<u>Exceptions</u>

Exceptions are run time anomalies or unusual condition that a program may encounter during execution.

Examples:

- Division by zero
- Access to an array outside of its bounds
- Running out of memory
- Running out of disk space

<u>Principles of Exception Handling:</u> Similar to errors, exceptions are also of two types. They are as follows:

- **Synchronous exceptions**: The exceptions which occur during the program execution due to some fault in the input data.

  For example: Errors such as out of range, overflow, division by zero
- **Asynchronous exceptions**: The exceptions caused by events or faults unrelated (external) to the program and beyond the control of the program.

  For Example: Key board failures, hardware disk failures

The exception handling mechanism of C++ is designed to handle only synchronous exceptions within a program. The goal of exception handling is to create a routine that detects and sends an exceptional condition in order to execute suitable actions. The routine needs to carry out the following responsibilities:

1. Detect the problem (Hit the exception)
2. Inform that an error has been detected (Throw the exception)
3. Receive error information (Catch the exception)
4. Take corrective action (Handle the exception)

An exception is an object. It is sent from the part of the program where an error occurs to the part of the program that is going to control the error

The Keywords try, throw, and catch

Exception handling mechanism basically builds upon three keywords:

- try
- catch
- throw

The keyword **try** is used to preface a block of statements which may generate exceptions.

Syntax of try statement:

```
try
{
        statement 1;
        statement 2;
}
```

When an exception is detected, it is thrown using a **throw** statement in the try block.

Syntax of throw statement

- throw (excep);
- throw excep;
- throw; // re-throwing of an exception

A **catch** block defined by the keyword ‘catch’ catches the exception and handles it appropriately. The catch block that catches an exception must immediately follow the try block that throws the exception

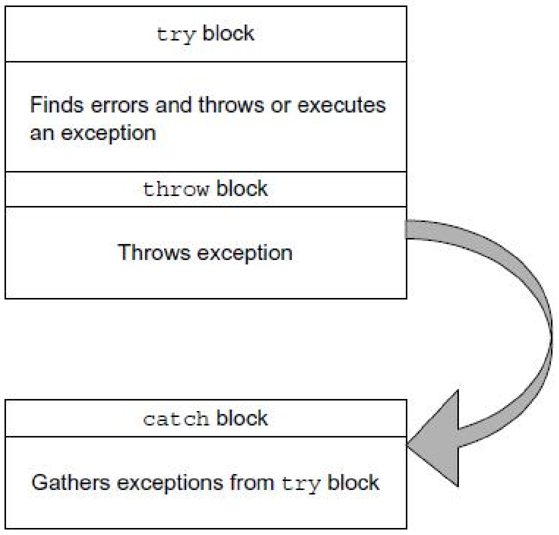
Syntax of catch statement:

```
try
{
        Statement 1;
        Statement 2;
}
catch ( argument)
{
        statement 3; // Action to be taken
}
```

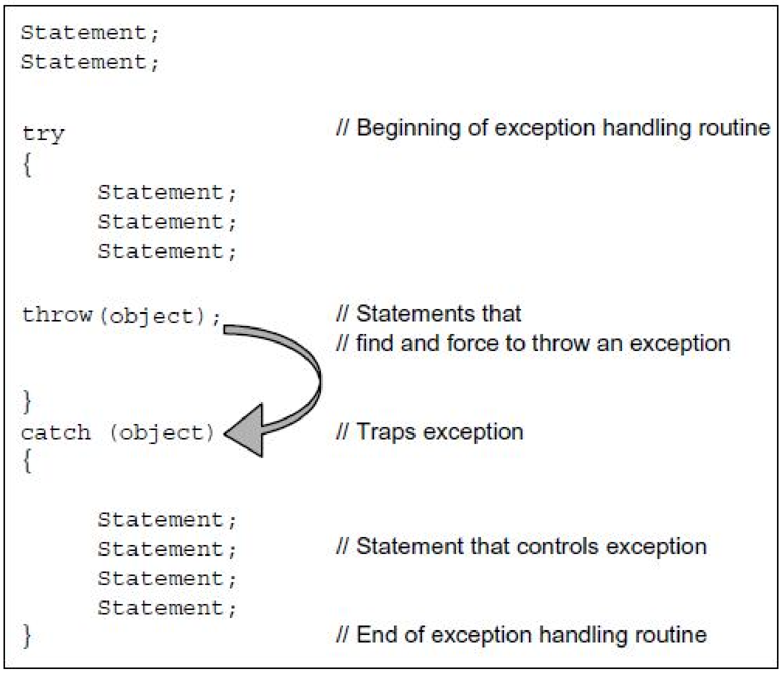
When an exception is found, the catch block is executed. The catch statement contains an argument of exception type, and it is optional. When an argument is declared, the argument can be used in the catch block. After the execution of the catch block, the statements inside the blocks are executed. In case no exception is caught, the catch block is ignored, and if a mismatch is found, the program is terminated.

Guidelines for Exception Handling

The C++ exception-handling mechanism provides three keywords; they are try, throw, and catch. The keyword try is used at the starting of the exception. The throw block is present inside the try block. Immediately after the try block, the catch block is present. Figure shows the try, catch, and throw statements.


try block
Finds errors and throws or executes an exception
throw block
Throws exception
catch block
Gathers exceptions from try block

As soon as an exception is found, the throw statement inside the try block throws an exception (a message for the catch block that an error has occurred in the try block statements). Only errors occurring inside the try block are used to throw exceptions. The catch block receives the exception that is sent by the throw block. The general form of the statement is as per the following figure.



```
Statement;
Statement;

try                          // Beginning of exception handling routine
{
      Statement;
      Statement;
      Statement;

throw(object);               // Statements that
                             // find and force to throw an exception

}
catch (object)               // Traps exception
{

      Statement;
      Statement;             // Statement that controls exception
      Statement;
      Statement;
}                            // End of exception handling routine
```

When the try block passes an exception using the throw statement, the control of the program passes to the catch block. The data type used by throw and catch statements should be same; otherwise, the program is aborted using the abort() function, which is executed implicitly by the compiler. When no error is found and no exception is thrown, in such a situation, the catch block is disregarded, and the statement after the catch block is executed.

