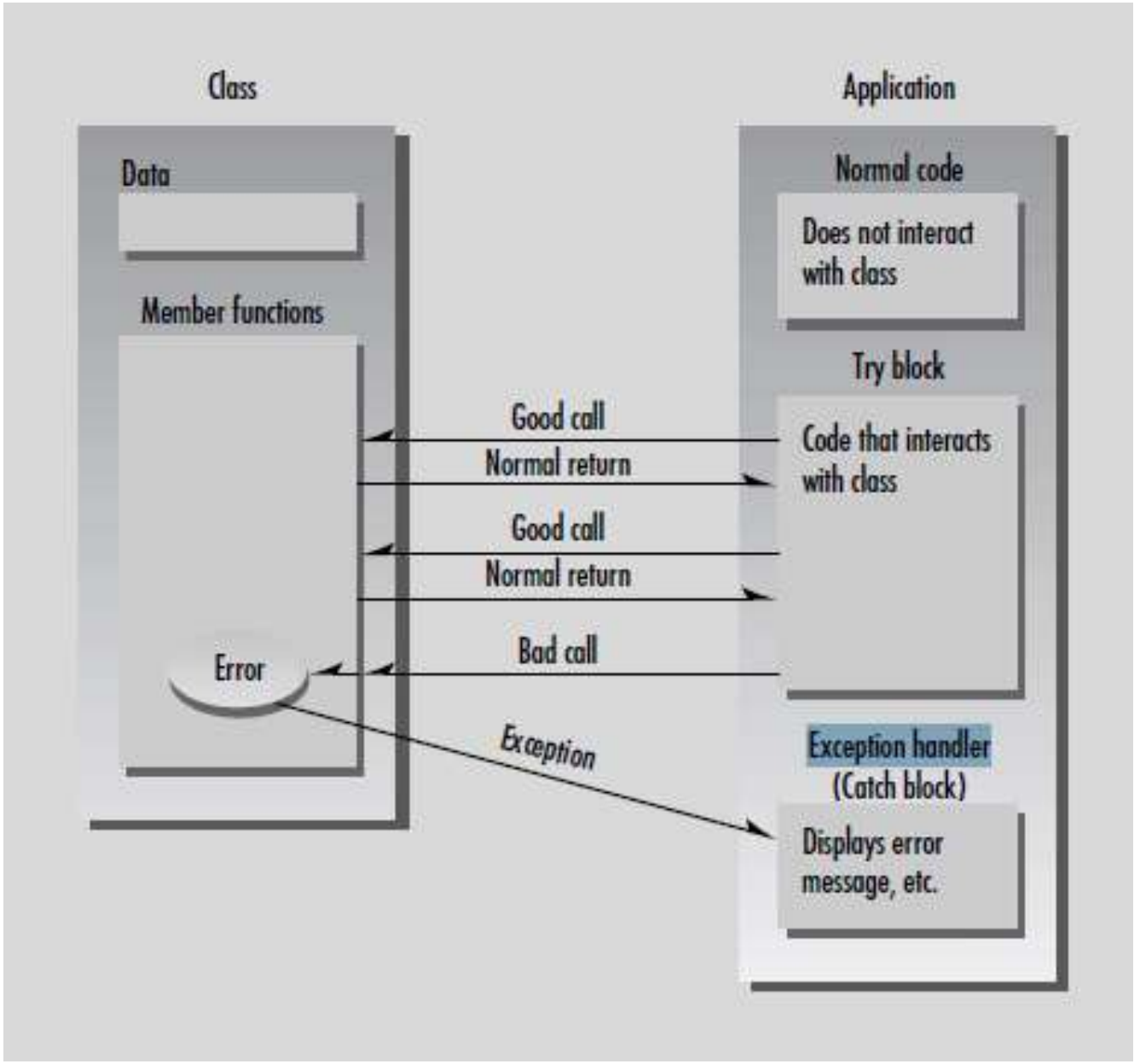
```
#include <iostream>
using namespace std;

class Test {
public:
        Test() { cout << "Constructor of Test " << endl; }
        ~Test() { cout << "Destructor of Test " << endl; }
};

int main()
{
        try {
                Test t1;
                throw 10;
        }
        catch (int i) {
                cout << "Caught " << i << endl;
        }
}
```

```
Constructor of Test
Destructor of Test
Caught 10
```

