1. Consider the following C++ code:

```
class Q
{
public:
    int x;
    Q();
    Q(const Q& q);
    Q& operator=(const Q& q);
};

Q::Q(): x(0)
{
    cout << "1";
}

Q::Q(const Q& q)
{
    cout << "2";
    x = q.x;
}

Q& Q::operator=(const Q &q)
{
    cout << "3";
    x = q.x;
    return *this;
}

void foo(Q q4, Q& q5)
{
    Q q6 = q4;
    q6 = q5;
}

void bar()
{
    Q q1;
    Q q2 = q1;
    q1 = q2;
    Q q3(q1);
    foo(q1, q2);
}
```

If the function ***bar*** is called, what would be the output printed?

2. If class A declares class B as its friend, then which of the following are true? Mark an "X" by each applicable answer.

_____ Class A can access private and protected members of Class B

_____ Class B can access private and protected members of Class A

_____ Class B can access private, but not protected members of Class A

_____ Class A can access private, but not protected members of Class B

_____ Class B can access protected, but not private members of Class A

_____ Class A can access protected, but not private members of Class A

3. Answer the following questions:

(a) Why do we sometimes write member functions of classes to be *pure virtual*?

(b) What is required for a class to become an *abstract base class*?

One pure virtual function

(c) Why do we use *abstract base classes*?

provide an appropriate base class from which other classes can inherit.

4. Consider the following declarations:

I. `const int* x`
II. `const int* const x;`
III. `int* const x;`
IV. `int* x;`
V. `int** x;`

Match declarations I-V with the following:

(a) _____ A pointer to an integer that can never change the address it points to, but the integer it points to can change in value.

(b) _____ A pointer to a pointer to an integer

(c) _____ A pointer whose address can never change, and whose integer value that it points to cannot change either.

(d) _____ A pointer to an integer whose value can never change, but the address that the pointer points to can.

5. A *self-indexed* array of size n is an integer array of which each of its elements is an integer between 0 and n-1, inclusive. Here is an example:

```
int arr[10] = {5, 4, 7, 3, 1, 9, 0, 5, 1, 2};
```

We consider each value in a *self-indexed* array to specify the index of some element in the array. The *successor* of an element *k* in an array *arr* is the element at index `arr[k]`. The successor of element 0 in the example array above is 5, since arr[0] is 5. Likewise, the successor of element 5 (the one at index 0) is 9.

Suppose we want to check if a *self-indexed array* has a *circuit* based on a particular starting element. We define a *circuit* such that if we start at some element *k*, the *path* of successors eventually leads back to the element *k***.**

Example: If we start at element 2, the successor is 7, whose successor is 5, whose successor is 9, whose successor is 2, which is what we started at. In this case, we have a circuit.

Another example: If we start at element 8, the successor is 1, whose successor is 4, whose successor is 1, and this pattern cycles between 1 and 4. Therefore, there is not a circuit.

Write the C++ function ***hasCircuit***, which takes a *self-indexed array* of integers, an integer *n* specifying the length of the array, and an integer *start* which is the starting element to check for a circuit with, and returns `true` or `false` for whether there is a circuit.

```
bool hasCircuit(int a[], int n, int start)
{
```

6. What is the difference in the behavior of the two commented lines in the code below?

```
std:vector<int> v{1,2,3};
cout << v[4] << endl;                // Line 1
cout << v.at(4) << endl;             // Line 2
```

7. Which of the following best describes the *static members* of a class?

_____ Data which is allocated for each object separately

_____ Data that never changes throughout the lifespan of a program

_____ Data which is common to all classes

_____ Data which is common to all objects of a class

_____ Data which is unmodifiable after initialization

8. What is the output of the following C++ code?

```
int array[10] = {4,6,2,3,-1,-3,2,2,-7,-9};

int* p = array;

for (int i = 0; i < 4; i++){
    int hops = *p;
    cout << *p << ',';
    p += hops;

}
cout << *p << endl;
```

9. Assuming X and Y are classes, which of the following declarations would be legal? Mark an "X" by each applicable statement.

_____ `X xx;`

_____ `Y yy;`

_____ `X& rxx = yy;`

_____ `Y& ryy = xx;`

_____ `X* px = new X;`

_____ `Y* py = new Y;`

_____ `X* px2 = py;`

_____ `Y* py2 = px;`

10. What is the output of the following C++ code?

```
void one(int a, int b){
    a = b + 1;
    b = a + 2;
}

void two(int& a, int& b){
    a = b + 1;
    b = a + 2;
}

void three(int& a, int b){
    a = b + 1;
    b = a + 2;
}


int main() {
    int a = 1;
    int& b = a;
    int* c = &b;
    int* d = new int;
    *d = a + b + *c;
    cout << *d << endl;
    one(a, *d);
    cout << a << "," << b << ',' << *c << ',' << *d << endl;
    two(a, *d);
    cout << a << ',' << b << ',' << *c << ',' << *d << endl;
    three(*c, *d);
    cout << a << ',' << b << ',' << *c << ',' << *d << endl;
    delete d;
}
```

11. Write the C++ function ***eraseDuplicateChars*** for removing duplicate characters from a string. For example, the function should mutate the string "Hello there" into "Helo thr".

