Exception Handling

Exceptions are the conditions that arises infrequently and unexpectedly and which are generally different from common programming errors such as syntax errors. These exceptions may lead to crash of the program and may cause abnormal exit from the program.

In a program, exception may be raised due to several reasons such as low disk space, low free main memory, divide by zero condition, pop from an empty stack, push to a full stack, memory read-write error, etc.

Exception handling :- is a mechanism that separates the detection and handling of circumstantial exceptional flow from normal flow.

C++ provides a mechanism to handle such exceptions which are generated due to some discrepencies in the program. This mechanism is called as try-throw-catch mechanism. In this mechanism, the section of the code which may generate an exception, is kept inside the 'try' section. If an exception is caught, then using the keyword 'throw', the control is pushed to a section called as 'catch' where appropriate action is ~~acte~~ taken to handle the exception. For example, consider the situation of 'divide by zero' exception. If appropriate measure not taken to handle this exception, then program may crash.

```
int main()
{ int a=10;
  int b=10;
  int c = a-b;
  int d;
    d = a/c;
    cout << d;
}
```

Above program will create an exception, 'divide by zero'. Hence, the program will crash and will be abnormally aborted. So, let we apply here the 'try-throw-catch' mechanism. So, now it will look as follows

```
int main()
  { int a= 10;
    int b = 10;
    int c = a-b;
    int d ;
  try
    { if (c != 0)
        d = a/c;
      else
        throw c;
    }
  Catch (int m)
    {
      cout << " exception is handled";
    }
}
```

now, the program will not undergo the abnormal abortion. Here, in the 'try' section, the exception is caught and using the statement 'throw c;', the control is being switched to the 'catch' section & exception is handled.

Here, in the statement 'throw c;', c is an integer.
Hence, in the 'catch' section, this c is being received by 'm' which is also an integer.

If ~~in place of~~ 'm' is of float-type then due to mismatch, the control will not be able to come under the 'catch' section and program will crash. So, we should be careful while throwing an exception that variable data-type must be matched.

multiple catch statement

- within a particular 'try' section, exceptions may be raised due to several reasons and hence, for each reason, we may need a separate catch section. This situation leads to the need of multiple catch statement.

For example:

```
int main( )
{
    int a = 10;
    float b = 20.5;
    char c = 'D';

    try
    {   if (condition 1)
            throw a;
        else if (Condition 2)
            throw b;
        else
            throw c;
    }
    catch ( int m)
    { cout << "Int exception handled"; }
    catch ( float n)
    { cout << "float exception handled"; }
```

```
catch ( char P )
    { cout << "Char exception handled";}
}
```

Here, depending upon the 'throw' statement, a relevant 'catch' section will be invoked. Here, implicit type conversion is not no done.

~~Even after taking multiple catch statement, it is possible that for~~
Sometime, it is also possible that ~~ex~~ all type of 'throw' statements can be handled by a single 'catch' section. So, in this case, multiple catch statement is not required. For such case, catch(...) is used.

```
int main()
    {   int a=10;
        float b=20.5;
        char c = 'D';
        try
            {   if (condition1)
                    {
                        throw a;
                    }
                else if ( condition2 )
                    { throw b;
                    }
                else
                    { throw c;
                    }
            }
    Catch(...)
        {   cout << "all exceptions are handled here";
        }
    }
```

Rethrow of exceptions

When a catch statement not only handles a particular exception but also throws the same exception to another catch statement which is part of next enclosing try-catch sequence.

Most of the times, exceptions are ~~caught~~ not caught in 'main' function but in deep nested function calls.

e.g.

```
main()
{ fun();
}

fun()
{ gun();
}

gun()
{ mun();
}
```

So, it is possible that exception is caught within the mun() function. So, in order to propogate the exception to the 'main' function, we need to 'rethrow' the exception.

```
main()
{
    try
    { fun();
    }
    catch (int x)
    { cout << "exception caught in main";
    }
}

void fun()
{
    try { gun();
    }
    catch (int y)
    { cout << "exception caught in fun";
      throw;
    }
}
```

```
void gun()
{
    try
        {
            mun();
        }
    catch(int z)
        {
            cout << "exception caught in gun";
            throw;
        }
}

void mun()
{
    int m=5;
    throw m;
}
```

In the above program, 1st fun() will be called. ~~then gun() and at~~ Inside the fun(), gun() will be called. Inside the gun(), mun() will be called. Let, within the mun(), exception is caught and being thrown by the statement 'throw m'. This 'throw' will be received by the 'catch' statement within gun() function. This 'catch' statement 1st prints "exception caught in gun" and then rethrows the exception which will be received by 'catch' statement inside the fun(). Further, this 'catch' statement prints "exception caught in fun" and then, rethrows the exception to the 'catch' statement within the 'main' function. Finally, this catch statement prints "exception caught in main" and not rethrows further. In this way, an exception raised in the mun() function propagates to the 'main' function.