1a.

int main()

{

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

int\* ptr = arr;

\*ptr = 10; // set arr[0] to 30 // need to swap this with the 30 pointer assignment

\*(ptr + 1) = 20; // set arr[1] to 20

ptr += 2;

ptr[0] = 30; // set arr[2] to 10 //need to swap this with the first pointer assignment

while (ptr >= arr)

{

// ptr--; // Error here

cout << \*ptr << endl; // print values

ptr--;

}

}

1b. This was fixed by making the pointer also a reference so that when you need to modify the pointer in the function the pointer in the main function is also changed. To fix this I put a ‘&’ in the parameter list of the function.

int main()

{

int nums[4] = { 5, 3, 15, 6 };

int\* ptr;

findMax(nums, 4, ptr);

cout << "The maximum is at address " << ptr << endl;

cout << "It's at position " << ptr - nums << endl;

cout << "Its value is " << \*ptr << endl;

}

void findMax(int arr[], int n, int\*& pToMax)

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

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

{

if (arr[i] > \*pToMax){

pToMax = arr + i;

cout << \*pToMax << endl;

}

}

}

1c. This function does not work becaseu the ‘ptr’ pointer points to nothing so nothing will be stored in it. So we must create a location for it to point to. I created an empty array for the pointer to point to.

void computeCube(int n, int\* ncubed)

{

\*ncubed = n \* n \* n;

}

int main()

{

int someNonNull[0];

int\* ptr = someNonNull;

computeCube(5, ptr);

cout << "Five cubed is " << \*ptr << endl;

}

1d. The issues were that the characters were not being compared. You need to make sure that the characters at each location are the same. So change the values for the two conditionals to ‘\*str1’ and ‘\*str2’ to make sure that they are checking the character value and not the address.

bool strequal(const char str1[], const char str2[])

{

while (\*str1 != 0 || \*str2 != 0)

{

if (\*str1 != \*str2){ // compare corresponding characters

return false;

}

str1++; // advance to the next character

str2++;

}

return true; // the ended the same and both reached 0 together

}

1e. The program is assuming that the values in the function are global and not local. The local value will be erased after the function is finished running. This program is doing pointers incorrectly by assuming that the values created locally will last forever.

Once the function has run once the values saved locally are erased. So this will maybe print a correct value for the first printed value, but after that everything will be junk because the function’s variables will have been erased. When we run the program, the value printed is the first value of the “junk” array. It is immediately erased because local variables are erased and then junk is printed.

2.

int main(){

double\* cat;

double mouse[5];

cat = mouse+4; // cat is now at location 4 (element 5)

\*cat = 25; // the value at location 4 is now 25

printArray(mouse, 5);

\*(mouse+3) = 42; // the value at location 3 is now 42

cat -= 3; // cat is now pointing to location 1 (element 2)

printArray(mouse, 5);

cat[1] = 54; // the value at location 2 (element 3) is now 54

printArray(mouse, 5);

cat[0] = 27; // the value at location 1 (element 2) is now

printArray(mouse, 5);

bool b = \*cat == \*(cat+1);

bool d = cat == mouse;

cout << "b is: " << b << endl;

cout << "d is: " << d << endl;

printArray(mouse, 5);

}

3a.

double mean(const double\* scores, int numScores)

{

const double\* ptr = scores;

double tot = 0;

int i = 0;

while ((ptr+i) != scores + numScores)

{

tot += \*(ptr+i);

i++;

}

return tot/numScores;

}

3b.

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

{

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

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

return &\*(str+k);

return nullptr;

}

3c.

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

{

while(\*str != 0){

if(\*str == chr)

return &\*(str);

str++;

}

return nullptr;

}

4.

the program prints the following as output

3 // this is printed because the first thing printed is the difference of &array[5] and the ptr which is 3 less than the value of &array[5] because ptr points to the value of array[2]

4 // this is printed because we swapped the values of array[0] and array[2]

79 // this is printed because we made the value at array[1] 79 with the line “\*(array+1) = 79;”

-1 // this is printed because we made the value at array[0] = -1 (this line: “\*ptr = -1;“, and the line before it to assign ptr to array[0]) and then swapped it with the value of array [2]

9 // this is printed because we made the value at array [3] = 9 with the lines: ptr += 2; and ptr[1] = 9)

22 // this location was never changed so we print regularly

19 // this location was never changed so we print regularly

5.

void removeS(char\* msg){

int i = 0;

while(\*(msg+i) != 0){

if(\*(msg+i) == 's' || \*(msg+i) == 'S'){

for(int j = i; \*(msg+j) != 0; j++){

cout << msg << endl;

\*(msg+j) = \*(msg+j+1);

}

i--;

}

i++;

}

}