

UNIT -5

Exception handling

Exception handling:-

Exceptions: Exceptions are runtime anomalies or unusual conditions that a program may encounter while executing. Anomalies might include conditions such as division by zero, accessing an array outside of its bounds or running out of memory or disk space. When a program encounters an exception condition, it must be identified and handled.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

Types of exceptions: There are two kinds of exceptions

1. Synchronous exceptions

1. Asynchronous exceptions

1. Synchronous exceptions: Errors such as “Out-of-range index” and “over flow” are synchronous exceptions

2. Asynchronous exceptions: The errors that are generated by any event beyond the control of the program are called asynchronous exceptions

The purpose of exception handling is to provide a means to detect and report an exceptional circumstance

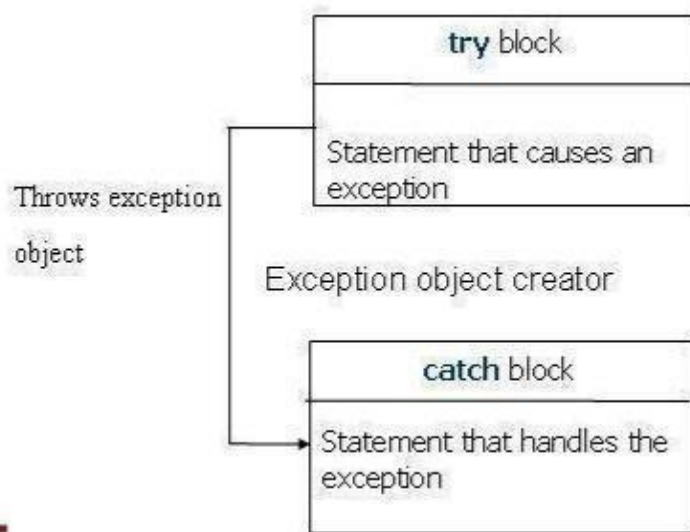
Exception Handling Mechanism:

An exception is said to be thrown at the place where some error or abnormal condition is detected. The throwing will cause the normal program flow to be aborted, in a raised exception. An exception is thrown programmatically, the programmer specifies the conditions of a throw.

In handled exceptions, execution of the program will resume at a designated block of code, called a catch block, which encloses the point of throwing in terms of program execution.

C++ exception handling is built upon three keywords: try, catch, and throw.

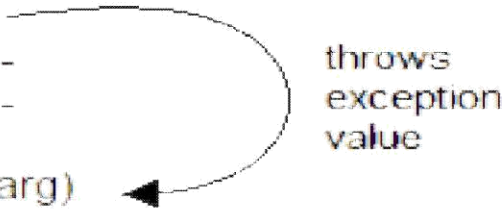
Try is used to preface a block of statements which may generate exceptions. This block of statements is known as try block. When an exception is detected it is thrown by using throw statement in the try block. Catch block catches the exception thrown by throw statement in the try block and handles it appropriately.



```

try
{
    -----
    -----
    throw val;
    -----
    -----
}
catch( data-type arg)
{
    -----
    -----
    -----
}

```



throws exception value

```

#include<iostream>
using namespace std;
int main()
{
    int a,b;
    cout<<"Enter any two integer values";
    cin>>a>>b;
    int x=a-b;
    try
    {
        if(x!=0)
        {
            cout<<"Result(a/x)="<<a/x<<endl;
        }
        else
        {
            throw x;
        }
    }
    catch(int ex)
    {
        cout<<"Exception caught:Divide By Zero \n";
    }
}

```

THROWING MECHANISM

When an exception is detected, it can be thrown by using throw statement in any one of the following forms

- throw(exception);
- throw exception;
- throw;

CATCHING MECHANISM:

Catch block is as below

```
Catch(data type arg)
{
//statements for handling
//exceptions
}
```

Multiple catch statements:

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

- When an exception is thrown, the exception handler are searched in order fore an appropriate match.
- It is possible that arguments of several catch statements match the type of an exception. In such cases the first handler that matches the exception type is executed

Write a Program to catch multiple catch statements

```
#include<iostream.h>
void test(int x)
{
    try
    {
        if(x==1) throw x;
        else
        if(x==0) throw 'x';
        else
        if(x== -1) throw 1.0;
        cout<<"end of try block"<<endl;
    }
}
```

```

        catch(char c)
        {
            cout<<"caught a character"<<endl;
        }
        catch(int m)
        {

            cout<<"caught an integer"<<endl;

        }

    catch(double d)
    {
        cout<<"caught a double"<<endl;
    }
}
int main()
{
    test(1);
    test(0);
    test(-1);
    test(2);
    return 0;
}

```

Output:

```

caught an integer
caught a character
caught a double
end of try block

```

Catch All Exceptions:

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

```

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

```

Write a Program to catch all exceptions

```

#include<iostream.h>
void test(int x)
{
    try
    {
        if(x==0) throw x;
        if(x==0) throw 'x';
        if(x==1) throw 1.0;
    }
    catch(...)
    {

```

```

cout<<"caught exception"<<endl;
}
}
int main()
{
test(-1);
test(0);
test(1);

return 0;
}

```

Re-throwing an Exception:

It is possible to pass exception caught by a catch block again to another exception handler. This is known as Re-throwing.

```

#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
    }
}
int main()
{
    cout<< "Main start ... "<<endl;
    try
    {
        MyHandler();
    }
    catch(const char*)
    {
        cout <<"Caught exception inside Main\n";
    }
    cout << "Main end";
    return 0;
}

```

Specifying Exceptions:

Specification of exception restrict functions to throw some specified exceptions only with the use of throw(exception list) in the the header of the function.

General form

Type function_name(argument list) throw(exceptions -list)

```
{
Statements
try
{
    statements
}

}
```

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

void test(int x)
throw(int,float,char)
{
    switch(x)
    {
        case 1:throw x;
                break;
        case 2:throw 'x';
                break;
        case 3:throw double(x);
                break;
        case 4:throw float(x);
                break;
    }
}
```

```
int main()
{
    try
    {
        test(4); //test(4) leads to abnormal termination
    }
    catch(int i)
    {
        cout << "Caught int type exception\n";
    }
    catch(float f)
    {
        cout << "Caught float type exception\n";
    }
    catch(char c)
    {
        cout << "Caught char type exception\n";
    }
}
```

```
    }  
    catch(double i)  
    {  
        cout << "Caught Double type exception\n";  
    }  
  
    return 0;  
}
```