#1: Give the output for the following program.

```cpp
#include <iostream>
#include <cctype>
using namespace std;

void Foo(int n, char& ch);

int main()
{
    int num = -5;
    char c = 'b';

    cout << num << "", " << c << endl;
    Foo(num, c);
    cout << num << "", " << c << endl;

    return 0;
}

void Foo(int n, char& ch)
{
    if (n < 0)
        n = n * -1;
    if (islower(ch))
        ch = toupper(ch);
}

ANSWER:
-5, b
-5, B
```

#2: Write a function inch2feet, which converts some inches to a feet-inch pair. For example, 59 inches can be converted to 4 feet 11 inches. This function receives three int arguments: the first is the inch in total, the second is the converted feet, and the third is the converted remaining inches. Think about what kind of parameter-passing method (call-by-value or call-by-reference) should be used for each parameter. This function returns nothing.

#3: Write a function equalsIgnoreCase. This function receives two const cstring parameters (say s1 and s2), and returns true if the two cstrings contained the same strings, ignoring the upper/lower cases, or false otherwise. Write the function using a pointer notation (*) and WITHOUT using any <cstring> library functions.

#4: Write a program/main() which demonstrates the functions asked in the two questions above (inch2feet, equalsIgnoreCase). Call each function at least twice, with different parameter values.

ANSWER: Sample code pasted at the end of this file.

#5: Give the output for the following piece of code.

ANSWER: Shown in the code.

```cpp
int ar[4] = {11, 22, 33, 44};
int *ptr;

ptr = ar;
cout << *ptr;       // output (1) 11
ptr++;
cout << *ptr;       // output (2) 22
cout << *ptr++;     // output (3) 22
cout << **ptr;      // output (4) 44
cout << *(ptr-2);   // output (5) 22
cout << *ptr - 2;   // output (6) 42
ptr = &ar[3];
```
cout << *ptr; // output (7) 44

#6: Given a piece of code below, state whether each of the marked lines (a) through (g) cause an error. If it is an error, state whether it's a syntax error (which compilers do not accept) or a run-time error (which compilers accept but causes error during execution). Assume the necessary library header files are #included in the code.

ANSWER: Shown in the code.

```cpp
int main()
{
    char b1[10],
        b2[10] = "my mother"; // (a)

cout << b2 << endl; // (b)
b1 = b2; // (c) -- syntax error (b1 is a pointer constant)
if (b1 == b2) // (d) -- neither syntax nor run-time error, but
cout << "two strings are equal"; // it doesn't do the intended test

char b4[5] = "one";
strcat(b4, ", two"); // (e) -- danger for run-time error (array
    // index out of bounds)

int num;
cout << "Enter a number: ";
cin >> num;
char b3[num]; // (f) -- syntax error (Size must be a constant for [];
    // if this were a dynamic array, a variable is ok)

char s[3] = {'A', 'B', 'C'};
int len = strlen(s); // (g) -- danger for run-time error
```

#7: Find errors in the following code. If there are no errors, say "Nothing is wrong".

```cpp
int *pint; pint = new int[100];
// ....
delete pint[100];
```

ANSWER: Syntax error -- pint[100] is not a pointer. To fix the code, change "delete pint[100];" to "delete [] pint;" (as intended).

#8: For the class Person shown in the class, add a method setName (*). This method sets/changes the name string of this person to the same as the parameter cstring (by deep-copy). Be sure to 'get rid of' the current/exisitng name string before making a new string.

```cpp
class Person
{
public:
    Person();
    Person(const char* n, int a);
    Person(const Person &per); // copy constructor
    ~Person();

    int getAge() const { return age; } 
    const char* getName() const { return name; }

    void setName(const char* n); // (*) WRITE THIS METHOD

    friend ostream & operator<<(ostream &out, const Person &p);

private:
    char *name; // cstring of various size (just enough for a given name)
    int age;
};
```

ANSWER:

```cpp
void Person::setName(const char* n)
```
delete [] name;
name = new char[strlen(n)+1];
strcpy(name, n);
}

--------------------------------------
#9: Using the class Person above, suppose this class does NOT implement the overloaded assignment (=) operator. Draw a picture after the following lines are executed.

int main()
{
    Person p1("Skip", 46), p2("Jim", 29);
    p2 = p1; // after this line
    ...
}

ANSWER: p2's name now points to p1's name cstring, and the previous cstring "Jim" is causing memory leak.

