Section 1: c++

1. A reference refers to another variable’s memory location where as a pointer has it’s own memory location and stores another variables memory location.
2.
   ```
   Foo temp;
   temp.CalculateMysteries();
   ```
3.
   ```
   Foo::TotalBars();
   ```
4. No, since it is a static method it can only access static variables. Unless the instance variable is passed by reference.
5. Yes since it is a static variable.
6.
   ```
   Foo &tempR = temp;
   tempR.CalculateMysteries();
   ```
7.
   ```
   Foo *tempP = new Foo();
   tempP->CalculateMysteries();
   ```
8. If you want to have the object to exist outside its scope.

Section 2: Good Coding Practices

1.
   ```
   If (hasACat == false)
     returnItForADog();
   ```
2.
   ```
   int x = some value from somewhere;
   int y = some value from somewhere;

   if(y > x) {
     x += 1;
   } else if(x > y) {
     y += 1;
   } else if(x == y) {
     x = x + y;
   }
   cout << "We're messing with numbers!" << endl;
   ```

Section 3: keywords/const/overloading

1. A const field prevents a variable from being changed like in a function that shouldn’t change the value it was given. A const method prevents the method from making any changes to the object.
2. In the initializer list.
3. Inheriting a class gives you the public variables of the parent class.
4. The virtual keyword calls the most derived version of the method in an inherited class. This concept is Polymorphism, which covers the overloading and overriding of methods in inherited classes.
5. You would overload a comparison operator if you need to compare custom objects.

Section 4: Version control & git

1. A branch is a copy of the current or previous version of the master branch. You would work on a branch in order to always have working code in the master branch.
2.
   ```
   git checkout master
   git fetch origin
   git merge origin/master
   git checkout my-fabulous-feature
   git merge master
   ```
3. A git pull pulls the remote branch of the local branch you are in. A pull request is for merging a branch with the master branch.
4. Is the code readable. Does it compile or run as it should. Is it adequately tested.

Section 5:

1. Run it in a terminal and compile it by hand.

Section 6:

1. A TEST_CASE is a test on a certain part of a code's functionality, while a SECTION tests a specific aspect of the functionality. A TEST_CASE should be testing something like a method of a class. While a SECTION would test each edge case the method may have. I would say the number of methods tested should be held to one but if two methods have similar inputs and outputs they can be put into one TEST_CASE. Each section should be testing all the aspects of one method/function.

Section 7:

1.

a. Singleton: Prevents multiple objects of a class from being created. For instance, only one map should be generated in a game. They are implemented by removing their copying abilities and making their constructor private and calling the constructor with a static method.

b. Flyweight: Reduces the number of large memory objects being used. In the case of a game having a bunch of images of trees can be reduced to using one image of a tree and reusing it for each tree object. It can be implemented by having a method return an existing object of that type or a new object that hasn't if it hasn't been created yet.

c. Prototype: Reduces the number class need to be rewritten by using inheritance. If a class is using most of the functions of another class it should inherit from the first one should inherit from the second.

d. Factory: Gives a user given options for creating a class object. A factory would be another class that is given an input that calls the main class with pre-defined parameters.

e. Iterator: Allows to quickly access a value that is in a data structure in a loop. To implement it on a class that has a data sturucture, you simply redefine the begin() end() iterator and const_iterator.

2. Since these are complicated to implement and there are useful libraries available.

Section 8:

1.
   ```
   template <typename T>
   void Swap(T & a, T & b){
   T temp = a;
   a = b;
   b = temp;
   }
   ```

2.
   ```
   int main(){
      Swap(1,3); // works
      Swap(2.3,2.9); // works
      // Swap(1,2.3); // doesn’t work
      // Swap(7.3,-1); // doesn’t work
   }
   ```

3.
   ```
   Swap((double)1,2.3);
   Swap(7.3,(double)-1);
   ```
4. You wouldn’t want to write all your functions as templates since templates are not as efficient as regular functions.

Section 9:

1. When a user interacts with a component the component sends a signal to a slot to perform a function.
2. As many as needed for both.

3.
   ```
   void testSignal(string out);
   void testSlot(string out);
   connect(button, SIGNAL(testSignal(out)),button,SLOT(testSlot(out)));
   ```
   you would call emit in the slot for when that slot is activated the signal is sent and whatever it does is done