Xavier Kuehn

1. In the public section of the class declaration, define a constructor with default values.

```
public:
fruit (int w = 1, int c = 2) {
     weight = w;
     Color = c;
}
```

2. Stack memory is used to store temporary variables created during runtime, they can be accessed quickly and do not need to be managed by the programmer. Memory is freed when the calling function exits.

3. Heap variables are essentially global in scope because once they are allocated on the heap, they can be accessed by any function as long as a valid memory address is used.

```
 int * f1() {                          int f2(int *ptr) {
     int *ptr = new int;                   return *ptr;
     return ptr;                       }
 }
```

4. A resource (memory) is leaked when it is allocated to the heap but not freed before the allocating function returns.

```
int main() {
     int *ptr = new int;
     *ptr = 3;
     std::cout << "The value is: " << *ptr << std::endl;
     return 0; // ptr not freed before return
}
```

5. The issue with using the automatic assignment operator for classes with dynamic memory is that the pointer values are set equal so they point to the same memory address and therefore the same value. (e.g. if there are two dynamically allocated arrays within a class, using the assignment operator will make it so that both object's arrays point to the same array values in heap memory.

```
x.data --->[x][x][x][x][x] // where x is any stored value
y.data -----^
```

6. The “this” keyword is a pointer that points to the class object for which the function is a member of. Since a friend function is not a member function but only has access to the private fields of a class, the “this” keyword cannot be used.

7. The output of this code is an allocation error. In the bookInfo function defined on line 8, the delete operator is used to free “this”, the function-calling object. However, on line 20, a new book is created that is not dynamically allocated and thus is placed on the stack and cannot be freed. When the subsequent line is executed, the program tries to free memory that was never allocated.

8. Below is the full implementation of the bag class “reserve” function

```
void bag::reserve(size_type new_capacity) {
      value_type *bigger_array;
      if (new_capacity == capacity) return;
      if (new_capacity < used) new_capacity = used;
      bigger_array = new value_type[new_capacity];
      copy(data, data + used, bigger_array);
      delete [] data;
      data = bigger_array;
      capacity = new_capacity;
}
```

9. Note: all A objects are initialized with a value of 5; foo changes pointer address then set value to 10.

```
OUTPUT:
value = 5
value = 5
```

Nothing is done to ao for the first cout statement on line 27 thus 5 is printed. bo.foo(ao) sets the private member A object of bo to ao but calling ob.setValue(10) within does not change the value of a directly, only the copy stored in bo. If instead on line 31, bo.ob.getValue() was called the second output line would be 10.

10. The program runs without issue, first “Computer::process()” is printed due to the initialization of the employee object and allocation of a new computer object. Then “Employee::foo()” is printed by line 26.