+-------------+        +-------------+
|              |        |              |
|              |        |              |
|     +-----+ |        |     +-----+ |
|     | o----> |        |     | o----> |
|     | S | k | i | p | \0 |     | J | i | m | \0 |
|     +-----+        +-----+        +-----+        +-----+
| name      |        | age        |
| 46        |
+-------------+        +-------------+

#10: For the class PFArray you wrote for HW#6, add a method deleteElementShift (*).

class PFArray
{
public:
    PFArray(int initCapacity = 1);
    int Capacity() const { return capacity; }
    int Size() const { return size; }
    int elementAt(int index) const;
    void addElement(int elt);
    void deleteElement(int elt);
    void deleteElementShift(int elt); // (*) WRITE THIS METHOD
    void print() const;

private:
    int* elementData; // pointer to a dynamic array
    int capacity;     // physical length of the array
    int size;         // # of elements inserted so far
};

ANSWER:

void PFArray::deleteElementShift(int elt)
{
    int i;
    for (i = 0; i < size; ++i) {
        if (elementData[i] == elt)
            break;
    }
    if (i < size) { // i.e., elt found
#11: For the class Loan you wrote for HW#5, add the following overloaded operators:

(a). operator== as a class member function. Two Loan's are the same if all member variables (amountBorrowed, yearlyRate, years) are the same.
(b). operator< as a regular function. One Loan is less than (<) another Loan when the total payment (over the whole loan term) is smaller.
(c) operator<< as a friend function. It prints the same way as the other method "printLoan()" does, except that operator<< outputs to the parameter stream (and returns it).

ANSWER:

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

class Loan
{
public:
    Loan() : amountBorrowed(0.0), yearlyRate(0.0), years(0) {}
    Loan(double a, double r, int y) : amountBorrowed(a), yearlyRate(r), years(y) {}

    double getAmountBorrowed() const { return amountBorrowed; }
    double getYearlyRate() const { return yearlyRate; }
    int getYears() const { return years; }

    void setAmountBorrowed(double a) { amountBorrowed = a; }
    void setYearlyRate(double r) { yearlyRate = r; }
    void setYears(int y) { years = y; }
    double monthlyPayment() const;

    bool operator==(const Loan& L2) const;
   _friend_ functions
    friend ostream& operator<<(ostream &out, const Loan &L);

private:
    double amountBorrowed;
    double yearlyRate;
    int years;
};

// Prototypes for regular functions
bool operator<(const Loan& m1, const Loan& L2); // (b)

--------------------
// (a)
bool Loan::operator==(const Loan& L2) const
{
    return (amountBorrowed == L2.amountBorrowed && yearlyRate == L2.yearlyRate 
            && years == L2.years);
}

// (b)
bool operator<(const Loan& m1, const Loan& L2) {
    double total1 = m1.monthlyPayment() * 12 * m1.getYears();
    double total2 = m2.monthlyPayment() * 12 * m2.getYears();
    return (total1 < total2);
}

// (c)
ostream& operator<<(ostream &out, const Loan &L)
{
    // Magic Formula -- Necessary lines to control the number of decimal digits
```
displayed
out.setf(ios::fixed);
out.setf(ios::showpoint);
out.precision(2);  // 2 decimal digits
out << "Loan: $" << L.amountBorrowed << " at " << L.yearlyRate << ": for " << L.years << "years."
return out;
}
--------------------------------------
#12: Which situation or situations do NOT use the copy constructor? Give all that apply.
A. Calling a function with a reference parameter (call-by-reference)
B. Calling a function with a value parameter (call-by-value)
C. Declaring a variable to be a copy of another existing object
D. Returning a value from a function.
E. Making an assignment to an already declared variable.

ANSWER: A, E
--------------------------------------
#13: Defined below are two simple classes, Rectangle and Box. Identify two problems with the Box class. You must explain the problem and provide a solution to what the programmer intended.

class Rectangle  // base class
{
private:
    double length, width;

public:
    Rectangle ( double l, double w ) : length(l), width(w) {}
    double area() { return length * width; }
};

class Box : public Rectangle  // derived class
{
private:
    double height;

public:
    Box( double l, double w, double h )
    {
        Rectangle( l,w );
        height = h;
    }

    double volume()
    {
        return length * width * height;
    }
};

ANSWER: There are 2 errors:
The constructor is in error. Since the base class Rectangle does not have the default constructor, it has to be called in the header in the derived class Box. The volume() in Box cannot access length and width because they are private in Rectangle.
The corrected class definition should be:

class Box : public Rectangle
{
private:
    double height;

public:
    Box( double l, double w, double h ) : Rectangle(l, w)
    {
        height = h;
    }

    double volume()
    {
return area() * height; // Rectangle::area() is not necessary  
// because Box doesn't have the method.

#14: Given the following classes B and D, give the output if each of the marked lines if it is legal (it may be illegal).
ANSWER: As shown in the code.

class B
{
    protected:
    int x;
    int y;
    public:
    B(int a, int b) : x(a), y(b) {}
    virtual void Fn1() { cout << x << endl; }
    virtual void Fn2() { cout << y << endl; }
};

class D : public B
{
    protected:
    int x;
    public:
    D(int a, int b, int c) : B(a, b), x(c) {}
    virtual void Fn1() { cout << B::x << x << y << endl; }
    virtual void Fn2(int a) { cout << (a * x) << endl; }
    virtual void Fn3() { cout << x << endl; }
};

int main()
{
    B b(3, 2);
    D d(1, 5, 7);
    d.Fn1(); // (1) legal, output is "175"
    d.Fn2(); // (2) illegal -- because of name hiding
    d.Fn3(); // (3) legal, output is "7"
    B *ptr = &d;
    ptr->Fn1(); // (4) legal, output is "175"
    ptr->Fn2(); // (5) legal, output is "5"
    ptr->Fn3(); // (6) illegal -- because B does not have a function Fn3
}

#15: Consider the hierarchy of classes shown in the following diagram.
Pet
  /  \
Cat  Dog
  \   
    LoudDog

The base class Pet is specified as an abstract class as shown in the following declaration. Each Pet has a name that is specified when it is constructed.

class Pet
{
    public:
    Pet(string name) { myName = name; }
    string getName() { return myName; }
    virtual string speak() = 0;
    private:
    string myName;
};

The derived class Dog has the partial class declaration shown below.

class Dog : public Pet
{
    public:
    Dog(string name) { /* implementation not shown */ }
    virtual string speak() { /* implementation not shown */ }
a. Given the class hierarchy shown above, write a complete class definition for the class Cat, including implementations of its constructor and method(s). The cat method speak returns "meow" when it is invoked.

**ANSWER:**

