Practical Lecture : Exception Handling

Quick Recap

Let’s take a quick recap of previous lecture –

- Basics of exception handling
- Exception handling mechanism
- Throwing mechanism
- Catching mechanism

Today’s

Today we are going to cover –

- Rethrowing an exception

Let’s Get Started-

Rethrowing an exception

In C++, try-catch blocks can be nested.

Also, an exception can be re-thrown using “throw; ”

Rethrowing an expression from within an exception handler can be done by calling throw, by itself, with no exception.

This causes current exception to be passed on to an outer try/catch sequence.

An exception can only be rethrown from within a catch block.

When an exception is rethrown, it is propagated outward to the next catch block.

Consider the example given below. Revisit this slide after you go through

Rethrowing an exception

```
int main()
{
    try
    {
        try
        {
            throw 20;
        }
        catch (int n)
        {
            cout << "Inner Catchn";
            throw;
        }
    }
catch (int x)
    {
        cout << "Outer Catchn";
    }
    return 0;
}
```

Output:
Inner Catchn
Outer Catchn

Rethrowing an exception

```
#include <iostream>
using namespace std;
void MyHandler()
{
   try
   {
       throw "hello" ;
   }
   catch (const char*)
   {
   cout <<"Caught exception inside MyHandler\n";
   throw; //rethrow char* out of function
   }
}
```

Rethrowing an exception

```
int main()
{
   cout<< “Main start”;
   try
   {
       MyHandler();
   }
   catch(const char*)
   {
      cout <<”Caught exception inside Main\n”;
   }
       cout << “Main end”;
       return 0;
}
```

Rethrowing an exception

Output:

```
Main start
Caught exception inside MyHandler
Caught exception inside Main
Main end
```

Explanation: The try block in the main() function calls function MyHandler(). The try block in function MyHandler() throws an exception “Hello”. The handler catch (const char*) catches this exception. The handler then rethrows char* out of function with the statement throw to the next dynamically enclosing try block: the try block in the main() function. The generic handler in main catch(...) catches char* exception.

Exception in function calls

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

void fun(int *ptr, int x)  // Dynamic Exception specification
{
    if (ptr == NULL)
        throw ptr;
    if (x == 0)
        throw x;
    /* Some functionality */
}
```

Exception in function calls

```
int main()
{
    try {
       fun(NULL, 0);
    }
    catch(...) {
        cout << "Caught exception from fun()";
    }
}
```

Explanation: If the compiler encounters an exception in a try block, it will try each handler in order of appearance. If the run time cannot find a matching handler in the current scope, the run time will continue to find a matching handler in a dynamically surrounding try block. In function fun(), the run time could not find a handler to handle the exception of type E thrown. The run time finds a matching handler in a dynamically surrounding try block: the try block in the main() function.

Points to remember

A catch block of the form catch(...) must be the last catch block following a try block or an error occurs.

This placement ensures that the catch(...) block does not prevent more specific catch blocks from catching exceptions intended for them.

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

If both base and derived classes are caught as exceptions then catch block of derived class must appear before the base class. If we put base class first then the derived class catch block will never be reached.

When an exception is thrown and not caught, the program terminates

Exception in Constructor / Destructor

```
class Test {
public:
    Test() { cout << "Constructor of Test " << endl; }
    ~Test() { cout << "Destructor of Test " << endl; }
};
int main()
{
    try {
        Test t1; //creating object of Test class using default constructor
        throw 10;
    }
    catch (int i) {
        cout << "Caught " << i << endl;
    }
}
```

```
Constructor of Test

Destructor of Test

Caught 10
```

Exception in inheritance

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

class Base {};
class Derived: public Base {};
int main()
{
   Derived d;
   try {
       throw d;
   }
   catch(Base b) {
        cout<<"Caught Base
Exception";
   }
catch(Derived d)
//This catch block is NEVER executed
 {
        cout<<"Caught Derived
Exception";
   }
   return 0;
}
```

Output:

```
Caught Base Exception
```

Note: Catching a base class exception before derived is not allowed by the compiler itself. Compiler might give warning about it, but compiles the code.

