Object Oriented Programming with C++

C++ Templates, Exception Handling

Lecturer : Astha Sharma

CHAPTER 6

Template and generic programming

(4 hrs)

Generic and template functions and classes

Specific and general programming

•Till now, we have been writing programs with specific codes for specific tasks.

•But cases may arise where the same program is to be used for different purpose.

•Template is a new concept which enable us to define generic classes and functions and thus provides support for generic programming.

•Generic programming is an approach where generic types are used as parameters in algorithms so that they work for a variety of suitable data types and data structures.

•A template can be used to create a family of classes or functions.

•Since the template is defined with a parameter that would be replaced by a specified data type at the time of actual use of the class or function, the templates are sometimes called parameterized classes or function.

Function templates

Lets see WHY we need function templates:

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

int maximum(int first,int second)
{
   if (first>second)
      {
         return first;
      }
   else
      {
       return second;
      }
}
float maximumfloat(float first, float second)
{
   if (first>second)
      {
         return first;
      }
   else
      {
       return second;
      }
}
int main()
{

   cout<<"The highest integer number is: "<<maximum(4,3)<<endl;
   cout<<"The highest float number is: "<<maximumfloat(4.522,5.622)<<endl;

}
```

Problem statement

•Here, we need to print the maximum between two numbers given during function call.

•In the first case we had two integers numbers and maximum was found out from maximum() function.

•In the second case we had two float numbers and maximum was found out from maximumfloat() function.

•Here, the objective was in both functions to find out and print the maximum but only the data type was different.

Problem statement

- So, the same tasks had to be done in two different functions that cause code repetition .

- In C++, we want code to be short and reuable.

- That’s why we need function templates.

Function templates

•Function templates are special functions that can operate with *generic types*.

•This allows us to create a function template whose functionality can be adapted to more than one type without repeating the entire code for each type.

•We write a generic function that can be used for different data types.

•Examples of function templates are sort(), max(), min(), printArray(), printnumber(),etc.

•Format:

```
template<class T>
return_type function_name (arguments of type T)
{
……………………..
…………………….. // Body of function with type T
……………………..
…………………….. // Wherever appropriate
}
```

Here, template and class are keywords.

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

template<class T>
T maximum(T first , T second)
{
    if (first>second)
        {
            return first;
        }
    else
        {
         return second;
        }
}

int main()
{

    cout<<"The highest integer number is: "<<maximum(4,3)<<endl;
    cout<<"The highest float number is: "<<maximum(10.522,5.622)<<endl;
    return 0;
}
```

- Make a function template which prints the sum of two numbers(integers and floats).

- Make a function template which prints the multiplication of two numbers(intergers and floats).

Function templates with multiple parameters

•Format:

```
template<class A, class B >
return_type function_name (arguments of type T)
{
..........................
.......................... // Body of function with type T
..........................
.......................... // Wherever appropriate
}
```

Here, template and class are keywords.

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

template<class A,class B>
void maximum(A first,B second)
{
   if (first>second)
      {
         cout<<first;
      }
   else
      {
         cout<<second;
      }
}

int main()
{

   cout<<"The highest between 4 and 3.3 is ";
   maximum(4,3.3);
   cout<<"\nThe highest between 10.522 and 1 is ";
   maximum(10.522,1);
   return 0;
}
```

- Make a function template which prints the sum of two numbers(first number is integer and second is float).

- Make a function template which prints the multiplication of two numbers(first number is float and second is integer).

Class templates

•Format:

```
Template<class T>
class class_name
{
.................... //class member
.................... // specifications with
.................... // anonymous type T
.................... // where or appropriate
};
```

Example:

```
#include<iostream>
template<class T1>
class Test
{
T1 a;
public:
void add (T1 x, T1 y)
{
a=x+y;
}
void show()
{
cout<<a<<"/n";
}
};
int main()
{
Test<float>testf;
testf.add(5.23,6.43);
testf.show();
return 0;
}
```

OUTPUT:

```
11.66
```

•Make a class template what will show the addition and subtraction of two float numbers.

•Make a class template what will show the multiplication and division of two intergers numbers.

Class Template with multiple parameters:

•FORMAT:

```
template <class T1, class T2, ........>;
class class_name
{
..................
..................
.................. // Body of the class
..................
};
```

```
#include<iostream>
using namespace std;
template <class T1, class T2>
class Test
{
T1 a;
T2 b;
public:
Test (T1 x, T2 y)
{
a = x;
b = y;
}
void show()
{
cout<<a<<“and”<<b<<“\n”;
}
};
int main()
{
Test<float, int>;
test1(1.23,123);
Test<int,char>;
test2(100,’W’);
test1.show();
test2.show();
return 0;
}
```

OUTPUT:

