Object Oriented Programming in C++

Nepal College of Information Technology

Course Instructor : Er. Rabina Chaudhary

Errors:

- Syntax error: occurs due to typing mistake in syntax or writing wrong syntax

- Semantic error: program having logical errors is compiled and executed successfully but doesn’t produce expected output
- Linker error: program is compiled successfully but error occur during linkage phase example when function prototype is written and that function is called in the program, but there is no function definition

Run time errors:

- Errors which occur when the program is running (during execution time)
- The program having runtime error is compiled successfully but the error occurs while execution of the program
- A program crash in unexpected way while running is caused by run time error

Exception handling :

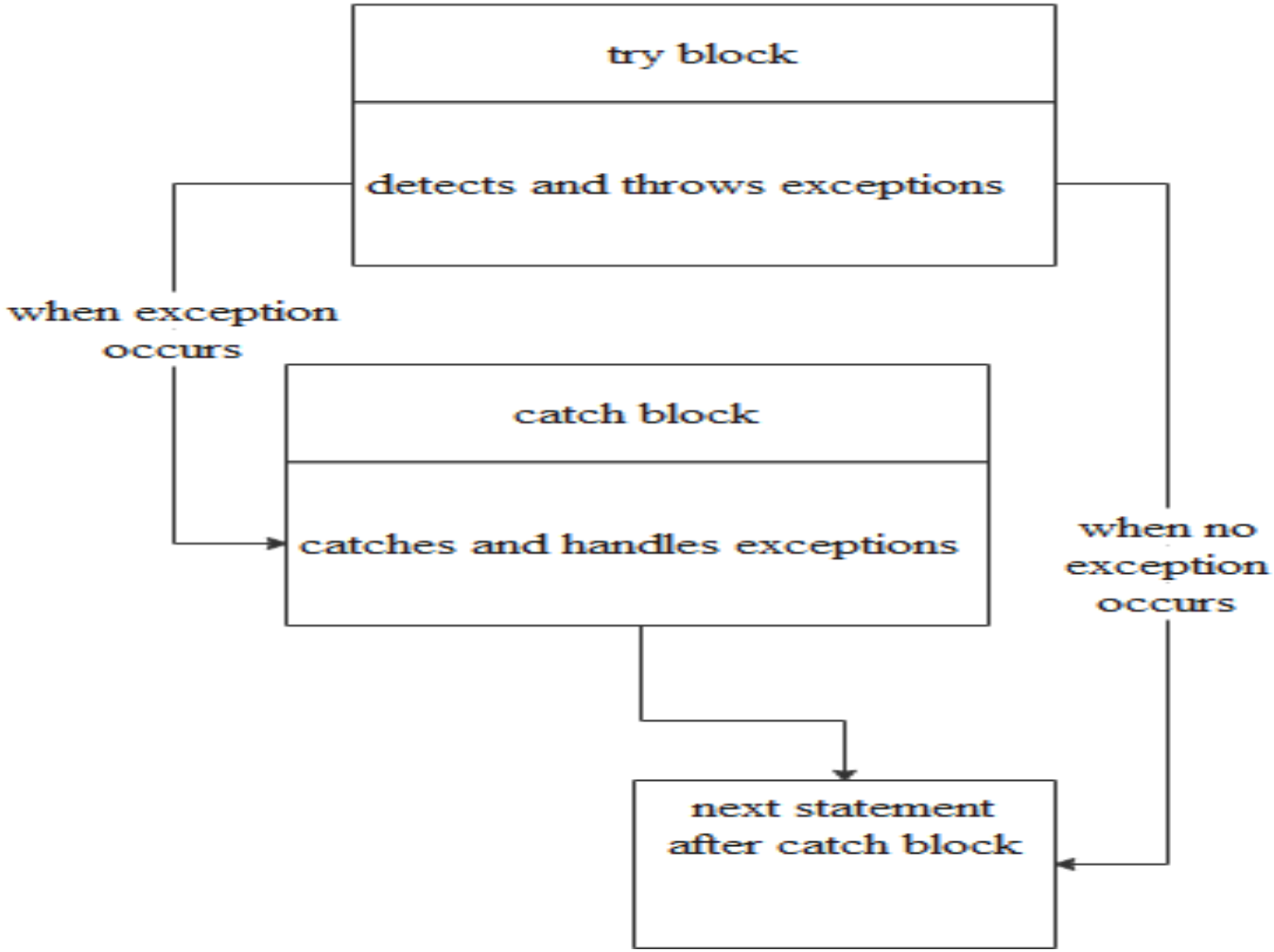
- C++ has mechanism to handle run time error called as exception handling mechanism
- Exception handing mechanism is responsible to detect and report exceptional circumstances so that appropriate action can be taken
- Exceptional handling mechanism performs the following tasks

