SDF II(15B11CI211)

EVEN Semester 2021

2nd Semester , First Year

Jaypee Institute Of Information Technology (JIIT), Noida

Module 6: Introduction to Exceptions, Try, Catch and Throw

Contents

- Types of Errors
- Introduction to Exceptions, Try, Catch and Throw
- Re-throwing exceptions, Exception and Inheritance
- Case Study of Exceptions

Types of Errors

- Errors can be broadly categorized into two types.
  - Compile Time Errors
  - Run Time Errors
- Compile Time Errors – Errors caught during compiled time is called Compile time errors. Compile time errors include library reference, syntax error or incorrect class import.
- Run Time Errors - They are also known as exceptions.

Difference between Error and Exception handling

- Errors hinder normal execution of program.
- Exception handling is the process of handling errors and exceptions in such a way that they do not hinder normal execution of the system.
- For example, User divides a number by zero, this will compile successfully but an exception or run time error will occur due to which our applications will be crashed. In order to avoid this we can include exception handling in our code.

Exception Handling in C++

- The process of converting system error messages into user friendly error message is known as Exception handling.
- This is one of the powerful feature of C++ to handle run time error and maintain normal flow of C++ application.
- Exception : An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's Instructions.

Exception Handling in C++: Examples of exception

- Exceptions are runtime anomalies that a program encounters during execution.
- It is a situation where a program has an unusual condition and the section of code containing it can't handle the problem.
- Exception includes condition such as division by zero, accessing an array outside its bound, running out of memory, etc.
- In order to handle these exceptions, exception handling mechanism is used which identifies and deal with such condition.

Exception handling mechanism

- Exception handling mechanism consists of following parts:
- Find the problem (Hit the exception)
- Inform about its occurrence (Throw the exception)
- Receive error information (Catch the exception)
- Take proper action (Handle the exception)

Exception Handling in C++

- Handling the Exception : Handling the exception is converting system error message into user friendly error message.
- Three keywords for Handling the Exception in C++ Language, they are;
  - Try
  - catch
  - throw

Exception Handling in C++: try block

- try: Try block consists of the code that may generate exception. Exception are thrown from inside the try block.
- try: represents a block of code that can throw an exception.
- "try" block groups one or more program statements with one or more catch clauses.

Exception Handling in C++ : throw block

- throw: Throw keyword is used to throw an exception encountered inside try block. After the exception is thrown, the control is transferred to catch block.

- Raising of an exception is done by "throw" expression.

Exception Handling in C++: catch block

- catch: Catch block catches the exception thrown by throw statement from try block. Then, exception are handled inside catch block.

- catch :The catch block defines the action to be taken, when an exception occur.