```
1.23 and 123
100 and W
```

•Make a class template that will display three things (integer, float and character)

Cases study: container class and the standard template library

•**Standard Template Library (STL)** is a collection of standard C++ template classes.

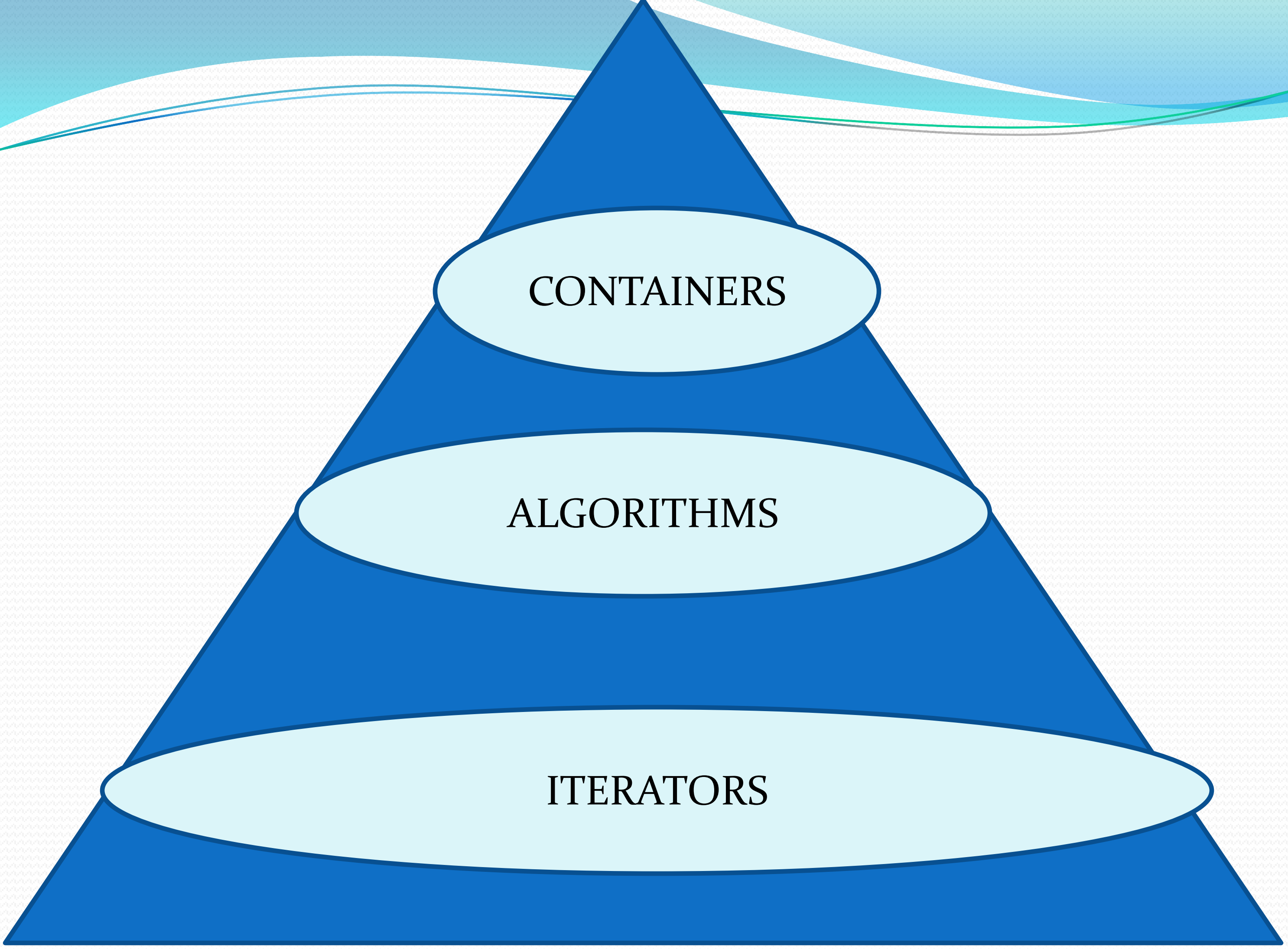
•It consists of generic methods/functions and classes to work with different forms of data.

•Standard Template Library is basically a generic library i.e. a single method/class can operate on different data types.

•So, as understood, we won't have to declare and define the same methods/classes for different data types.

Advantages of STL:

1. STL saves a lot of effort
2. STL **reduces** the redundancy of the code
3. STL leads to the **increased** optimization of the code blocks



COMPONENTS OF STL

Container class

•In Standard Template Library, Containers create and store data and objects.

•Containers can be described as the objects that hold the data of the same type.

•Containers are used to implement different data structures for example arrays, list, trees, etc.

•They are basically template-based generic classes.

Iterators

•As the name suggests, iterators are used for working upon a sequence of values.

•They are the major feature that allow generality in STL.

•They help the algorithms to apply on the container classes.

•They are very important in STL as they act as a bridge between algorithms and containers.

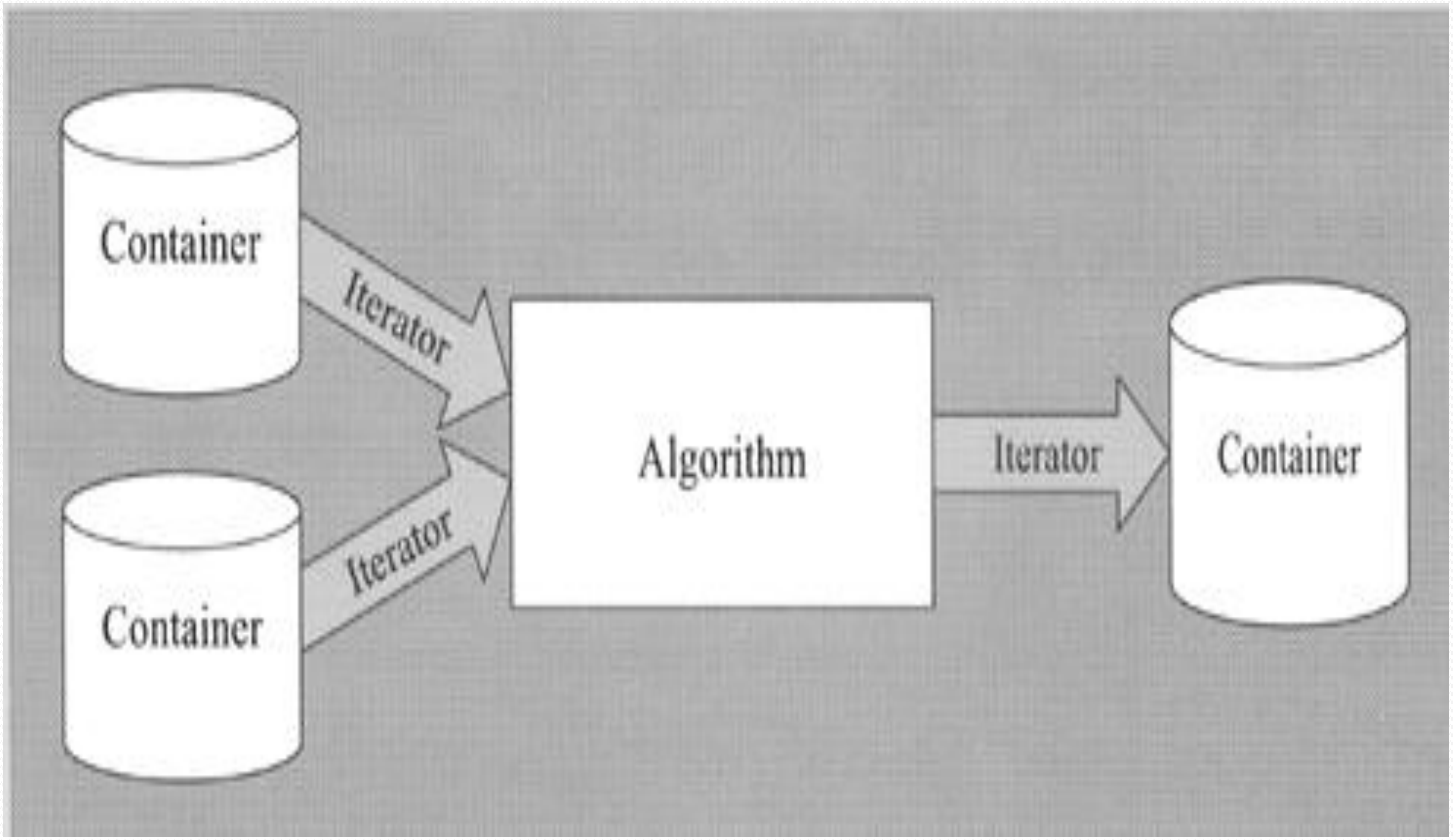
•Iterators always point to containers and in fact algorithms actually, operate on iterators and never directly on containers.

Algorithms

•Algorithms are the methods or functions that act on containers.

•By using algorithms provided by STL, we can have methods to search, sort, modify, transform or initialize the contents of container class objects.

•Algorithms provided by STL have built-in functions that can directly operate on complex data structure instead of having to write the algorithms ourselves.


Container
Iterator
Container
Iterator
Algorithm
Iterator
Container

EXCEPTION HANDLING

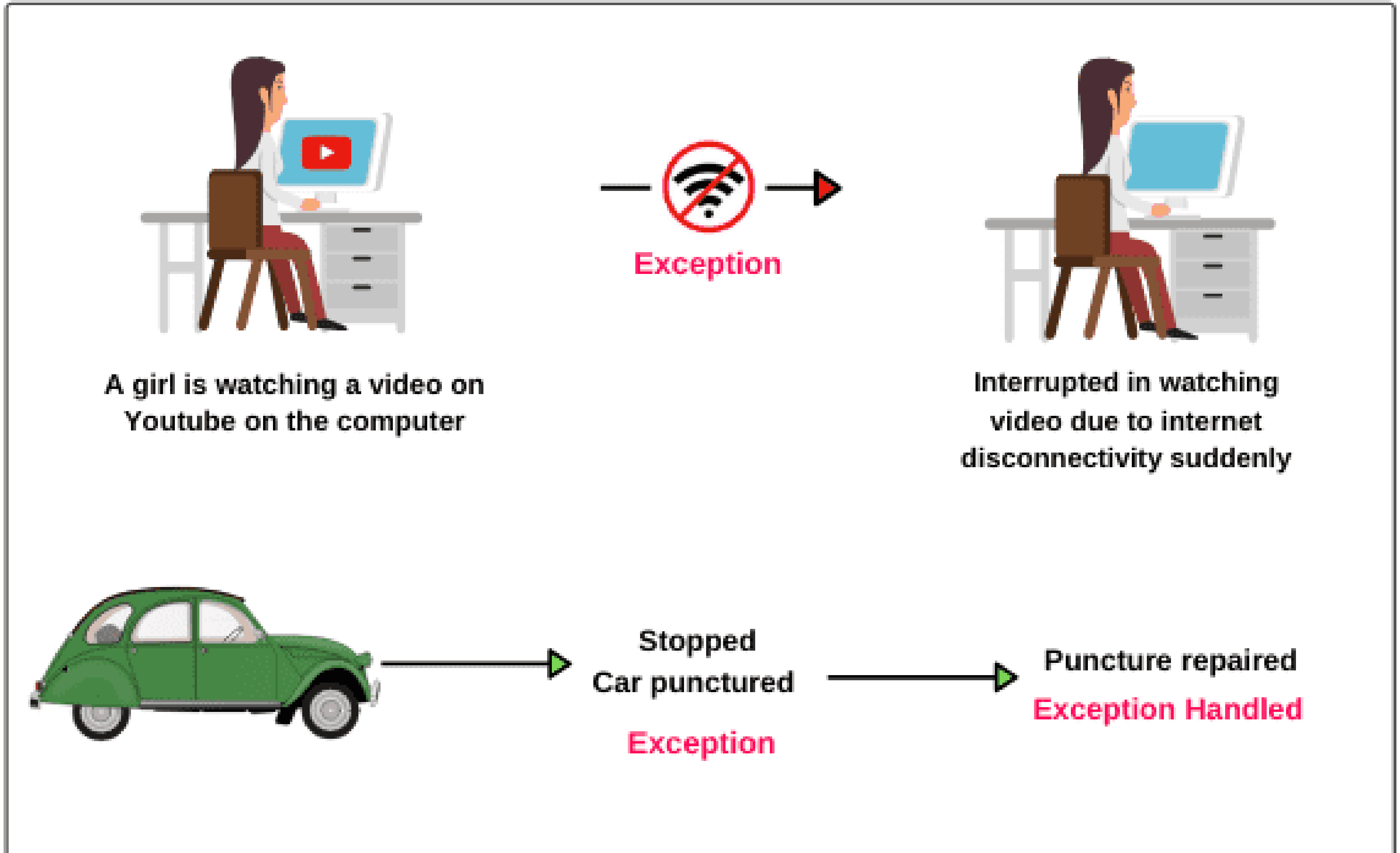
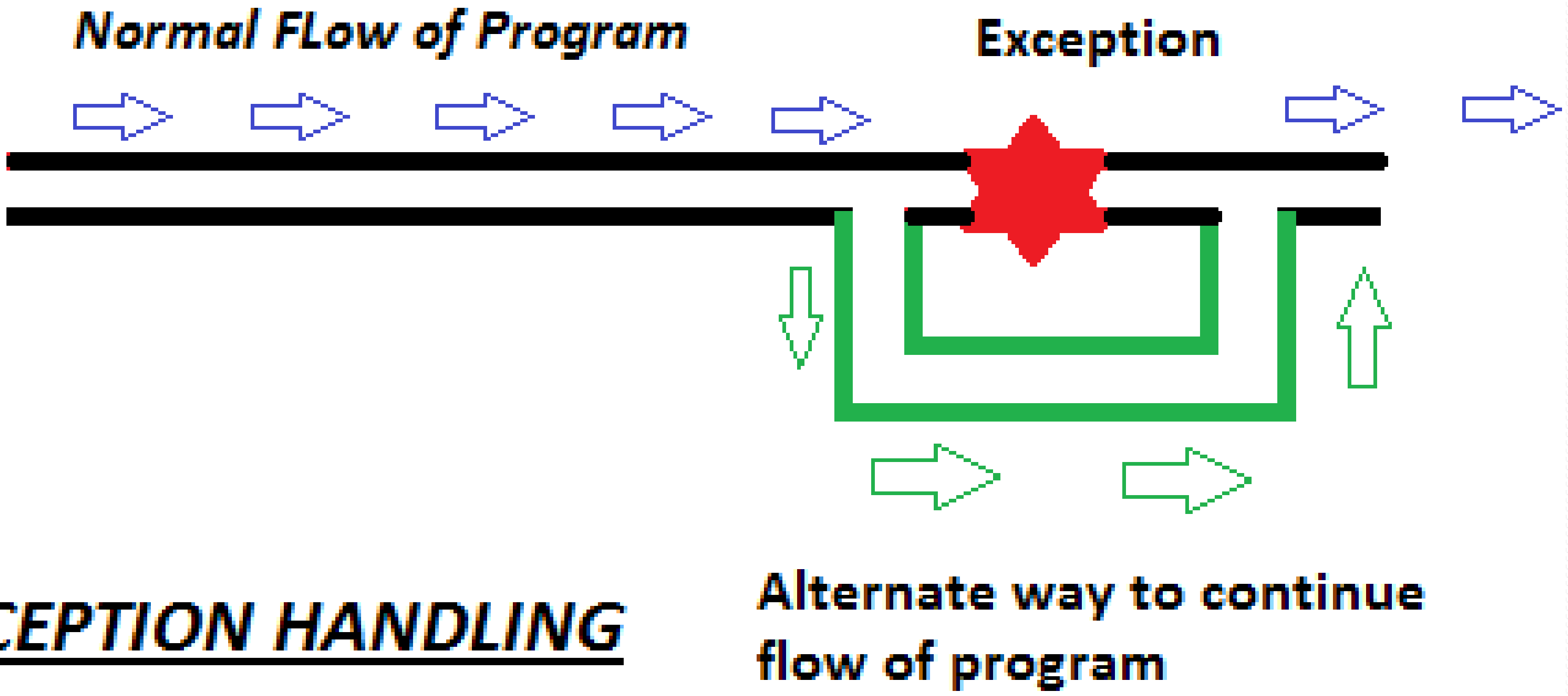

Exception
A girl is watching a video on Youtube on the computer
Interrupted in watching video due to internet disconnectivity suddenly
Stopped
Car punctured
Exception
Puncture repaired
Exception Handled

Fig: Realtime Example of Exception Handling



EXCEPTION HANDLING

•An exception is a problem that arises during the execution of a program.

