EXCEPTION HANDLING

Exception Handling

- ***Exception:*** Errors occurring at run time.
  - such as running out of memory, not being able to open a file, trying to initialize an object to an impossible value.
- ***Exception handler:*** Code that is designed to deal with the exception
  - CATCH BLOCK
- ***Throwing an exception:*** Process of generating & passing the exception
  - throw *exception*;

Syntax:

```
try
{
    //one or more statements
    //atleast one of which should be capable of throwing an
    exception
}
catch(parameter)
{
            //one or more statements capable of handling
          //exceptions
}
```

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

Uses three words:

try

catch

throw

- Keywords
  - throw, try and catch
- **try block:** Identifies start of an exception-handling block of code

Syntax:

```
try
{
        //code that might cause an exception
}
```

Exception Handler

- **Catch block:** each try block must be followed by one or more catch blocks (exception handlers)
  - Receives a thrown exception and processes it

Syntax:

```
catch( exception  x )
{
        ....
}
```

Throwing an exception

- **throw:**
  - *transfers program control to the exception handler*

- *Once an exception has been thrown, control passes to the **catch** expression and the **try** block is terminated. **That is, catch is not called, program execution is transferred to it.***

No catch (exception handler)

- *If you throw an exception for which there is no applicable* ***catch*** *statement*
  - *an abnormal program termination may occur.*
- *Throwing an unhandled exception causes the standard library function* ***terminate( )*** *to be invoked. By default,* ***terminate( )*** *calls* ***abort( )*** *to stop the program*

Example-1

```
3  int main()
4  {
5      int A[] = { 10, 0, 2, 5 }, N = 20, Res;
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 }
```

Example-1

```
3  int main()
4  {
5      int A[] = { 10, 0, 2, 5 }, N = 20, Res, size = 4;
6      try
7      {
8          for(int i = 0; i < size; i ++ )
9          {
10             if( A[i] == 0 )
11                 throw "Divide by Zero..";
12             Res = N / A[i];
13             cout<<Res;
14         }
15     }
16     catch( const char *msg )
17     { cout<<"\n Exception caught.."<<msg; }
18
19     cout<<"\n End of program..";
20 }
```

```
2
Exception caught..Divide by Zero..
End of program..
```

Example-2:

```
24  int main()
25  {
26      cout<<"Start\n";
27      Xhandler(0);
28      Xhandler(1);
29      Xhandler(2);
30      cout<<"\nEnd";
31  }
```

```
3   void Xhandler( int test )
4   {
5       try
6       {
7           if( test == 0 ) throw "Value is zero";
8           if( test == 1 ) throw 1;
9           if( test == 2 ) throw 'T';
10      }
11      catch( int i )
12      {
13          cout <<"\nCaught Exception #: " << i;
14      }
15      catch( char c )
16      {
17          cout <<"\nCaught Exception #: " << c;
18      }
19      catch(const char *str)
20      {
21          cout <<"\nCaught a string: " << str;
22      }
23  }
```

```
Start


Caught a string: Value is zero
Caught Exception #: 1
Caught Exception #: T
End
```

Specifying the Exception class

```
class   Exception_Class
{    //note: empty class body     };
```

- This is used to connect a throw statement with a catch block.

Example - 3

```
3   class Zero{};
4   class Negative{};
5
6   void exception_test(int Arr[], int size )
7   {
8       try
9       {
10          for( int i = 0; i < size; i++ )
11          {
12              if( Arr[i] == 0 )
13                  throw Zero();
14              if( Arr[i] < 0 )
15                  throw Negative();
16              cout<<Arr[i]<<"\n";
17          }
18      }
19      catch( Zero )
20      { cout<<"\nException caught...Zero \n"; }
22      catch( Negative )
23      { cout<<"\nException caught...Negative \n"; }
24  }
```

```
25  int main()
26  {
27      int A[] = { 10, 20, -10, 40 };
28      int size_A = 4;
29      int B[] = { 11, 0, 22 };
30      int size_B = 3;
31      exception_test( A, size_A );
32      exception_test( B, size_B );
33      cout<<"End of program..";
34  }
```

```
10
20

Exception caught...Negative
11

Exception caught...Zero
End of program..
```

Example - 4.a

```
3   const int MAX = 3;
4
5   class Stack
6   {
7       int st[MAX], top;
8   public:
9       class FULL {};  //exception classS
10      class EMPTY{}; //exception class
11
12      Stack()
13      { top = -1; }
14      void push(int var)
15      {
16          if(top >= MAX-1)
17              throw FULL();
18          st[++top] = var;
19      }
```

```
20      int pop()
21      {
22          if(top < 0)
23              throw EMPTY();
24          return st[top--];
25      }
26  };
```

```
Exception: Stack Full
End of program..
```

```
27  int main()
28  {
29      Stack s1;
30      try
31      {
32          s1.push(11);
33          s1.push(22);
34          s1.push(33);
35          s1.push(44);
36          cout<< s1.pop() << endl;
37          cout<< s1.pop() << endl;
38      }
39      catch( Stack::FULL )  //exception handler
40      { cout << "Exception: Stack Full" << endl; }
41
42      catch( Stack::EMPTY )  //exception handler
43      { cout << "Exception: Stack empty" << endl; }
44
45      cout <<"End of program.."<< endl;
46  }
```