Exception Handling in C++

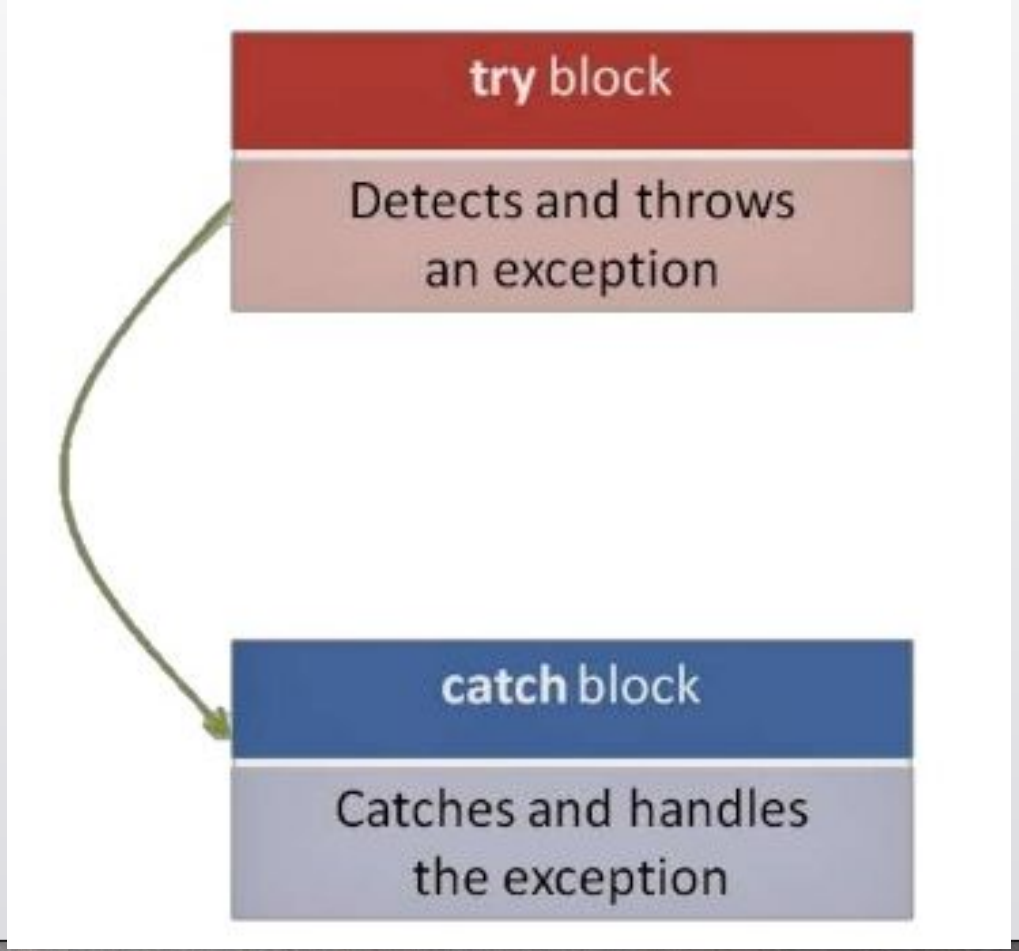
Syntax of Exception Handling

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

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

Exception Handling in C++

Syntax of Exception Handling

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

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



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;
   ... ... ...
}
```

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.

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

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.

```
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 without Exception Handling

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

Note: Abnormally terminates program

Catch block

- The catch block contain the code to handle exception.
- The catch block is similar to function definition.

```
catch(data-type arg)
{
            ----------
            ----------
            ----------
};
```

- Data-type specifies the type of exception that catch block will handle, Catch block will receive value, send by throw keyword in try block.

Multiple Catch Blocks


```
#include <iostream>
using namespace std;
int main() {
int a=2;
try {
if(a==1) throw a; //throwing integer exception
else if(a==2) throw 'A'; //throwing character exception
else if(a==3) throw 4.5; //throwing float exception
}
catch(int a) { cout<<"\nInteger exception caught."; }
catch(char ch) { cout<<"\nCharacter exception caught."; }
catch(double d) { cout<<"\nDouble exception caught."; }
cout<<"\nEnd of program."; }
```

```
#include <iostream>
using namespace std;
int main()
{
int a=2;
try {
if(a==1) throw a; //throwing integer exception
else if(a==2) throw 'A'; //throwing character exception
else if(a==3) throw 4.5; //throwing float exception
}
catch(int a) { cout<<"\nInteger exception caught."; }
catch(char ch) { cout<<"\nCharacter exception caught."; }
catch(double d) { cout<<"\nDouble exception caught."; }
cout<<"\nEnd of program."; }
```

```
/tmp/7CyGPjfUu1.o
Character exception caught.
End of program.
```

Example of Exception Handling

```
#include <iostream>
using namespace std;
int main() {
int n1,n2,result;
cout<<"Enter 1st number : ";
cin>>n1;
cout<<"Enter 2nd number : ";
cin>>n2;
try {
if(n2==0) throw n2; //Statement 1
else {
result = n1 / n2;
cout<<"The result is : "<<result;
}
} catch(int x)
{ cout<<"Can't divide by : "<<x;
} cout<<"\nEnd of program."; }
```

```
#include <iostream>
using namespace std;
int main() {
int n1,n2,result;
cout<<"Enter 1st number : ";
cin>>n1;
cout<<"Enter 2nd number : ";
cin>>n2;
try {
if(n2==0) throw n2; //Statement 1
else {
result = n1 / n2;
cout<<"The result is : "<<result;
}
} catch(int x)
{ cout<<"Can't divide by : "<<x;
} cout<<"\nEnd of program."; }
```

```
/tmp/1nV9ikmuCa.o
Enter 1st number : 20
Enter 2nd number : 5
The result is : 4
End of program.
```

```
#include <iostream>
using namespace std;
int main() {
int n1,n2,result;
cout<<"Enter 1st number : ";
cin>>n1;
cout<<"Enter 2nd number : ";
cin>>n2;
try {
if(n2==0) throw n2; //Statement 1
else {
result = n1 / n2;
cout<<"The result is : "<<result;
}
} catch(int x)
{ cout<<"Can't divide by : "<<x;
} cout<<"\nEnd of program."; }
```

```
/tmp/1nV9ikmuCa.o
Enter 1st number : 20
Enter 2nd number : 0
Can't divide by : 0
End of program.
```