Exception Handling



Exception in classes

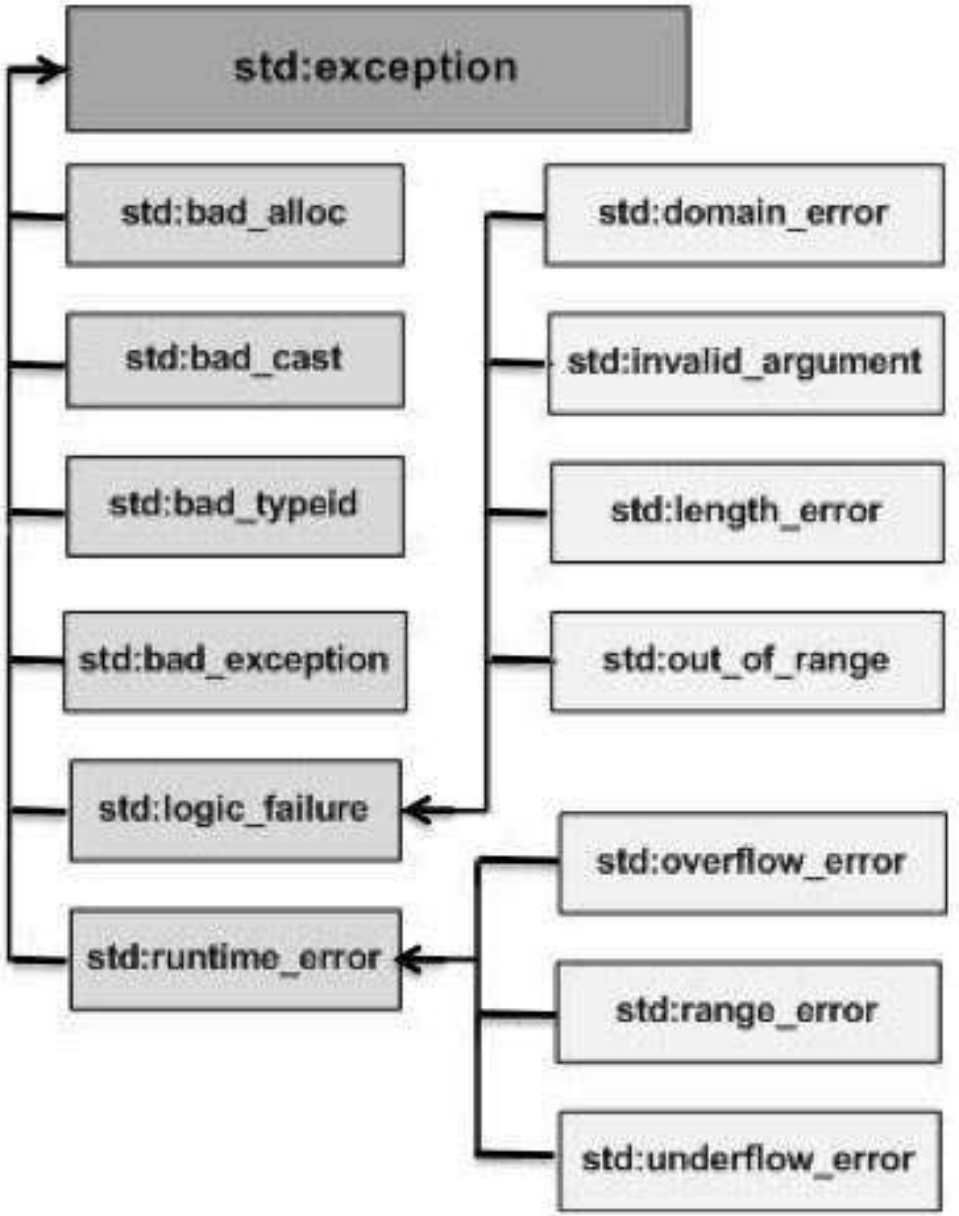
Always write derived class catch before base catch.

```
class Base {
};
class Derived : public Base {
};
int main()
{
        Derived d;
        // Some other functionalities
        try {
                throw d;
        }
        catch (Derived d) {
                cout << "Caught Derived Exception";
        }
        catch (Base b) {
                cout << "Caught Base Exception";
        }

        getchar();
        return 0;
}
```

