Programming Project 2

Mighty Polymorphic Nemo

Fishies:

To get warmed up lets implement a Fish class with definition given below:

```
class Fish {
public:
      Fish(int capacity, std::string name);// Constructor
      Fish(const Fish &other);             // Copy Constructor
      ~Fish();                             // Destructor
      Fish &operator=(const Fish &other);  // Assignment Operator

      void remember(char c);   // Remember c
      void forget();           // Clears memory by filling in '.'
      void printMemory() const;// Prints memory

      std::string getName();

protected:
      const char* getMemory() const;// Returns memory
      int getAmount() const;        // Returns amount remembered
      int getCapacity() const;      // Returns memory capacity

private:
      // TODO: declare any private member variables/functions here

};
```

This models a creature that is intelligent enough to remember some capacity of characters at a time. But, a Fish might not use up its entire capacity, it might just remember some amount that is less than its capacity. For example a Fish might be able to remember 10 characters, but may only have encountered 2 so far.

Implement Fish's member functions:

- `Fish(int capacity, std::string name);`
  Dynamically allocate capacity-many characters to for the Fish's memory. Initialize its memory with dot('.')s. If capacity is not positive, then set it to 3 by default. Since, this is a brand new fish it uses none of its capacity. Also, save the Fish's name.

- `Fish(const Fish &other);`
  Copy constructor for a Fish, when declaring a new Fish we can make it an exact copy of another.

- `~Fish();`
  Clean up dynamically allocated memory.

- `Fish &operator=(const Fish &other);`
  Overloaded assignment operator so we can assign one declared fish to be an exact copy of another declared fish.

- `void remember(char c);`
  Implement remember. Store the character c into the Fish's memory. If you already have the max capacity characters memorized, then discard the oldest character in memory to open up a free slot. (This is an example of LRU (Least Recently Used) replacement.). Update the amount of characters remembered appropriately.

- `void forget();`
  Implement forget, clear memory by filling dot('.')s into memory. This implies that there are no characters remembered.

- `void printMemory() const;`
  Print to console what is in Fish's memory.

- `std::string getName();`
  Gets the Fish's name.

- `const char* getMemory() const;`
  Gets the Fish's memory.

- `int getAmount() const;`
  Gets the amount of characters remembered.

- `int getCapacity() const;`
  Gets the Fish's capacity.

Below is an example using this new Fish class:

```
int main() {
	Fish *nemo = new Fish(3, "nemo");
	nemo->remember('a');
	nemo->printMemory();// prints "a.. "
	nemo->remember('b');
	nemo->remember('c');
	nemo->printMemory();// prints "abc"
	nemo->remember('d');
	nemo->printMemory(); // prints "bcd"

	Fish *dory = new Fish(*nemo);
	dory->printMemory(); // prints "bcd"

	nemo->forget();
	nemo->printMemory(); // prints "..."

	*dory = *nemo;
	dory->printMemory(); // prints "..."
	return 0;
}
```

Bubbles:

Let's create another class based on our Fish class called Yellowtang. Yellowtang is just like any other fish except in the presence of bubbles Yellowtang gets excited and can't remember anything other than BUBBLES! Below is Yellowtang's class definition:

```
class Yellowtang :public Fish {
public:
      Yellowtang(int capacity, std::string name);     // Constructor
      Yellowtang(const Yellowtang &other);            // Copy Constructor
      Yellowtang &operator=(const Yellowtang &other); // Assignment Operator

      // Overridden functions
      virtual void remember(char c);
      virtual void printMemory() const;
      virtual void forget();

private:
      // TODO: declare any private member variables/functions here
};
```

Since, we're making a Base class out of Fish and we intend on overriding three of Fish's functions we must make them virtual, and we should also make the destructor virtual as well, because ALWAYS make base class destructor virtual! Implement Yellowtang's member functions:

- `Yellowtang(int capacity, std::string name);`
  Implement Yellowtang's constructor which must call its Fish constructor and then initializes any member variables.

- `Yellowtang(const Yellowtang &other);`
  Copy constructor for a Yellowtang, calls Fish's copy constructor to copy its Fish component and makes copies of any Yellowtang specific member variables.

