CAP444
OBJECT ORIENTED PROGRAMMING USING C++

Unit6

Created By:
Kumar Vishal
(SCA), LPU

Unit-6

Exception handling :

- principles of exception handling,
- exception handling mechanism,
- Multiple catch statements,
- catching multiple exceptions,
- re-throwing exceptions,
- exceptions in constructors and destructors,
- controlling uncaught exceptions

Real Life Example

❖Exception handling

- Exception is an abnormal condition.
- Exception is an event that disturbs the normal flow of the program.
- Result out of expectation

Exception Handling Keywords:

- **Try :** The try block contain statements which may generate exceptions.
- **Throw :** When an exception occur in try block, it is thrown to the catch block using throw keyword.
- **Catch :**The catch block defines the action to be taken, when an exception occur.

Syntax:

```
try {
  // Block of code to try
  throw exception; // Throw an
exception when a problem arise
}
catch () {
  // Block of code to handle
exception
}
```

```
#include <iostream>

using namespace std;
float division(float x, float
y)
{
   return x/y;
}
int main()
{
   cout<<division(5.0f,0);
   return 0;
}
```

```
#include <iostream>
using namespace std;
float division(float x, float y) {
  if( y == 0 ) {
    throw "Attempted to divide by zero!";
  }
  return (x/y);
}
int main()
{
try{
  cout<<division(5.0f,0);
}
catch(const char *excep)
{
cerr<<excep;
}
  return 0;
}
```

Standard error stream (cerr): cerr is the standard error stream which is used to output the errors. It is an instance of the ostream class.

Which of the following statements are true about Catch handler?

i) It must be placed immediately after try block .

ii) It can have multiple parameters.

iii) There must be only one catch handler for every try block.

iv) There can be multiple catch handler for a try block .

v) Generic catch handler can be placed anywhere after try block.

A. Only i, iv, v
B. Only i, ii, iii
C. Only i, iv
D. Only i, ii

multiple catch statements:

A **single try statement** can have multiple catch statements. Execution of particular catch block depends on the type of exception thrown by the throw keyword. If throw keyword send exception of integer type, catch block with integer parameter will get execute.

multiple catch statements:

```
try
 {
            throw value
  }
  catch(type1 arg)
  {
            //catch block1
  }
  catch(type2 arg)
  {
            //catch block2
  }
  ………
  ………
```

. If throw keyword send exception of integer type, catch block with integer parameter will get execute.

Which of the following statements are correct?

A. The execution of a throw statement is called throwing an exception.

B. You can throw a value of any type.

C. The catch block contains the code that are executed when an exception occurs.

D. All of the above

Catch All Exceptions

We can use the catch statement with three dots as parameter (...) so that it can handle all types of exceptions.

```
catch(...)
{
}
```

Note: The catch(...) must be the last catch block.

Rethrowing Exceptions

- An exception to be thrown from inner catch block to outer catch block is called rethrowing exception.
- Rethrowing exception is possible in case of Nested try-catch statement.

```
try{
          try{
               throw val;
                     }
         catch(type a)                          throws exception
             {
                     throw val
                     }
     }                          rethrows exception
catch(type b)
{

}
```

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Enter two integers: ";
  int number1, number2;
  cin >> number1 >> number2;
  try
  {
    if (number2 == 0)
      throw number1;
 cout << number1 << " / " << number2 <<
" is "
      << (number1 / number2) << endl;
    cout << "C" << endl;
  }
  catch (int e)
  {
    cout << "A";
  }
  cout << "B";   return 0;
}
```

If you enter 1 0, what is the output of the following code?

Options:

A. A

B. B

C. C

D. AB

Exceptions in Constructors and Destructors

- If an exception is raised in a constructor, memory might be allocated to some data members and might not be allocated for others. This might lead to memory leakage problem.
- Similarly, when an exception is raised in a destructor, memory might not be deallocated which may again lead to memory leakage problem.
- So, it is better to provide exception handling within the constructor and destructor to avoid such problems.

Syntax:

```
class division
{
        division()
        {
                try{
                 }
                catch()
                {
                        }
        }

};
```

```
~division()
        {
                try{
                 }
                        catch()
                        {
                        }
        }
```

If inner catch handler is not able to handle the exception then_________ .

A. Compiler will look for outer try handler
B. Program terminates abnormally
C. Compiler will check for appropriate catch handler of outer try block
D. None of the above

What will be output?

```
#include <iostream>
using namespace std;
int main()
{
   try
   {
     throw 'b';
   }
   catch (int param)
   {
      cout << "Int Exception";
   }
   catch (...)
   {
      cout << "Default Exception";
   }
   cout << "After Exception";
   return 0;
}
```

A. Default Exception After Exception
B. Int Exception After Exception
C. Int Exception
D. Default Exception

Uncaught exceptions

In some case when an exception is thrown but isn’t caught because the exception handling subsystem fails to find a matching catch block for that particular exception.

If there’s no exception handler then main() terminates.

Example:

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

int main()
{
    int x=5;
    try{
        throw 0;
    }
    catch(char c)
    {
        cout<<"exception";
    }
    return 0;
}
```

What function will be called when we have a uncaught exception?

A. catch
B. throw
C. terminate
D. none of the mentioned

```
#include <iostream>
using namespace std;
int main()
{
    try
    {
      throw 10;
    }
    catch (...)
    {
       cout << "Default Exceptionn";
    }
    catch (int param)
    {
       cout << "Int Exceptionn";
    }
    return 0;
}
```

A. Default Exception
B. Int Exception
C. Compiler Error
D. None of the above

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

int main()
{
  int P = -1;
  try {
    cout << "Inside try";
    if (P < 0)
    {
      throw P;
      cout << "After throw";
    }
  }
  catch (int P ) {
    cout << "Exception Caught";
  }
  cout << "After catch";
  return 0;
}
```

A. Inside try Exception Caught After throw After catch

B. Inside try Exception Caught After catch

C. Inside try Exception Caught

D. Inside try After throw After catch

Any Query?

Unit-6 End