Exception Handling

Errors can be broadly categorized into two types:
1. Compile Time Errors
2. 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. An exception caught during run time creates serious issues.

Errors hamper 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. So, we need to handle this situation in our program.

In C++, all exceptions are derived from `std::exception` class. If we don't handle the exception, it prints exception message and terminates the program.

**C++ Exception Classes**

In C++ standard exceptions are defined in `<exception>` class that we can use inside our programs. The arrangement of parent-child class hierarchy is shown here:
Following is brief discretion of C++ common exception classes:

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>std::exception</td>
<td>It is an exception and parent class of all standard C++ exceptions.</td>
</tr>
<tr>
<td>std::logic_failure</td>
<td>It is an exception that can be detected by reading a code.</td>
</tr>
<tr>
<td>std::runtime_error</td>
<td>It is an exception that cannot be detected by reading a code.</td>
</tr>
<tr>
<td>std::bad_exception</td>
<td>It is used to handle the unexpected exceptions in a c++ program.</td>
</tr>
<tr>
<td>std::bad_cast</td>
<td>This exception is generally thrown by dynamic_cast.</td>
</tr>
<tr>
<td>std::bad_typeid</td>
<td>This exception is generally thrown by typeid.</td>
</tr>
<tr>
<td>std::bad_alloc</td>
<td>This exception is generally thrown by new.</td>
</tr>
</tbody>
</table>

**The keywords try, throw and catch**

In C++, Error handling is done by three keywords:-
- Try
- Catch
- Throw

**Syntax:**
```
Try
{
    //code
    throw parameter;
}
catch(exceptionname ex)
{
    //code to handle exception
}
```

**Try:** Try block is intended to throw exceptions, which is followed by catch blocks. We can have only one try block.

**Catch:** Catch block is intended to catch the error and handle the exception condition. We can have multiple catch blocks.

**Throw:** It is used to throw exceptions to exception handler i.e. it is used to communicate information about error. A throw expression accepts one parameter and that parameter is passed to handler.

**Example without try/catch:**
```
#include <iostream>
using namespace std;
float division(int x, int y) {
```
return (x/y);
}
int main () {
    int i = 50;
    int j = 0;
    float k = 0;
    k = division(i, j);
    cout << k << endl;
    return 0;
}

OUTPUT
Floating point exception (core dumped)

Same Example without try/catch:

#include <iostream>
using namespace std;
float division(int x, int y) {
    if (y == 0) {
        throw "Attempted to divide by zero!";
    }
    return (x/y);
}
int main () {
    int i = 25;
    int j = 0;
    float k = 0;
    try {
        k = division(i, j);
        cout << k << endl;
    } catch (const char* e) {
        cerr << e << endl;
    }
    return 0;
}

OUTPUT
Attempted to divide by zero!

Example for multiple catch statement
Below program contains multiple catch statements to handle exception.

#include <iostream>
using namespace std;
int main() {
    int x[3]={-1,2,};
    for(int i=0;i<2;i++){
        int ex=x[i];
        try {
            if (ex > 0)
                throw ex;
            else
                throw 'x';
        } catch (int ex) {

Provided By Shipra Swati, PSTC, Vaishali, Bihar
cout << " Integer exception \n";
} 
catch (char ex) {
    cout << " Character exception \n";
}
}

**OUTPUT**
Character exception
Integer exception

**Example for generalized catch statement**
Below program contains generalized catch statement to catch uncaught errors. Catch(...) takes care of all type of exceptions.

```cpp
#include <iostream>
using namespace std;
int main()
{
    int x[3]={-1,2,};
    for(int i=0;i<2;i++)
    {
        int ex=x[i];
        try {
            if (ex > 0)
                throw x;
            else
                throw 'ex';
        } catch (int ex) {
            cout << " Integer exception \n" ;
        } catch (char ex) {
            cout << " Character exception \n" ;
        } catch (...) {
            cout << "Special exception \n";
        }
    }
    return 0;
}
```

**OUTPUT**
Integer exception
Special exception

**Creating own Exception Classes**
The new exception can be defined by overriding and inheriting exception class functionality. In the simple example of user-defined exception in which std::exception class is used to define the
#include <iostream>
#include <exception>
using namespace std;
class MyException : public exception{
   public:
   const char * what() const throw()
   {
      return "Attempted to divide by zero!\n";
   }
};
int main()
{
   try
   {
      int x, y;
      cout << "Enter the two numbers : \n";
      cin >> x >> y;
      if (y == 0)
      {
         MyException z;
         throw z;
      }
      else
      {
         cout << "x / y = " << x/y << endl;
      }
   } catch(exception& e)
   {
      cout << e.what();
   }
}

OUTPUT:

1ST RUN:
Enter the two numbers :
50
70
x / y = 0

2ND RUN:
Enter the two numbers :
40
0
Attempted to divide by zero!