Exception in inheritance

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

class Base {};
class Derived: public Base {};
int main()
{
   Derived d;
   try {
       throw d;
   }
 catch(Derived d)
 {
        cout<<"Caught Derived
Exception";
   }
```

```
catch(Base b) {
        cout<<"Caught Base
Exception";
   }
```

Output:

```
Caught Derived Exception
```

Note: If we change the order of catch statements then both catch statements become reachable. Above is the modified program and it prints *“Caught Derived Exception”*

Standard exceptions

<This slide is only for knowledge, and won’t be included for exam>

C++ provides a list of standard exceptions defined in <exception> which we can use in our programs.

1. std::exception :An exception and parent class of all the standard C++ exceptions.
2. std::bad_alloc : This can be thrown by new.
3. std::range_error : This is occurred when you try to store a value which is out of range.
4. std::underflow_error: This is thrown if a mathematical underflow occurs.
5. std::overflow_error: This is thrown if a mathematical overflow occurs.

Advanatages of exception handling

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.

Programmers can deal with them at some level within the program

If an error can't be dealt with at one level, then it will automatically be shown at the next level, where it can be dealt with.

Advanatages of exception handling

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)

Example revisited of try catch

Dynamic memory allocation failure program

Memory allocation failure

If memory allocation using new is failed in C++ then how it should be handled?

When an object of a class is created dynamically using new operator, the object occupies memory in the heap.

Below are the major thing that must be kept in mind:

1. What if sufficient memory is not available in the heap memory, and how it should be handled? - using try and catch block
2. If memory is not allocated then how to avoid the project crash? – prevent memory crash by throwing an exception

Memory allocation failure

```
#include <iostream>
using namespace std;
int main()
{
    // Allocate huge amount of memory
    long MEMORY_SIZE = 0x7fffffff;
     // Put memory allocation statement
    // in the try catch block
    try {
        char* ptr = new char[MEMORY_SIZE];
         // When memory allocation fails, below line is not be executed
        // & control will go in catch block
        cout << "Memory is allocated“ << " Successfully" << endl;
    }
```

Memory allocation failure

```
// Catch Block handle error
    catch (const bad_alloc& e) {

        cout << "Memory Allocation“ << " is failed: “    << e.what()   <<
endl;
    }

    return 0;
}
```

Output:

```
Memory Allocation is failed: std::bad_alloc
```

The above memory failure issue can be resolved without using the try-catch block. It can be fixed by using nothrow version of the new operator.

Memory allocation failure

The nothrow constant value is used as an argument for operator new and operator new[] to indicate that these functions shall not throw an exception on failure but return a null pointer instead.

By default, when the new operator is used to attempt to allocate memory and the handling function is unable to do so, a bad_alloc exception is thrown.

But when nothrow is used as an argument for new, and it returns a null pointer instead.

This constant (nothrow) is just a value of type nothrow_t, with the only purpose of triggering an overloaded version of the function operator new (or operator new[]) that takes an argument of this type.

Note

Teachers are encouraged to discuss the solutions of the following programs by executing them if required.

Practice question

What will be the output of the following program?

```
class Base {};
class Derived: public Base {};
int main(){
   Derived d;
   try {
       throw d;
   }
   catch(Base b) {
        cout<<"Caught Base Exception";
   }
   catch(Derived d) {
        cout<<"Caught Derived
Exception";
   }
 }
```

Practice question

What will be the output of the following program?

```
class Base {};
class Derived: public Base {};
int main(){
   Derived d;
   try {
       throw d;
   }
   catch(Base b) {
        cout<<"Caught Base Exception";
   }
   catch(Derived d) {
        cout<<"Caught Derived
Exception";
   }
 }
```

Output:

Caught Base Exception

Practice question

What will be the output of the following program?

