**Project A:**

1. This program first prints a line of 10 uninitialized integers, thus printing out garbage values from whatever is stored in memory at those addresses. It then generates and prints 10 random positive or negative integers. It compiled and ran correctly the first time.
2. I changed the array of integers to an array of doubles, and now it randomly generates 10 double values instead of integer values.

**Project B:**

**//Prototypes:**

**double ftoc(double);**

**double ctof(double);**

**int main() {**

**double temperaturesInitialCelsius[5] = {39.5, 10.8, 22.6, 18.9, 15.7};**

**double temperaturesFahrenheit[5];**

**double temperaturesFinalCelsius[5];**

**cout << "\nPrinting Temperatures in Celsius: " << endl;**

**for(int i = 0; i < 5; i++)**

**cout << temperaturesInitialCelsius[i] << " ";**

**cout << "\nConverting to Fahrenheit..." << endl;**

**for(int i = 0; i < 5; i++)**

**temperaturesFahrenheit[i] = ftoc(temperaturesInitialCelsius[i]);**

**cout << "\nPrinting Temperatures in Fahrenheit: " << endl;**

**for(int i = 0; i < 5; i++)**

**cout << temperaturesFahrenheit[i] << " ";**

**cout <<"\nConverting back to Celsius..." << endl;**

**for(int i = 0; i < 5; i++)**

**temperaturesFinalCelsius[i] = ctof(temperaturesFahrenheit[i]);**

**cout << "\nPrinting Temperatures in Celsius Again: " << endl;**

**for(int i = 0; i < 5; i++)**

**cout << temperaturesFinalCelsius[i] << " ";**

**return 0;**

**}**

**double ftoc(double temperature){**

**return (temperature \* 1.8) + 32;**

**}**

**double ctof(double temperature){**

**return (temperature - 32) \* (5.0/9.0);**

**}**

**Project C:**

**2.** No, the variable X has no meaning outside this function. No, the X in main is not changed.

**3.** The value in the argument, x, is not changed by the function at all.

**4.** This does work. We can use the second add1 function by specifying 2 parameters, as that matches the signature of the second add1 function : add1(int p, int q). This is via overloading.

**5.** This program does not compile because when you call add1(int), the compiler cannot differentiate between add1(int p) and add1(int& p), therefore you get an error. It shows up when the function is called.

**6.** The program now compiles and works (and changes the input parameter value). This is because you cannot have two signatures of a function that seem exactly the same in terms of calling. ie. Add1(int p) and add1(int& p). It does not know which one to use in that case.

**7.** This function works differently from the others because it is “pass by reference” versus “pass by value”. What this means is that you are passing in the memory address of the value you put in instead of just a copy of the value itself – therefore if you modify that value within the function, you are actually modifying the value in memory, instead of modifying a temporary copy of the value. The function does actually modify the original input value because of the p++ statement, and the fact that your signature is “**int&** p”, which I believe essentially means to pass the “pointer” to p in memory instead of to make a copy of the value of p.