Tutorial Link https://course.testpad.chitkarauniversity.edu.in/tutorials/Exception Handling/5b390373c6a1d0259e728f4b

TUTORIAL

# Exception Handling

Topics

Exception handling is a mechanism that provides resources to identify and report an exceptional circumstance in order to take proper action. The error handling code performs the following tasks:

1. Identify the problem (hit the exception).
2. Notify that an error has occurred (throw the exception).
3. Accept the error message (catch the exception).
4. Take corrective actions (handle the exception).

## Exception handling mechanism

Exception handling mechanism is basically built upon three keywords:

1. **try:** The keyword try is the name of block that encapsulates the statements which may generate the exceptions. This block of statements is known as try block.
2. **throw:** When an exception is identified, it is thrown by using a statement throw. The throw statement is written inside the try block.
3. **catch:** the keyword catch defines the catch block which catches and handles the exception thrown by the throw statement inside the try block.

When try block passes an exception using throw statement, the control of the program is passed to the catch block. The catch block must immediately follow the try block that throws the exception. Also remember that the data types used by the throw and catch statements must be same. If they do not match the program is aborted with the help of function abort ( ) which is invoked by the compiler by default. When no error has occurred and no exception is thrown, then the catch block is ignored and the statement after the catch block is executed. The point at which the throw is executed is called throw point. Once an exception is thrown to the catch block, control cannot return back to the throw point.

```
Statement;
try
{ .....................
.....................
throw( exception); // throws exception
.....................
}
catch( type arg) // catches exception
{
.....................
.....................
.....................
}
.................................
```

Let us take a program that shows the concept of exception handling.

```cpp
#include<iostream>

using namespace std;

int main()
{
  int m,n;
  m = 35;
  n = 0;
  try
  {
    if ( n != 0)
    {
      cout <<"Result ( m/n) = "<< m/n<<endl;
    }
    else
    {
      throw (n); // throws int object
    }
  }
  catch(int n) // catches an exception
  {
    cout<<"Exception handled; n = "<< n <<endl;
  }
  return 0;
}


```

The above program detects and catches a division-by-zero problem. The exception is thrown by using an object x of int type. As the data type of exception object is int, therefore catch statement uses int type argument and displays an appropriate message.

## Multiple catch statements

If a program has more than one condition to throw an exception, then multiple catch blocks with a single try block are used. The format of multiple catch statements are as follows:

```
try
{
       // try block
}
catch( type1 arg)
{
       // catch block 1
}
catch( type2 arg)
```

```
{
        // catch block2
}
        …………………….
        …………………….
catch( type1 arg)
{
        // catch block 1
}
```

When an exception is thrown, the catch blocks are searched in order. The very first catch block that yields a match is executed. After executing the catch block, the control goes to the first statement after the last catch block. Let us take an example.

```cpp
#include<iostream>

using namespace std;

void Number(int value)
{
  try
  {
    if (value == 0) throw 'x'; //char
    else
      if(value > 0) throw value; // int
    else
      if(value < 0) throw 1.0; // double
    cout<<"End of try block\n";
  }
  catch (char ch)
  {
    cout<<"Caught a null value\n";
  }
  catch ( int m)
  {
    cout<<"Caught a positive value\n";
  }
  catch ( double d)
  {
    cout<<"Caught a negative value\n";
  }
  cout<<"End of try-catch block\n";
}

int main()
{
  Number(7);
  Number(0);
  Number(-2);
  return 0;
}

```

In the above program, the function Number ( ) is first invoked with positive number 7 and throws an int exception. After that the function is again invoked with 0 and throws a character exception and in the last it is invoked with negative value -2 and throws an exception of double type.

## Catching multiple exceptions

It is possible to define a single catch block to handle all possible types of exceptions. In such situation the syntax of catch block is as under:

```
catch ( …. )
{
       // statements for handling all exceptions
}
```

The previous example can be handled using single catch block.

```
#include<iostream>

using namespace std;
```

```

void Number(int value)
{
  try
  {
    if (value == 0) throw 'x'; //char
    else
      if(value > 0) throw value; // int
    else
      if(value < 0) throw 1.0; // double
    cout<<"End of try block\n";
  }
  catch(...)
  {
    cout<<"Caught an exception\n";
  }
  cout<<"End of try-catch block\n";
}

int main()
{
  Number(7);
  Number(0);
  Number(-2);
  return 0;
}

```

The above program is the same as the previous one. The difference is that rather than using multiple catch blocks, a single catch block is defined here.

## Rethrowing exceptions

Sometimes an exception received by the catch block is passed again to another exception handler. This is known as rethrowing of exceptions.

The above throw statement is used without any argument. For example:

```cpp
#include<iostream>

using namespace std;

void Division ( int x, int y )
{
  cout <<"inside function\n";
  try
  {
    if ( y == 0)
      throw y; // int
    else
      cout<<"division = "<< x/y <<endl;
  }
  catch ( int y)
  {
    cout<<"divide by zero inside function\n";
    throw;
  }
  cout<<"End of function\n";
}

int main()
{
  cout<<"inside main \n";
  try
  {
    Division(20, 0);
  }
  catch(int e) // catches an exception
  {
    cout<<"caught divide by zero inside main\n";
  }
  cout<<"End of main ( ) function\n";
  return 0;
}

```

In the above program, the catch block inside the function accepts the exception and throws again using the statement throw without any argument.

## Restricting exceptions

We can restrict a function to throw only certain specified exceptions outside of it. This is accomplished by adding a throw list clause to the function definition. The general form of this as shown here:

```
return_type Function_Name ( arg-list) throw (type-list)
{
        ……………………….
        ………………………
}
```

The type-list specifies the type of exceptions that may be thrown. Throwing an exception that is not supported by the function causes abnormal program termination. We can restrict a function from throwing any exception by using empty type-list. That is throw( ) //Empty list let us take an example

C++

```
#include<iostream>

using namespace std;

void Number(int value) throw( int, char, double)
{
  if ( value == 0) throw 'x'; //char
  else
    if ( value > 0) throw value; // int
  else
    if ( value < 0) throw 1.0; // double
}

int main()
{
  try
  {
    Number(0);
  }
  catch(char ch)
  {
    cout<<"caught a null value\n";
  }
  catch(int m)
  {
    cout<<"caught a positive value\n";
  }
  catch(double d)
  {
    cout<<"caught a negative value\n";
  }
  return 0;
}

```

In the above program the function Number ( ) may only throw integer, character and double exceptions. If any other type of exception is tried to throw, an abnormal program termination will occur.

## Define New Exceptions

You can define your own exceptions by inheriting and overriding exception class functionality. Following is the example, which shows how you can use std::exception class to implement your own exception in the standard way −

C++

```cpp
#include <iostream>
#include <exception>
```

```
using namespace std;

class CQ1 : public exception {
  public:
   const char * what () const throw () {
      return "Code Quotient";
   }
};

int main() {
  CQ1 o1;
   try {
      throw o1;
   } catch(CQ1& e) {
      cout << "CQ1 caught" << endl;
      cout << e.what() << endl;
   } catch(exception& e) {
      cout << "Exception caught" << endl;
   }
}
```

Here, what() is a public method provided by exception class and it has been overridden by all the child exception classes. This returns the cause of an exception.