



## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

# Module 37: Programming C++

## Exceptions (Error handling in C++): Part 2

Partha Pratim Das

Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur

*ppd@cse.iitkgp.ernet.in*

Tanwi Mallick  
Srijoni Majumdar  
Himadri B G S Bhuyan



# Module Objectives

## Module 37

Partha Pratim  
Das

### Objective & Outline

#### Exceptions in C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

#### Summary

- Understand the Error handling in C++



# Module Outline

## Module 37

Partha Pratim  
Das

### Objective & Outline

#### Exceptions in C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

#### Summary

- Exception Fundamentals
  - Types of Exceptions
  - Exception Stages
- Exceptions in C
  - C Language Features
    - Return value & parameters
    - Local goto
  - C Standard Library Support
    - Global variables
    - Abnormal termination
    - Conditional termination
    - Non-local goto
    - Signal
  - Shortcomings
- Exceptions in C++
  - Exception Scope (try)
  - Exception Arguments (catch)
  - Exception Matching
  - Exception Raise (throw)
  - Advantages



# Expectations

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- Separate Error-Handling code from Ordinary code
- Language Mechanism rather than of the Library
- Compiler for Tracking Automatic Variables
- Schemes for Destruction of Dynamic Memory
- Less Overhead for the Designer
- Exception Propagation from the deepest of levels
- Various Exceptions handled by a single Handler



# try-throw-catch

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

```
void f() {  
    A a;  
    try {  
        B b;  
        g();  
        h();  
    }  
    catch (UsrExcp& ex) {  
        cout <<  
            ex.what();  
    }  
    return;  
}  
  
class UsrExcp:  
    public exceptions {}  
  
void g()  
{  
    A a;  
    UsrExcp ex("From g()");  
  
    throw ex;  
  
    return;  
}
```

- g() called



# try-throw-catch

## Module 37

Partha Pratim Das

Objective & Outline

Exceptions in C++

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

Summary

```
void f() {  
    A a;  
    try {  
        B b;  
        g();  
        h();  
    }  
    catch (UsrExcp& ex) {  
        cout <<  
            ex.what();  
    }  
    return;  
}  
  
class UsrExcp:  
    public exceptions {}  
  
void g()  
{  
    A a;  
    UsrExcp ex("From g()");  
    throw ex;  
    return;  
}
```

- g() successfully returns



# try-throw-catch

## Module 37

Partha Pratim Das

Objective & Outline

Exceptions in C++

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

Summary

```
void f() {  
    A a;  
    try {  
        B b;  
        g();  
        h();  
    }  
    catch (UsrExcp& ex) {  
        cout <<  
            ex.what();  
    }  
    return;  
}  
  
class UsrExcp:  
    public exceptions {}  
  
void g()  
{  
    A a;  
    UsrExcp ex("From g()");  
    throw ex;  
    return;  
}
```

- g() called and exception raised
- Exception caught by catch clause



# Exception Flow

## Module 37

Partha Pratim Das

### Objective & Outline

#### Exceptions in C++

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

#### Summary

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

class MyException : public exception {};
class MyClass {};

void h() { MyClass a;

    //throw 1;

    //throw 2.5;

    //throw MyException();

    //throw exception();

    //throw MyClass();
}

void g() { MyClass a;
    try {
        h();
    }
    catch (int) { cout << "int"; }
    catch (double) { cout << "double"; }
    catch (...) { throw; }
}
```

```
void f() { MyClass a;
    try {
        g();
    }
    catch (MyException) { cout << "MyException"; }
    catch (exception) { cout << "exception"; }
    catch (...) { throw; }
}

int main() {
    try {
        f();
    }
    catch (...) { cout << "Unknown"; }

    return 0;
}
```





# try Block: Exception Scope

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- try block
  - Consolidate areas that might throw exceptions
- function try block
  - Area for detection is the entire function body
- Nested try block
  - Semantically equivalent to nested function calls

### Function try

```
void f()
{
    try {
        throw E();
    }
    catch (E& e) {
    }
}
```

### Nested try

```
try {
    try { throw E(); }
    catch (E& e) { }
}
catch (E& e1) {
}
```



# try-throw-catch

## Module 37

Partha Pratim Das

Objective & Outline

Exceptions in C++

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

Summary

```
void f() {  
    A a;  
    try {  
        B b;  
        g();  
        h();  
    }  
    catch (UsrExcp& ex) {  
        cout <<  
            ex.what();  
    }  
    return;  
}
```

```
class UsrExcp:  
    public exceptions {}  
  
void g()  
{  
    A a;  
    UsrExcp ex("From g()");  
  
    throw ex;  
  
    return;  
}
```

- try Block



# catch Block: Exception Arguments

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- catch block

- Name for the Exception Handler
- Catching an Exception is like invoking a function
- Immediately follows the try block
- Unique Formal Parameter for each Handler
- Can be simply a Type Name to distinguish its Handler from others



# try-throw-catch

## Module 37

Partha Pratim Das

Objective & Outline

Exceptions in C++

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

Summary