1. Find the problem (hit the exception)
2. Inform that error has occurred (throw the exception)
3. Receive the error information (catch the exception)
4. Take corrective actions (handle the exception)

- Error handling code consists of two segments

1. One segment to detect errors and throw exceptions
2. The other segment to catch the exception and take appropriate actions

Exception handling mechanism consists of three key words



General format of try, throw and catch:

```
…
 try
  {
        …
        throw exception;
        …
  }
 catch( type argument)
  {
        //block of statements that handles the exception
  }
```

try block:

- try block is a block of code for which particular exceptions will be analyzed and detected
- try block encloses the block of code that we want to analyze for exception
- If the try block detects exceptions, the exception is thrown to catch block using throw key word and the program control also is transferred to catch block
- If no error is found in try block, the control never goes into catch block

Syntax:

```
try
{
    //code to be monitored for execution
    throw exception;
}
```

catch block:

- Syntax

```
catch (type argument)
{
        //block of statements that handles the exception
}
```

- Type is the type of exception that it handles and argument is parameter name
- Catch statement catches an exception whose type matches with the type of catch argument
- When the exception is caught, the code in the catch block is executed

throw statement:

- An exception is thrown to a particular catch block using throw keyword
- When the exception is thrown, it will be caught by the catch statement associated with the try block
- When the try block throws an exception, the program control is transferred to the catch block

Program to handle division by zero exception.

```
#include<iostream.h>
#include<conio.h>

void main()
{
                char error[]="ERROR!!! division by zero!!!";
                float num1,num2,result;
                cout<<"Enter two numbers :";
                cin>>num1>>num2;
  try
  {
                if(num2!=0)
                {
                result =num1/num2;
                cout<<"The  result after division is "<<result<<endl;
                }
                else
                                throw error;
  }
catch(char e[] )
  {
                                cout<<"The  denominator must be a non zero"<<endl<<e<<endl;
  }
  cout<<endl<<"This  is last statement "<<endl;
  getch();
}
```

Multiple Catch Block:

- It is possible for a program segment to have more than one condition to throw an exception
- In this case, we can associate more than one catch statement with a try statement
- To handle different types of situation, we can specify two or more catch blocks, each catching different types of exceptions
- When exception is thrown, each catch statement is inspected in order and first one whose type matches with that of exception is executes and other catch blocks are ignored and execution continues after the try catch blocks

Multiple catch block

```
try
        {
                //try block
        }
catch(type1 argument)
        {
                //catch block1
        }
catch(type2 argument)
        {
                //catch block2
        }
catch(typeN argument)
        {
                //catch blockN
        }
```

Example program:

```
#include<iostream>
using namespace std;
int main()
{
        int array[5]={1,2,3,4,5};
        int deno,size;
        try
        {
                cout<<"Enter size of array!!!";
                cin>>size;
                if(size>5)
                {
                throw 1;
                }

                cout<<"Enter denominator value :";
                cin>>deno;
                if(deno==0)
                {
                throw 'a';
                }
                for(int i=0;i<size;i++)
                {
                cout<<"the result is "<<array[i]/deno<<endl;
                }
        }
```

Program continued…

```
catch(int a)
        {
                cout<<"Exception Caught!!!"<<a<<endl;
        }
catch(const char a[])
        {
                cout<<"Exception Caught!!!"<<a<<endl;
        }

        return 0;

}
```

Catching all types of exceptions by single catch block:

- We can use a single catch block to catch all types of exception

Syntax to define catch block to catch all types of exception:

```
catch(…)
{
        //statements to process all the exceptions
}
```

Example program:

```
#include<iostream>
using namespace std;
int main()
{
        int array[5]={1,2,3,4,5};
        int deno,size;
        try
        {
                cout<<"Enter size of array!!!";
                cin>>size;
                if(size>5)
                {
                throw 1;
                }

                cout<<"Enter denominator value :";
                cin>>deno;
                if(deno==0)
                {
                throw 'a';
                }
                for(int i=0;i<size;i++)
                {
                cout<<"the result is "<<array[i]/deno<<endl;
                }
        }
```

Program Continued…

```
catch(...)
{
cout<<"Exception Caught"<<endl;
}

return 0;
}
```