CS 31 Solutions Week 9

Please Fill Out End of Year LA Feedback Form:
http://tinyurl.com/LA-Feedback-F24

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

Concepts: Pointers, Dynamic Allocation

Reading Problems

1. What will the following program output?:

```
#include <iostream>
using namespace std;
void showarray(int* a, int l);
int main() {
      char s1[16] = "spring";
      char* p1;
      int a[8] = {3, 6, 9, 12, 15};
      int b[8] = {0,0,0,0,0,0,0,0};
      int* x = &b[0];
      int* p2;

      showarray(a, 8);
      showarray(x, 8);

      for (int i = 0; i < 8; i++){
            x[i] = a[i] + *(a+7-i);
      }
      showarray(x, 8);

      p2 = &a[2];
      p2 = p2 -1;
      (*p2)++;
      p2--;
      (*p2) += p2[0];
```

```
	showarray(a, 8);

	p1 = s1;
	while (*p1) {
		(*p1)--;
		p1++;
	}
	cout << s1 <<endl;
	return 0;
}

void showarray(int* a, int l) {
	int i;
	for (i = 0; i < l; i++) {
		cout << *(a+i)<<" ";
	}
	cout<<endl;
}
```

```
3 6 9 12 15 0 0 0 // default set uninitialized elements to 0 in array
0 0 0 0 0 0 0 0
3 6 9 27 27 9 6 3
6 7 9 12 15 0 0 0
roqhmf
```

2. Find the **six** errors in the following code, and write the fixes.

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

const int NAME_LEN = 100;

class Cat {
	int m_age;
	char m_name[NAME_LEN];
	string m_type;
public: (1) // classes are private by default!
	Cat(int age, const char name[], string type) {
		m_age = age;
		m_name = name;		(2)
		strcpy(m_name, name); (2) // name is a c-string, so use
strpy()
		m_type = type;
	}
```

```
            void introduce() {
                cout << "Hi! I am a " + m_type + " cat" << endl;
        }
};

struct Sheep {
        string m_name;
        int m_age;
        Sheep(int age) {
                m_age = age;
        }
        void introduce() {
                cout << "Hi! I am " + m_name + " the sheep" << endl;
        }
};(3)
Don't forget the semicolon! (3)

int main() {
        Cat* schrodinger = new Cat(5, "Schrodinger's cat", "Korat");
        schrodinger->introduce();
        cout << schrodinger->m_age << endl; (4)
        Private variables cannot be accessed outside a class
        declaration.
       cout << schrodinger->getAge() << endl;(4)

        Sheep dolly(6);
        dolly->introduce(); (5)
        dolly.introduce(); (5) // the -> operator is only used if you
        have a pointer. If dolly was a Sheep*, then you would use ->

        delete schrodinger;
        delete dolly;   (6)
        Do not delete dolly, because it was not created on the heap
        (not using the new operator)!
        Every execution of delete should correspond to an execution
        of new. (6)
}
```

The errors are highlighted in **red** and the fixes are in **green**! Each error and fix can be matched by the corresponding number.

What will the program above successfully print once all the fixes have been made?

Hi! I am a Korat cat!
5
Hi! I am the sheep!

3. Find the **4 errors** in the following class definitions so the main function runs correctly.

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

class Account {
public:
      Account(int x) {
            cash = x;
      }
      int cash;
}; // Don't forget the semicolon!

class Billionaire {
public:
      Billionaire(string n) : account(10000){
// : account(10000) is special notation since `account` has no
default constructor. This says "pass the value 10000 into account's
constructor." You'll learn more about this in CS 32!
            offshore = new Account(1000000000);
// We need the new keyword to allocate a pointer object with a longer
lifespan than a typical variable. This way, the account point doesn't
go out of scope once the constructor exits.
            name = n;
      }
      // now that we use dynamic allocation (new), we need a
destructor.
      ~Billionaire() {
            delete offshore;
      }

      Account account;
      Account* offshore;
      string name;
};

int main() {
```

```
        Billionaire jim = Billionaire("Jimmy");
        cout << jim.name << " has " << jim.account.cash +
        jim.offshore->cash << endl;
    }
```