Example - 4.b

```
 3  const int MAX = 3;
 4
 5  class Stack
 6  {
 7      int st[MAX], top;
 8  public:
 9      class FULL {};  //exception classS
10      class EMPTY{}; //exception class
11
12      Stack()
13      { top = -1; }
14      void push(int var)
15      {
16          if(top >= MAX-1)
17              throw FULL();
18          st[++top] = var;
19      }
```

```
20      int pop()
21      {
22          if(top < 0)
23              throw EMPTY();
24          return st[top--];
25      }
26  };
```

```
22
11
Exception: Stack empty
End of program..
```

```
27  int main()
28  {
29      Stack s1;
30      try
31      {
32          s1.push(11);
33          s1.push(22);
34
35          cout<< s1.pop() << endl;
36          cout<< s1.pop() << endl;
37          cout<< s1.pop() << endl;
38      }
39      catch( Stack::FULL )  //exception handler
40      { cout << "Exception: Stack Full" << endl; }
41
42      catch( Stack::EMPTY )  //exception handler
43      { cout << "Exception: Stack empty" << endl; }
44
45      cout <<"End of program.."<< endl;
46  }
```

Catching All Exceptions

```
catch(...)
{
        // process all exceptions
}
```

Example:5

```
19  int main()
20  {
21      cout << "Start\n";
22      Xhandler(0);
23      Xhandler(1);
24      Xhandler(2);
25      cout<<"End";
26  }
```

```
3   void Xhandler(int test)
4   {
5       try
6       {
7           if(test==0)
8               throw test; // throw int
9           if(test==1)
10              throw 'a'; // throw char
11          if(test==2)
12              throw 123.23; // throw double
13      }
14      catch(...)
15      { // catch all exceptions
16      cout << "Caught One!\n";
17      }
18  }
```

OUTPUT

```
Start
Caught One!
Caught One!
Caught One!
End
```

Exceptions with Arguments

Example-6

```
4   class Zero //Exception class
5   {
6       string msg;
7   public:
8       Zero() { }
9
10      Zero(string m)
11      {
12          msg = m;
13      }
14
15      void show_msg()
16      {
17          cout<<msg;
18      }
19  };
```

```
21  int main()
22  {
23      int A[] = { 10, 0, 2 }, N = 20, Res, size = 3;
24      try
25      {
26          for(int i = 0; i < size; i ++ )
27          {
28              if( A[i] == 0 )
29                  throw Zero("Divide by Zero..");
30              Res = N / A[i];
31              cout<<Res;
32          }
33      }
34      catch( Zero Z )
35      {
36       cout<<"\n Exception caught:";
37       Z.show_msg();
38      }

39      cout<<"\n End of program..";
40  }
```

```
2
Exception caught:Divide by Zero..
End of program..
```

Re-throwing exceptions

- If a need arises to rethrow an expression from within an exception handler,
  - This causes the current exception to be passed on to an outer **try/catch** sequence.
- An exception can only be re-thrown from within a **catch** block (or from any function called from within that block).
- An exception that is rethrown, will not be recaught by the same **catch** statement. It will propagate outward to the next **catch** statement.

Example-7

```
3  class My_Exception{};
4
5  void test()
6  {
7      try
8      {
9          throw My_Exception();
10     }
11
12     catch( My_Exception e )
13     {
14         cout<<"Caught My_Exception inside test()\n";
15         throw e; // rethrow out of function
16     }
17 }
```

```
18 int main()
19 {
20     cout << "Start\n";
21
22     try
23     { test(); }
24
25     catch( My_Exception e )
26     { cout<<"Caught in main\n"; }
27
28     cout<<"End";
29 }
```

```
Start
Caught My_Exception inside test()
Caught in main
End
```

terminate( ) and unexpected( )

- **terminate( )** and **unexpected( )** are called when something goes wrong during the exception handling process.
- These functions are supplied by the Standard C++ library.
- Prototypes:

  void terminate( );

  void unexpected( );
- These functions require the header **<exception>**.
- The unexpected( ) function is called when a function attempts to throw an exception that is not allowed by its throw list. By default, unexpected( ) calls terminate( ).

```
3  class EXCEP1{};
4  class EXCEP2{};
5
6  int main()
7  {
8      try
9      {
10         // Some statements..
11         //if( some_condition )
12             throw EXCEP2();
13     }
14
15     catch( EXCEP1 )
16     {
17         cout<<"Caught Excep1..\n";
18     }
19     cout<<"End";
20 }
```

```
terminate called after throwing an instance of 'EXCEP2'


--------------------------------
```