* In C++, exceptions are runtime anomalies or abnormal conditions that a program encounters during its execution. The process of handling these exceptions is called exception handling. Using the exception handling mechanism, the control from one part of the program where the exception occurred can be transferred to another part of the code.  
  So basically using exception handling in C++, we can handle the exceptions so that our program keeps running.  
  **What is a C++ Exception?**An exception is an unexpected problem that arises during the execution of a program our program terminates suddenly with some errors/issues. Exception occurs during the running of the program (runtime).  
  **Types of C++ Exception**There are two types of exceptions in C++
  + **Synchronous:** Exceptions that happen when something goes wrong because of a mistake in the input data or when the program is not equipped to handle the current type of data it’s working with, such as dividing a number by zero.
  + **Asynchronous**: Exceptions that are beyond the program’s control, such as disc failure, keyboard interrupts, etc.
* **C++ try and catch**C++ provides an inbuilt feature for Exception Handling. It can be done using the following specialized keywords: try, catch, and throw with each having a different purpose.  
  **Syntax of try-catch in C++  
  try** {   
  *// Code that might throw an exception* **throw** SomeExceptionType("Error message");  
   }   
  **catch**( ExceptionName e1 ) {   
   *// catch block catches the exception that is thrown from try block* }   
  **1. try in C++**The try keyword represents a block of code that may throw an exception placed inside the try block. It’s followed by one or more catch blocks. If an exception occurs, try block throws that exception.  
  **2. catch in C++**The catch statement represents a block of code that is executed when a particular exception is thrown from the try block. The code to handle the exception is written inside the catch block.  
  **3. throw in C++**An exception in C++ can be thrown using the throw keyword. When a program encounters a throw statement, then it immediately terminates the current function and starts finding a matching catch block to handle the thrown exception.  
  ***Note:*** *Multiple catch statements can be used to catch different type of exceptions thrown by try block.*The try and catch keywords come in pairs: We use the try block to test some code and If the code throws an exception we will handle it in our catch block.  
  **Why do we need Exception Handling in C++?**The following are the main advantages of exception handling over traditional error handling:
  + **Separation of Error Handling Code from Normal Code***:* There are always if-else conditions to handle errors in traditional error handling codes. These conditions and the code to handle errors get mixed up with the normal flow. This makes the code less readable and maintainable. With try/catch blocks, the code for error handling becomes separate from the normal flow.
  + **Functions/Methods can handle only the exceptions they choose***:* A function can throw many exceptions, but may choose to handle some of them. The other exceptions, which are thrown but not caught, can be handled by the caller. If the caller chooses not to catch them, then the exceptions are handled by the caller of the caller.   
    In C++, a function can specify the exceptions that it throws using the throw keyword. The caller of this function must handle the exception in some way (either by specifying it again or catching it).
  + **Grouping of Error Types***:* In C++, both basic types and objects can be thrown as exceptions. We can create a hierarchy of exception objects, group exceptions in namespaces or classes, and categorize them according to their types.
* lø 
  11 
  12 
  13 
  14 
  15 
  16 
  17 
  18 
  int main() { 
  Snt new int (løøøøøøøøøøøøoøøl ; 
  allocation is successfull\n"; 
  delete (Ip; 
  catch(const exception &e) 
  cout<<"Exception Occured: "«e.what(   
     
  std:exception 
  std:bad alloc 
  std 
  std:bad_typeid 
  std:bad_exception 
  std:runtime error 
  std:domain error 
  std:invalid_argument 
  std: length_error 
  std:out_of_range 
  std:overflow error 
  std: range_error 
  std:underflow error   
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
  // C++ program to demonstate the use of try,catch and throw  
  // in exception handling.  
  #include <iostream>  
  #include <stdexcept>  
  using namespace std;  
  int main()  
  {  
   // try block  
   try {  
   int numerator = 10;  
   int denominator = 0;  
   int res;  
   // check if denominator is 0 then throw runtime  
   // error.  
   if (denominator == 0) {  
   throw runtime\_error(  
   "Division by zero not allowed!");  
   }  
   // calculate result if no exception occurs  
   res = numerator / denominator;  
   //[printing result after division  
   cout << "Result after division: " << res << endl;  
   }  
   // catch block to catch the thrown exception  
   catch (const exception& e) {  
   // print the exception  
   cout << "Exception " << e.what() << endl;  
   }  
   return 0;  
  }  
  **Output**Exception Division by zero not allowed!  
     
  // C++ program to demonstate the use of try,catch and throw  
  // in exception handling.  
  #include <iostream>  
  using namespace std;  
  int main()  
  {  
   int x = -1;  
   // Some code  
   cout << "Before try \n";  
   // try block  
   try {  
   cout << "Inside try \n";  
   if (x < 0) {  
   // throwing an exception  
   throw x;  
   cout << "After throw (Never executed) \n";  
   }  
   }  
   // catch block  
   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)   
     
     
  **Properties of Exception Handling in C++  
  Property 1  
  *There is a special catch block called the ‘catch-all’ block, written as catch(…), that can be used to catch all types of exceptions.***   
  **Example**In the following program, an int is thrown as an exception, but there is no catch block for int, so the catch(…) block will be executed.
  + CPP
* // C++ program to demonstate the use of catch all  
  // in exception handling.  
  #include <iostream>  
  using namespace std;  
  int main()  
  {  
   // try block  
   try {  
   // throw  
   throw 10;  
   }  
   // catch block  
   catch (char\* excp) {  
   cout << "Caught " << excp;  
   }  
   // catch all  
   catch (...) {  
   cout << "Default Exception\n";  
   }  
   return 0;  
  }  
  **Output**Default Exception  
     
  **Property 2  
  *Implicit type conversion doesn’t happen for primitive types.*Example**In the following program, ‘a’ is not implicitly converted to int.   
     
  //// C++ program to demonstate property 2: Implicit type  
  /// conversion doesn't happen for primitive types.  
  // in exception handling.  
  #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  
  **Property 3  
  *If an exception is thrown and not caught anywhere, the program terminates abnormally.*Example**In the following program, a char is thrown, but there is no catch block to catch the char.   
     
  // C++ program to demonstate property 3: If an exception is  
  // thrown and not caught anywhere, the program terminates  
  // abnormally in exception handling.  
  #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'  
  We can change this abnormal termination behavior by writing our unexpected function.  
  ***Note****: A derived class exception should be caught before a base class exception.***Property 4  
  *Unlike Java, in C++, all exceptions are unchecked, i.e., the compiler doesn’t check whether an exception is caught or not (See*** [***this***](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/) ***for details). So, it is not necessary to specify all uncaught exceptions in a function declaration. However,exception-handling it’s a recommended practice to do so.***   
  // C++ program to demonstate property 4 in exception  
  // handling.  
  #include <iostream>  
  using namespace std;  
  // This function signature is fine by the compiler, but not  
  // recommended. Ideally, the function should specify all  
  // uncaught exceptions and function signature should be  
  // "void fun(int \*ptr, int x) throw (int \*, int)"  
  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;  
  }  
  **Output**Caught exception from fun()  
  **Limitations of Exception Handling in C++**The exception handling in C++ also have few limitations:
  + Exceptions may break the structure or flow of the code as multiple invisible exit points are created in the code which makes the code hard to read and debug.
  + If exception handling is not done properly can lead to resource leaks as well.
  + It’s hard to learn how to write Exception code that is safe.
  + There is no C++ standard on how to use exception handling, hence many variations in exception-handling practices exist.