1) A.

int main() {

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

int\* ptr = arr;

\*ptr = 10;

\*ptr + 1 = 20; // bug 1: \*ptr + 1 is a pointer, not an int or

// double

ptr += 2;

ptr[0] = 30;

while (ptr >= arr) {

ptr--;

cout << ' ' << \*ptr; // bug 2: order of

//statements is switched

}

cout << endl;

}

correct version:

int main()

{

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

int\* ptr = arr;

\*ptr = 10; // set arr[0] to 10

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

ptr += 2;

ptr[0] = 30; // set arr[2] to 30

while (ptr >= arr)

{

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

ptr--;

}

cout << endl;

}

B. As is, the function passes the pointer ptr by value, not by reference. To fix it one need only change the function signature to:

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

c. P isn’t initialized in the main function, so when the hypotenuse function tries to access \*p, it leads to undefined behavior. This is the revised main function:

int main()

{

double result = 0;

double\* p = &result;

hypotenuse(1.5, 2.0, p); // for some reason doesn't change p?

cout << "The hypotenuse is " << \*p << endl;

}

D. The condition for the while loop compares pointers with integers, which is invalid. The if condition and final return statement also compare two pointers instead of the characters they point to. A correct implementation of the function:

// return true if two C strings are equal

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

{

while (\*str1 != 0 && \*str2 != 0) // zero bytes at ends

{

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

return false;

str1++; // advance to the next character

str2++;

}

return strlen(str1) == strlen(str2); // both ended at same time?

}

E. In the main function, ptr tries to point to an array that only exists in the environent of the computeSquares function.

2)

a. string\* fp;

b. string fish[5];

c. fp = &fish[4]; // fp[0] = fish[4], fp[-4] = fp[0]

d. \*fp = “yellowtail”;

e. \*(fish + 3) = “salmon”;

f. fp -= 3;

g. fp[1] = “perch”; // check on xcode

h. fish[(fp-fish)] = “eel”;

i. bool d = (fp == fish);

j. bool b = (\*fp == \*(fp + 1));

3)

a.

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

{

double tot = 0;

int k = 0;

while (k < nScores)

{

tot += \*(scores+k);

k++;

}

return tot/nScores;

}

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

{

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

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

return str+k;

return nullptr;

}

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

{

while (\*str != 0)

{

if (\*str == chr)

return str;

else

str++;

}

return nullptr;

}

4. It would print out:

diff=1 // ptr is initialized to point to the 2nd element in the

//array, then it’s shifted 2 spaces so ptr=&array[4]

4 // swap2 changes the value of array[0] from 5 to 4

79 // \*(array+1) = 79 reassigns the value of array[1]

5 // swap2 changes the value of array[2] from 4 to 5

9 // ptr[1] = 9 reassigns value of ptr[1], or arr[3]

-1 // \*ptr = -1 changes value of arr[4], since ptr was

initialized to &arr[2] and was later incremented by 2

19 // arr[5] is not changed by the program

5.

void deleteG(char\* ptr)

{

// if an element is G or g, shift every element following it

// one space to the left

// also, the test case for this made me chuckle

while (\*ptr != '\0')

{

if (tolower(\*ptr) == 'g')

{

for (int i = 0; \*(ptr+i) != '\0'; i++)

{

\*(ptr+i) = \*(ptr+i+1);

}

}

else

ptr++;

}

}