Exception Handling

Exception

- Exceptions are the run time anomalies or unusual conditions that a program may encounter while executing.
- Examples- division by zero, access to an array outside of its bounds, running out of memory space.
- Because exceptions are outside the normal operation of a program, **default action** is to write out an error message and terminate the offending process.

Types of exceptions

- Synchronous – Errors such as “out of range” and “over-flow” are synchronous.
- Asynchronous – Errors that are caused by events beyond the control of the program (keyboard interrupts) are called asynchronous exceptions.
  - **Exception handling in C++ is designed to handle only synchronous exceptions.**

Exception handling mechanism

- Find the problem (*Hit the exception*)
- Inform that an error has occurred.(*Throw an exception)*
- Receive the error information. (*catch the exception*)
- Take corrective actions. (*Handle the exception*)

- ***try block-*** A block of statements which may generate exceptions.
- When an ***exception*** is detected, it is thrown using a ***throw*** statement in the try block.
- A catch block defined by the keyword ***catch*** ‘catches’ the e***xception*** ‘thrown’ by the throw statement in the try block and handles it appropriately.

NOTE:

The ***catch*** block that catches an exception must immediately follow the ***try*** block that throws the exception.

```
……
……
try
{
    ……
    throw exception;          //block of statements which
    …….                       //detects and throws an exception
}
catch(type arg)                          //catches the exception
{
    ……
    ……                        //block of statement that handles
}                             // the exceptions
……
```

Example

```
#include <iostream.h>
int main()
{
int a,b;
cout<<“Enter the values of a
    and b”;
cin>>a;
cin>>b;
int x= a-b;
try
{
    if(x!=0)
    {
    cout<<“Result(a/x)=“<<a/x;
    }
    else                    //there is an
                            //exception
    {
    throw(x);               //throws  int
    }                       //object
}
catch(int i)
{
    cout<<“exception
                caught:x=“<<x;
}
cout<<“end”;
return 0;
}
```

Output

First run:

Enter values of a and b

20 15

Result (a/x)= 4

End

Second run:

10 10

Exception caught : x= 0

End

Throw point outside the try block

```
type function(arg_list)                    //function with exception
{
    .....
    throw(object);                         //Throws exception
    .......
}
.....
.....
try
{
    ......                 //invoke function here.
    ......
}
catch(type arg)                    //catches the exception
{
    ..........
    ..........             //Handles exception here
}
```

Example

```
#nclude <iostream.h>
void divide(int x, int y, int z)
{
cout<<“We are inside the
    function”;
    If((x-y)!=0)
    {
          int R= z/(x-y);
          cout<<“Result=“<<R;
    }
    else
    {
     throw(x-y);
    }
}
```

```
int main()
{
    try
    { cout<<“We are inside the
    try block”;
    divide(10,20,30);
    divide(10,10,20);
    }
    catch(int i)
    {
          cout<<“caught the
          exception”;
    }
    return 0;
}
```

Multiple catch statements


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

Multiple catch statements

```
#include<iostream.h>
Void test(int x)
{
    try
    {       if(x==1) throw x;                   //int
            else if(x==0) throw 'x';            //char
            else if (x== -1) throw 1.0;         //double
            cout<<"End of try block";
    }
    catch(char c)                               //catch 1
    {       cout<<"caught a character";}
    catch(int m)
    {       cout<<"caught an integer";}
    catch(double d)
    {       cout<<"caught a double";}
    cout<<"End of try-catch system";
 }
```

Multiple catch statements

```
Int main()
{
Cout<<“testing multiple catches”;
Cout<<“x==1”;
Test(1);
Cout<<“x==0”;
Test(0);
Cout<<“x==-1”;
Test(-1);
Cout<<“x==2”;
Test(2);                    //does not throw any exception
Return 0;                   //and control passes to the next
                            //statement after last catch
}
```

Catch all exceptions

- In some situations, we may not be able to predict all possible types of exceptions and therefore may not be able to design independent **catch** handlers to catch them.
- In such situations, we can force a **catch** statement to catch all exceptions instead of a certain type alone.

```
catch(...)
{
        // Statements for processing
       // all exceptions
}
```

Example

```
#include<iostream>
using namespace std;
void test (int x)
{
    try
    {
        if(x==0) throw x;          //int
if(x== -1) throw 'x';       // char
if(x == 1) throw 1.0;              // float
    }
     catch(…)        // catch all
    {
       cout<<"caught an exception";
    }
 }
```

```
int main()
 {
    cout<<"testing generic catch";
    test(-1);
    test(0);
    test (1);
    return 0;
  }
```

OUTPUT:

```
testing generic catch
caught an exception
caught an exception
caught an exception
```

Catching Class types as Exception

```
#include<iostream.h>
#include<string.h>
class error
 {
   int err_code;
   char *err_desc;
  public:
    error(int c, char *d)
    {
    err_code=c;
    err_desc=new char[strlen (d)];
    strcpy(err_desc, d);
   }
    void err_display(void)
   {
    cout<<“Error code:”<<err_code<<“error
            description:”<< err_desc;
};
int main()
{
    try
     {
    cout<<“Press any key:”;
    getch();
  throw error(99, “test exception”);
}
Catch( error e)
{
 cout<<“exception caught successfully”;
e.err_display();
 }
getch();
return 0;
}
```

OUTPUT:

Press any key

Exception caught successfully.

Error code: 99

Error description: Test exception

Rethrowing an exception

- Rethrowing causes the current exception to be thrown to the next enclosing **try/catch** sequence and is caught by a **catch** statement listed after that enclosing **try** block.
- In such situations, we may invoke **throw** without any arguments as:

```
throw;
```

Example

```
#include<iostream>
using namespace std;
void divide(double x, double y)
{
 cout<<" Inside function";
 try
{
   if(y==0.0)
      throw y;       //Throwing double
   else
     cout<<" Division= " << x/y;
}
catch( double)    // Catch a double
{
   cout<<" caught double inside function";
   throw;            // Rethrowing double
}
 cout<<" end of function";
}
int main()
{
  cout<<" Inside main";
 try
 {
   divide(10.5,2.0);
   divide(20.0, 0.0);
 }
 catch(double)
{
cout<<"caught double inside main";
 }
 cout<<"end of main";
 return 0;
}
```

- OUTPUT:

Inside main

Inside function

Division =5.25

End of function

Inside function

Caught double inside function

Caught double inside main

End of main

- When an exception is rethrown, it will not be caught by same **catch** statement or any other **catch** in that group.
- It will be caught by an appropriate **catch** in the outer **try/catch** sequence only