TATA ELXSI

OBJECT ORIENTED PROGRAMMING USING C++
Module 9

Learning & Development Team

Exception Handling

Objectives

- What are Exceptions?
- Handling Exceptions in C++ - try, catch, throw
- Catching any exception using ellipsis,
- Nested try blocks
- Standard exceptions
- Custom Exception

Error vs exception

- What is error?
- What types of error you know?
- **Then What is exception?**
- **How to handle exception?**

Try – throw -catch block

- Exceptions provide a way to react to exceptional circumstances (like runtime errors) in programs by transferring control to special functions called **handlers**.
- To catch exceptions, a portion of code is placed under exception inspection.
- This is done by enclosing that portion of code in a try-block.
- When an exceptional circumstance arises within that block, an exception is thrown that transfers the control to the exception handler.
- If no exception is thrown, the code continues normally and all handlers are ignored.

Handling exception

- An exception is thrown by using the throw keyword from inside the try block.
- Exception handlers are declared with the keyword catch, which must be placed immediately after the try block:

```
int main() {
  try {
    throw 20;
  }
  catch (int e) {
    cout << "An exception occurred. Exception  " << e << '\n';
  }
  return 0;
}
```

Catching any exception using ellipsis

- If the catch statement specifies an ellipsis (...) instead of a type, the catch block handles every type of exception.
- an ellipsis handler must be the last handler for the associated try block.
- Use catch(...) with caution;
- do not allow a program to continue unless the catch block knows how to handle the specific exception that is caught.
- Typically, a catch(...) block is used to log errors and perform special cleanup before program execution is stopped.

Handling multiple exception

- Multiple handlers (i.e., catch expressions) can be chained; each one with a different parameter type.

- Only the handler whose argument type matches the type of the exception specified in the throw statement is executed.

```
try {
      // code here
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

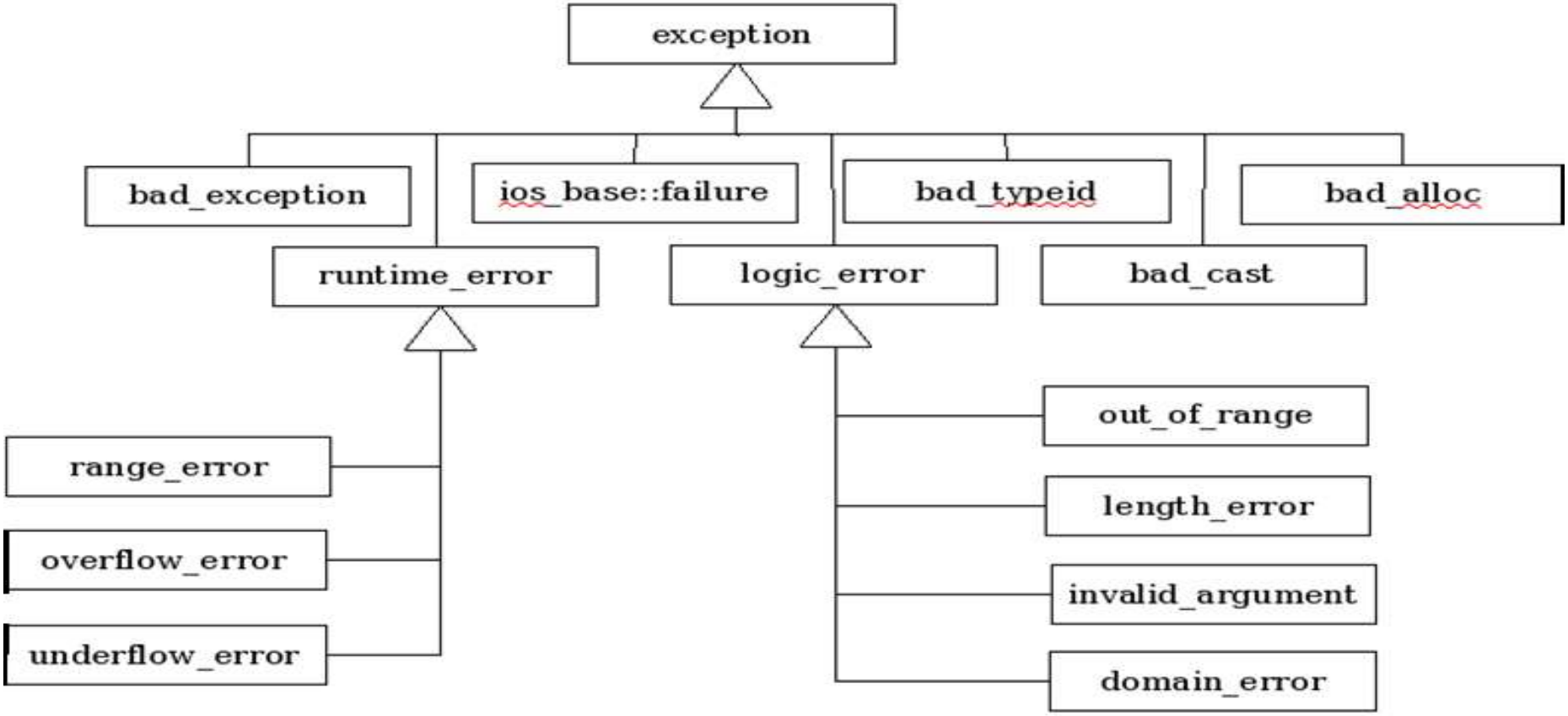
Nested try blocks

- It is also possible to nest try-catch blocks within more external try blocks.
- In these cases, we have the possibility that an internal catch block forwards the exception to its external level.
- This is done with the expression throw; with no arguments.

For example:

```
try {
                try { // code here     }
                catch (int n) { throw;}
     }
catch (...) {
       cout << "Exception occurred";
          }
```

Standard exceptions



Standard exceptions

- All exceptions thrown by components of the C++ Standard library throw exceptions derived from this exception class.
- These are:

exception	Description
bad_alloc	thrown by new on allocation failure
bad_cast	thrown by dynamic_cast when it fails in a dynamic cast
bad_exception	thrown by certain dynamic exception specifiers
bad_typeid	thrown by typeid
bad_function_call	thrown by empty function objects
bad_weak_ptr	thrown by shared_ptr when passed a bad weak_ptr

Standard exceptions

- Also deriving from exception, header **<exception>** defines two generic exception types that can be inherited by custom exceptions to report errors:

exception	description
logic_error	error related to the internal logic of the program
runtime_error	error detected during runtime

Class std::bad_alloc

If operator new fails then , throws bad_alloc.

Ex: **// bad_alloc example**

```
#include <iostream>      // std::cout
#include <new>           // std::bad_alloc

int main ()
{
   try {
     int* myarray= new int [ 10000]; //request to the large memory new may fail
    }
    catch (std::bad_alloc& ba)   {
         std::cerr << "bad_alloc caught: " << ba.what() << '\n';
     }
     return 0;
}
```

User Defined Exception

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

```
int main(){
  try  {
    throw MyException();
  }
catch(MyException& e)
  {
    std::cout << "MyException caught";
    std::cout << e.what() << std::endl;
  }
  catch(std::exception& e)
  {
    //Other errors
  }
}
```

Are you ready to solve…

1. In C++ we can write nested try block.
   a. True
   b. False

   Ans: **a. True**

2. Which exception is thrown by new on allocation failure.

   a. bad_new b. bad_alloc c. alloc_fail d. bad_except

   Ans: **b. bad_alloc**

End of Module 9

Disclaimer

- Some examples and concepts have been sourced from the below links and are open source material
  - **http://cppreference.com**
  - **www.cplusplus.com**

- References:
  - *C++: The Complete Reference* - 4th Edition by Herbert Schildt, Tata McGraw-Hill publications.
  - *The C++ Programming Language*- by Bjarne Stroustrup.
  - *Practical C++ Programming*- by Steve Oualline, O'Reilly publications.

Thank You

Learning & Development Team

ITPB Road Whitefield

Bangalore 560 048 India

Tel +91 80 2297 9123

Fax +91 80 2841 1474

e-mail info@tataelxsi.com

www.tataelxsi.com