C++ Associate Programing

Exception Handling in C++

Lesson Objectives

- *Overview Exceptions*
- *Why Exception Handling?*
- *Exception Handling in C++*

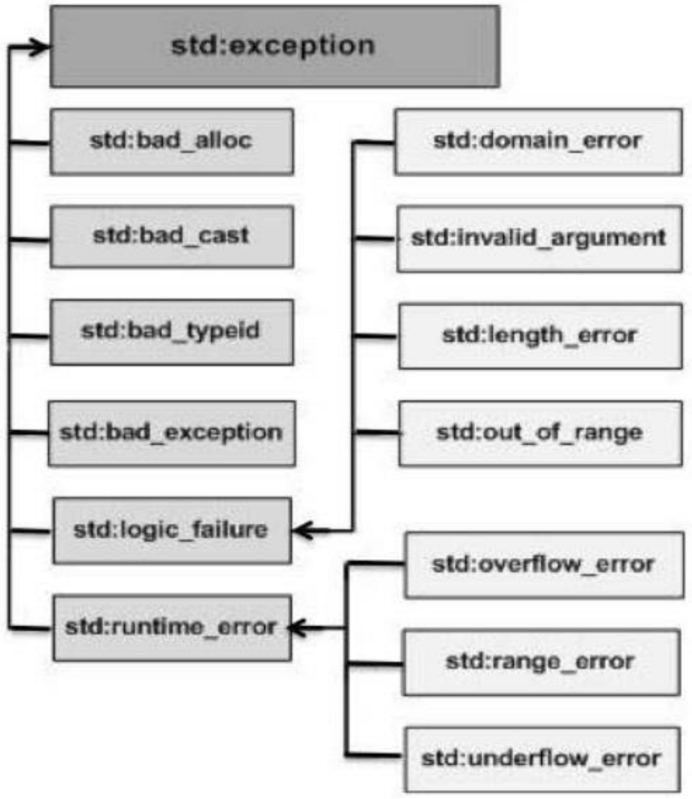
Section 1

OVERVIEW EXCEPTIONS

Overview Exceptions

- One of the advantages of C++ over C is Exception Handling. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. There are two types of exceptions: a)Synchronous, b)Asynchronous (Ex:which are beyond the program’s control, Disc failure etc). C++ provides following specialized keywords for this purpose.
- try: represents a block of code that can throw an exception.
- catch: represents a block of code that is executed when a particular exception is thrown.
- throw: Used to throw an exception. Also used to list the exceptions that a function throws, but doesn’t handle itself.

Overview Exceptions: C++ Standard Exceptions



C++ provides a list of standard exceptions defined in <exception> which we can use in our programs. These are arranged in a parent-child class hierarchy shown below:

Overview Exceptions: 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, for example a std::vector and std::bitset<>::operator[]().
- **std::runtime_error -** An exception that theoretically cannot be detected by reading the code.
- **std::overflow_error -** This is thrown if a mathematical overflow occurs.
- **std::range_error -** This is occurred when you try to store a value which is out of range.
- **std::underflow_error -** This is thrown if a mathematical underflow occurs.

Section 2

WHY EXCEPTION HANDLING?

Why Exception Handling? (1)

Separation of Error Handling code from Normal Code:

- In traditional error handling codes, there are always if else conditions to handle errors.
- These conditions and the code to handle errors get mixed up with the normal flow.
- This makes the code less readable and maintainable.
- With try catch blocks, the code for error handling becomes separate from the normal flow.

Why Exception Handling? (2)

Functions/Methods can handle any exceptions they choose:

- A function can throw many exceptions, but may choose to handle some of them.

- The other exceptions which are thrown, but not caught can be handled by caller. If the caller chooses not to catch them, then the exceptions are handled by caller of the caller.

- In C++, a function can specify the exceptions that it throws using the throw keyword. The caller of this function must handle the exception in some way (either by specifying it again or catching it)

Why Exception Handling? (3)

Grouping of Error Types:

- In C++, both basic types and objects can be thrown as exception.
- We can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types.

Section 3

EXCEPTION HANDLING IN C++

Exception Handling in C++ (1)

```
#include <iostream>
using namespace std;
int main()
{
  int x = -1;
  // Some code
  cout << "Before try \n";
  try {
    cout << "Inside try \n";
    if (x < 0)
    {
      throw x;
      cout << "After throw (Never executed) \n";
    }
  }
  catch (int x ) {
    cout << "Exception Caught \n";
  }
  cout << "After catch (Will be executed) \n";
  return 0;
}
```

Following is a simple example to show exception handling in C++. The output of program explains flow of execution of try/catch blocks.

Output:

```
Before try
Inside try
Exception Caught
After catch (Will be executed)
```

Exception Handling in C++ (2)

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

int main()
{
   try  {
      throw 10;
   }
   catch (char *excp)  {
      cout << "Caught " << excp;
   }
   catch (...)  {
      cout << "Default Exception\n";
   }
   return 0;
}
```

There is a special catch block called 'catch all' catch(…) that can be used to catch all types of exceptions. **For example**, in the following program, an int is thrown as an exception, but there is no catch block for int, so catch(…) block will be executed.

Output:

```
Default Exception
```

Exception Handling in C++ (3)

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

int main()
{
   try  {
     throw 'a';
   }
   catch (int x)  {
      cout << "Caught " << x;
   }
   catch (...)  {
      cout << "Default Exception\n";
   }
   return 0;
}
```

Implicit type conversion doesn't happen for primitive types. **For example**, in the following program 'a' is not implicitly converted to int

Output:

```
Default Exception
```

Exception Handling in C++ (4)

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

int main()
{
   try  {
     throw 'a';
   }
   catch (int x)  {
      cout << "Caught ";
   }
   return 0;
}
```

If an exception is thrown and not caught anywhere, the program terminates abnormally. For example, in the following program, a char is thrown, but there is no catch block to catch a char.

Output:

```
terminate called after throwing an instance of 'char'

This application has requested the Runtime to terminate it in an
unusual way. Please contact the application's support team for
more information.
```

Exception Handling in C++ (5)

```
#include <iostream>
using namespace std;
int main()
{
   try {
      try  {
         throw 20;
      }
      catch (int n) {
          cout << "Handle Partially ";
          throw;   //Re-throwing an exception
      }
   }
   catch (int n) {
      cout << "Handle remaining ";
   }
   return 0;
}
```

In C++, try-catch blocks can be nested. Also, an exception can be re-thrown using “throw; ”

Output:

```
Handle Partially Handle remaining
```

Exception Handling in C++ (6)

```
#include <iostream>
using namespace std;
class Test {
public:
  Test() { cout << "Constructor of Test " << endl;
}
 ~Test() { cout << "Destructor of Test "  << endl;
}
};
int main() {
  try {
    Test t1;
    throw 10;
  } catch(int i) {
    cout << "Caught " << i << endl;
  }
}
```

When an exception is thrown, all objects created inside the enclosing try block are destructed before the control is transferred to catch block.

Output:

```
Constructor of Test
Destructor of Test
Caught 10
```

Fpt Software | CAMPUS

Thank you

Q&A