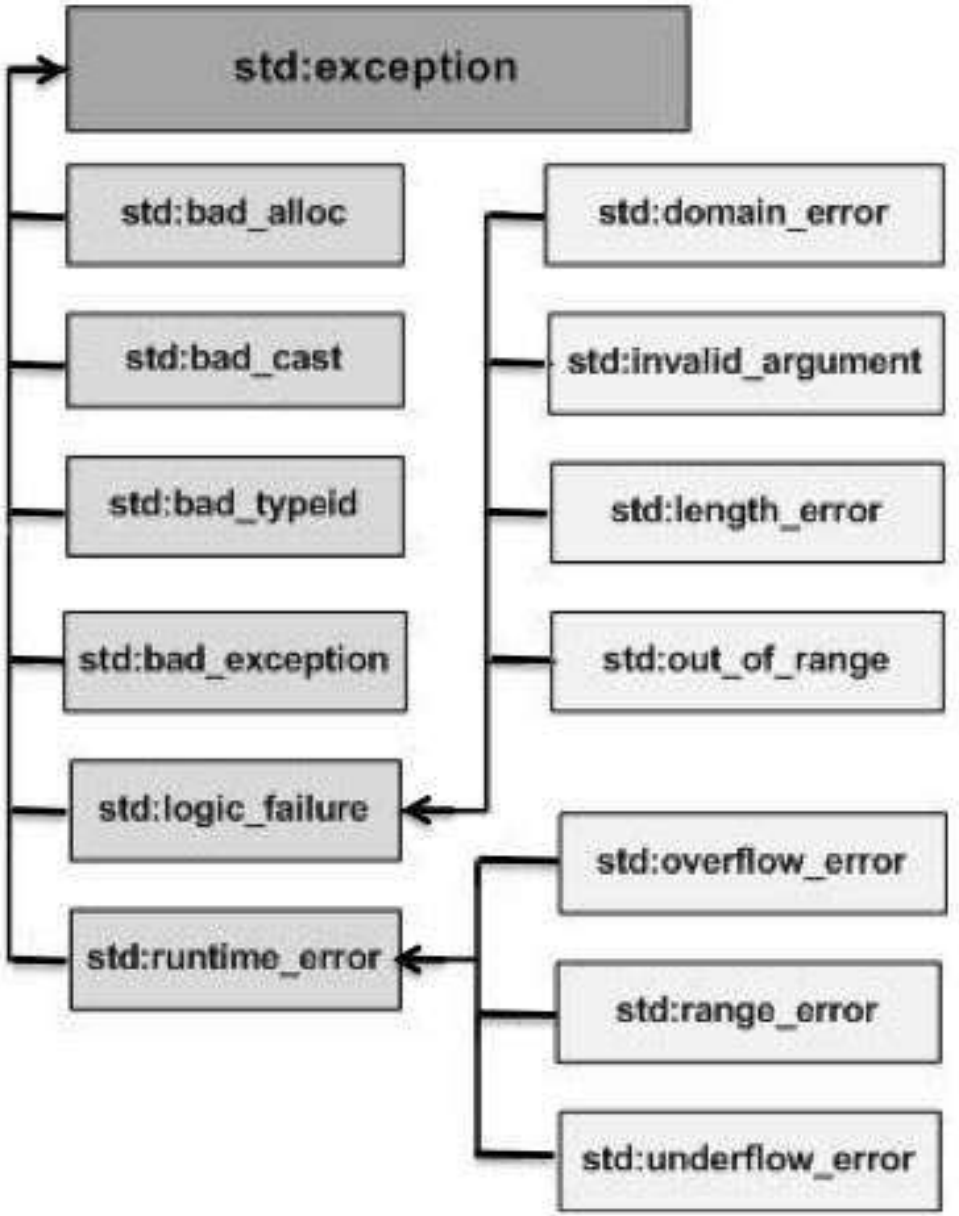
C++ standard exception

1	**exception** An exception and parent class of all the standard C++ exceptions.
2	**bad_alloc** This can be thrown by **new**.
3	**bad_cast** This can be thrown by **dynamic_cast**.
4	**bad_exception** This is useful device to handle unexpected exceptions in a C++ program.
5	**bad_typeid** This can be thrown by **typeid**.
6	**logic_error** An exception that theoretically can be detected by reading the code.
7	**domain_error** This is an exception thrown when a mathematically invalid domain is used.
8	**invalid_argument** This is thrown due to invalid arguments.



C++ standard exception

9	**length_error** This is thrown when a too big string is created.
10	**out_of_range** This can be thrown by the 'at' method, for example a vector and bitset<>::operator[]().
11	**runtime_error** An exception that theoretically cannot be detected by reading the code.
12	**overflow_error** This is thrown if a mathematical overflow occurs.
13	**range_error** This is occurred when you try to store a value which is out of range.
14	**underflow_error** This is thrown if a mathematical underflow occurs.



Out of range and length error

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string sentence;
    string str1, str2, str3;

    try
    {
        sentence = "Testing string exceptions!";
        cout << "Line 5: sentence = " << sentence
             << endl;

        str1 = sentence.substr(8, 20);
        cout << "Line 8: str1 = " << str1
             << endl;

        str2 = sentence.substr(28, 10);
        cout << "Line 10: str2 = " << str2
             << endl;

        str3 = "Exception handling. " + sentence;
        cout << "Line 12: str3 = " << str3
             << endl;

    }
    catch (out_of_range re)
    {
        cout << "Line 14: In the out_of_range "
             << "catch block: " << re.what()
             << endl;
    }
    catch (length_error le)
    {
        cout << "Line 16: In the length_error "
             << "catch block: " << le.what()
             << endl;
    }

    return 0;
}
```

```