Example Continued

```
#include <iostream>
using namespace std;
int main() {
int n1,n2,result;
cout<<"Enter 1st number : ";
cin>>n1;
cout<<"Enter 2nd number : ";
cin>>n2;
try {
if(n2==0) throw n2; //Statement 1
else {
result = n1 / n2;
cout<<"The result is : "<<result;
}
} catch(int x)
{ cout<<"Can't divide by : "<<x;
} cout<<"\nEnd of program."; }
```

```
/tmp/7CyGPjfUu1.o
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

Exercise

- Modify the above program to handle following exceptions using multiple catch blocks (Refer: Sample snapshot)
- 1. Exception : Division by zero
- 2. Exception: Division is less than 1
- 3. Exception : Unknown

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

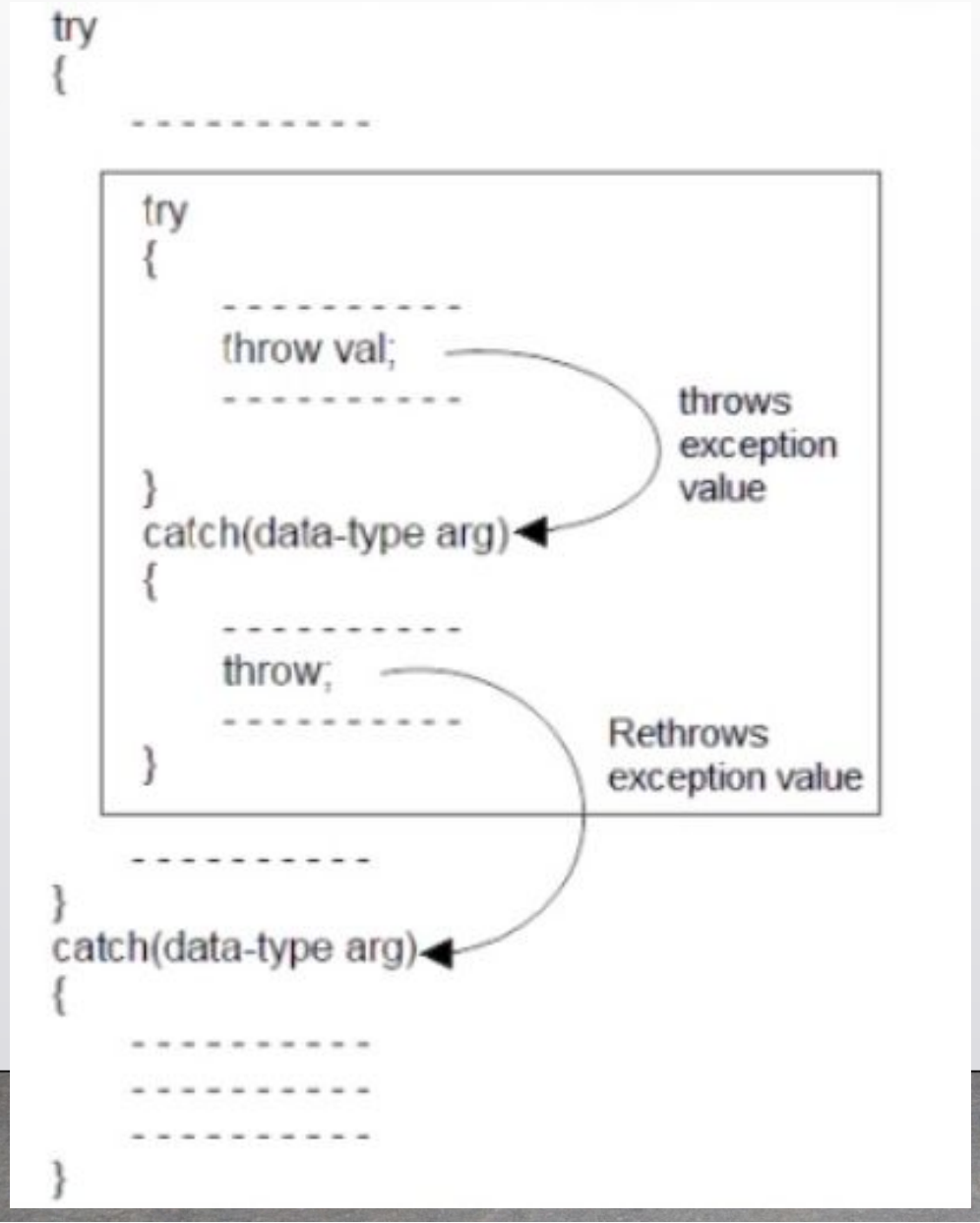
```
Enter 1st number: 5
Enter 2st number: 20
n1/n2 = 0.25
```

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

Rethrowing Exceptions

- Rethrowing exception is possible, where we have an inner and outer try-catch statements (Nested try-catch).
- An exception to be thrown from inner catch block to outer catch block is called rethrowing exception.

Syntax of Rethrowing Exceptions



Example


