1a.

\*ptr + 1 should be \*(ptr+1) to modify the next element in the array.

Ptr--; should be after the cout line.

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.

findDisorder has a defective parameter of int\* p which is a pointer passed by value. Instead, for the pointer to be passed by reference and have the changes of the void function carry over to the main function, the parameter must be set to int\*& p. Without this change, the modification made to p from void function would not have any effect on ptr from main function.

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;

}

c.

double\*p; declares a pointer when the function intends on generating a value (object), not an address. double\*p; must be changed to double p; to declare a double, and the hypotenuse must pass in the address of that double by typing &p instead of p. Lastly, the main function should print p as \*p implies p was a pointer.

int main()

{

double p;

hypotenuse(1.5, 2.0, &p);

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

}

d.

The problem is the assumption that you are able to compare the entirety of cstrings to one another, and that you are able to increment or compare elements without specifying position.

Use forloop instead of whileloop to encompass whether the loop reached a null byte in either strings, and the forloop also encompasses the increment aspect of the string since you can’t increment entire string by a statement like str1++;. The parameter must be str1[i] != ‘\0’ instead of str1 != ‘\0’ which compares the entire cstring to see if it was just consisting of the nullbyte. Additionally, the if statement and return statement, the comparison must be (str1[i] != str2[i]) and str1[i]==str2[i] respectively which compares the strings in the ith position since str1!=str2 won’t do what you intend (compare memory location).

An alternative fix to this code would be to add \* to the beginning of all instances of str1 and str2 from the initial code (excluding the parameters). This fix would change comparisons of addresses to comparisons of objects at those addresses.

// return true if two C strings are equal

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

{

for(int i = 0; str1[i] != '\0' || str2[i] != '\0';i++)

{

if (str1[i] != str2[i]) // compare corresponding characters

return false;

}

return str1[i]==str2[i]; // both ended at same time?

}

e.

The function tried to modify the cstring arr (in a pointer function) and have the pointer ptr (in main function) point to elements in arr. However, the pointer function returns arr which is position 0 of arr.

The array arr[10] is only a local array which disappears after the function computeSquares is complete. Thus, the pointer in the main function simply can’t access arr[10].

2a. string\* fp;

b. string fish[5];

c. fp = &fish[4];

d. \*fp = “yellowtail”;

e. \*(fish + 3) = “salmon”;

f. fp=fp-3;

g. fp[1]=”basa”;

h. fp[0]=”sole”;

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

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

3a.

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

{

const double\* ptr = scores;

double tot = 0;

int i = 0;

while (i!=nScores)

{

tot += \*(ptr+i);

i++;

}

return tot / nScores;

}

b.

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;

}

c.

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

{

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

if (\*str == chr)

return str;

return nullptr;

}

4.

int main()

{

int array[6] = { 5, 3, 4, 17, 22, 19 };

int\* ptr = minimart(array, &array[2]); //initializes pointer to position 2 of array

//since value of element at &array[0] >

//&array[2]

ptr[1] = 9; //pointer in next position is set to 9 (element 2 of array)

ptr += 2; //pointer moves 2 places forward to position 2 (3rd element)

\*ptr = -1; //pointer at position 2(element 3) is set to value -1

\*(array + 1) = 79; //object at position 1 of array (element 2) is set to value 79

cout << "diff=" << &array[5] - ptr << endl; //prints "diff=1" since ptr was left at

//position 2. Thus (by rule of increasing consecutive

//order), position 5 - position 4 = 1.

swap1(&array[0], &array[1]); //swaps pointers at position 0 and 1 at the array with

//each other. Now &array[0] points to 3 and &array[1]

//points to 5

swap2(array, &array[2]); //swaps element at &array[0] with element at &array[2].

//Since &array[0] points to 3 (position 1), 3 and 4 are

//swapped in the array.

for (int i = 0; i < 6; i++) //cycles through first 6 elements in the for-loop.

cout << array[i] << endl; //prints "4 79 5 9 -1 19"(‘\n’ after each element)

}

//chronologically, the list transitions as follows:

//5 3 4 17 22 19 ptr at position 2

//5 3 4 9 22 19

//5 3 4 9 -1 19 ptr at position 4

//5 79 4 9 -1 19 pointer at pos 0 and 1 swapped

//4 79 5 9 -1 19 object at pos 0 and 2 swapped

5.

void deleteG(char\* ptr) {

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

if (\*ptr == 'g' || \*ptr == 'G') {

char\* a = ptr;

while (\*(a + 1) != '\0') {

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

a++;

}

\*a = '\0';

ptr--;

}

ptr++;

}

}