•A C++ exception is a response to an exceptional circumstances that arises while a program is running such as an attenpt to divide by zero.

•Exception provides a way to transfer control from one part of a program to another.

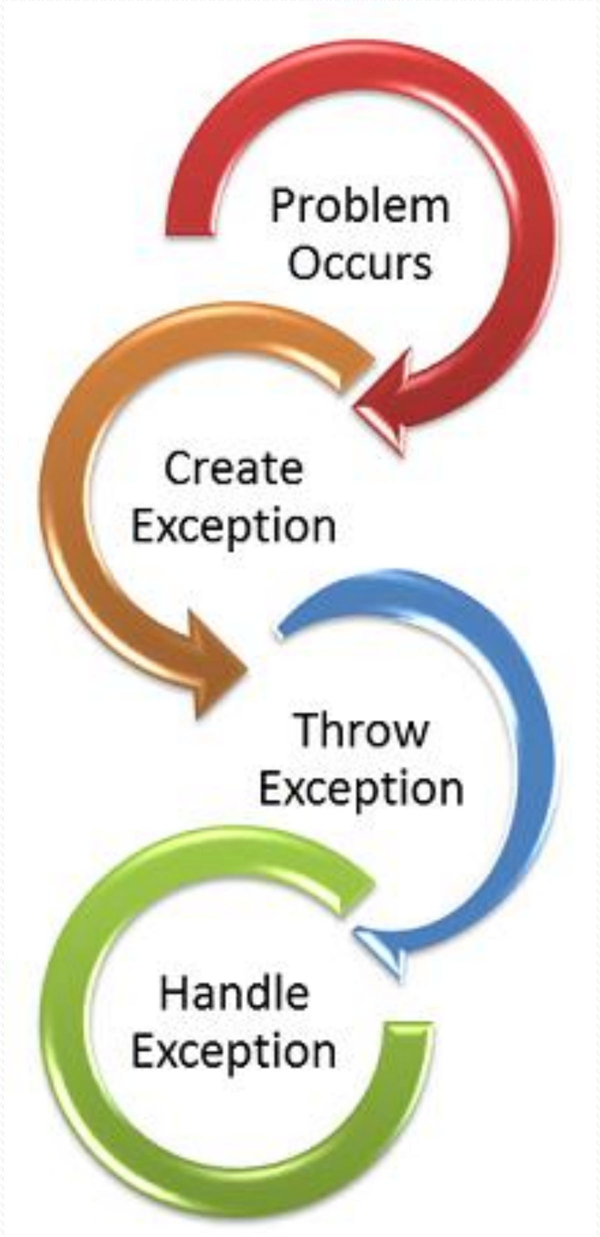
•C++ exception handling is built upon three keywords:
try, catch, throw

•The process of converting system error messages into user friendly error messages is called exceptional handling.

•This is one of the most important features of C++ to handle run time errors and maintain the normal flow of the program.

•EXCEPTION:

•An exception is an event , which occurs during the execution of a program that disrupts the normal flow of the program’s instruction.


Problem Occurs
Create Exception
Throw Exception
Handle Exception

a) *throw:* A program throws an exception when a problem shows up. This is done using a throw keyword.

b) *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.

c) *try:* A try block identifies a block of code for which particular exceptions will be activated. It is followed by one or more catch blocks.

1. try:

- *Try block consist of the program that may generate exception.*
- *Exceptions are thrown inside try block*
- *“try” block groups one or more program statements with one or more catch clauses.*

2. Throw:

- *"throw" keyword is used to throw the exception encountered inside of the try block.*
- *After the exception is thrown , the control is transferred to the catch block.*
- *There can be one or several throw statements used in exception handling.*

3. Catch:

- "catch" keyword is used to catch the exception case thrown from throw statement inside the try block.
- There can be one or several catch statements to catch the thrown statements used in exception handling.
- "catch" keyword is used to denote a catch statement.

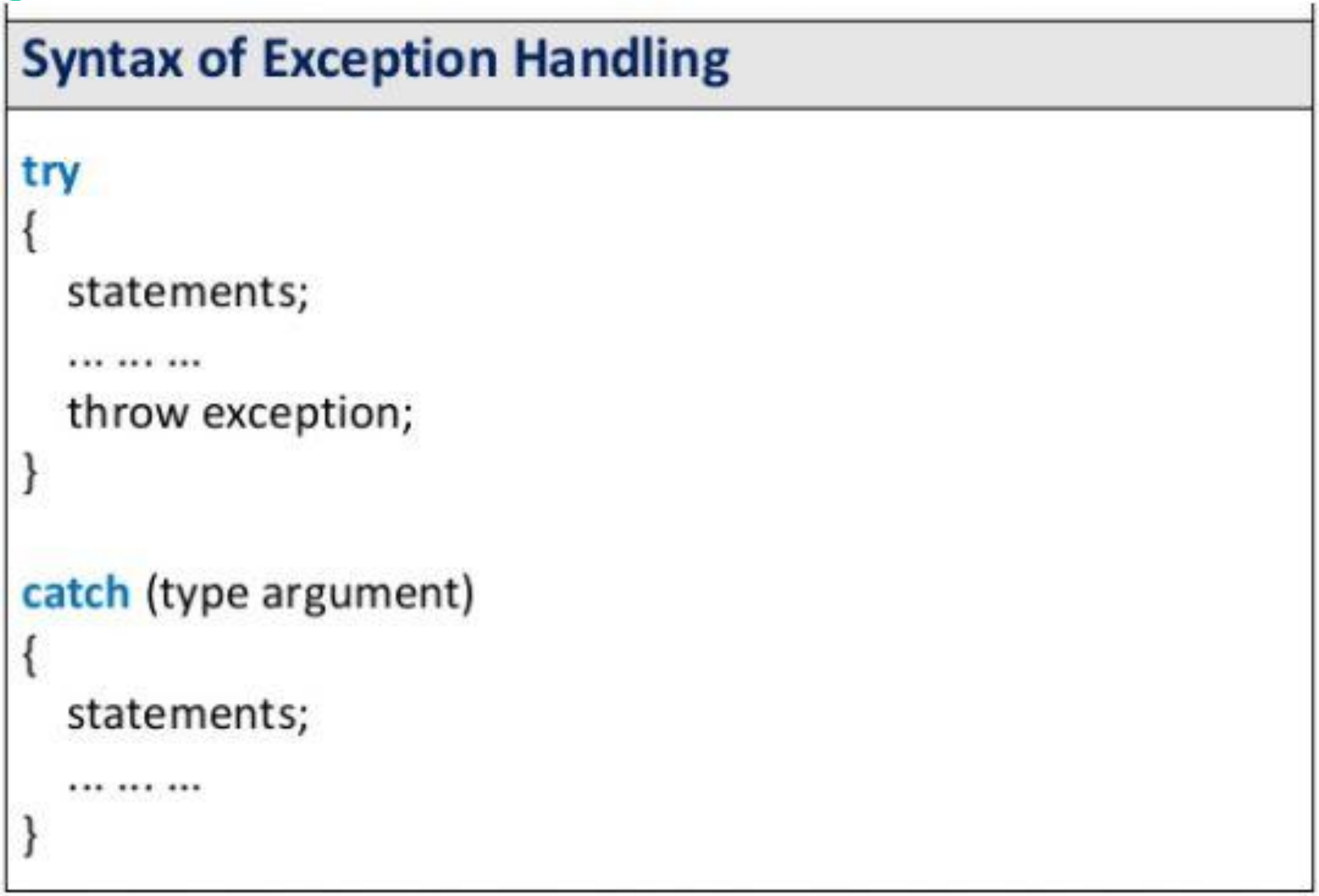
RULES :

- “try” block and “catch” block should be there together in any program.
- “throw” statements lie inside the try block.
- One or more “throw” blocks can exist according to the number of exceptions we want to handle.
- One or more “catch” blocks can exists according to the paths we want to define for given “throw” blocks.

The try and catch keywords come in pairs:

Example:

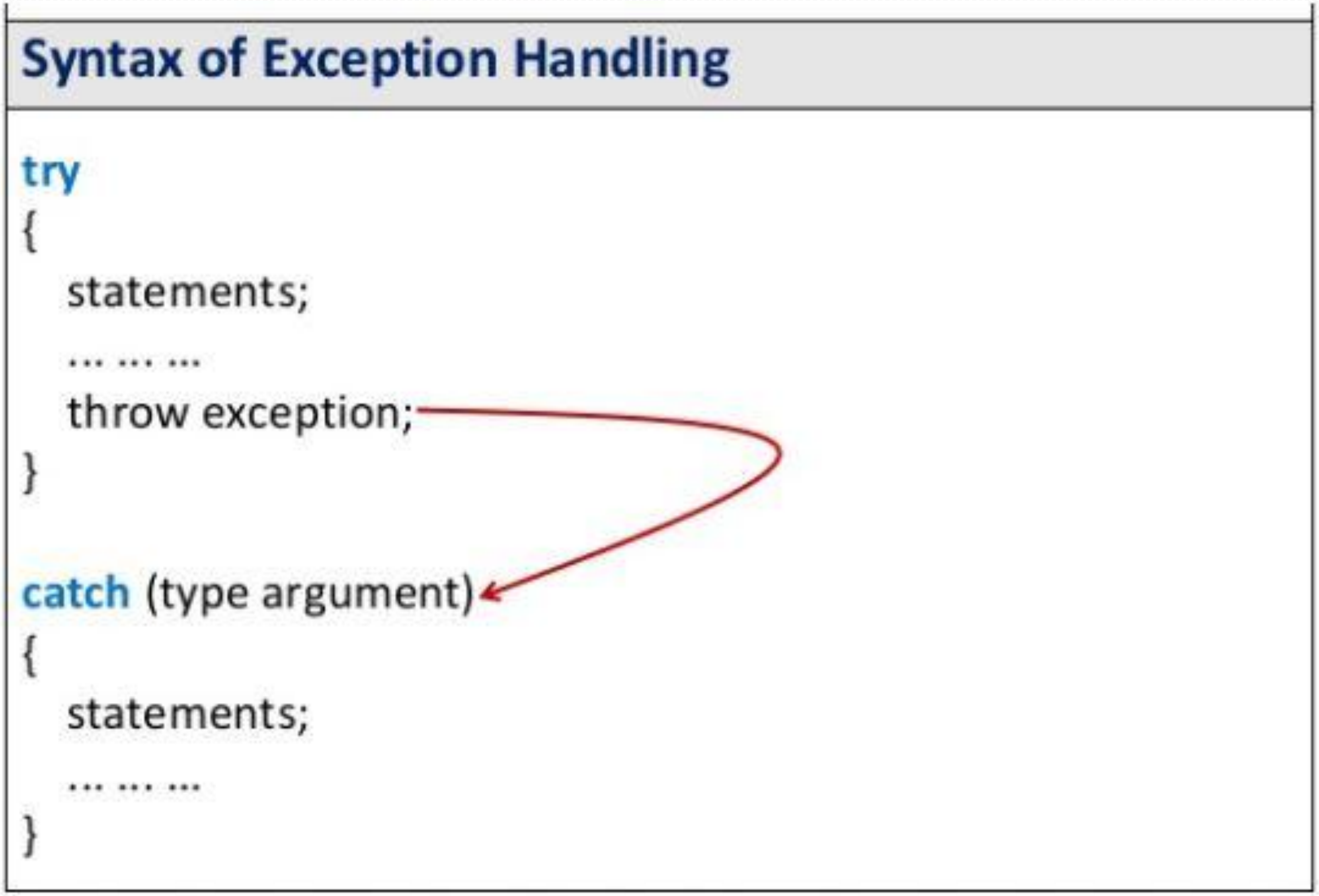
```
try {
  // Block of code to try
  throw exception; // Throw an exception when a problem arise
}
catch () {
  // Block of code to handle errors
}
```



Syntax of Exception Handling