```
#include <iostream>
using namespace std;
int main()
{ int a=1;
try {
    try {
    throw a;
   }
catch(int x) {
cout<<"\nException in inner try-catch block.";
throw x; }
}
catch(int n) {
cout<<"\nException in outer try-catch block.";
}
cout<<"\nEnd of program.";
}
```

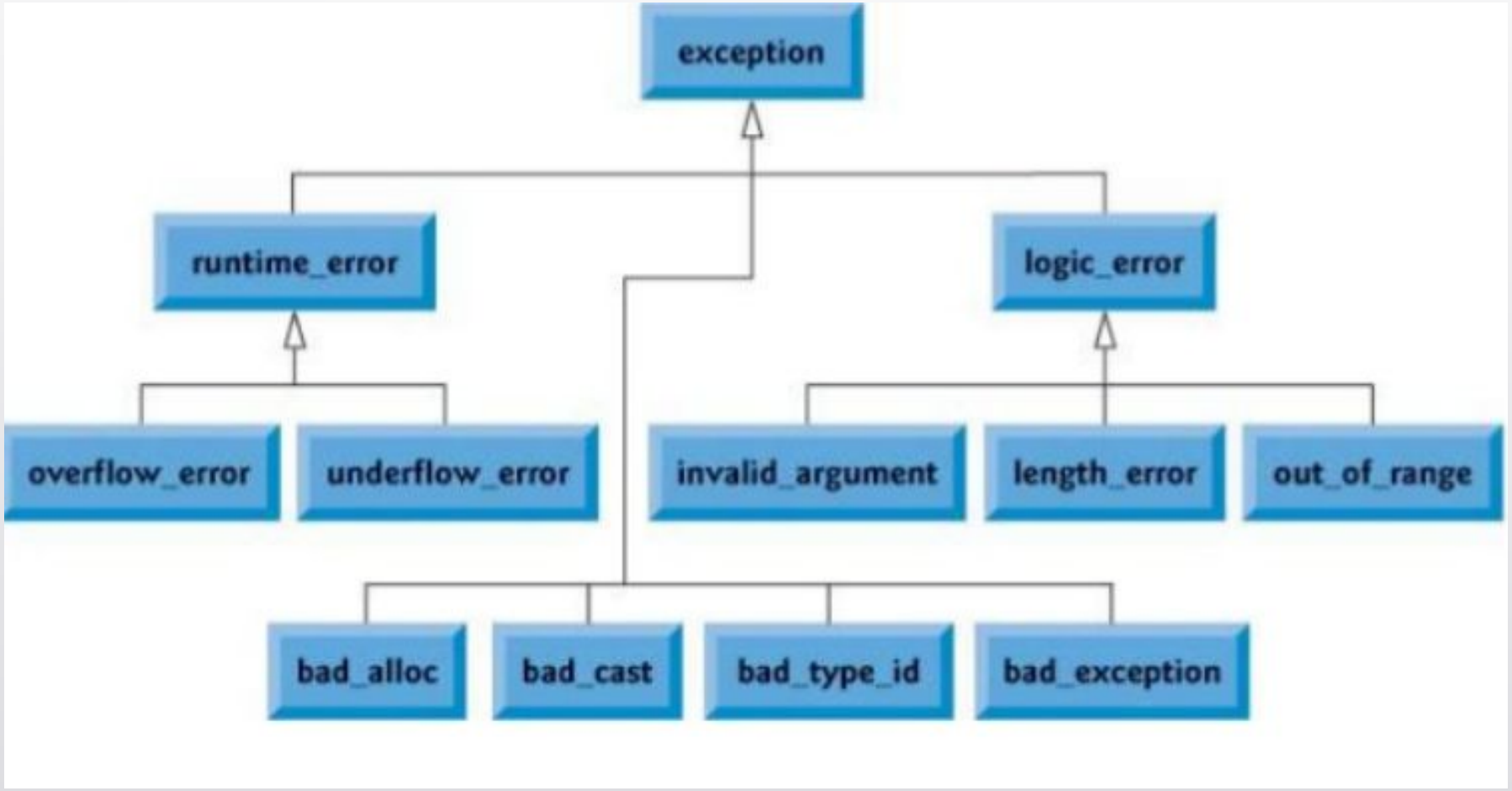
```
#include <iostream>
using namespace std;
int main()
{ int a=1;
try {
     try {
     throw a;
    }
catch(int x)
{
cout<<"\nException in inner try-catch block.";
throw x; }
}
catch(int n) {
cout<<"\nException in outer try-catch block.";
}
cout<<"\nEnd of program.";
}
```