Line 5: sentence = Testing string exceptions!
Line 8: str1 = string exceptions!
Line 14: In the out_of_range catch block: invalid string position
```

Bad alloc

```
#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)                               //Line 6
    {
        cout << "Line 7: In the bad_alloc catch "
             << "block: " << be.what() << "."
             << endl;                                  //Line 7
    }

    return 0;                                          //Line 8
}
```

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.
```

User define 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)                                 //Line 9
          throw divByZero();                            //Line 10

      quotient = dividend / divisor;                    //Line 11
      cout << "Line 12: Quotient = " << quotient
           << endl;                                     //Line 12
  }
  catch (divByZero)                                     //Line 13
  {
      cout << "Line 14: Division by zero!"
           << 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: Division by zero!
```

User define exception class

```
// User-defined exception class.

#include <iostream>
#include <string>

using namespace std;

class divisionByZero
{
public:
    divisionByZero()
    {
        message = "Division by zero";
    }

    divisionByZero(string str)
    {
        message = str;
    }

    string what()
    {
        return message;
    }

private:
    string message;
};
```

```
int main()
{
    int dividend, divisor, quotient;

    try
    {
        cout << "Line 3: Enter the dividend: ";
        cin >> dividend;
        cout << endl;

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

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

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

    return 0;
}
```

User define exception class

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
```

```
if (divisor == 0)
   throw divisionByZero("Found division by zero");
```

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
```

Multiple Exceptions by making own classes