```
try
{
   statements;
   ... ... ...
   throw exception;
}

catch (type argument)
{
   statements;
   ... ... ...
}
```

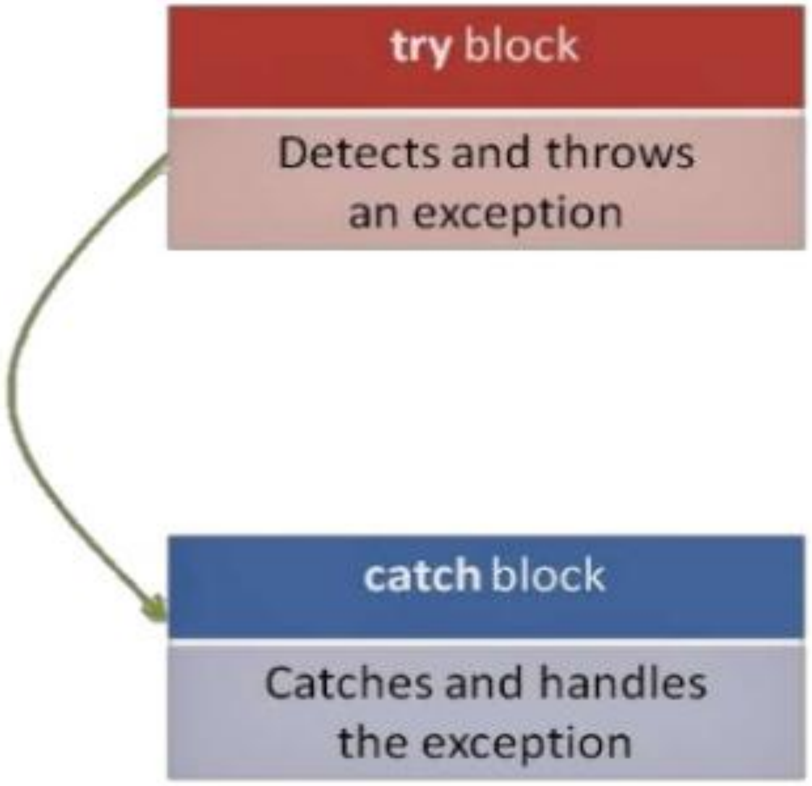


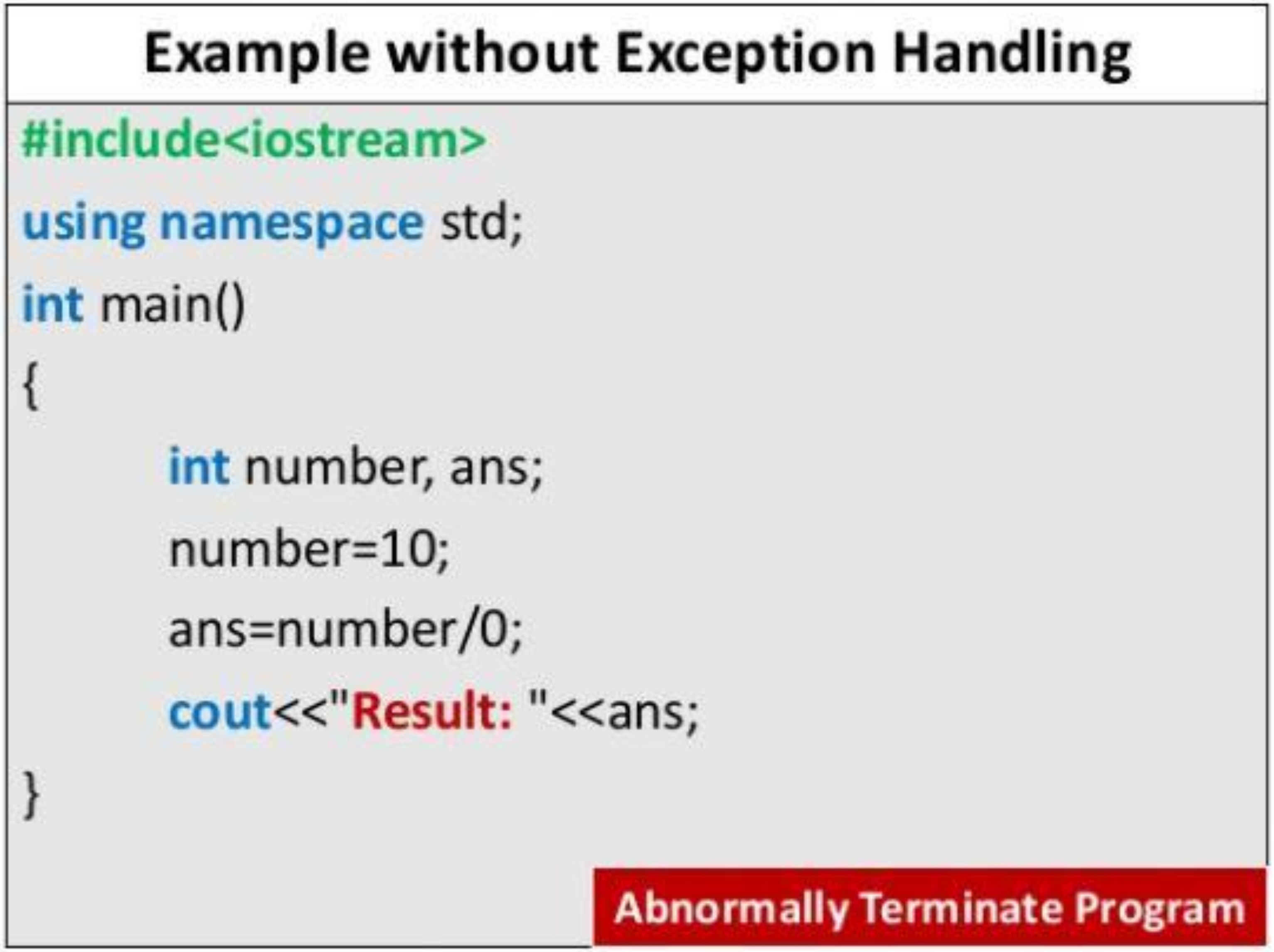
Syntax of Exception Handling