You must *mutate* the string given in the argument of the function (recall that std::string is mutable, and has an *erase* member function).

You may **not** create any new string objects, nor create any containers (such as arrays) in writing the function.

```
void eraseDuplicateChars(std::string& s)
{
```

12. Consider the following three kinds of function headers.

```
void func(int x);        // A
void func(int* x);       // B
voic func(int& x);       // C
```

For each of the following use cases, indicate whether each function defined by headers A, B, and C above are appropriate for it by marking an "X" by it.

(a) Function that can change an `int` variable's value.

_____ A _____ B _____ C

(b) Allows passing an array to the function.

_____ A _____ B _____ C

(c) Guarantees that the original variable won't be modified by the function call.

_____ A _____ B _____ C

13. Consider the following C++ code that uses exceptions.

```
void f(int val) {
    cout << val << ',';
    switch (val) {
        case 0:
            cout << 0 << ',';
        case 1:
            throw 10;
        case 2:
            throw string("hello");
        default:
            throw 'A';
    }
}

void g() {
    for(int i = 0; i < 4; ++i) {
        try {
            cout << i << ',';
            f(i);
            cout << i << ',';
        } catch (int i) {
            cout << i << ',';
        } catch (char c) {
            cout << c << ',';
        } catch (string s) {
            cout << s << ',';
        }
    }
}
```

What is the output generated when the function ***g*** is called?

14. Suppose we wish to implement a LinkedList class, which naturally, implements a linked list. The class is partially declared below:

```
class LinkedList {
public:
    LinkedList();
    LinkedList(const LinkedList &list);
    LinkedList &operator=(const LinkedList &list);
    ~LinkedList();

    // All other member functions, such as add() and remove()

private:
    struct Node {
        int value;
        Node* next;
    };

    Node* head;
    unsigned int size;
};
```

Assuming all other member functions are implemented properly, write the *destructor* for this LinkedList class, which should deallocate all memory utilized to store the nodes of the list, starting at the head. Ensure your code does not produce any memory leaks.

```
LinkedList::~LinkedList()
{
```

15. Which of the following is/are regarding C++ standard library iterators?

_____ An iterator typically holds an address, and the **++** operator applied to the iterator always increases that address

_____ The **auto** keyword can be used to allow the compiler to deduce the proper type for an iterator automatically.

_____ For a valid standard library container `c`, when the expression `c.end() - c.begin()` is well defined, it returns the same value as `c.size()`

_____ A **`const std::vector<std::string>::iterator`** can be used to iterate over a **const** vector container.

_____ For a valid standard library container `c`, the iterator returned by `c.end()` refers to the last position in `c`.

16. Consider the following code fragment:

```
int a[2];
int* x = new int[3];
int* y = new int[5];
int* t = y;
y = x;
x = a;
delete t;
delete y;
delete x;
```

What bugs does the code have? Indicate them clearly on the code above. You may describe the bugs below, but clearly indicate which line(s) of code each bug you describe refers to.

17. Consider the following C++ class definitions:

```
class Animal {
public:
    Animal(const std::string& givenSound): sound{givenSound} {
        std::cout << "A";
    }

    std::string getSound() {
        return sound;
    }

    virtual std::string getType() {
        return "Animal";
    }

private:
    std::string sound;
};

class Pig: public Animal {
public:
    Pig() : Animal("Oink") {
        std::cout << "P";
    }

    std::string getType() {
        return "Pig";
    }
};

class Chicken: public Animal {
public:
    Chicken(): Animal("Cluck") {
        std::cout << "C";
    }

    std::string getType() {
        return "Chicken";
    }
};

class Cow: public Animal {
public:
    Cow() : Animal("Moo") {
        std::cout << "M";
    }

    std::string getType() {
        return "Cow";
    }
};
```

17 (continued). Referring to the class declarations of **Animal, Pig, Chicken,** and **Cow** on the previous page, write the output that is printed when the following code fragment is executed.

```
Animal cow = Cow();
Animal chicken1 = Chicken();
Animal* chicken2 = &chicken1;
Animal* pig = new Pig();
std::cout << std::endl;
std::cout << cow.getSound() << ": " << cow.getType() << std::endl;
std::cout << chicken2->getSound() << ": " << chicken2->getType() << std::endl;
std::cout << pig->getSound() << ": " << pig->getType() << std::endl;
```