GeeksforGeeks

Related Articles

Exception Handling in C++

Difficulty Level : Medium   •   Last Updated : 03 Feb, 2021

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: a)Synchronous, b)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.

Why Exception Handling?

Following are main advantages of exception handling over traditional error handling.

1) *Separation of Error Handling code from Normal Code:* In traditional error handling codes, there are always if else conditions to handle errors. 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.

2) *Functions/Methods can handle any 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 caller. If the caller chooses not to catch them, then the exceptions are handled by 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)

3) *Grouping of Error Types:* In C++, both basic types and objects can be thrown as exception. We can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types.

Exception Handling in C++

1) Following is a simple example to show exception handling in C++. The output of program explains flow of execution of try/catch blocks.

```cpp
#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) 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.

```cpp
#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. For example, in the following program 'a' is not implicitly converted to int

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

int main()
{
    try  {
        throw 'a';
    }
    catch (int x)  {
        cout << "Caught " << x;
    }
    catch (...)  {
        cout << "Default Exception\n";
    }
    return 0;
}
```
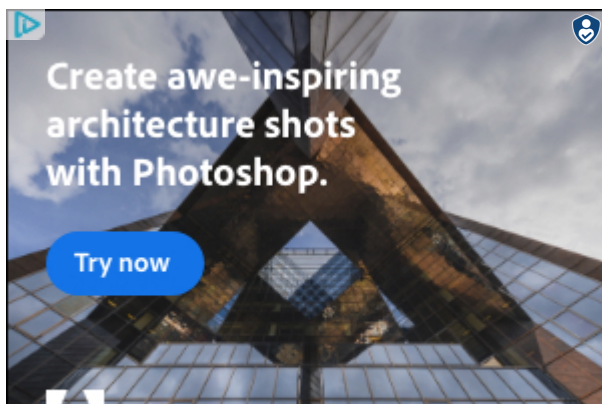
Output:

```
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.

```cpp
#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.
```

We can change this abnormal termination behavior by writing our own unexpected function.

5) A derived class exception should be caught before a base class exception. See this for more details.

6) Like Java, C++ library has a standard exception class which is base class for all standard exceptions. All objects thrown by components of the standard library are derived from this class. Therefore, all standard exceptions can be caught by catching this type

7) Unlike Java, in C++, all exceptions are unchecked. Compiler doesn't check whether an exception is caught or not (See this for details). For example, in C++, it is not necessary to specify all uncaught exceptions in a function declaration. Although it's a recommended practice to do so. For example, the following program compiles fine, but ideally signature of fun() should list unchecked exceptions.

```cpp
#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()
```

A better way to write above code

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

// Here we specify the exceptions that this function
// throws.
void fun(int *ptr, int x) throw (int *, int) // Dynamic Exception specification
{
    if (ptr == NULL)
        throw ptr;
```

```cpp
        if (x == 0)
            throw x;
        /* Some functionality */
}

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

(**Note :** The use of Dynamic Exception Specification has been deprecated after C++11, one of the reason maybe because it can randomly abort your program. This can happen when you throw an exception of an another type which is not mentioned in the dynamic exception specification, your program will abort itself, because in that scenario program calls(indrectly) terminate(), and which is by default call abort()).

Output:

```
 Caught exception from fun()
```

8) In C++, try-catch blocks can be nested. Also, an exception can be re-thrown using "throw; "

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

```cpp
#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;
    }
}
```

Output:

```
 Constructor of Test
 Destructor of Test
```

```
Caught 10
```

10) You may like to try Quiz on Exception Handling in C++.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Want to learn from the best curated videos and practice problems, check out the C++ Foundation Course for Basic to Advanced C++ and C++ STL Course for foundation plus STL. To complete your preparation from learning a language to DS Algo and many more, please refer **Complete Interview Preparation Course.**

♡ **Like**   0

I< Previous   Next >I

RECOMMENDED ARTICLES   Page : **1** 2 3

01 **Exception handling and object destruction | Set 1**
12, Dec 11

05 **C++ | Exception Handling | Question 4**
24, Nov 13

02 **Python Exception Handling**
12, Mar 16

06 **C++ | Exception Handling | Question 5**
24, Nov 13

03 **C++ | Exception Handling | Question 1**
21, Nov 13

07 **C++ | Exception Handling | Question 6**
24, Nov 13

C++ | Exception Handling | Question

04   ~~C++ | Exception Handling | Question~~
         3
         24, Nov 13

08   C++ | Exception Handling | Question
         7
         24, Nov 13

● ● ●

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy	Normal	Medium	Hard	Expert

Improved By :   slodha95,   pritiprajapati314,   crawler

Article Tags :   cpp-exception,   C Language,   C++,   School Programming

Practice Tags :   CPP

Improve Article   Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us
Careers

Privacy Policy

Contact Us

Copyright Policy

Learn

Algorithms
Data Structures

Languages

CS Subjects

Video Tutorials

Practice

Courses

Company-wise

Topic-wise

How to begin?

Contribute

Write an Article

Write Interview Experience

Internships

Videos