- `Yellowtang &operator=(const Yellowtang &other);`
  Implement Yellowtang's overloaded assignment operator. Like all assignment operator overloads this should check to see if Yellowtang is being assigned to itself. Afterwards, like the copy constructor, should just call Fish's overloaded assignment operator to assign its Fish. Then copy any Yellowtang specific member variables, then as always return itself.

- `virtual void remember(char c);`
  Yellowtang remembers things exactly the way any other Fish does, but in addition keeps track if it has seen any bubbles, represented by the character 'o'.

- `virtual void printMemory() const;`
  Yellowtang print's its memory just like any other Fish, except if Yellowtang remembers any bubbles, in that case Yellowtang just prints "BUBBLES!"

- `virtual void forget();`
  Yellowtang forgets just like Fish, but in addition to clearing its memory it also forgets that it has seen bubbles.

Below is example usage of Yellowtang:

```
int main() {
	Yellowtang *bubbles = new Yellowtang(3, "bubbles");
	bubbles->remember('t');
	bubbles->printMemory(); // prints "t.."
	bubbles->remember('o');
	bubbles->printMemory(); // prints "BUBBLES!"
	bubbles->remember('a');
	bubbles->remember('b');
	bubbles->printMemory(); // prints "BUBBLES!"
	bubbles->remember('c');
	bubbles->printMemory(); // prints "abc"
	return 0;
}
```

Obnoxious Memory:

Butterflyfish is another type of Fish, except has an extended memory in addition to Fish's memory. Not only does it store characters in its limited capacity memory, it also has memory that can expand once it reaches its capacity. Not only that, it remembers how many times it has seen any character. Similar to Fish's memory, the number of unique characters Butterflyfish as seen is not necessarily its capacity. Butterflyfish's definition is provided below:

```
class Butterflyfish :public Fish {
public:
    Butterflyfish(int capacity, std::string name);        // Constructor
    Butterflyfish(const Butterflyfish &other);            // Copy Constructor
    Butterflyfish &operator=(const Butterflyfish &other); // Assignment Operator
    virtual ~Butterflyfish();                              // Destructor

    // Overridden Functions
    virtual void remember(char c);
    virtual void printMemory() const;

private:
    // TODO: declare any private member variables/functions here

};
```

- `Butterflyfish(int capacity, std::string name);`
  Implement Butterflyfish's constructor which must call its Fish constructor and then initializes its own member variables. Butterflyfish's dynamically allocated extended memory starts of the same size as Fish's. Recall that Butterflyfish's extended memory keeps track of two things: any unique characters encountered and the number of times it has seen them. It starts off not having seen anything.

- `Butterflyfish(const Butterflyfish &other);`
  Copy constructor for a Butterflyfish, calls Fish's copy constructor to copy its Fish component and makes copies of any Butterflyfish specific member variables. Pay special mind to any dynamically allocated member variables.

- `Butterflyfish &operator=(const Butterflyfish &other);`
  Implement Butterflyfish overloaded assignment operator. Like all assignment operator overloads this should check to see if Butterflyfish is being assigned to itself. Afterwards, like the copy constructor, should just call Fish's overloaded assignment operator to assign its Fish. Then copy any Butterflyfish specific member variables, like the copy constructor pay special mind to any dynamically allocated member variables, then as always return itself.

- `virtual void remember(char c);`
  Butterflyfish remembers things exactly the way any other Fish does, but in addition keeps of unique characters and a count of each time it has seen those characters. If its extended memory fills up to its capacity, Butterflyfish expands its character and counter memory by doubling its capacity.

- `virtual void printMemory() const;`
  Butterflyfish print's its memory just like any other Fish, but in addition to that it says "I'm Obnoxious!", then prints out each of the unique characters it has seen along with the number of times it has seen them.

Example usage below:

```
int main() {

    Butterflyfish *tad
          = new Butterflyfish(3, "tad");

    tad->printMemory();
    tad->remember('a');
    tad->remember('x');
    tad->remember('a');
    tad->remember('b');
    tad->remember('c');
    tad->remember('a');
    tad->remember('d');
    tad->printMemory();
    tad->forget();
    tad->printMemory();
    return 0;
}
```

```
...
I'm Obnoxious!
cad
I'm Obnoxious!
I've seen:
  a 3 times
  x 1 times
  b 1 times
  c 1 times
  d 1 times
...
I'm Obnoxious!
I've seen:
  a 3 times
  x 1 times
  b 1 times
  c 1 times
  d 1 times
```

