Exam Review 2 Solutions:

Write Output:

1.	Constructed Copied Copied Deleted A is 4 Deleted A is 3 Constructed Copied Copied Deleted A is 8 Deleted A is 7 Copied Copied Deleted A is 14 Deleted A is 11 Deleted A is 4 Copied Deleted A is 15 Deleted A is 14
2.	1 0 -3 4 3 0
3.	7 11 15 19 23 10
4.	11 6 11 7 12 11
5.	Constructed Constructed Copied Copied Destroyed Destroyed Copied Destroyed Destroyed 17 10 Destroyed

6.	30
7.	0 1 2 3 4 0 1 2 3 4 5 5 5 5 5 5

Find Errors:

1.	**friend class B;** //compiler doesn't know what B is yet, as it reads file from top to bottom
2.	**b.foo()**;//should be b -> foo()
3.	//Not a member function because no A:: **int foo()** {
4.	**a[i] = b[i]**;//cannot change a constant void f1(const int * a) { **a++** //forgot ; but a++ is legal ***a = 5;**//Changing a constant } void f2(int a) { a += 2;//Will compile, but it won't really do anything } void f3(const int &a) { **a += 5;**//cannot modify a const } int f4(int *a) { a += 5;//did not dereference a, so it will change the memory address return a }
5.	**c = a + b**;//Operator + not overloaded
6.	Forgot semicolon after class declaration, operator function name is incorrect, and operator<< cannot be implemented as a member function of A, it must be a friend function (friend ostream& operator<<(ostream& out, const A& a);)
7.	Uses Java style of functions. You don't need a public or private before functions
8.	Both functions named "f1" and take in same parameters
9.	Modifying constant values and copy function cannot access private variables of class A as it is not a friend function.
10.	/* Solution to FE10 Explanation: const int * c and int const * c behave in the same way, where c is a variable pointer to a constant integer int * const c1 -> c1 is constaht pointer to variable integer int const * const c2 -> constant pointer to constant integer Functions with const will not be able to modify any variables in the object */ //Pointer can change but not value void f1(const int * p, int p_size) {

	p++; //OK p[0] = 10; //Bad *p = 0; //Bad } //Pointer can change but not value void f2(int const * a, int a_size) { a[0] = 55; //Bad *a = 45;//Bad } //Pointer can't be changed void f3(int * const p, int p_size) { p++;//Bad *p = 0;//OK } class A { public: int a, b; const int c; A(): a(0), b(0), c(105) {} ~A(); void g()const; int g1(); } void f4(const A &a) { a.g(); a.g1();//Calls non-const function. Bad } //Modifies things in the class void A::g()const { a++; b--; } int A::g1() { return c + 1; }
11.	~A() {delete [] a;}//Unnecessary deletion as “a” is not a pointer
12.	//This can never be a member function as C++ does not allow it ostream& A::operator<< (ostream& out) {

Write Code: (note that there are multiple solutions for these, but these are the ones I made. You could also put all of this in one .C file, but that is not good programming practice.)

FileName:	**Main.C**	**Class.C**	**Class.h**

1.

```
#include "class.h"

int main() {
   char a[] = "Hello", b[] =
"world", c[] = "Hi";
   String sa(a), sb(b), sc(c);

   sc = sa + sb;
   cout << sc << endl;
   sc += sa;
   cout << sa << endl;
}
```

```
#include "class.h"

String::String() {
   str = new char[0];
   len = 0;
}
String::String(const char
*a) {
   str = new char[strlen(a)];
   strcpy(str, a);
   len = strlen(a);
}
String::~String() {
   delete [] str;
}
char*
String::getCharArray() {
   return str;
}
int String::length() {
   return len;
}
char String::charAt(int a) {
   if (a < len && a > 0) {
      return str[a];
   } else {
      cerr << "Bad Index" <<
endl;
      return '\0';
   }
}
String&
String::operator+(const
String& a) {
   char *temp = new
char[len + a.len];
   strcpy(temp, str);
   strcat(temp,a.str);
   String *toReturn = new
String(temp);
   return *toReturn;
}
String&
String::operator+=(const
String& a) {
   char *temp = new
char[len + a.len];
   strcpy(temp, str);
```

```
#ifndef CLASS_H
#define CLASS_H

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

class String {
   char *str;
   int len;
   public:
      String();
      String(const char
*);
      ~String();
      char*
getCharArray();
      int length();
      char charAt(int);
      String&
operator+(const
String&);
      String&
operator+=(const
String&);
      friend ostream&
operator<<(ostream&,
const String&);
};
#endif
```

		strcat(temp,a.str); delete [] str; str = temp; return *this; } ostream& operator<<(ostream& out, const String& a) { out << a.str; return out; }	
2.	#include "class.h" int main() { Stack stk; for (int i = 0; i < 5; i++) { stk.push(i); } cout << stk.peek() << endl; while (!stk.isEmpty()) { cout << stk.pop() << endl; } }	#include "class.h" Stack::Stack() { head = NULL; } Stack::~Stack() { while (head != NULL) { pop(); } } void Stack::push(int a) { if (head == NULL) { head = new Node(a, NULL); } else { Node *temp = new Node(a, head); head = temp; } } int Stack::peek() { if (head != NULL) { return head -> data; } cerr << "ERROR" << endl; throw -1; } int Stack::pop() { if (head != NULL) { int toReturn = head -> data; Node *toDelete = head; head = head -> child; delete toDelete; return toReturn; }	#ifndef CLASS_H #define CLASS_H #include <iostream> using namespace std; class Node; class Stack { Node *head; public: Stack(); ~Stack(); void push(int); int peek(); int pop(); bool isEmpty(); }; class Node { int data; Node *child; public: Node(int, Node *); friend class Stack; }; #endif

		cerr << "ERROR" << endl; throw -1; } bool Stack::isEmpty() { return head == NULL; } Node::Node(int a, Node *b) { data = a; child = b; }	
3.	#include "class.h" int main() { Complex a(1,0), b(1,1), c(0,2); Complex d(0,0); d = a + b; c += d; b = c - a; a -= c; cout << a << b << c << d << endl; }	#include "class.h" Complex::Complex(int r, int i) { real = r; imag = i; } Complex::~Complex() { //Don't really need one } Complex& Complex::operator+(const Complex& a) { Complex *toReturn = new Complex(a.real + real, a.imag + imag); return *toReturn; } Complex& Complex::operator-(const Complex& a) { Complex *toReturn = new Complex(a.real - real, a.imag - imag); return *toReturn; } Complex& Complex::operator+=(const Complex& a) { real += a.real; imag += a.imag; return *this; } Complex& Complex::operator-=(const Complex& a) {	#ifndef CLASS_H #define CLASS_H #include <iostream> using namespace std; class Complex { int real, imag; public: Complex(int,int); ~Complex(); Complex& operator+(const Complex&); Complex& operator-(const Complex&); Complex& operator+=(const Complex&); Complex& operator-=(const Complex&); friend ostream& operator<<(ostream&, const Complex&); }; #endif

		real -= a.real; imag -= a.imag; return *this; } ostream& operator<<(ostream& out, const Complex& a) { out << "(" << a.real << " + " << a.imag << "i)"; return out; }	

As always, the solutions are on Github:
https://github.com/Nesdood007/CSCE240S2017/tree/master/Worksheets/ExamReview2

The questions are split up by category (Write Code, Write Output, and Find Errors) and numbered based on the number in the worksheet. You can download and compile it yourself if you like.