1a.

int main()

{

int arr[3] = { 5, 10, 15 };

int\* ptr = arr;

\*ptr = 10;

\*(ptr + 1) = 20;

ptr += 2;

ptr[0] = 30;

while (ptr >= arr)

{

cout << ‘ ‘ << \*ptr;

ptr--;

}

cout << endl;

}

1b. The function would not do what it is supposed to because the pointer p was passed by

value, thus modifying it within the function findDisorder, would not modify ptr in the main

function. We fix this by passing ptr by reference in which p is a reference to ptr.

void findDisorder(int arr[], int n, int\*& p)

{

for (int k = 1; k < n; k++)

{

if (arr[k] < arr[k-1])

{

p = arr + k;

return;

}

}

p = nullptr;

}

int main()

{

int nums[6] = { 10, 20, 30, 40, 50 };

int\* ptr;

findDisorder(nums, 6, ptr);

if (ptr == nullptr)

cout << “The array is ordered” << endl;

else

{

cout << “The disorder is at address “ << ptr << endl;

cout << “It’s at position “ << ptr - nums << endl;

cout << “The item’s value is “ << \*ptr << endl;

}

}

1c. The program may not work because passing the pointer p as a parameter for the

function will only have resultPtr = p, but both pointers do not contain a memory address,

thus when assigning \*resultPtr would result in an error. We fix this by changing p to a

double, and passing the address of p as a parameter, thus resultPtr would now point at

the address of p and calling \*resultPtr would modify p.

#include <iostream>

#include <cmath>

using namespace std;

void hypotenuse(double leg1, double leg2, double\* resultPtr)

{  
 \*resultPtr = sqrt(leg1\*leg1 + leg2\*leg2);

}

int main()  
{  
 double p;  
 hypotenuse(1.5, 2.0, &p);  
 cout << "The hypotenuse is " << p << endl;  
 }

1d. The problem with the implementation is that the loop is not iterating to look at each

character of the c string because str1 only refers to array position 0, and calling str1++

does not iterate to position 1.

// return true if two C strings are equal

bool match(const char str1[], const char str2[])  
{

int pos = 0;  
 while ( \*(str1+pos) != 0 && \*(str2+pos) != 0) // zero bytes at ends  
 {

if ( \*(str1+pos) != \*(str2+pos) ) // compare corresponding characters  
 return false;  
 pos++; // advance to the next character  
 }  
 return true; // both ended at same time  
}  
  
 int main()  
 {  
 char a[10] = "pointed";  
 char b[10] = "pointer";  
  
 if (match(a,b))  
 cout << "They're the same!\n";  
 }

1e. This program is declaring an array and setting values into each element of the array

within separate functions, but the array exists only within the scope of that

function. Thus, this program is trying to follow the pointer ptr, but ptr does not point to

the array declared in computeSquares because it does not exist within the scope of the

main function and prints out garbage values instead.

2.

1. string\* fp;
2. string fish[5];
3. fp = fish + 4;
4. \*fp = “yellowtail”;
5. \*(fish + 3) = “salmon”;
6. fp -= 3;
7. fp[1] = “trout”;
8. fp[0] = “eel”;
9. bool d = ( fp == fish );
10. bool b = ( \*fp == \*(fp + 1) );

3a.

double computeAverage(const double\* scores, int nScores)

{

int k = 0;

double tot = 0;

while (k < nScores)

{

tot += \*(scores + k);

k++;

}

return tot/nScores;

}

3b.

// This function searches through str for the character chr.

// If the chr is found, it returns a pointer into str where

// the character was first found, otherwise nullptr (not found).

const char\* findTheChar(const char\* str, char chr)

{

for (int k = 0; \*(str + k) != 0; k++)

if (\*(str + k) == chr)

return str + k;

return nullptr;

}

3c.

const char\* findTheChar(const char\* str, char chr)

{

for(; \*(str) != 0; str++)

if (\*(str) == chr)

return str;

return nullptr;

}

4.

diff=1

4

79

5

9

-1

19

This program prints diff=1 because after calling minimart(array, &array[2]), ptr points to position 2 in array and then we did ptr += 2, thus ptr now points to position 4. By pointer arithmetic, &array[5] - ptr which is &array[4] gives us 5 - 4 = 1.

Following the order of the main function, ptr[1], which is array[3], is changed to the value 9. We incremented the ptr by 2, which is now at array[4] and we changed its value to -1. Then by calling \*(array + 1) = 79, we changed the value at array[1] to 79. When we called swap1, this had no effect on the elements of the array because only the pointers were being swapped, not the elements of the array. Then by calling swap2, we swapped value of the elements at array[0] and array[2]. Now our array has elements { 4, 79, 5, 9, -1, 19 } and we cout each element in order.

5.

void deleteG(char\* c)

{

while(\*c != 0)

{

if (\*c == 'g' || \*c == 'G')

{

for (char\* pos = c; \*pos != 0; pos++)

{

\*pos = \*(pos + 1);

}

}

c++;

}

}