1. This code does not work because in a and b are not pointers. This method will not change the values that get passed into the function’s parameters, due to the nature of c being pass by value. To fix this, change “a” and “b” to a pointer, and when assigning their values, and in doing so will change the value in the same memory address as what was passed in the functions parameters. We would also have to make temp an integer, and not a pointer because when we make temp a pointer and make it equal to \*a, we are saying that temp is also pointing to the same memory location, and has the same data as “\*a” so when we update the value of \*a,temp will also have that same updated value. When assigning \*b to temp, it will also have the updated data from \*a. Thus, making it a regular integer prevents that scenario. Working code is below.

void swap(int \*a, int \*b){

int temp = \*a;

\*a = \*b;

\*b = temp;

}

2.

If we know we are copying the string, and we know the length of it; we should specify explicitly that exact amount in the string we are copying to. After we are finish copying hello, the null terminator doesn't get added on to copy. We accomplish this by assigning the last character of the copy array to the null terminating string ‘\0’; due to the fact this code snippet is not a method, I feel it is appropriate to assign copy[5] instead of making int i = 0 outside the for loop and then assigning copy[i] = ‘\0’

char copy [6];

const char\* string = "hello";

for (int i = 0; string [i] != '\0' && i <7; ++i){

copy [i] = string[i];

}

copy[5] = '\0';

printf("%s\n", copy);

3.Suppose you have a function that takes an array as an argument in the usual way, which is

declared as:

void f(int\* array, int length);

Suppose you also have an array declared as:

int a[128];

How would you pass a sub-array containing only the third through sixth elements of

At o the function f?

f(a+2, 4);

4.returning array on the stack causes it to go out of scope when it gets returned. To solve this problem, we save the array on the heap. To do this we, do: double \*array = new double [length];

double\* allocateArray(int length) {

double \*array = new double [length];

return array;

}

5. The last char in a string needs to be a null terminator. Thus, we need to allocate one more element in the string array, and add a null terminator as the last character.

char string[6];

string[0] = ’h’;

string[1] = ’e’;

string[2] = ’l’;

string[3] = ’l’;

string[4] = ’o’;

string[5] = ’\0’;

printf("%s\n", string);