


# Exceptions

For : COP 3330.  
Object oriented Programming (Using C++)  
<http://www.compgeom.com/~piyush/teach/3330>


Piyush Kumar

Material from "4th ed. Programming and Problem solving with C++" Dale, Weems




## Exceptions

- Exceptions are run-time anomalies, such as division by zero, that require immediate handling when encountered by your program. The C++ language provides built-in support for raising and handling exceptions. With C++ exception handling, your program can communicate unexpected events to a higher execution context that is better able to recover from such abnormal events.
- Useful when the code that detects the problem cannot handle it (Exception-Detection code). Control must be transferred to the code that can handle such error. (Exception-Handling code).




## Exceptions in C++

- Communication between exception-detection and exception-handling parts of the program in C++. It involves:
  - throw expressions
  - try blocks
  - exception classes
- The try, throw, and catch statements implement exception handling.




## Exceptions in C++

- If not handled properly, exceptions can cause the program to :
  - Crash
  - Falls into unknown state
- An exception handler is a section of program code that is designed to execute when a particular exception occurs
  - Resolve the exception
  - Lead to known state, such as exiting the program



## Standard Exceptions

- Exceptions Thrown by
  - the Language
    - new
  - Standard Library Routines
  - User code, using *throw* statement



## The *throw* Statement

**Throw:** to signal the fact that an exception has occurred; also called *raise*

Syntax

throw Expression

## The try-catch Statement

How one part of the program catches and processes the exception that another part of the program throws.

### TryCatchStatement

```
try
    Block
catch (FormalParameter)
    Block
catch (FormalParameter)
```

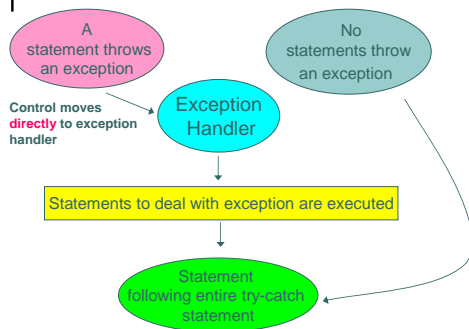
### FormalParameter

```
{
    DataType VariableName
    ...
}
```

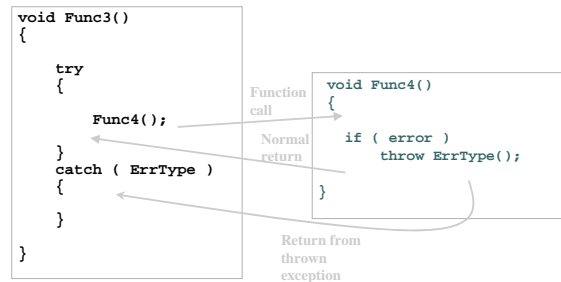
## Example of a try-catch Statement

```
try
{
    // Statements that process personnel data and may throw
    // exceptions of type int, string, and SalaryError
}
catch ( int )
{
    // Statements to handle an int exception
}
catch ( string s )
{
    cout << s << endl; // Prints "Invalid customer age"
    // More statements to handle an age error
}
catch ( SalaryError )
{
    // Statements to handle a salary error
}
```

## Execution of try-catch



## Throwing an Exception to be Caught by the Calling Code



## Practice: Dividing by ZERO

Apply what you know:

```
int Quotient(int numer,    // The numerator
            int denom )   // The denominator
{
    if (denom != 0)
        return numer / denom;
    else
        //What to do?? do sth. to avoid program
        //crash
}
```

## A Solution

```
int Quotient(int numer,    // The numerator
            int denom )   // The denominator
{
    if (denom == 0)
        throw DivByZero();
    //throw exception of class DivByZero
    return numer / denom;
}
```

## A Solution

```
// "quotient.cpp" - Quotient program
#include <iostream>
#include <string>
using namespace std;
int Quotient(int, int);
class DivByZero // Exception class
{
};
int main()
{
    int numer; // Numerator
    int denom; // Denominator
    cout << "Enter numerator and denominator: ";
```

```
    cin >> numer >> denom;
    while (cin)
    {
        try
        {
            cout << "Their quotient: "
                << Quotient(numer, denom) << endl;
        }
        catch (DivByZero)
        {
            cout << "*** Denominator can't be 0"
                << endl;
        }
        cout << "Enter numerator and denominator: ";
        cin >> numer >> denom;
    }
    return 0;
}
```

```
int Quotient(/* in */ int numer, // The numerator
            /* in */ int denom) // The denominator
{
    if (denom == 0)
        throw DivByZero();
    return numer / denom;
}
```