# CSCI251/CSCI851 Autumn-2020
# Advanced Programming (**LT10**)

Lecture Tutorial 10

# From the Lab: Something exceptional

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

class tornadoException
{
private:
        string message;
public:
        tornadoException();
        tornadoException(int m);
        string what();
};
```

C++11 → : `to_string`…

```
tornadoException::tornadoException()
{message = "Tornado: Take cover NOW!“;}

tornadoException::tornadoException(int m)
{message = "Tornado: " + to_string(m) + " kilometers away!";}

string tornadoException::what()
{return message;}
```

```
int main()
{
        int m;
        try{
                cout << "Enter how close the Tornado is in km : ";
                cin >> m;
                if ( m==0 )
                        throw tornadoException();
                else
                        throw tornadoException(m);

        }
        catch (tornadoException &e)
        {
                cout << e.what() << endl;
        }
}
```

- No attempt to be robust here…
- … and it really doesn't demonstrate detection and handling separation!

```
void generateTornado()
{
        int m;
        do
        {
                cout << "Enter how close the Tornado is in km : ";
                cin >> m;
                if (m < 0 ) {cout << "Positive only!" << endl;}
        } while ( m < 0 );
                if ( m==0 )
                        throw tornadoException();
                else
                        throw tornadoException(m);
}
```

```
int main()
{
        try{
                generateTornado();
        }
        catch (tornadoException &e){
                cout << e.what() << endl;
        }
        catch (...){
                cout << “What went wrong here?" << endl;
                throw;
        }
}
```

```
template <typename T>
void showData(T x, int number, char symbol)
{
  for(int i=0; i < number; ++i)
    cout << symbol;
  cout << x;
  for(int i=0; i < number; ++i)
    cout << symbol;
  cout << endl;
}
```

Symbolic.cpp

```
int main()
{
  char letter = 'P';
  int integer= 47;
  double money= 39.25;
  string name = "Bob";
  showData(letter,5,'+');
  showData(integer,3,'*');
  showData(money,3,'0');
  showData(name,4,'a');
}
```

# Building and using Libraries …

- Lecture set S5a.
- This can be covered fairly quickly, and the syntax isn't a big deal.
- 3.2 : The exact size doesn't really matter but you expect M3 to be smaller than the other 2, but not by much.

M1: 13,448 bytes

M2: 13,448 bytes

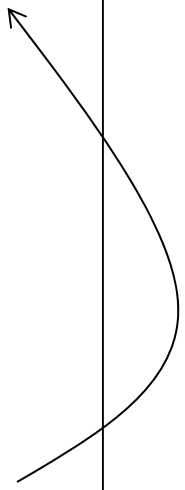
M3: 13,208 bytes

- 3.3: M1 and M2 can both run, M3 cannot.
  - M1 doesn't directly use a library and M2 is statically linked.
  - M3 won't be able to run if the library file libcode.so is removed.
- This is expected because M3 is dynamically linked so tries to load the library at run time.

- 3.4 will likely only work on capa.
  - You will have to use capa for A3 because we will be provided a library function that has to be used in a similar way.
- 3.4(c): The function `mash`, being called from the library, is a hash function.
- It takes a string input of arbitrary size and produces two digits (00-99).
  - This isn't quite the same as a two digit integer, or a integer of at most two digits, because you can have 03, rather than the single digit number 3.

- 3.4d:
  - The pigeonhole principle states that if you have N boxes and N+1 items, each of which must go into one of your boxes, there must be a box with at least 2 items in it.
  - For a hash function, arbitrary sizes input and only 100 outputs, we must have multiple inputs that hash to the same thing.
  - Now we are mostly interested in what happens when we restrict our attention to two digit inputs.
    - Without even looking at the specific output, we will in a moment, we can infer there must be a loop or cycle in the input-output chain.
    - We generate a input-output chain by starting with any value, say such as 83, mash it, then mash the result and continue doing so.
    - We must eventually get a value we have already seen in the chain because there are only a finite number of outputs so even if we were initially getting different values we will run out of different values.

```
$ ./Mash 83
83 : 61
61 : 53
53 : 69
69 : 74
74 : 06
06 : 88
88 : 30
30 : 95
95 : 37
37 : 90
90 : 35
35 : 49
49 : 03
03 : 55
55 : 20
20 : 72
72 : 39
39 : 27
27 : 45
45 : 02
02 : 49
```



Here go all the Input : Output pairs.

```
00 : 62
01 : 77
02 : 49
03 : 55
04 : 39
05 : 70
06 : 88
07 : 30
08 : 84
09 : 12
10 : 23
11 : 98
12 : 03
13 : 80
14 : 62
15 : 70
16 : 74
17 : 06
18 : 54
19 : 28
20 : 72
21 : 11
22 : 27
23 : 18
24 : 36
```

```
25 : 31
26 : 39
27 : 45
28 : 13
29 : 41
30 : 95
31 : 41
32 : 22
33 : 27
34 : 45
35 : 49
36 : 15
37 : 90
38 : 60
39 : 27
40 : 28
41 : 18
42 : 73
43 : 05
44 : 49
45 : 02
46 : 81
47 : 24
48 : 78
49 : 03
```

```
50 : 00
51 : 49
52 : 71
53 : 69
54 : 24
55 : 20
56 : 91
57 : 40
58 : 51
59 : 47
60 : 54
61 : 53
62 : 33
63 : 39
64 : 40
65 : 07
66 : 94
67 : 45
68 : 12
69 : 74
70 : 84
71 : 75
72 : 39
73 : 13
74 : 06
```

```
75 : 95
76 : 23
77 : 90
78 : 49
79 : 20
80 : 90
81 : 47
82 : 61
83 : 61
84 : 65
85 : 65
86 : 26
87 : 89
88 : 30
89 : 59
90 : 35
91 : 85
92 : 24
93 : 76
94 : 21
95 : 37
96 : 04
97 : 82
98 : 22
99 : 12
```