**Exception Handling in C++**

**1)** Normal process.

|  |
| --- |
| #include <iostream>  using namespace std;    int main()  {     int x = -1;       // Some code     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;  } |

Output:

Before try

Inside try

Exception Caught

After catch (Will be executed)

**2)** ‘catch all’ catch(…) that can be used to catch all types of exceptions.

|  |
| --- |
| #include <iostream>  using namespace std;    int main()  {      try  {         throw 10;      }      catch (char excp)  {          cout << "Caught " << excp;      }      catch (...)  {          cout << "Default Exception\n";      }      return 0;  } |

Output:

Default Exception

**3)** Implicit type conversion doesn’t happen for primitive types. ‘a’ is not implicitly converted to int so catch all is executed

|  |
| --- |
| #include <iostream>  using namespace std;    int main()  {      try  {         throw 'a';      }      catch (int x)  {          cout << "Caught " << x;      }      catch (...)  {          cout << "Default Exception\n";      }      return 0;  } |

Output:

Default Exception

**4)** 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:

(run time error – some message like this)

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.

**5)**In C++, try-catch blocks can be nested. Also, an exception can be re-thrown using “throw; ”

|  |
| --- |
| #include <iostream>  using namespace std;    int main()  {      try {          try  {              throw 20;          }          catch (int n) {               cout << "Handle Partially ";               throw;   //Re-throwing an exception          }      }      catch (int n) {          cout << "Handle remaining ";      }      return 0;  } |

Output:

Handle Partially Handle remaining

A function can also re-throw a function using same “throw; “. A function can handle a part and can ask the caller to handle remaining.

**6)**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 testInstance;      throw 10;    } catch(int i) {      cout << "Caught " << i << endl;    }  } |

Output:

Constructor of Test

Destructor of Test

Caught 10

**7)**Multiple exceptions.

#include<iostream>

#include <string>

using namespace std;

class DivideByZero

{

public:

DivideByZero(string m) {msg = m;}

string msg;

};

void method1();

void method2();

void method3();

void method4();

int main()

{

try{

method3();

}

catch (DivideByZero input)

{

cout<<input.msg<<endl;

}

catch (string all)

{

cout<<all<<endl;

}

return 0;

}

void method1()

{

method2();

}

void method2()

{

char ch;

cout << “string exception?” << endl;

cin << ch;

if (ch == ‘Y’|| ch == ‘y’) throw string ("hello class\n");

method4();

}

void method3()

{

method1();

}

void method4()

{

throw DivideByZero("divided by zero in method4”);

}

Output: hello class 🡸 ch = y or Y

Divided by zero in method4 🡸if ch not = y or Y

8) Some standard exceptions:

bad\_alloc <new> failure to allocate storage

bad\_array\_new\_length <new> invalid array length

bad\_cast <typeinfo> execution of an invalid dynamic-cast

bad\_exception <exception> signifies an incorrect exception was thrown

bad\_function\_call <functional> thrown by "null" function

bad\_typeid <typeinfo> using typeinfo on a null pointer

bad\_weak\_ptr <memory> constructing a shared\_ptr from a bad weak\_ptr

domain\_error <stdexcept> parameter outside the valid range

invalid\_argument <stdexcept> invalid argument

length\_error <stdexcept> length exceeds its maximum allowable size

out\_of\_range <stdexcept> argument value not in its expected range

runtime\_error <stdexcept> errors detectable when the program executes

overflow\_error <stdexcept> arithmetic overflow error.

underflow\_error <stdexcept> arithmetic underflow error.

range\_error <stdexcept> range errors in internal computations

regex\_error <regex> errors from the regular expression library.

system\_error <system\_error> from operating system or other C API

failure <ios> Input or output error

// try block around call to division

double division(int a, int b)

{

if( b == 0 ) {

throw "Division by zero condition!";

}

return (a/b);

}

catch(string msg) {cout << msg << endl;} // after try block

------------------------------------------------------------------------------------

double division (int numerator, int denominator)

{

if (denominator == 0)

throw std::overflow\_error("Divide by zero exception!");

return numerator / denominator;

}

try {

i = division (10, 0);

}

catch (std::overflow\_error e)

{

cout << e.what() << endl;

}

Output:

Divide by zero exception!

Unable to allocate memory – no throw …..

try {

.....

int\* a = new int;

.....

}

catch(bad\_alloc ex)

{

cerr << "Out of memory!";

exit(1);

}