```
try
{
   statements;
   ... ... ...
   throw exception;
}

catch (type argument)
{
   statements;
   ... ... ...
}
```

Exception Handling in C++

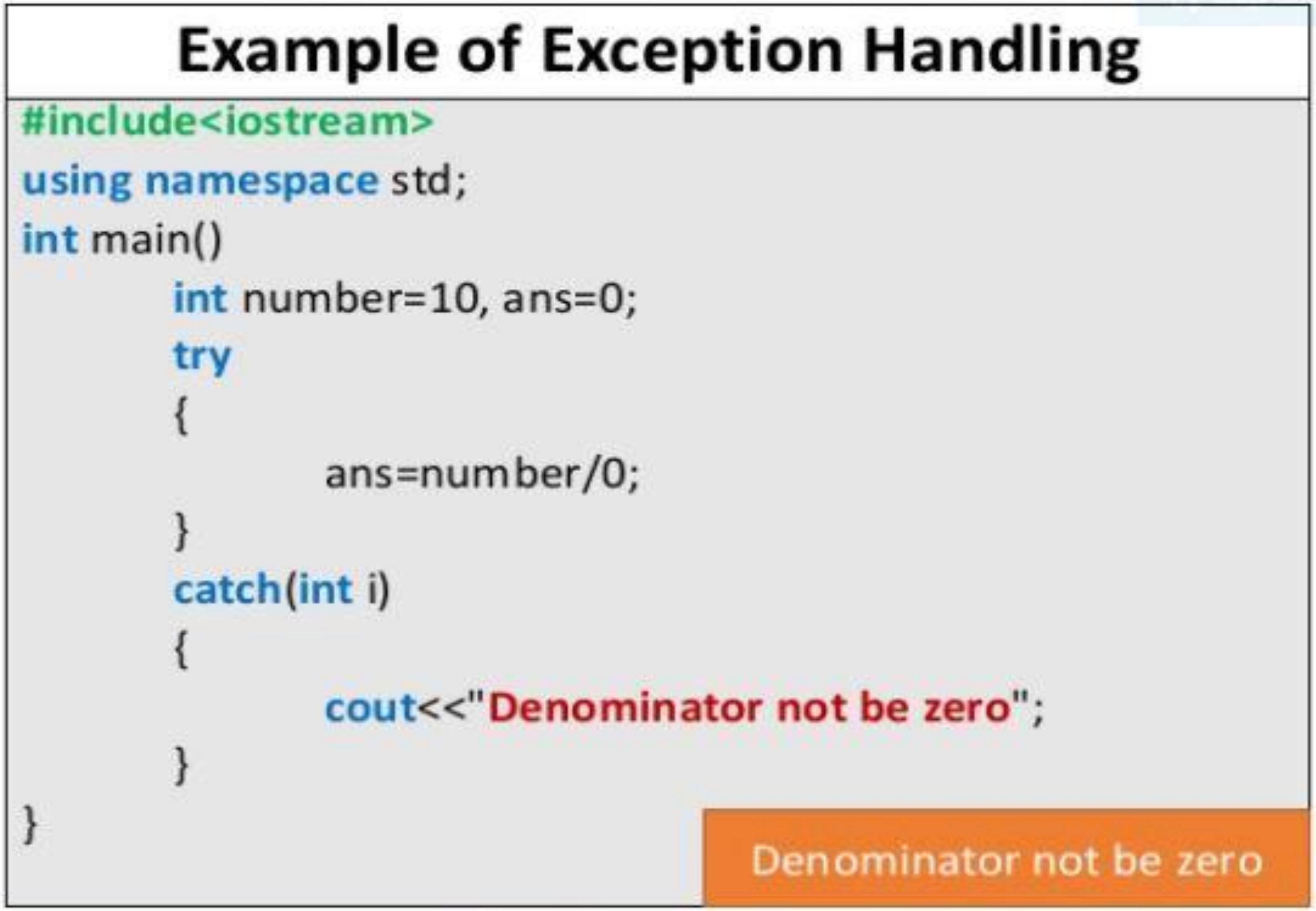

try block
Detects and throws an exception
catch block
Catches and handles the exception



Example without Exception Handling

```
#include<iostream>
using namespace std;
int main()
{
        int number, ans;
        number=10;
        ans=number/0;
        cout<<"Result: "<<ans;
}
```

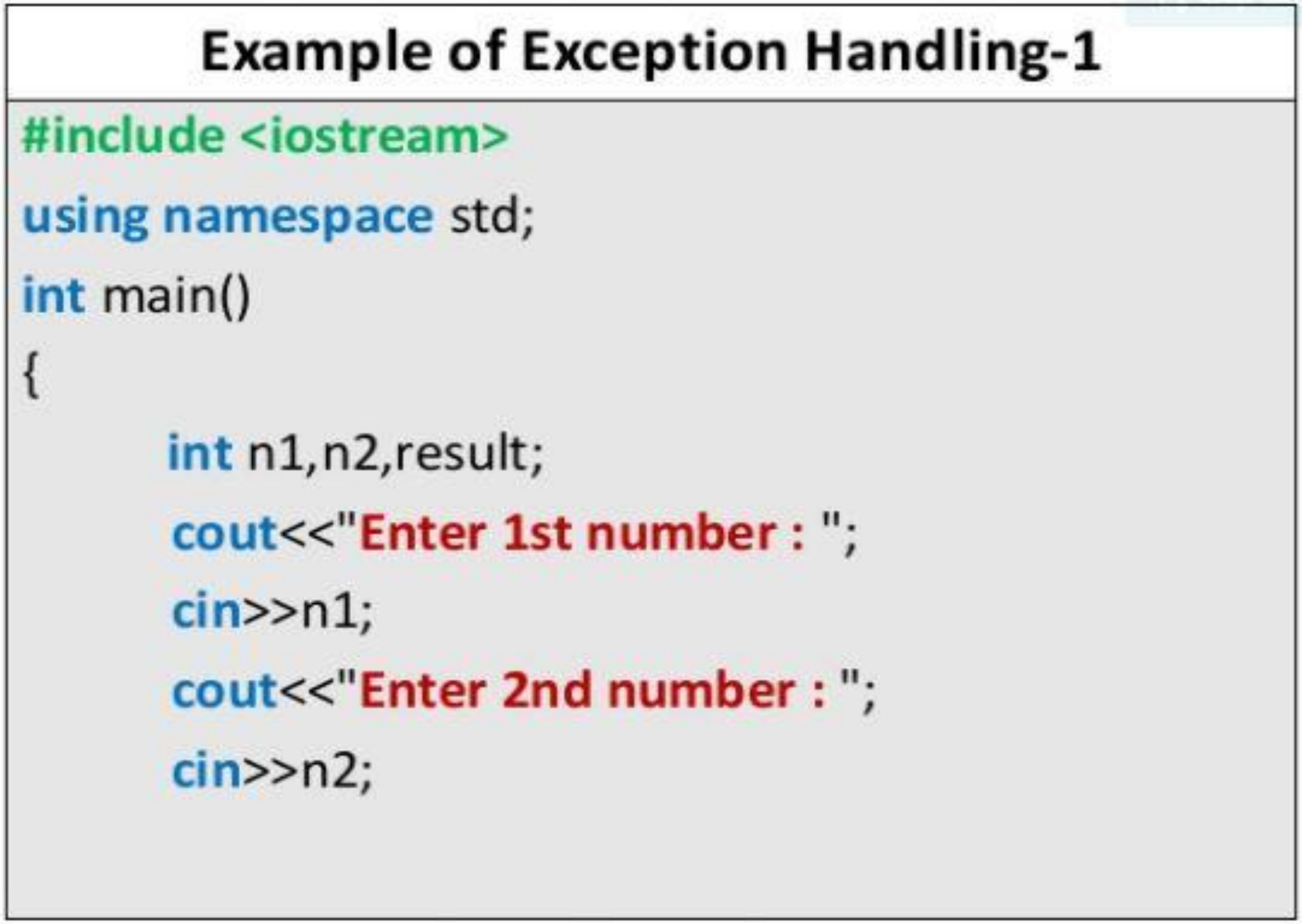
Abnormally Terminate Program



Example of Exception Handling

```
#include<iostream>
using namespace std;
int main()
          int number=10, ans=0;
          try
          {
                    ans=number/0;
          }
          catch(int i)
          {
                    cout<<"Denominator not be zero";
          }
}
```

```
Denominator not be zero
```



Example of Exception Handling-1

```
#include <iostream>
using namespace std;
int main()
{
        int n1,n2,result;
        cout<<"Enter 1st number : ";
        cin>>n1;
        cout<<"Enter 2nd number : ";
        cin>>n2;
```

Example of Exception Handling-2

```
  try {
      if(n2==0)
          throw n2;   //Statement 1
      else {
          result = n1 / n2;
          cout<<"\nThe result is : "<<result;
      }
  }
  catch(int x) {
      cout<<"\nCan't divide by : "<<x;
  }

cout<<"\nEnd of program.";

