EXCEPTION HANDLING

EXAMPLE

```
class MyException : public exception
{
  char * what () const throw ()
  {
    return "C++ Exception";
  }
};
```

```
void main()
{
  int a=0;
  try
  {
     if (a==0)
    throw MyException();
  }
  catch(MyException& e)
  {
   cout << "MyException caught" ;
   cout << e.what();
  }
  catch(exception& e)
  {
    //Other errors
  }
}
```

EXCEPTION HANDLING

- **An exception is a problem that arises during the execution of a program.**
- **Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.**

TRY, CATCH & THROW

- **throw:** A program throws an exception when a problem shows up. This is done using a **throw** keyword.
- **catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
- **try:** A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

TRY & CATCH

```
try
{
  // protected code
}
catch( ExceptionName e1 )
{
  // catch block
}
catch( ExceptionName e2 )
{
  // catch block
}
catch( ExceptionName eN )
{
  // catch block
}
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

THROWING EXCEPTIONS

- Exceptions can be thrown anywhere within a code block using **throw** statements.
- The operand of the throw statements determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

EXAMPLE

```
double division(int a, int b)
{
  if( b == 0 ) // type of exception
  {
    throw "Division by zero condition!";
  }
  return (a/b);
}
```

CATCHING EXCEPTIONS

- **The catch block following the try block catches any exception.**
- **You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.**

EXAMPLE WITHOUT EXCEPTION HANDLING

```
int a=20,b=0;
c=a/b;


// displays error
```

EXAMPLE WITH EXCEPTION HANDLING
- DIVIDE BY ZERO EXCEPTION

```
int a=20,b=0;
try
{
 if( b == 0 )
   {
     throw "Division by zero condition!";
   }
else
  c=a/b;
}
catch (char* msg)
{
    cout << msg << endl;
   }
```

MULTIPLE CATCH BLOCKS

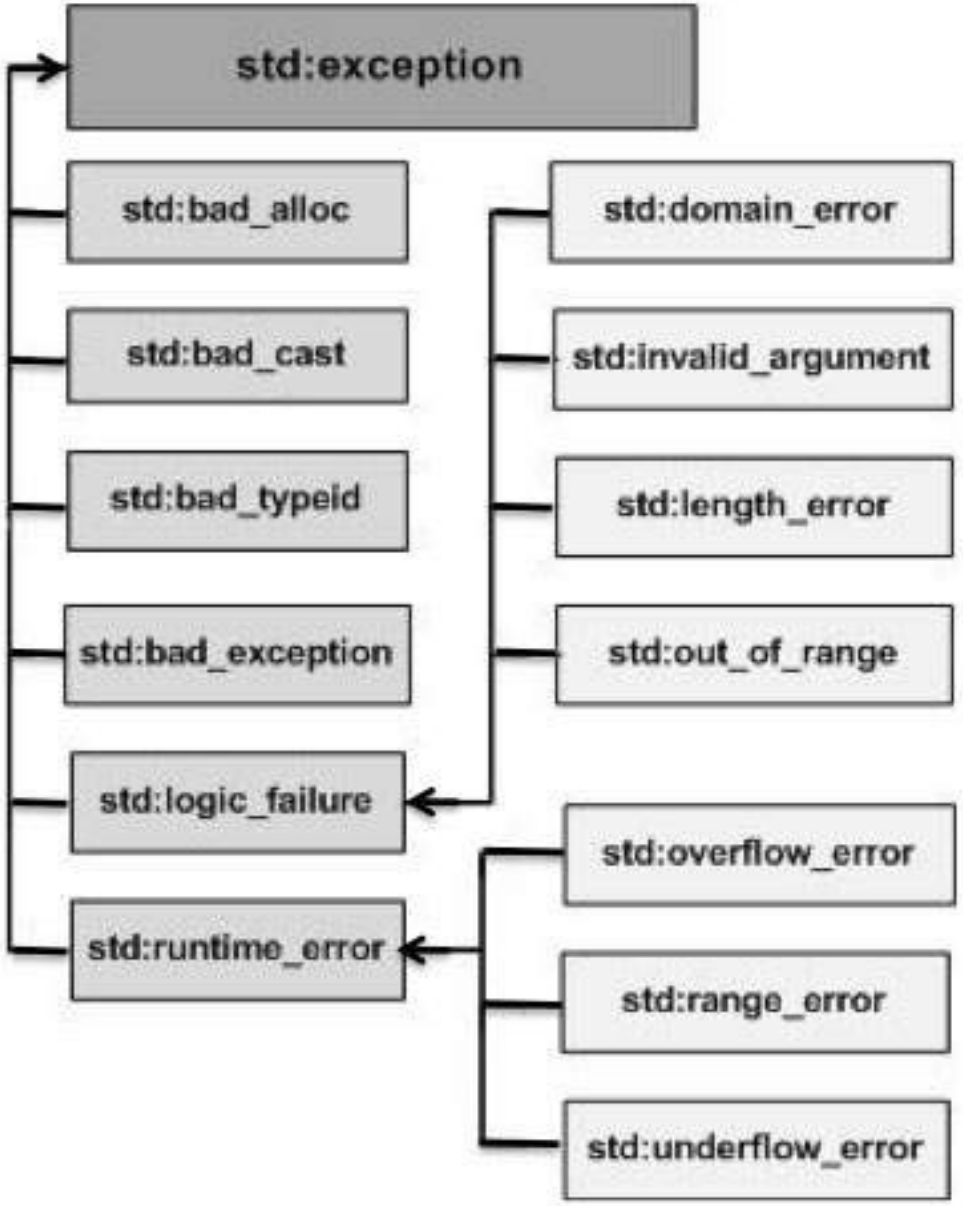
```
void test(int x)
{
  try
  {
          if(x>0)
            throw x;
      else
            throw 'x';
  }

  catch(int x)
  {
          cout<<"Catch a integer and that integer
is:"<<x;
  }

  catch(char x)
  {
          cout<<"Catch a character and that character
is:"<<x;
  }
}
```

```
void main()
{
  clrscr();
  cout<<"Testing multiple catches
n:";
  test(10);
  test(0);
  getch();
}
```

C++ STANDARD EXCEPTIONS:



std::exception	**An exception and parent class of all the standard C++ exceptions.**
std::bad_alloc	**This can be thrown by new.**
std::bad_cast	**This can be thrown by dynamic_cast.**
std::bad_exception	**This is useful device to handle unexpected exceptions in a C++ program**
std::bad_typeid	**This can be thrown by typeid.**
std::logic_error	**An exception that theoretically can be detected by reading the code.**
std::domain_error	**This is an exception thrown when a mathematically invalid domain is used**
std::invalid_argument	**This is thrown due to invalid arguments.**
std::length_error	**This is thrown when a too big std::string is created**
std::out_of_range	**This can be thrown by the at method from for example a std::vector and std::bitset<>::operator[]().**
std::runtime_error	**An exception that theoretically can not be detected by reading the code.**
std::overflow_error	**This is thrown if a mathematical overflow occurs.**
std::range_error	**This is occured when you try to store a value which is out of range.**

DEFINE NEW EXCEPTIONS

- You can define your own exceptions by inheriting and overriding **exception** class functionality.
- Allows you can use std::exception class to implement your own exception in standard way