Lecture 22

Exception Handling in C++

We cannot ensure that code will execute normally at all times. There can be certain situations that might force the code written by us to malfunction, even though it's error-free. This malfunctioning of code is called **Exception**.

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.

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.

When an exception has occurred, the compiler has to throw it so that we know an exception has occurred. When an exception has been thrown, the compiler has to ensure that it is handled properly, so that the program flow continues or terminates properly. This is called the **handling of an exception.**
Thus in C++, we have three keywords i.e. **try**, **throw** and **catch** which are in exception handling.

The general syntax for exception block is:

```
try{
 ….
 # Code that is potentially about to throw exception goes here
 ….
 throw exception;
 }
 catch(exception type) {
 …
 #code to handle exception goes here
 }
```

As shown above, the code that might potentially malfunction is put under the try block. When code malfunctions, an exception is thrown. This exception is then caught under the catch block and is handled i.e. appropriate action is taken.

Why Exception Handling?

Following are main advantages of exception handling over traditional error 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.
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)
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 namespace or classes, categorize them according to types.

C++ Exceptions:
When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an exception (throw an error).

C++ try and catch:

Exception handling in C++ consists of three keywords: try, throw and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

We use the try block to test some code: If the age variable is less than 18, we will throw an exception, and handle it in our catch block.

In the catch block, we catch the error and do something about it. The catch statement takes a parameter: in our example we use an int variable (myNum) (because we are throwing an exception of int type in the try block (age)), to output the value of age.

If no error occurs (e.g. if age is 20 instead of 15, meaning it will be be greater than 18), the catch block is skipped:

Exception Handling in C++

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

```
#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;
}
```

Output:

```
Before try

Inside try

Exception Caught

After catch (Will be executed)
```

2) 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.

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

int main()
{
```

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

Output:

```
Default Exception
```

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

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

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

Output:

```
Default Exception
```

4) 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.

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

int main()
{
	try {
```

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

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.
```