}
```

Output of the Previous Program

```
Enter 1st number : 45
Enter 2nd number : 0

Can't divide by : 0
End of program
```

```
Enter 1st number : 12
Enter 2nd number : 20

The result is : 0
End of program
```

Output of the Previous Program

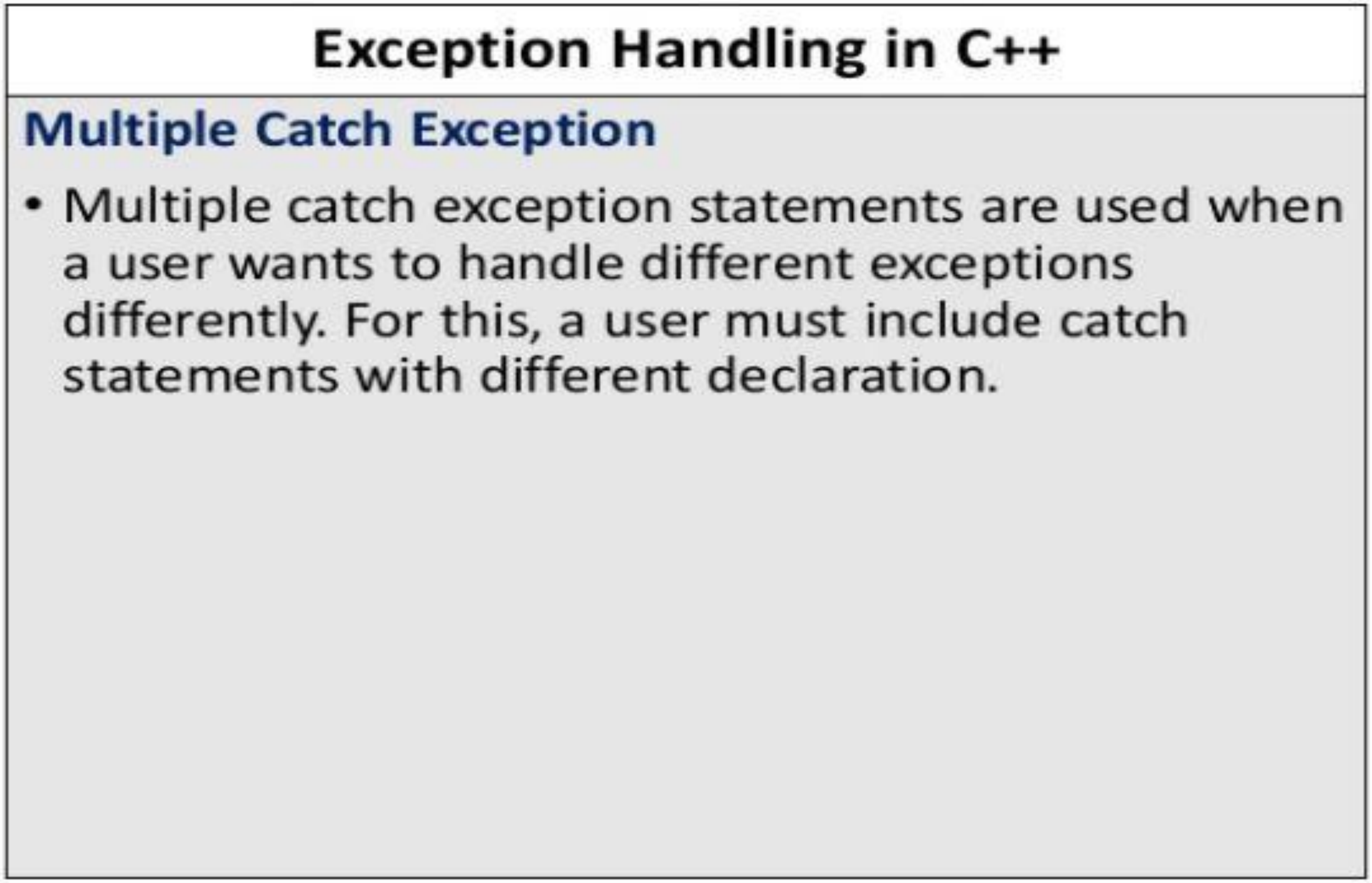
```
Enter 1st number : 45
Enter 2nd number : 0


Can't divide by : 0
End of program
```

```
Enter 1st number : 12
Enter 2nd number : 20


The result is : 0
End of program
```

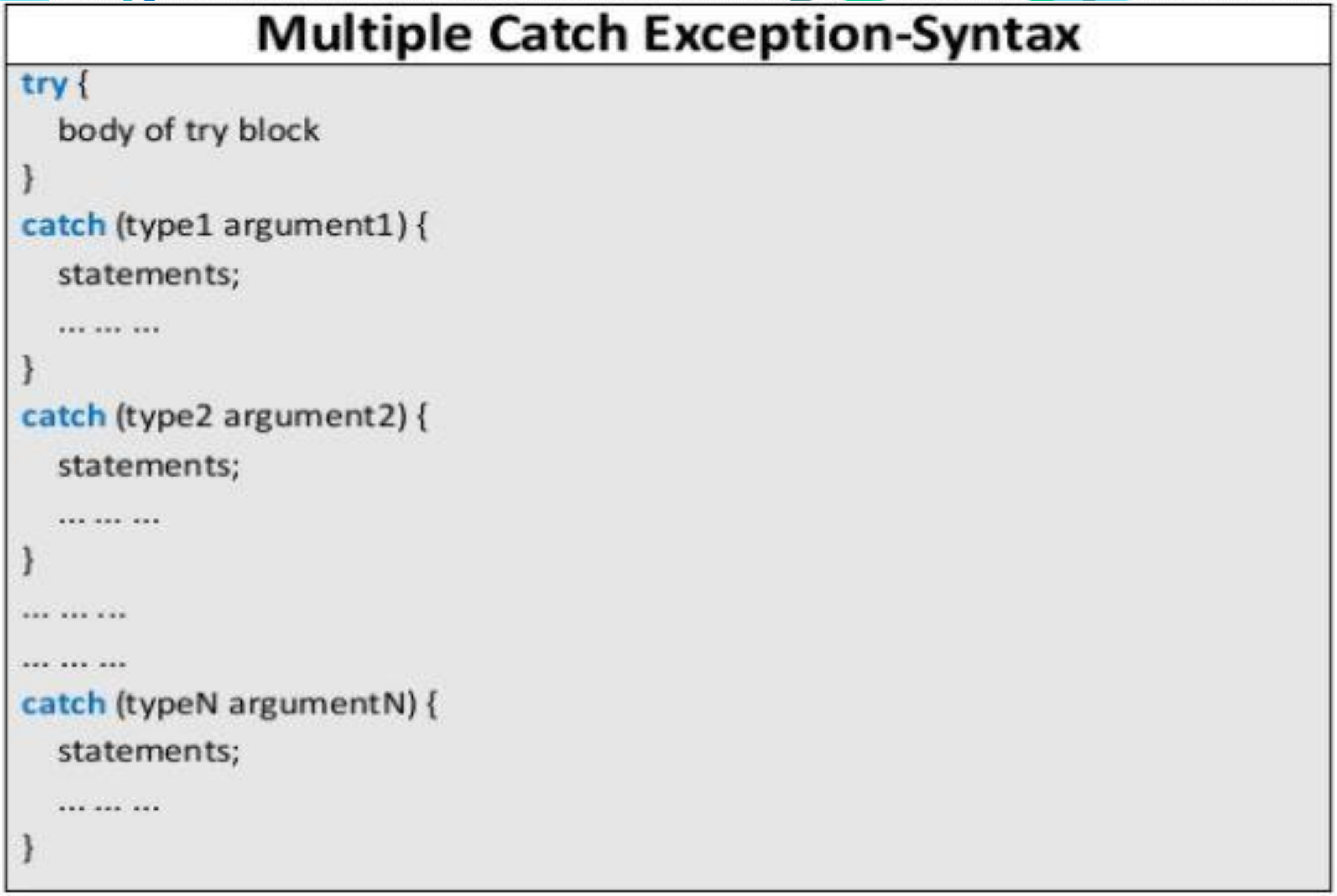
Why the answer is zero ??
Simple Answer is because we can’t handle this condition



Exception Handling in C++

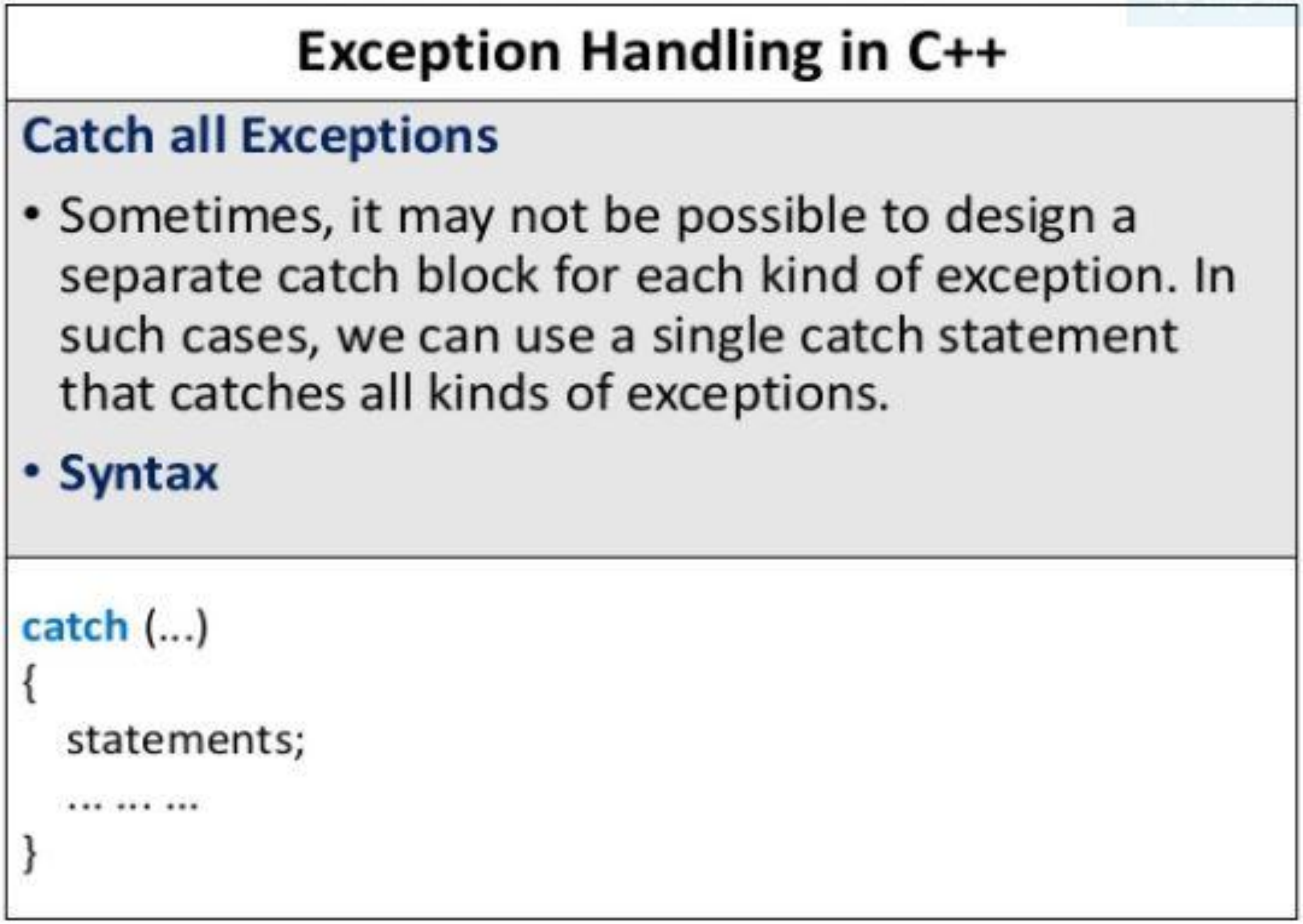
Multiple Catch Exception

- Multiple catch exception statements are used when a user wants to handle different exceptions differently. For this, a user must include catch statements with different declaration.



Multiple Catch Exception-Syntax