```
int main(){
    try    {
       throw 'a';
    }
    catch (int param)     {
        cout << "int exceptionn";
    }
    catch (...)     {
        cout << "default exceptionn";
    }
    cout << "After Exception";
    return 0;
}
```

Practice question

What will be the output of the following program?

```
int main(){
    try    {
       throw 'a';
    }
    catch (int param)     {
        cout << "int exceptionn";
    }
    catch (...)     {
        cout << "default exceptionn";
    }
    cout << "After Exception";
    return 0;
}
```

Output:

```
default Exception
After Exception
```

Practice question

What will be the output of the following program?

```
int main() {
    try     {
       throw 10;
    }
    catch (...)     {
        cout << "default exception";
    }
    catch (int param)     {
        cout << "int exception";
    }

    return 0;
}
```

Options:
1. default exception
2. int Exception
3. Compile error
4. default exception int exception

Practice question

What will be the output of the following program?

```
int main() {
    try     {
       throw 10;
    }
    catch (...)     {
        cout << "default exception";
    }
    catch (int param)     {
        cout << "int exception";
    }

    return 0;
}
```

Options:
1. default exception
2. int Exception
3. Compile error
4. default exception int exception

Practice question

Which of the following is true about exception handling in C++?
1. When an exception is rethrown, it is propagated outward to the next catch block.
2. A catch block of the form catch(...) must be the last catch block following a try block or an error occurs.
Options:
1. 1 only
2. 2 only
3. Both are true
4. Both are false

Practice question

Which of the following is true about exception handling in C++?
1. When an exception is rethrown, it is propagated outward to the next catch block.
2. A catch block of the form catch(...) must be the last catch block following a try block or an error occurs.

Options:

1. 1 only
2. 2 only
3. Both are true
4. Both are false

Practice question

What happens in C++ when an exception is thrown and not caught anywhere like in the following program?

```
#include <iostream>
using namespace std;
 int fun() throw (int)
{
    throw 10;
}

 int main() {
   fun();
   return 0;
}
```

Options:

1. Compile error
2. Abnormal program termination
3. Program doesn’t print anything and terminates normally
4. None of the above

Practice question

What happens in C++ when an exception is thrown and not caught anywhere like in the following program?

```
#include <iostream>
using namespace std;
 int fun() throw (int)
{
    throw 10;
}

 int main() {
   fun();
   return 0;
}
```

Options:

1. Compile error
2. Abnormal program termination
3. Program doesn’t print anything and terminates normally
4. None of the above

Practice Question

Write a c++ program to accept a character from keyboard. If it is not an alphabet, not a number then throw an appropriate exception and catch it using multiple catch statements and generalized catch.

Hint:

```
If not an alphabet
      throw("not an alphabet")
Else if not a number
      throw "not a number
Else
      throw "Special char"
```

Practice Question

Write a c++ program to accept a character from keyboard. If it is not an alphabet, not a number then throw an appropriate exception and catch it using multiple catch statements and generalized catch.

Hint:

```
If not an alphabet
      throw(“not an alphabet”)
Else if not a number
      throw “not a number
Else
      throw “Special char”
```

Practice Question

```
#include<iostream>
using namespace std;
int main(){
      char ch;
      cout<<"Enter a char";
      cin>> ch;
      try
      {
      If ((!isalpha(ch)) && (!isdigit(ch)))
            throw ch;
      else if (!isalpha(ch))
            throw "not an alphabet";
      else if (!isdigit(ch))
            throw 1;
      }
```

Practice Question

```
 catch(const char* ex)
{
           cout<<ex<<endl;
}
catch (int n)
{
 cout <<"not a number"<<endl;
}
catch (...)
{
cout<< " It is a special character";
 }
return 0;

}
```

Assignment

Write a c++ program to accept 5 numbers from user in an array and handle the exceptions for positive , -ve numbers and equal to 0 numbers using multiple try catch statements.

Sample Input: 1 -2 0 2 -4

Sample output:

```
1- Positive number
-2 – negative number
0 – Zero
2 -positive number
-4 – negative number
```

Any
Questions ??

Thank You!

See you guys in next class.