```
void f() {  
    A a;  
    try {  
        B b;  
        g();  
        h();  
    }  
    catch (UsrExcp& ex) {  
        cout <<  
            ex.what();  
    }  
    return;  
}
```

```
class UsrExcp:  
    public exceptions {}  
  
void g()  
{  
    A a;  
    UsrExcp ex("From g()");  
  
    throw ex;  
  
    return;  
}
```

- catch Block



# try-catch: Exception Matching

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- Exact Match
  - The catch argument type matches the type of the thrown object
    - No implicit conversion is allowed
- Generalization / Specialization
  - The catch argument is a public base class of the thrown class object
- Pointer
  - Pointer types – convertible by standard conversion



# try-throw-catch

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

```
void f() {  
    A a;  
    try {  
        B b;  
        g();  
        h();  
    }  
    catch (UsrExcp& ex) {  
        cout <<  
            ex.what();  
    }  
    return;  
}
```

```
class UsrExcp:  
    public exceptions {}  
  
void g()  
{  
    A a;  
    UsrExcp ex("From g()");  
    throw ex;  
    return;  
}
```

- Expression Matching



# try-catch: Exception Matching

Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- In the order of appearance with matching
- If Base Class catch block precedes Derived Class catch block
  - Compiler issues a warning and continues
  - Unreachable code (derived class handler) ignored
- `catch(...)` block must be the last catch block because it catches all exceptions
- If no matching Handler is found in the current scope, the search continues to find a matching handler in a dynamically surrounding try block
  - Stack Unwinds
- If eventually no handler is found, `terminate()` is called



# throw *Expression*: Exception Raise

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- *Expression* is treated the same way as
  - A function argument in a call or the operand of a return statement
- Exception Context
  - `class Exception ;`
- The *Expression*
  - Generate an Exception object to throw
    - `throw Exception();`
  - Or, Copies an existing Exception object to throw
    - `Exception ex;`
    - `...`
    - `throw ex; // Exception(ex);`
- Exception object is created on the Free Store





# try-throw-catch

## Module 37

Partha Pratim Das

Objective & Outline

Exceptions in C++

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

Summary

```
void f() {  
    A a;  
    try {  
        B b;  
        g();  
        h();  
    }  
    catch (UsrExcp& ex) {  
        cout <<  
            ex.what();  
    }  
    return;  
}
```

```
class UsrExcp:  
    public exceptions {}  
  
void g()  
{  
    A a;  
    UsrExcp ex("From g()");  
    throw ex;  
    return;  
}
```

- throw Expression



# throw *Expression*: Restrictions

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- For a UDT Expression
  - Copy Constructor and Destructor should be supported
- The type of Expression cannot be
  - An incomplete type (like void, array of unknown size or of elements of incomplete type, Declared but not Defined struct / union / enum / class Objects or Pointers to such Objects)
  - A pointer to an Incomplete type, except void\*, const void\*, volatile void\*, const volatile void\*



# (re)-throw: Throwing Again?

- Re-throw

- catch may pass on the exception after handling
- Re-throw is not same as throwing again!

## Throws again

```
try { ... }  
catch (Exception& ex) {  
    // Handle and  
    ...  
    // Raise again  
    throw ex;  
    // ex copied  
    // ex destructed  
}
```

## Re-throw

```
try { ... }  
catch (Exception& ex) {  
    // Handle and  
    ...  
    // Pass-on  
    throw;  
    // No copy  
    // No Destruction  
}
```

Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary



# Advantages

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- **Destructor-savvy:**
  - Stack unwinds; Orderly destruction of Local-objects
- **Unobtrusive:**
  - Exception Handling is implicit and automatic
  - No clutter of error checks
- **Precise:**
  - Exception Object Type designed using semantics
- **Native and Standard:**
  - EH is part of the C++ language
  - EH is available in all standard C++ compilers



# Advantages

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- **Scalable:**

- Each function can have multiple try blocks
- Each try block can have a single Handler or a group of Handlers
- Each Handler can catch a single type, a group of types, or all types

- **Fault-tolerant:**

- Functions can specify the exception types to throw; Handlers can specify the exception types to catch
- Violation behavior of these specifications is predictable and user-configurable



# Module Summary

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

- Discussed exception (error) handling in C++
- Illustrated try-throw-catch feature in C++ for handling errors
- Demonstrated with examples



# Instructor and TAs

## Module 37

Partha Pratim  
Das

Objective &  
Outline

Exceptions in  
C++

Exception Scope  
(try)

Exception  
Arguments  
(catch)

Exception  
Matching

Exception Raise  
(throw)

Advantages

Summary

Name	Mail	Mobile
Partha Pratim Das, <i>Instructor</i>	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, <i>TA</i>	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, <i>TA</i>	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, <i>TA</i>	himadribhuyan@gmail.com	9438911655