Programming in Modern C++

Module M37: Exceptions (Error handling in C++): Part 2

Partha Pratim Das

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

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

- Introduced the concept of exceptions
- Discussed error handling in C
- Illustrated various language features and library support in C for handling errors
- Demonstrated with examples

- Understand the Error handling in C++

1. Exceptions in C++
   - `try-throw-catch`
   - Exception Scope (`try`)
   - Exception Arguments (`catch`)
   - Exception Matching
   - Exception Raise (`throw`)
   - Advantages
   - `std::exception`

2. Module Summary

Exceptions in C++

- Separate *Error-Handling code* from *Normal 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

Error Handling Dynamics: C and C++

Header	Caller	Callee
C Scenario		
| ```c
#include <stdio.h>
#include <stdbool.h>
#include <setjmp.h>
``` | ```c
int main() {
    if (setjmp(jbuf) == 0) {
        printf("g() called\n");
        g();
        printf("g() returned\n");
    }
    else printf("g() failed\n"); // On longjmp
    return 0;
}
``` | ```c
jmp_buf jbuf;
void g() {
    bool error = false;
    printf("g() started\n");
    if (error)
        longjmp(jbuf, 1);
    printf("g() ended\n");
    return;
}
``` |
| **C++ Scenario** | | |
| ```cpp
#include <iostream>
#include <exception>
using namespace std;
``` | ```cpp
int main() {
    try {
        cout << "g() called\n";
        g();
        cout << "g() returned\n";
    }
    catch (Excp&) { cout << "g() failed\n"; }
    return 0;
}
``` | ```cpp
class Excp: public exception {};
void g() {
    bool error = false;
    cout << "g() started\n";
    if (error)
        throw Excp();
    cout << "g() ended\n";
    return;
}
``` |

try-throw-catch

Module M37

Partha Pratim
Das

Objectives &
Outlines

Exceptions in
C++

try-throw-catch

Exception Scope
(try)

Exception Arguments
(catch)

Exception Matching

Exception Raise
(throw)

Advantages

std::exception

Module Summary

Caller	Callee

```cpp
int main() {
    try {
        cout << "g() called\n";
        g();
        cout << "g() returned\n";
    }
    catch (Excp&) { cout << "g() failed\n"; }
    return 0;
}
```

```cpp
class Excp: public exception {};
void g() {
    bool error = false;
    cout << "g() started\n";
    if (error)
        throw Excp();
    cout << "g() ended\n";
    return;
}
```

(1) g() called	(2) g() successfully returned

g() called
g() started
g() ended
g() returned

try-throw-catch

Module M37

Partha Pratim Das

Objectives & Outlines

Exceptions in C++

try-throw-catch

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

std::exception

Module Summary

Caller	Callee
```int main() { try { cout << "g() called\n"; g(); cout << "g() returned\n"; } catch (Excp&) { cout << "g() failed\n"; } return 0; }```	```class Excp: public exception {}; class A {}; void g() { A a; bool error = true; cout << "g() started\n"; if (error) throw Excp(); cout << "g() ended\n"; return; }```
(1) g() called

(5) Exception caught by catch clause
(6) Normal flow continues |

(2) Exception raised
(3) Stack frame of g() unwinds and destructor of a called
(4) Remaining execution of g() and cout skipped |

```
g() called
g() started
g() failed
```

Module M37

Partha Pratim Das

Objectives & Outlines

Exceptions in C++

try-throw-catch

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

std::exception

Module Summary

```cpp
#include <iostream>
#include <exception>
using namespace std;
class MyException: public exception { };
class MyClass { public: ~MyClass() { } };
void h() { MyClass h_a;
    //throw 1;                 // Line 1
    //throw 2.5;               // Line 2
    //throw MyException();     // Line 3
    //throw exception();       // Line 4
    //throw MyClass();         // Line 5
    // Stack unwind, h_a.~MyClass() called
    // Passes on all exceptions
void g() { MyClass g_a;
    try { h();
        bool okay = true; // Not executed
    }
    // Catches exception from Line 1
    catch (int) { cout << "int\n"; }
    // Catches exception from Line 2
    catch (double) { cout << "double\n"; }
    // Catches exception from Line 3-5 & passes on
    catch (...) { throw; }
}   // Stack unwind, g_a.~MyClass() called
```

```cpp
void f() { MyClass f_a;
    try { g();
        bool okay = true; // Not executed
    }
    // Catches exception from Line 3
    catch (MyException) { cout << "MyException\n"; }
    // Catches exception from Line 4
    catch (exception) { cout << "exception\n"; }
    // Catches exception from Line 5 & passes on
    catch (...) { throw; }
}   // Stack unwind, f_a.~MyClass() called

int main() {
    try { f();
        bool okay = true; // Not executed
    }
    // Catches exception from Line 5
    catch (...) { cout << "Unknown\n"; }