```
using namespace std;
const int MAX = 3;                  //stack holds 3 integers
////////////////////////////////////////////////////////////
class Stack
   {
   private:
      int st[MAX];                  //stack: array of integers
      int top;                      //index of top of stack
   public:
      class Full { };               //exception class
      class Empty { };              //exception class
//--------------------------------------------------------
      Stack()                       //constructor
         { top = -1; }
//--------------------------------------------------------
      void push(int var)            //put number on stack
         {
         if(top >= MAX-1)           //if stack full,
            throw Full();           //throw Full exception
         st[++top] = var;
         }

//--------------------------------------------------------
      int pop()                     //take number off stack
         {
         if(top < 0)                //if stack empty,
            throw Empty();          //throw Empty exception
         return st[top--];
         }
   };
////////////////////////////////////////////////////////////
int main()
   {
   Stack s1;

   try
      {
      s1.push(11);
      s1.push(22);
      s1.push(33);
//    s1.push(44);                          //oops: stack full
      cout << "1: " << s1.pop() << endl;
      cout << "2: " << s1.pop() << endl;
      cout << "3: " << s1.pop() << endl;
      cout << "4: " << s1.pop() << endl;  //oops: stack empty
      }
   catch(Stack::Full)
      {
      cout << "Exception: Stack Full" << endl;
      }
   catch(Stack::Empty)
      {
      cout << "Exception: Stack Empty" << endl;
      }
   return 0;
   }
```

Throw and rethrow

Output : Main start
Caught exception inside MyHandler
Caught exception inside Main
Main end

```
#include <iostream>
using namespace std;
void MyHandler()
{
   try
   {
      throw “hello”;
   }
   catch (const char*)
   {
   cout <<”Caught exception inside MyHandler\n”;
   throw; //rethrow char* out of function
   }
}
int main()
{
   cout<< “Main start”;
   try
   {
      MyHandler();
   }
   catch(const char*)
   {
     cout <<”Caught exception inside Main\n”;
   }
      cout << “Main end”;
      return 0;
}
```

Throw and rethrow

Output : Caught Base b, which is actually a Derived
Caught Base b, which is actually a Derived

```
class Base
{
public:
   Base() {}
   virtual void print() {cout << "Base"; }
};
class Derived: public Base
{
public:
   Derived() {}
   void print() override {  cout << "Derived"; }
};
int main()
{
   try
   {
      try
      {
         throw Derived{};
      }
      catch (Base& b)
      {
          cout << "Caught Base b, which is actually a ";
         b.print(); cout << "\n";
               throw;
      }
   }
   catch (Base& b)
   {
       cout << "Caught Base b, which is actually a ";
      b.print();
       cout << "\n";
   }
}
```

Throw and rethrow

```
    catch (divisionByZero)   //Line 18
    {
        throw;               //Line 19
    }
}
```

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

```
Line 8: Enter the dividend: 34

Line 11: Enter the divisor: 5

Line 17: Quotient = 6
```

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

```
Line 8: Enter the dividend: 56

Line 11: Enter the divisor: 0

Line 4: In main: Found division by 0!
```

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

using namespace std;

void doDivision()

int main()
{
    try                                                          //Line 1
    {
        doDivision();                                            //Line 2
    }
    catch (divisionByZero divByZeroObj)                          //Line 3
    {
        cout << "Line 4: In main: "
             << divByZeroObj.what() << endl;                     //Line 4
    }

    return 0;                                                    //Line 5
}

void doDivision()
{
    int dividend, divisor, quotient;                             //Line 6

    try                                                          //Line 7
    {
        cout << "Line 8: Enter the dividend: ";                  //Line 8
        cin >> dividend;                                         //Line 9
        cout << endl;                                            //Line 10

        cout << "Line 11: Enter the divisor: ";                  //Line 11
        cin >> divisor;                                          //Line 12
        cout << endl;                                            //Line 13

        if (divisor == 0)                                        //Line 14
            throw divisionByZero("Found division by 0!");        //Line 15

        quotient = dividend / divisor;                           //Line 16
        cout << "Line 17: Quotient = " << quotient
             << endl;                                            //Line 17
    }
```