/* Write a program to illustrate division by zero exception. */

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

int main()
{
        int a, b;
        cout<<"Enter the values of a and b"<<endl;
        cin>>a>>b;
        try{
                if(b!=0)
                        cout<<a/b;
                else
                        throw b;
        }
        catch(int i)
        {
          cout<<"Division by zero:  "<<i<<endl;
        }
        return 0;
}
```

Output:

```
Enter the values of a and b
2
0

Division by zero: 0
```

/* Write a program to illustrate array index out of bounds exception. */

```
#include <iostream>
using namespace std;
int main() {
        int a[5]={1,2,3,4,5},i;
        try{
                i=0;
                while(1){
                        if(i!=5)
                        {
                                cout<<a[i]<<endl;
                                i++;
                        }
                        else
                                throw i;
                }
        }
        catch(int i)
        {
                cout<<"Array Index out of Bounds Exception: "<<i<<endl;
        }
        return 0;
}
```

Output:

```
1
2
3
4
5
Array Index out of Bounds Exception: 5
```

/* Write a C++ program to define function that generates exception. */

```
#include<iostream>
using namespace std;
void sqr()
{
        int s;
        cout<<"\n Enter a number:";
        cin>>s;
        if (s>0)
        {
                        cout<<"Square="<<s*s;
        }
        else
        {
                        throw (s);
        }
}
int main()
{
        try
        {
                sqr();
        }
        catch (int j)
        {
                cout<<"\n Caught the exception \n";
        }
        return 0;

}
```

Ouput:

```
Enter a number:10
Square=100
Enter a number:-1
Caught the exception
```

Multiple catch Statements

It is also possible that a program segment has more than one condition to throw an exception. In such cases, we can associate more than one catch statement with a try (similar to switch statement). The format of multiple catch statements is as follows:

```
 try
{
        // try block
}
catch (type1 arg)
{
        // catch section1
}
catch (type2 arg)
{
        // catch section2
}
.......
.......
catch (typen arg)
{
        // catch section-n
}
```

When an exception is thrown, the exception handlers are searched in the order for an appropriate mach. The first handler that yields a match is executed. After executing the handler, the control goes to the first statement after the last catch block for that try. When no match is found the program will terminate.

It is possible that arguments of several catch statements match the type of exception. In such cases, the first handler that matches the exception type is executed.

/*Write a C++ program to throw multiple exceptions and define multiple catch statement. */

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

void num (int k)
{
        try
        {
                if (k==0) throw k;
                else
                        if (k>0) throw 'P';
                else
                        if (k<0) throw 1.0;
                cout<<"*** end of try block ***\n";
        }
        catch(char g)
        {
                cout<<"Caught a positive value \n";
        }
        catch (int j)
        {
                cout<<"caught an null value \n";
        }
        catch (double f)
        {
                cout<<"Caught a Negative value \n";
        }
        cout<<"*** end of try catch ***\n \n";
}
int main()
{
        cout<<"Demo of Multiple catches"<<endl;
        num(0);
        num(5);
        num(-1);
        return 0;
}
```

Output:

```
        Demo of Multiple catches
        caught an null value
        *** end of try catch ***
         Caught a positive value
        *** end of try catch ***
         Caught a Negative value
        *** end of try catch ***
         Caught a positive value
        *** end of try catch ***
```

Catching Multiple Exceptions

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

Syntax:

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

/* **Write a C++ program to** `catch` **multiple exceptions.*/**

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

void num (int k)
{
        try
        {
                if (k==0) throw k;
                else
                        if (k>0) throw 'P';
                else
                        if (k<0) throw 1.0;
        }
        catch(...)
        {
                cout<<"Caught an Exception"<<endl;
        }
}
int main()
{
        cout<<"Demo of Multiple catches"<<endl;
        num(0);
        num(5);
        num(-1);
        return 0;

}
```

Output:

```
Demo of Multiple catches
Caught an Exception
Caught an Exception
Caught an Exception
```

Specifying Exceptions

It is possible to restrict a function to throw only certain specified exceptions. This is achieved by adding a throw list clause to the function definition. The general form of using an exception specification is:

```
return_type fucntion_name (parameter list) throw (data type list)
{
        // function body
}
```

The data type list indicates the type of exception that is permitted to be thrown. If we want to deny a function from throwing any exception, declaring the data type list void as per the following statement can do this.

```
throw(); // void or vacant list
```

/* Write a C++ program to restrict a function to `throw` only specified type of exceptions. */

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

void check (int k) throw (int)
{
        if (k==1) throw 'k';
        else
                if (k==2) throw k;
        else
                if (k==-2) throw 1.0;
}
int main()
{
        try {
                        check(1);
                        check(-2);
                        check(3);
        }
        catch (char g)
        {
                cout<<"Caught a character exception \n";
        }
        catch (int j)
        {
                cout<<"Caught a character exception \n";
        }
        catch (double s)
        {
                cout<<"Caught a double exception \n";
        }
        cout<<"\n End of main()";
        return 0;
}
```