    cout << "End of main()\n";
}
```

Module M37

Partha Pratim
Das

Objectives &
Outlines

Exceptions in
C++

try-throw-catch

Exception Scope
(try)

Exception Arguments
(catch)

Exception Matching

Exception Raise
(throw)

Advantages

std::exception

Module Summary

try Block: Exception Scope

- `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
```cpp
void f()
    try {
        throw E();
    }
    catch (E& e) {
    }
```
Note: The usual curly braces for the function scope are not to be put here

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

Module M37

Partha Pratim
Das

Objectives &
Outlines

Exceptions in
C++

try-throw-catch

Exception Scope
(try)

**Exception Arguments
(catch)**

Exception Matching

Exception Raise
(throw)

Advantages

std::exception

Module 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 simply be a Type Name to distinguish its Handler from others

- **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

- 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

- *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*

Module M37

Partha Pratim
Das

Objectives &
Outlines

Exceptions in
C++

try-throw-catch

Exception Scope
(try)

Exception Arguments
(catch)

Exception Matching

Exception Raise
(throw)

Advantages

std::exception

Module Summary

throw *Expression*: Restrictions

- For a UDT Expression
  - Copy Constructor and Destructor should be supported
- The type of Expression cannot be an incomplete type or a pointer to an incomplete type
  - No 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
  - No 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	**Re-throw**

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

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

Module M37

Partha Pratim Das

Objectives & Outlines

Exceptions in C++

try-throw-catch

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

std::exception

Module Summary

Advantages

- **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

Module M37

Partha Pratim
Das

Objectives &
Outlines

Exceptions in
C++

try-throw-catch

Exception Scope
(try)

Exception Arguments
(catch)

Exception Matching

Exception Raise
(throw)

Advantages

std::exception

Module Summary

Advantages

- **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 M37

Partha Pratim
Das

Objectives &
Outlines

Exceptions in
C++

try-throw-catch

Exception Scope
(try)

Exception Arguments
(catch)

Exception Matching

Exception Raise
(throw)

Advantages

std::exception

Module Summary

All objects thrown by components of the standard library are derived from this class. Therefore, all standard exceptions can be caught by catching this type by reference.

```
class exception {
public:
    exception() throw();
    exception(const exception&) throw();
    exception& operator=(const exception&) throw();
    virtual ~exception() throw();
    virtual const char* what() const throw();
}
```

Sources: `std::exception` and `std::exception` in `C++11, C++14, C++17 & C++20`

Sources: Standard Library Exception Hierarchy

Module M37

Partha Pratim Das

Objectives & Outlines

Exceptions in C++

try-throw-catch
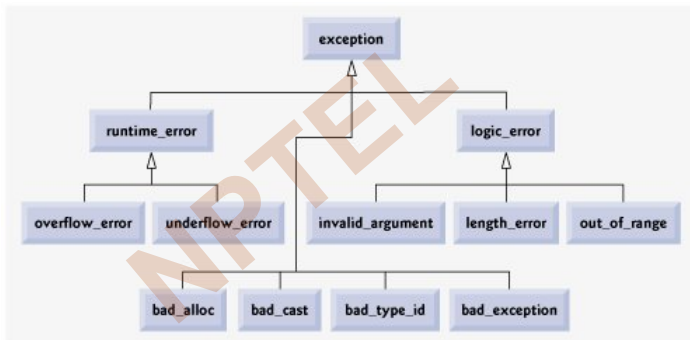
Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

std::exception

Module Summary

Exceptions in Standard Library: `std::exception`

- `logic_error`: Faulty logic like violating logical preconditions or class invariants (may be preventable)
  - `invalid_argument`: An argument value has not been accepted
  - `domain_error`: Situations where the inputs are outside of the domain for an operation
  - `length_error`: Exceeding implementation defined length limits for some object
  - `out_of_range`: Attempt to access elements out of defined range
- `runtime_error`: Due to events beyond the scope of the program and can not be easily predicted
  - `range_error`: Result cannot be represented by the destination type
  - `overflow_error`: Arithmetic overflow errors (Result is too large for the destination type)
  - `underflow_error`: Arithmetic underflow errors (Result is a subnormal floating-point value)
- `bad_typeid`: Exception thrown on typeid of null pointer
- `bad_cast`: Exception thrown on failure to dynamic cast
- `bad_alloc`: Exception thrown on failure allocating memory
- `bad_exception`: Exception thrown by unexpected handler

Sources: `std::exception` and `std::exception` in C++11, C++14, C++17 & C++20

Module M37

Partha Pratim Das

Objectives & Outlines

Exceptions in C++

try-throw-catch

Exception Scope (try)

Exception Arguments (catch)

Exception Matching

Exception Raise (throw)

Advantages

std::exception

Module Summary

Exceptions in Standard Library: `std::exception`: C++98, C++11, C++14, C++17 & C++20

- **`logic_error`**
  - ○ **`invalid_argument`**
  - ○ **`domain_error`**
  - ○ **`length_error`**
  - ○ **`out_of_range`**
  - ○ **`future_error`** (C++11)
- **`bad_optional_access`** (C++17)
- **`runtime_error`**
  - ○ **`range_error`**
  - ○ **`overflow_error`**
  - ○ **`underflow_error`**
  - ○ **`regex_error`** (C++11)
  - ○ **`system_error`** (C++11)
    - ▷ **`ios_base::failure`** (C++11)
    - ▷ **`filesystem::filesystem_error`** (C++17)
  - ○ **`txtion`** (TM TS)
  - ○ **`nonexistent_local_time`** (C++20)
  - ○ **`ambiguous_local_time`** (C++20)
  - ○ **`format_error`** (C++20)

- **`bad_typeid`**
- **`bad_cast`**
  - ○ **`bad_any_cast`** (C++17)
- **`bad_weak_ptr`** (C++11)
- **`bad_function_call`** (C++11)
- **`bad_alloc`**
  - ○ **`bad_array_new_length`** (C++11)
- **`bad_exception`**
- **`ios_base::failure`** (until C++11)
- **`bad_variant_access`** (C++17)

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