```cpp
class Cat : public Pet
{
public:
    Cat(string name) : Pet(name) {}
    virtual string speak() { return "meow"; }
};
```

b. Assume that class Dog has been declared as shown above. If the string dog-sound (whatever that is) is returned by the Dog method speak, then the LoudDog method speak returns a string dog-sound repeated two times. Given the class hierarchy shown previously, write a complete class definition for the class LoudDog, including implementation of its constructor and method(s).

**ANSWER:**

```cpp
class LoudDog : public Dog
{
public:
    LoudDog(string name) : Dog(name) {}
    virtual string speak()
    {
        string bark = Dog::speak();
        return (bark + ", " + bark);
    }
};
```

c. Consider the following partial definition of class Kennel:

```cpp
class Kennel
{
private:
    Pet* petList[5]; // all elements are pointers to objects of Pet's subclasses
public:
    ...
    void allSpeak() { /* to be implemented */ }
}
```

Write the method allSpeak(). For each Pet in the Kennel, its name followed by the result of call to its speak method is printed, one line per Pet.

**ANSWER:**

```cpp
void Kennel::allSpeak()
{
    for (int i = 0; i < 5; ++i)
        cout << petList[i]->speak() << endl;
}
```
cout << "- > " << origInches << " is " << ansFt << " feet and " << ansIn << " inches.\n";
}

//==== Code to call & test Question 3
cout << "\n====== Test Question 3 =========\n"
for (int i = 0; i < 6; i+=2) {
    cout << "equalsIgnoreCase("" << words[i] << "," << words[i+1] << ") returns ";
    cout << (equalsIgnoreCase(words[i],words[i+1]) ? "true" : "false") << endl;
} // ensure NULL pointers are covered

cout << "equalsIgnoreCase(NULL,""C++\") returns ";
cout << (equalsIgnoreCase(NULL,"C++") ? "true" : "false") << endl << endl;

system("pause");
return 0;
}

//*** Question 2
void inch2feet(int orig, int& ft, int& in)
{
    ft = orig / 12;
    in = orig % 12;
}

//*** Question 3
bool equalsIgnoreCase(const char *s1, const char *s2)
{
    // special case, for possible null pointer(s)
    if (s1 == NULL || s2 == NULL)
        return false;
    else {
        // general case; examine the cstrings
        const char *ptr1 = s1;
        const char *ptr2 = s2;

        while (*ptr1 != '\0' && *ptr2 != '\0') {
            if (toupper(*ptr1) != toupper(*ptr2))
                return false;
            else {
                ++ptr1;  ++ptr2;
            }
        }

        if (*ptr1 != '\0' || *ptr2 != '\0') // i.e., not the same length
            return false;
        else
            return true;
    }
}

/*
 ===== Test Question 2 ==========

Enter the original inches: 59
- > 59 is 4 feet and 11 inches.

Enter the original inches: 60
- > 60 is 5 feet and 0 inches.

===== Test Question 3 ==========

equalsIgnoreCase("Naruto","NARUTO") returns true
equalsIgnoreCase("12 3","12 34") returns false
equalsIgnoreCase(",","C++") returns false
equalsIgnoreCase(NULL,"C++") returns false
*/