Output: Jimmy has 1000010000

Programming Problems

1. After being defined by the above code, Jim the Billionaire funded a cloning project and volunteers himself as the first human test subject. Sadly, all his money isn't cloned, so his clone has his name, but has $0. Add the needed function to the Billionaire class so the following main function produces the following output.

```
int main() {
      Billionaire jim = Billionaire("Jimmy");
      Billionaire jimClone = jim;

      cout << jimClone.name << " has " << jimClone.account.cash
      + jimClone.offshore->cash<< endl;

      cout << jim.name << " has " << jim.account.cash +
      jim.offshore->cash << endl;
}

Billionaire(const Billionaire &b) //-> Review for section
      : account(0), name(b.name)
{
      offshore = new Account(0);
}
```

Output: Jimmy has 0
Jimmy has 1000010000

2. Implement a `Netflix` class which holds `Show` objects in a "watching queue". The capacity cannot exceed 100.

```
class Netflix {
public:
      Netflix(int capacity);
      void watch(string name);
      bool add(string name);
      void cleanUp();
      ~Netflix();
private:
      int num_shows; // number of shows in queue
      Show* queue[100];
      int m_capacity;
      // Hint: you can use this function in cleanUp()
      void remove_from_queue(int index) {
```

```
            delete queue[index];
            for (int i = index; i < num_shows - 1; i++) {
                  queue[i] = queue[i+1];
            }
            num_shows--;
      }
};

class Show {
public:
      Show(string name);
      void watch();
      bool isWatched(){
            return is_watched;
      }
      string getName(){
            return name;
      }
private:
      string name;
      bool is_watched;
};
```

Implement all of the functions highlighted in blue.

1. `Netflix(int capacity)` -- declare a Netflix object with a maximum capacity for the number of shows in queue.
2. `void watch(string name)` -- tells the Netflix object that you want to watch a particular show (as a result when cleanUp is called, the show you watched should be removed from the queue)
3. `bool add(string name)` -- add a new show to your queue. If the addition is successful (queue is not full), return True, else return False.
4. `void cleanUp()` -- clean up the queue and remove all shows that have been watched. Update the number of shows to reflect this change
5. `~Netflix()` -- destructor, make sure you remember to delete everything you have created on the heap!
6. `Show(string name)` -- declare a Show object with a name
7. `void watch()` -- updates the Show object from unwatched to watched. All shows are initially “unwatched”

Sample use case:

```
int main() {
      Netflix n(3);
      n.add("Stranger Things"); // returns True
      n.add("The Office"); // returns True
      n.add("Arrested Development"); // returns True
      n.add("Sherlock"); // returns False
      n.watch("The Office");
      n.cleanUp();
      n.add("Sherlock"); // returns True
}
```

```
Netflix::Netflix(int capacity) {
      m_capacity = capacity;
      num_shows = 0;
}

void Netflix::watch(string name) {
      for(int i = 0; i < num_shows; i++) {
            if (queue[i]->getName() == name) {
                  queue[i]->watch();
            }
      }
}

bool Netflix::add(string name) {
      if (num_shows == m_capacity)
            return false;
      else {
            queue[num_shows] = new Show(name);
            num_shows++;
            return true;
      }
}

void Netflix::cleanUp() { //****
      int toClean[100];
      int counter = 0;
      for(int i = 0; i < num_shows; i++) {
        if (queue[i]->isWatched()) {
            toClean[counter] = i;
            counter++;
        }
      }
      for(int j = 0; j < counter; j++) {
```

```
            remove_from_queue(toClean[j]);
      }
} //*****

Netflix::~Netflix() {
      for(int i = 0; i < num_shows; i++) {
            // NOTE: the array isn't dynamically allocated, but the
            objects inside are, so to avoid a memory leak, iterate
            over the array, deleting each element

            delete queue[i];
      }
}

Show::Show(string name) {
      this->name = name;
      is_watched = false;
}

void Show::watch() {
      is_watched = true;
}
```