```
try {
    body of try block
}
catch (type1 argument1) {
    statements;
    ... ... ...
}
catch (type2 argument2) {
    statements;
    ... ... ...
}
... ... ...
... ... ...
catch (typeN argumentN) {
    statements;
    ... ... ...
}
```

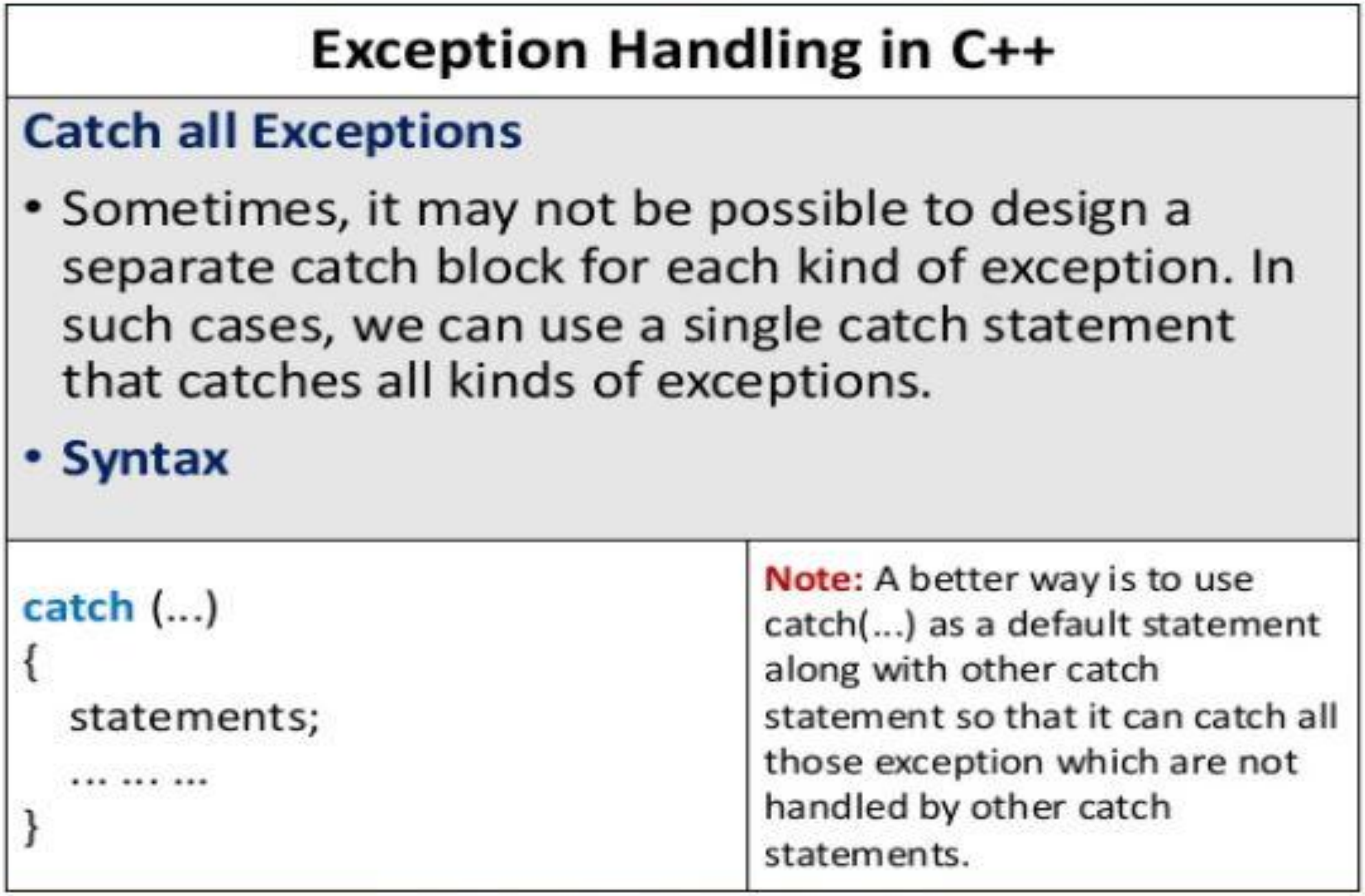


Exception Handling in C++

Catch all Exceptions

- Sometimes, it may not be possible to design a separate catch block for each kind of exception. In such cases, we can use a single catch statement that catches all kinds of exceptions.
- **Syntax**

```
catch (...)
{
   statements;
   ... ... ...
}
```



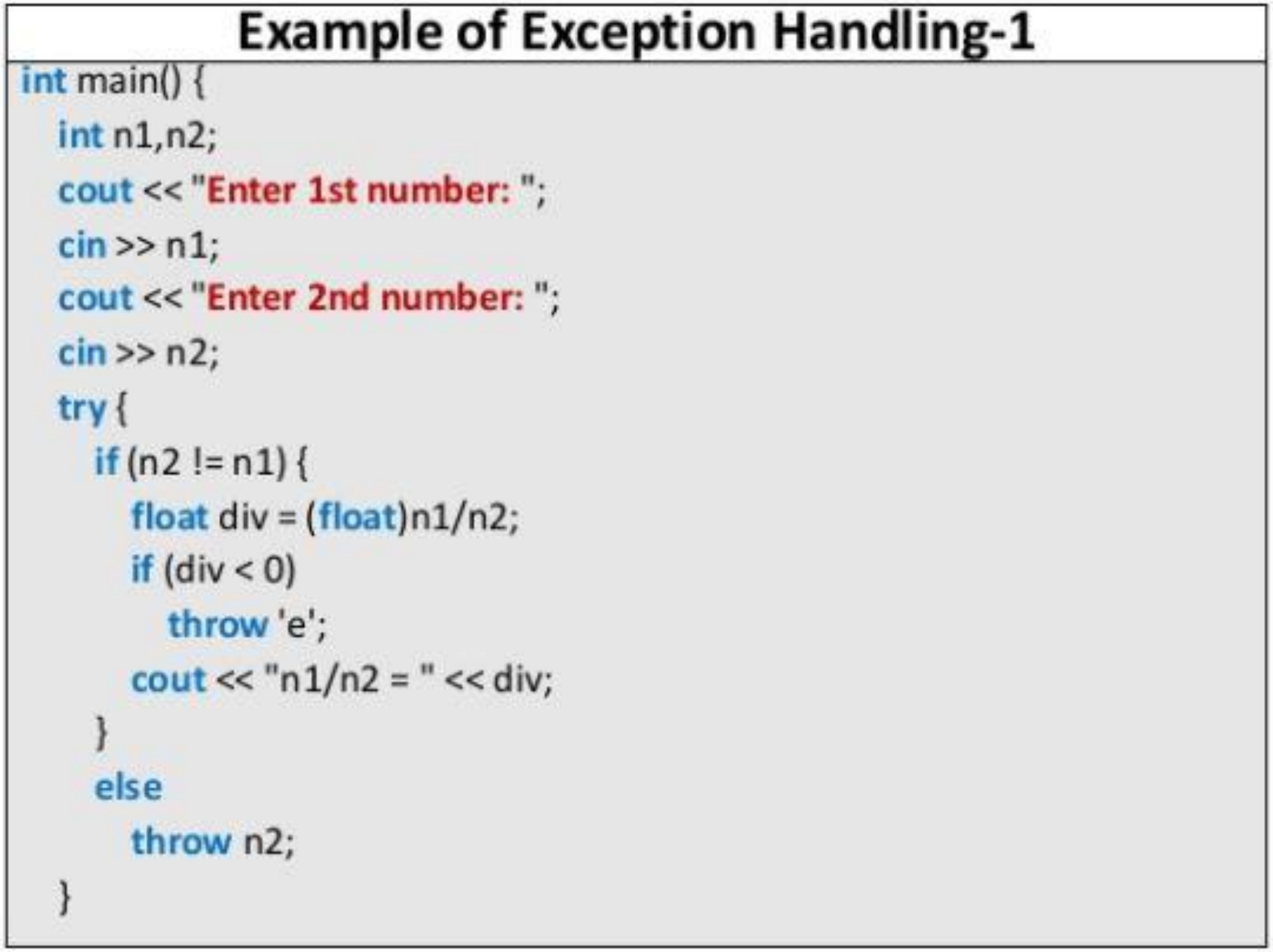
Exception Handling in C++

Catch all Exceptions

- Sometimes, it may not be possible to design a separate catch block for each kind of exception. In such cases, we can use a single catch statement that catches all kinds of exceptions.
- **Syntax**

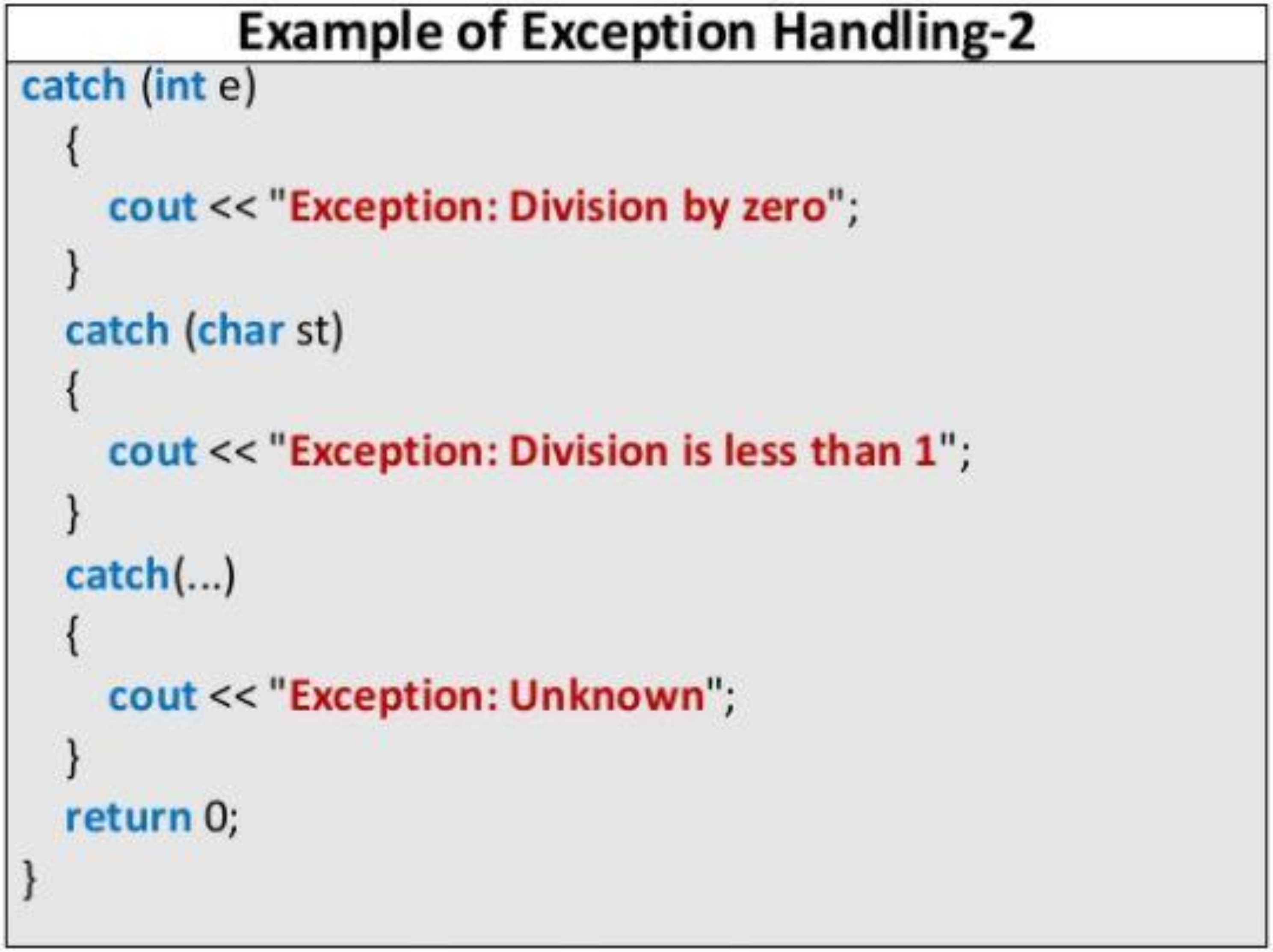
```
catch (...)
{
   statements;
   ... ... ...
}
```

Note: A better way is to use catch(...) as a default statement along with other catch statement so that it can catch all those exception which are not handled by other catch statements.



Example of Exception Handling-1

