Exception Handling C++

What are exceptions

- Exceptions are runtime anomalies that a program may detect
- e.g.
  - Division by 0
  - Access to an array outside it's bounds
  - Exhaustion of the free memory on heap

Exception Handling

- C++ provides built in features to raise and handle exceptions.
- These language features activates a runtime mechanism to communicate exceptions between two unrelated portions of C++ program.
- C++ exception handling is built upon three keywords: **try, catch,** and
- **throw.**
- try block is a block surrounded by braces in which exception may be thrown
- A catch block is the block immediately following a try block, in which exceptions are handled

Syntax -

```
try{
    // something unusual but still predictable
}
catch (out of memory) {
    // take some action
}
catch (File not found) {
    // take other action
}
```

While using Exception Handling …

- **Note**
  - When an exception is raised, program flow continues after catch block.
  - Control never comes back to the point from where exception is thrown
  - Memory leakage in context with exception when object is created on heap.
- Duplicate cleanup code that is common to both normal and exceptional path of control

```
// First Demo
int main() {
    cout << "Start\n"; try {
    // start a try block
    cout << "Inside try block\n";
    throw 100; // throw an error
    cout << "This will not execute";
    }
    catch (int i) { // catch an error
        cout << "Caught an exception -- value is: ";
        cout << i << "\n";
    }
    cout << "End";
    return 0;
}
}
```

Throwing an Exception from function

- If a function is throwing an exception which is not handled locally, it will be escalated to the calling function. If the calling function is not handling the exception, it will be escalated to the next outer scope, till the time it is not handled.

- If no handler is found then program is terminated, by invoking terminate() function.

- We can define our own terminate() function, and set it using the set_terminate()

```
void myTerminate() {
    // do whatever required
  }
 void main () { set_terminate(myTerminate);
      // regular execution
 }
```

set_terminate is defined in <exception>

Throwing an Exception from function

```
void fun( int num )
{
    if( num <= 0 )
        throw num;
    else
        throw ‘p’;
    cout<<"num = "<<num<<endl;
}
```

```
int main ()
{
    try {
        fun( 0 );
    }
    catch(int excNum ) {
        cout<<"exception generated: "<<excNum<<endl;
    }
    catch(char ch) {
        cout<<"char exception generated: "<<ch;
    }
    return 0;
}
```

Multiple catch blocks

- Execution is similar to switch-case
- Once a matched catch block signature is found, other catch blocks are not executed
- Order of catch blocks is important. Specific first and general at last.
- Most general is Catch everything indicated by catch(…)

If Exceptions Are Ignored…

- If exceptions are ignored or not handled properly, the program is terminated.
- This will happen in cases where:
  - An exception is thrown out of a **try** block.
  - No appropriate catch block has been defined to handle an exception.
- In such a case, the standard termination function - **terminate()** is executed.
  - The function executes the **abort()** function.

Throwing Exceptions of Class Type

You can create your own classes to represent exception.

```
class MyException {
   char str_what[80];
   int what;
   public:
      int get_what() { return what;}
      char *get_str_what() { return str_what ;}
      MyException() {
         *str_what = 0;
         what = -999;
      }
      MyException(char *s, int e) {
         strcpy(str_what, s);
         what = e;
      }
   void print() { cout<<str_what<<"Error code: "<<what<<endl;
   }
};
```

```
int main() {
   int i;
   try {
      cout << "Enter a positive number: "; cin >> i;
      if(i<0)
         throw MyException("Not Positive", i);
      }
      catch (MyException e) {
      cout << e.get_what() << "\n";
      cout << e.get_str_what() << "\n";

      }
   return 0;
}
```

Re-throwing an Exception

You could re-throw an expression from a exception handler, using the keyword throw.

```
void fun()
{
    try {
        throw "hello"; // throw a char *
  }
  catch(const char *) {
    cout << "Caught char * inside fun\n";
    throw ; // rethrow char * out of function
  }
}
```

```
int main()
{
    try{
        fun();
    }
    catch(const char *) {
        cout << "Caught char * inside main\n";
    }
}
```