University of Hertfordshire

7COM1025
Programming for Software Engineers
Lecture 20

Dr. Renato C. de Amorim

THE ONE HUNDRED UNDER FIFTY

CATCHING BASE CLASS EXCEPTIONS

```
#include <iostream>
using namespace std;
class B{};
class D: public B{};
int main(){
    D derived;
    try{
        throw derived;
    }
    catch(B b){
        cout<<"Catch a base class"<<endl;
    }
    catch (D b){
        cout<<"this won't execute"<<endl;
    }
    return 0;
}
```

CATCHING ALL EXCEPTIONS

```
#include <iostream>
using namespace std;
void Xhandler(int test){
    try{
        if (test==0) throw test;
        if (test==1) throw 'a';
        if (test==2) throw 123.123;
    }
    catch(...){
        cout<<"Catch an exception"<<endl;
    }
}
int main(){
    cout<<"start"<<endl;
    Xhandler(0);
    Xhandler(1);
    Xhandler(2);
    cout<<"end"<<endl;
    return 0;
}
```

SPECIFYING EXCEPTIONS THROWN BY A FUNCTION

```
#include <iostream>
using namespace std;
void Xhandler(int test) throw(int, char, double) {
    if (test==0) throw test;
    if (test==1) throw 'a';
    if (test==2) throw 123.123;

}
int main(){
    cout<<"start"<<endl;
    try{
        Xhandler(0);
        Xhandler(1);
        Xhandler(2);
    }
    catch(int i){
        cout<<"Catch int exception"<<endl;
    }
    catch(char i){
        cout<<"Catch char exception"<<endl;
    }
    catch(double i){
        cout<<"Catch double exception"<<endl;
    }
    cout<<"end"<<endl;
    return 0;
}
```

PROBLEM 20.1

Create your own exception class. The purpose of this class is that you it should contain all relevant information about an exception and you can throw an object of this class when throwing an exception.

You class should have fields for:
- Description of error
- Error ID (an int).
- Line number of the error (you can get a line number with __LINE__, notice there are two underscore in each side)
- Function name (you can get a function name using __func__)

TEMPLATES

```
#include <iostream>
using namespace std;
template <class DataType> void swapargs(DataType &a, DataType &b){
    DataType temp;
    temp = a;
    a=b;
    b=temp;
}
int main()
{
    int i=10, j=20;
    float x=10.1F, y=12.2F;
    cout<<"original i,j: "<<i<<','<<j<<endl;
    cout<<"original x,y: "<<x<<','<<y<<endl;
    swapargs(i,j);
    swapargs(x,y);
    cout<<"new i,j: "<<i<<','<<j<<endl;
    cout<<"new x,y: "<<x<<','<<y<<endl;
    return 0;
}
```

TWO GENERIC TYPES

```
#include <iostream>
using namespace std;
template <class Type1, class Type2>
    void myfunc(Type1 x, Type2 y){
    cout<<x<<' '<<y<<endl;}
int main(){
    myfunc(1,1.23);
    myfunc("Hello", -23.4F);
    return 0;
}
```

SPECIALIZING A TEMPLATE FUNCTION

```
#include <iostream>
using namespace std;
template <class DataType>
void swapargs(DataType &a, DataType &b){
    DataType temp;
    cout<<"use template function"<<endl;
    temp = a;
    a=b;
    b=temp;
}
void swapargs(int &a, int &b){
    int temp;
    cout<<"use int function"<<endl;
    temp = a;
    a=b;
    b=temp;
}
```

```
int main()
{
    int i=10, j=20;
    float x=10.1F, y=12.2F;
    cout<<"original i,j: "<<i<<','<<j<<endl;
    cout<<"original x,y: "<<x<<','<<y<<endl;
    swapargs(i,j);
    cout<<"new i, j: "<<i<<','<<j<<endl;
    swapargs(x,y);
    cout<<"new x, y: "<<x<<','<<y<<endl;
    return 0;
}
```

PROBLEM 20.2

Write a function called add. This function should return the sum of two parameters.
If the parameters are string, add should return the concatenation of the two parameters.

GENERIC CLASSES

```
#include <iostream>
using namespace std;
template <class DataType> class MyClass{
    DataType x,y;
public:
    MyClass(DataType a, DataType b){
        x=a;
        y=b;
    }
    DataType div() {return x/y;}
};
int main()
{
    MyClass<double> d_ob(10.0, 3.0);
    cout<<"Double division: "<<d_ob.div()<<endl;
    MyClass<int> i_ob(10, 3);
    cout<<"Integer division: "<<i_ob.div()<<endl;
    return 0;
}
```

CLASS WITH 2 GENERIC DATA TYPES

```
#include <iostream>
using namespace std;
template <class T1, class T2> class MyClass{
    T1 i;
    T2 j;
public:
    MyClass(T1 a, T2 b){i=a; j=b;}
    void show(){cout<<i<<", "<<j<<endl;}
};
int main(){
    MyClass<int, double> ob1(10,0.23);
    MyClass<char, char *> ob2('A', "string");
    ob1.show();
    ob2.show();
    return 0;
}
```

PROBLEM 20.3

We need a class that expands the capabilities of a STL vector. Using composition, create a class containing a vector. Your class should have two extra methods:
mean()
-returns the mean
std()
-returns the sample standard deviation

Your class should work as a container for any data type. However, if the data type is string, mean() and std() should throw an exception.

EXPLICIT CLASS SPECIALIZATION

```
#include <iostream>
using namespace std;
template <class T> class MyClass{
    T x;
public:
    MyClass (T a){
        cout<<"Inside Generic class"<<endl;
        x=a;
    }
    T getx(){return x;}
};
template <> class MyClass<int>{
    int x;
public:
    MyClass(int a){
    cout<<"Inside int class"<<endl;
    x=a;
    }
    int getx(){return x;}
};
```

```
int main(){
    MyClass<double> d(10.1);
    cout<<"Double: "<<d.getx()<<endl;
    MyClass<int> i(5);
    cout<<"Int: "<<i.getx()<<endl;
    return 0;
}
```