```
/tmp/7CyGPjfUu1.o
Exception in inner try-catch block.
Exception in outer try-catch block.
End of program.
```

C++ Standard Exceptions

- C++ provides a list of standard exceptions defined in **<exception>** which we can use in our programs.
- These are arranged in a parent-child class hierarchy.
- C++ provides a range of built in exceptions. The base class for all exceptions classes is **exception**

C++ Standard Exceptions



C++ Standard Exceptions

exception	An exception and parent class of all the standard C++ exceptions.
bad_alloc	This can be thrown by new.
bad_cast	This can be thrown by dynamic_cast.
bad_exception	This is useful device to handle unexpected exceptions in a C++ program
bad_typeid	This can be thrown by typeid function.

logic_error	An exception that theoretically can be detected by reading the code.
domain_error	This is an exception thrown when a mathematically invalid domain is used
invalid_argument	This is thrown due to invalid arguments.
length_error	This is thrown when a too big std::string is created
out_of_range	This can be thrown by the at method from for example a vector and bitset<>::operator[]().

C++ Standard Exceptions Continued

runtime_error	An exception that theoretically can not be detected by reading the code.
overflow_error	This is thrown if a mathematical overflow occurs.
range_error	This is occured when you try to store a value which is out of range.
underflow_error	This is thrown if a mathematical underflow occurs.

Example


```
#include <iostream>
using namespace std;
int main()
{
const unsigned long SIZE = 10000;
//memory size
char* ptr; //pointer to memory
try
{
ptr = new char[SIZE]; //allocate SIZE bytes
}
catch(bad_alloc) //exception handler
{
cout << "\nbad_alloc exception: can't allocate
memory.\n";
return(1);
}
delete[] ptr; //deallocate memory
cout << "\nMemory use is successful.\n";
return 0;
}
```

Exception Handling and Inheritance

- In inheritance, while throwing exceptions of derived classes, care should be taken that catch blocks with base type should be written after the catch block with derived type.
- Otherwise, the catch block with base type catches the exceptions of derived class types too.

Example


```
#include <iostream>
using namespace std;
class Base {};
class Derived : public Base {};
int main() {
try {
throw Derived(); }
      catch(Base b) {
            cout<<"Base object caught"; }
      catch(Derived d) {
            cout<<"Derived object caught"; }
return 0;
}
```

```
// Online C++ compiler to run C++ program online
#include <iostream>
using namespace std;
class Base {};
class Derived : public Base {};
int main() {
try {
throw Derived(); }
catch(Base b) {
cout<<"Base object caught"; }
catch(Derived d) {
cout<<"Derived object caught"; }
return 0; }
```

