1.a.

for the line \*ptr + 1 = 20;, the expression is not assignable because it’s basically assigning the value 20 to the sum of the value of the pointer of the first element and 1, which doesn’t make sense. Also, one should output the value of the pointer before changing the pointer (I’m referring to the code in the while loop)

Here’s one way to fix it:

;

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;

ptr--;

}

cout << endl;

}

1.b.

The pointer ptr gets reset every time it goes from the void function into the main function according to the debugger (not passed by reference). Therefore, I added a reference symbol in the parameters in the void function.

The function now looks like this:

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, 20, 40, 30, 50 };

int\* ptr;

findDisorder(nums, 6, ptr);

if (ptr == NULL)

cout << "The array is ordered" << endl;

else

{

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

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

cout << "The item's value is " << \*ptr << endl;

}

}

1.c

The pointer in the main function is initialized without any memory address. Hence, the pointer doesn’t point to anything. And since its type is double, so I added the line double x; p=&x to assign the address in x to the pointer 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;

double x;

p=&x;

hypotenuse(1.5, 2.0, p);

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

}

1.d.

The first problem with this function is that even though it’s a Boolean function, the program in spec still returns str1==str2 which is wrong for Boolean type of function. Therefore I changed it to true. The other problem is that str1 != str2 does not compare the corresponding character. To compare the character, one have to add a ‘\*’ to indicate its value.

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 true; // both ended at same time?

}

int main()

{

char a[10] = "Lien";

char b[10] = "Lin";

if (match(a,b))

cout << "They're the same guy!\n";

}

1.e.

One of the problem with this program is that, in the first function, the returns a pointer which only points to the first element of the array. In addition. once f() is run, the operating system puts value in the array’s memory location and therefore would change the value output by the function.

2.

a. string \*fp;

b. string fish[5];

c. fp=&fish[4];

d. \*fp="yellowtail";

e. \*(fish+3)="salmon";

f. fp-=3;

g. \*(fp+1)="tuna";

h. fp[0]="eel";

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

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

3.

**a.**

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

{

const double\* ptr = scores;

int adder=0;

double tot = 0;

for (adder; adder<nScores; adder++)

{

tot +=\*(ptr+adder);

}

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 NULL;

}

**c.**

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

{

return strchr(str, chr);

}

4.

The program outputs the following:

**diff=1 ---1**

**4 ---2**

**79 ---3**

**5 ---4**

**9 ---5**

**-1 ---6**

**19 ---7**

For the first line of output, the “diff= “ is followed after cout, and the ‘1’ comes from &array[5] – ptr.

Let me discuss how this function works:

For the first line, it begins with a declaration of the an integer array {5,3,4,17,22,19}.

For the next line, it initialized a pointer and the pointer points to the result of the minimart function.

Minimart function returns the pointer that points to the smallest value comparing to the other one. Since array is array[0] and the value is 5, and array[2] the value is 4, so array[2] is the pointer ptr points to. Therefore, the pointer points to the third element of the array. Hence, the third element of the array is now set to 4.

Since the pointer points to the second value of the array, ptr[1]=9 sets the forth value of the array to 9.

And ptr++ change the pointer location to point to the 5th element of the array. And the next line \*ptr=-1 sets the fourth element of the array to -1.

\*(array+1) is a pointer and it points to the second element of the array (with an index of 1). And \*(array+1)=79, sets the second element’s value to 79.

Therefore, before the diff part, the pointer points to the fifth value of the array and the &array[5] is the location of storage for the sixth element of array.

According to IBM:

If you have two pointers that point to the same array, you can subtract one pointer from the other. This operation yields the number of elements in the array that separate the two addresses that the pointers refer to.

Therefore, the since the two pointer differs by one element, the value of diff certainly is 1.

Now the array is {5, 79 , 4, 9, -1, 19}

Then the program calls swap1, which swapped the memory location and value of the two pointer, &array[0] and &array[1]. However, at the end of the function, it did not pass by reference and therefore this function does nothing.

Then the function calls swap2, which swaps the value of the pointer, but unlike swap1, it passes by reference in the end and therefore it swaps the value of array[0] and array[2]. Then the result we get is:

{4, 79 , 5, 9, -1, 19}

The last part of the for loop prints out the entire array.

5.

void deleteG(char \*c)

{

char \*ptr=c;

while (\*c!=0)

{

if (toupper(\*c)=='G')

{

while (\*c!=0)

{

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

c++;

}

c=ptr; //goes back to the starting location.

}

c++;

}

}