```
int main() {
  int n1,n2;
  cout << "Enter 1st number: ";
  cin >> n1;
  cout << "Enter 2nd number: ";
  cin >> n2;
  try {
    if (n2 != n1) {
      float div = (float)n1/n2;
      if (div < 0)
        throw 'e';
      cout << "n1/n2 = " << div;
    }
    else
      throw n2;
  }
```



Example of Exception Handling-2

```
catch (int e)
   {
      cout << "Exception: Division by zero";
   }
   catch (char st)
   {
      cout << "Exception: Division is less than 1";
   }
   catch(...)
   {
      cout << "Exception: Unknown";
   }
   return 0;
}
```

Output of the Previous Program

```
Enter 1st number: 20
Enter 2st number: 5
n1/n2 = 4
```

```
Enter 1st number: 5
Enter 2st number: 20
n1/n2 = 0.25
```

```
Enter 1st number: -1
Enter 2st number: 20
Exception: Division is less than 1
```

```
#include<iostream>
using namespace std;
int main(){
int a,b;
float division;
try{
cout<<"enter a and b"<<endl;
cin>>a>>b;
if(b == 0)
{
throw (0);
}     else
{
 cout<<"a/b="<<a/b<<endl;
}
}
catch(int n)
{
cout<<"You cannot divide by zero."<<endl;
}
return 0;
}
```

Consider the following example:

Example

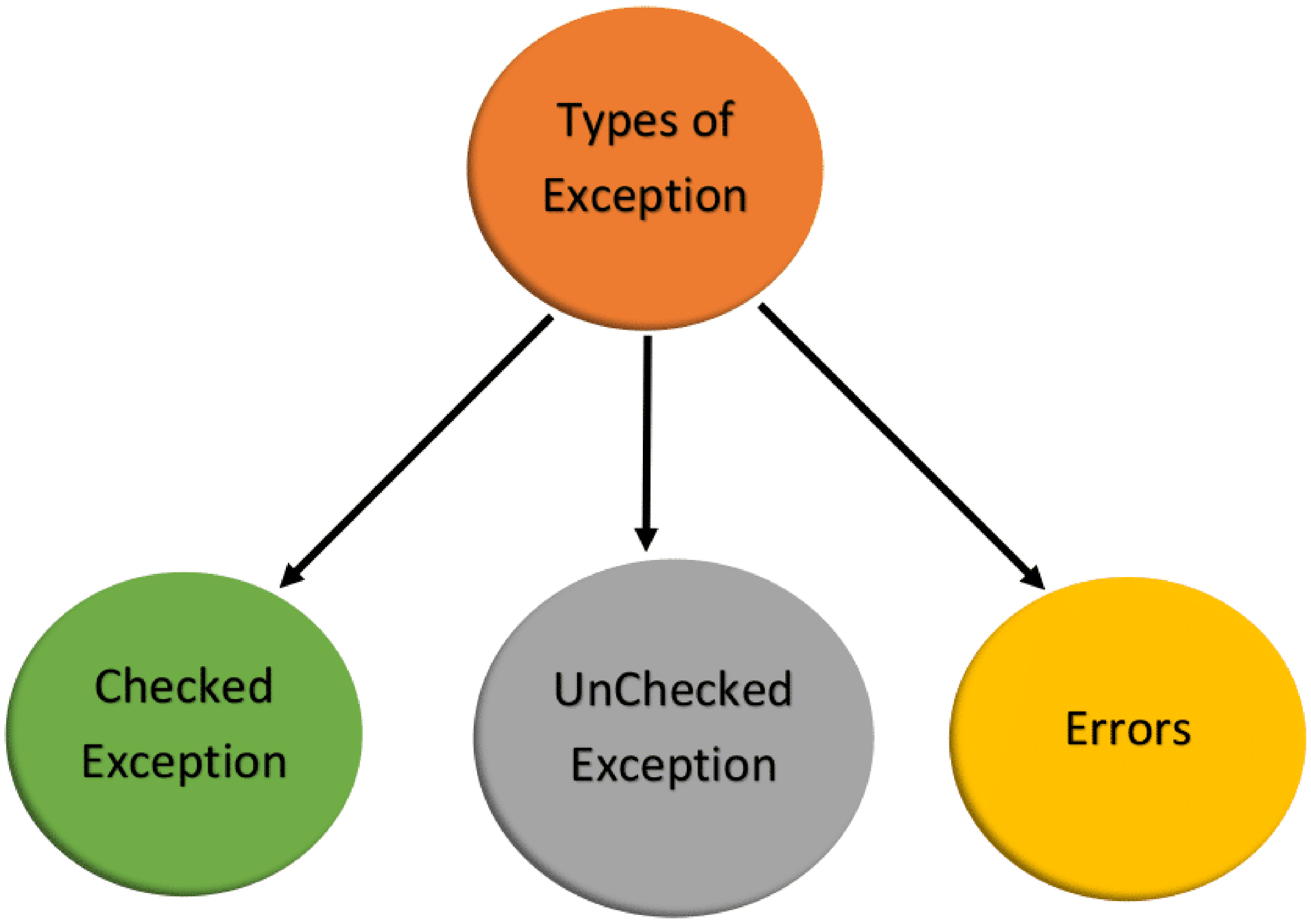
```
try {
  int age = 15;
  if (age >= 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw (age);
  }
}
catch (int myNum) {
  cout << "Access denied - You must be at least 18 years old.\n";
  cout << "Age is: " << myNum;
}
```

Example explained:

•We use the try block to test some code: If the age variable is less than 18, we will throw an exception, and handle it in our catch block.

•In the catch block, we catch the error and do something about it.

•The catch statement takes a **parameter**: in our example we use an int variable (myNum) (because we are throwing an exception of int type in the try block (age)), to output the value of age.
If no error occurs (e.g. if age is 20 instead of 15, meaning it will be be greater than 18), the catch block is skipped:


Types of Exception
Checked Exception
UnChecked Exception
Errors