Polyquarium:
Let's now create an aquarium to house all our Fish!

```
const int MAX_FISH = 20;

class Aquarium {
public:
      Aquarium();                    // Default Constructor
      bool addFish(Fish* fish);      // Add Fish to Aquarium
      Fish *getFish(int n);          // Get Fish at nth index
      void oracle();                 // Read the Fish minds
      void feed(std::string food);   // Put food into Aquarium
      void startle();                // Makes the Fish forget

private:
      Fish * m_fish[MAX_FISH];       // Pointers to Fish.
      int m_nFish;                   // Number of Fish.
};
```

- `Aquarium();`
  Initially there are no Fish in the Aquarium.
- `bool addFish(Fish* fish);`
  Implement addFish, which takes a pointer to Fish. If the Fish cannot be added because the Aquarium already has MAX_FISH-many Fish residing, return false and don't add any. Otherwise, return true.
- `Fish *getFish(int n);`
  Implement getFish, which returns the pointer to the nth Fish that is added. Return nullptr if there is less than n Fish in the Aquarium, or if n is an invalid position.
- `void oracle();`
  Oracle reads the Fishies' minds; It prints the memory of all Fish in the Aquarium along with indicating which Fish had those thoughts.
- `void feed(std::string food);`
  Feed the Fish by providing a string of characters, each Fish in turn will encounter a character and remember that character, once encountered no other Fish can encounter that character. For example, suppose you have two Fish and if you feed the Fish "abc", the first Fish will encounter the 'a', the second Fish will encounter the 'b', finally the first Fish will encounter the 'c'.
- `void startle();`
  Startle will cause all the Fish to forget.

Example usage and output:

```
int main() {

   Fish *nemo = new Fish(3, "nemo");
   nemo->remember('a');
   nemo->remember('c');

   Fish *dory = new Fish(*nemo);

   Butterflyfish *tad
      = new Butterflyfish(4, "tad");
   tad->remember('a');
   tad->remember('a');

   Yellowtang *bubbles
      = new Yellowtang(3, "bubbles");
   bubbles->remember('t');

   std::cout << "----------AQUARIUM" << std::endl;
   Aquarium aq;
   aq.addFish(nemo);
   aq.addFish(dory);
   aq.addFish(tad);
   aq.addFish(bubbles);
   aq.oracle();
   std::cout << "----------Feed" << std::endl;
   aq.feed("abcdefghijkl");
   aq.oracle();
   std::cout << "----------Bubbles!" << std::endl;
   aq.feed("oooo");
   aq.oracle();
   std::cout << "----------Boo!" << std::endl;
   aq.startle();
   aq.oracle();

   delete bubbles;
   delete tad;
   delete dory;
   delete nemo;

   return 0;
}
```

```
----------AQUARIUM
nemo ac.
nemo ac.
tad aa..
I'm Obnoxious!
I've seen:
  a 2 times
bubbles t..
----------Feed
nemo aei
nemo bfj
tad acgk
I'm Obnoxious!
I've seen:
  a 2 times
  c 1 times
  g 1 times
  k 1 times
bubbles dhl
----------Bubbles!
nemo eio
nemo fjo
tad cgko
I'm Obnoxious!
I've seen:
  a 2 times
  c 1 times
  g 1 times
  k 1 times
  o 1 times
bubbles BUBBLES!
----------Boo!
nemo ...
nemo ...
tad ....
I'm Obnoxious!
I've seen:
  a 2 times
  c 1 times
  g 1 times
  k 1 times
  o 1 times
bubbles ...
```

Submission:

Do not make any changes to the public interfaces of any class except to make the indicated functions in the base class virtual. Implement all four classes, you should have:

fish.h	yellowtang.h	butterflyfish.h	aquarium.h
fish.cpp	yellowtang.cpp	butterflyfish.cpp	aquarium.cpp

You will have your own main.cpp for testing, but do not include it with your submission. Combine everything into a zip file name <lastname>_<id>.zip, for example nguyen_123456.zip. Note that when you resubmit on canvas it will postpend your file name with a number indicating its submission order, this is ok. If I take these 8 files, I must be able to compile them using VS2017 without any errors or warnings. Be sure to you do not introduce any compilation or link errors.