EXCEPTION HANDLING

PRINCIPLES OF EXCEPTION HANDLING

Like errors, exceptions are also of two types. They are as follows:

(a) Synchronous exceptions
(b) Asynchronous exceptions

- C++ has a well-organized object-oriented method to control run-time errors that occur in the program.
- The goal of exception handling is to create a routine that detects and sends an exceptional condition in order to execute suitable action.
- The routine needs to carry following responsibilities:
  1. Detect the problem
  2. Warn that an error has come
  3. Accept the error message
  4. Perform accurate actions without troubling the user
- An exception is an object.
- It is sent from the part of the program where an error occurs to that part of the program which is going to control the error.
- Exception provides an explicit pathway that contains errors to the code that controls the errors.

5.6.1 THE KEYWORDS try, throw AND catch

- Exception handling technique passes control of program from a location of exception in a program to an exception handler routine linked with the **try block**.
- An exception handler routine can only be called by throw statement.

**1. The Keyword try**

- The try keyword is followed by a series of statements enclosed in curly braces.

**Syntax of try statement**

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

**2. The Keyword throw**

- The function of throw statement is to send the exception found.
- The declaration of throw statement is as given below:

```cpp
Syntax of throw statement
throw (excep);
throw excep;
throw // re-throwing of an exception
```
• The argument excep is allowed in any type and it may be a constant.
• The catch block associated with try block catches the exception thrown.
• The control is transferred from try block to catch block.
• The throw statement can be placed in function or in nested loop but it should be in try block.
• After throwing exception, control passes to the catch statement.

5.6.2 EXCEPTION HANDLING MECHANISM
• C++ exception handling mechanism provides three keywords; try, throw, and catch.
• The keyword try is used at the starting of exception.
• The throw block is present inside the try block.
• Immediately after try block, catch() block is present.
• The throw statement inside the try block throws an exception (a message) for catch() block that an error has occurred in the try block statement.
• Only errors occurring inside the try block are used to throw exception.
• The catch block receives the exception send by the throw block.
• When try block passes an exception using throw statement, the control of the program passes to the catch block.
• The data types used by throw and catch statements must be the same otherwise program gets aborted using abort() function, that is executed implicitly by the compiler.
• When no error is found and no exception is thrown, in such a situation catch block is disregarded and the statement after the catch block is executed.

5.6.3 MULTIPLE CATCH STATEMENTS
• We can also define multiple catch blocks, in try blocks.
• Such program also contains multiple throw statements based on certain conditions.
• The format of multiple catch statement is given below:

```cpp
try
{
    // try section
}
catch (object1)
{
    // catch section1
}
catch (object2)
    // catch section2
}
.......
As soon as an exception is thrown, the compiler searches for appropriate by matching catch ( ) block.

- The matching catch ( ) block is executed and control passes to the successive statement after the last catch ( ) block.
- In case no match is found, the program terminates.
- In multiple catch ( ) statement, if objects of many catch statements are similar to type of an exception, in such a situation the first catch ( ) block that matches is executed.

### 5.6.4 Catching Multiple Exceptions

- It is also possible to define single or default catch( ) block from one or more exceptions of different types.
- In such a situation, a single catch block is used for catch exceptions thrown by multiple throw statements.

```cpp
    catch( )
    {
        // statements for handling
        // all exceptions
    }
```

**Example program to catch multiple exceptions**

```cpp
# include <iostream.h>
void num (int k)
{
    try
    {
        if (k==0) throw k;
        else
            if (k>0) throw 'P';
            else
                if (k<0) throw .0;
                cout <<"*** try block ***\n";
    }
    catch (...)
    {
        cout <<"\n Caught an exception\n";
    }
    int main( )
```
{  
    num(0);
    num(5);
    num(-1);
    return 0;
}

OUTPUT
Caught an exception
Caught an exception
Caught an exception

5.6.5 RETHROWING EXCEPTION

- It is also possible to pass again the exception received to another exception handler i.e., an exception is thrown from catch( ) block and this is known as rethrowing of exception.
- The following statement accomplishes this:
  
  throw;

- The above throw statement is used without any argument. This statement throws the exception caught to the next try catch statement.

Example program to rethrow an exception

```cpp
#include <iostream.h>
void sub( int j, int k)
{
    cout <<"inside function sub ( )\n";
    try
    {
        if (j==0)
            throw j;
        else

            cout <<"Subtraction = "<<j-k <<"\n";
    }
    catch (int)
    {
        cout <<"Caught null value \n";
        throw;
    }
    cout <<"** End of sub ( ) ***\n";
}
int main( )
{
    cout <<"\ninside function main ( )\n";
    try
    {
```
sub (8,5);
sub (0,8);
}
catch (int)
{
    cout <<"caught null inside main ( ) \n";
}
cout <<"end of function main ( ) \n";
return 0;
}

5.6.6 SPECIFYING EXCEPTION

- The specified exceptions are used when we want to bind the function to throw only specified exception.
- Using a throw list condition can do this.
- The format for such exception is given below.

```
Specifying Exceptions
data-type function_name (parameter list) throw (data type list)
{
    Statement 1;
    Statement 2; Function definition
    Statement 3;
}
```
- The data type list indicates the type of exceptions to be thrown.
- If we want to deny a function from throwing exception, we can do this by declaring data type list void as per the following statement.

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

Example program to restrict a function to throw only specified type of exceptions.

```
#include <iostream>
void check (int k) throw (int,double)
{
    if (k==1) throw 'k';
    else if (k==2) throw k;
    else if (k==-2) throw 1.0;
    cout <<"\n End of function check ( )";
}
void main ( )
{
    try {
        cout <<"k==1\n";
```
check(1);

    cout <<"k==2\n";
    check(2);
    cout <<"k==-2\n";
    check(-2);
    cout <<"k==3\n";
    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 <<"\nEnd of main ( ) ";