```
/tmp/3u59qe1yCs.o
Base object caught
```

```
// Online C++ compiler to run C++ program online
#include <iostream>
using namespace std;
class Base {};
class Derived : public Base {};
int main() {
try {
throw Derived(); }
catch(Base b) {
cout<<"Base object caught"; }
catch(Derived d) {
cout<<"Derived object caught"; }
return 0; }
```

```
/tmp/3u59qe1yCs.o
Base object caught
```

Exception Handling and Inheritance

- In the previous program even though the exception thrown is of the type Derived it is caught by the catch block of the type Base.
- To avoid that we have to write the catch block of Basetype at last in the sequence

Example


```
#include <iostream>
using namespace std;
class Base {};
class Derived : public Base {};
int main() {
try {
throw Derived(); }
catch(Derived d) {
cout<<"Derived object caught"; }
catch(Base b) {
cout<<"Base object caught"; }
return 0; }
```

```
// Online C++ compiler to run C++ program online
#include <iostream>
using namespace std;
class Base {};
class Derived : public Base {};
int main() {
try {
throw Derived(); }
catch(Derived d) {
cout<<"Derived object caught"; }
catch(Base b) {
cout<<"Base object caught"; }
return 0; }
```

```
/tmp/3u59qe1yCs.o
Derived object caught
```

Case Studies on Exception Handling

Example-1


```
#include <iostream>
using namespace std;
class Distance
{  private:
    int feet;
    float inches;
public:
Distance() //constructor (no args)
{ feet = 0; inches = 0.0; }
Distance(int ft, float in) //constructor (two args)
{
if(in >= 12.0) //if inches too big,
     throw "inches value is too large."; //throw exception
feet = ft;
inches = in;
}
void getdist() //get length from user
{
cout << "\nEnter feet:" ;
cin >> feet;
cout << "Enter inches:" ;
cin >> inches;
if(inches >= 12.0) //if inches too big,
 throw "inches value is too large."; //throw
exception
}
void showdist() //display distance
{ cout << feet << "-" << inches; }
};
```

```
int main()
{
try
{
Distance dist1(17, 3.5); //2-arg constructor
Distance dist2; //no-arg constructor
dist2.getdist(); //get distance from user
//display distances
cout << "\ndist1 = ";
dist1.showdist();
cout << "\ndist2 =";
dist2.showdist();
}
catch(const char *str) //catch exceptions
{
cout << "Initialization error:"<<str;
}
cout << endl;
return 0;
}
```

```
Enter feet:12
Enter inches:22
Initialization error:inches value is too large.
```

Example-2


```
#include <iostream>
using namespace std;
#include <string.h>
#include <stdlib.h>
const int SZ=50;
class String //user-defined string type
{
private:
    char str[SZ]; //holds a string
public:

String()
{ strcpy(str, ""); }

String( const char s[]) {
    if( strlen(s)>80)
       throw "String Overflow";
    strcpy(str, s);
}
void display() const //display the String
{ cout << str; }


String operator + (String ss) const //add Strings
{
String temp; //make a temporary String
if( strlen(str) + strlen(ss.str) < SZ )
{
strcpy(temp.str, str); //copy this string to temp
strcat(temp.str, ss.str); //add the argument string
}
else
{ throw 1;}
return temp; //return temp String
}
```

```
int main()
{

try{
String s1("\nMerry Christmas!");
String s2("Happy new year!“);
String s3; //uses constructor 1
s1.display(); //display strings
s2.display();
s3 = s1 + s2; //add s2 to s1,
//assign to s3
s3.display(); //display s3
cout << endl;
String s4("What a wonderful day it is!!!!, Enjoy!!!!!");
String s5;
s5=s3+s4;
s5.display();
}
catch(const char* str)
{
   cout<<"Initialization Error -"<<str;
}
catch(int i)
{
  cout<<"Concatenation cannot be performed, string overflow";
}
return 0;
}
```

OUTPUT

Merry Christmas!Happy new year!
Merry Christmas!Happy new year!
Concatenation cannot be performed, string overflow

References

- https://www.slideshare.net/AdilAslam4/exception-handling-in-c-69353237
- https://www.tutorialspoint.com/cplusplus/pdf/cpp_exceptions_handling.pdf
- C, the complete reference Book by Herbert Schildt