Chap 15 – Exceptions, RTTI, Dynamic Casting

1.

a. Write a function called *GetNum()* that prompts the user to enter a number between 1 and 10. If the number entered is less than 1 it will throw an *underflow_error* exception. If the number entered is greater than 10 then it will throw an *overflow_error* exception. Both of these exceptions are in the exception header file ie. #include <exception>

To test that this works have your main() look like this:

```
while (1) {
        try {
                i = GetNum();
                cout << i << endl;
        }
        catch (underflow_error & e){
                cout << e.what() <<  endl;
        }
        catch (overflow_error & e) {
                cout << e.what() << endl;
        }
}
```

b. Modify this code slightly so that *main()* calls a function called *doit()* instead if *GetNum()*. The function *doit()* simply returns the value passed back from *GetNum()*. Will the code work pretty much the same?

2. Do #2 on pg 949
3. Do #3 on pg 949

4. Create a class **Gum** that has one function called *me()* that returns the string "class Gum". Then create three classes that derive from class **Gum** called **Bubble**, **Chewing**, and **Experimental** and implement the *me()* function for each one that returns the string "class Bubble", "class Chewing", and "class Experimental".

   Now create a class called **GumMachine** with a function called *dispense()* that will return a pointer to **Gum** but each call to dispense() will allocate a different type of Gum (that is **Bubble**, **Chewing**, and **Experimental**).

   Call *dispense()* 6 times and have it print out the string returned from me();

   Note: the me() method is NOT virtual in this example. If it were this would be easy to do. Instead you will need to use *dynamic_cast* to cast to the right type of pointer.

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

class Gum {
public:
        virtual void empty(){} {//dynamic_cast needs one virtual method
        string me() const {//your code here};
};

class Chewing : public Gum {
public:
        string me() const {//your code here }
};

class Bubble: public Gum {
public:
        string me() const {//your code here }
};

class Experimental : public Gum{
public:
        string me() const {//your code here }
};

class GumMachine {
private:
        //your code here
public:
        Gum* dispense() {// return either a Chewing, Bubble or Experimental object }
};

int main(){
        GumMachine gummachine;

        for (int i = 0; i < 6; i++) {
                Gum* g = gummachine.dispense();

                //your code here

                delete g;
        }
}
```

The output should look like the following:

```
class Chewing
class Bubble
class Experimental
class Chewing
class Bubble
class Experimental
```

5. Copy your code from exercise 4 (don't overwrite it), remove all the *me()* methods from the subclasses and rewrite the *me()* method in **Gum** so that it will work **exactly the same** as in exercise 4.

   Hint: There is a way to get the type name of a variable. But what variable to use? ;-)