1. a.

int main()

{

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

int\* ptr = arr;

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

ptr++;

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

ptr++;

\*ptr = 30; // set arr[2] to 30

while (ptr >= arr)

{

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

ptr--;

}

cout << endl;

}

b.

The function doesn’t work because the main routine passes a copy of the pointer ptr to the function. Although the pointer is changed in the function, it is not returned back to the main routine. To fix this, add a reference in the declaration of the pointer p in the function title.

Fixed 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.

The main routine does not work because the pointer p is used before initialization. Since the compiler does not know where p points to, it is considered bad access when you try to store something in the position where p points to. To fix this, create a variable result of type double (double result). Then initialize the pointer to point to result.

Fixed main routine:

int main()

{

double result = 0;

double\* p = &result;

hypotenuse(1.5, 2.0, p);

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

}

d.

The function does not work because it constantly run into the issue that two pointers are compared. Since they are declared to be different pointers, comparing them would always return false. To fix this, add “\*” in front of every pointer to indicate that the pointed values are compared.

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 \*str1 == \*str2; // both ended at same time?

}

e. The problem with the program is that the function returns a local variable. The returned pointer points to the first element in the array arr. However, when the program leaves the function, the arr variable no longer exists. As a result, the program prints random values from the computer memory instead of what’s in the array arr.

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] = “pike”;

h. fp[0] = “basa”;

i. bool d = (fp == &fish[0]);

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

3. a.

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

{

const double\* ptr = scores;

double tot = 0;

for (int k = 0; k < nScores; k++)

{

tot += \*(ptr+k);

}

return tot/nScores;

}

b.

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

{

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

if (\*str == chr)

return str;

}

return nullptr;

}

4.

#include <iostream>

using namespace std;

int\* minimart(int\* a, int\* b) //returns the address of the smaller //value between array and array[2]

{

if (\*a < \*b)

return a;

else

return b;

}

void swap1(int\* a, int \*b) //swap the addresses of a and b, but //does nothing to the original array

{

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int \*b) //swap the values that a and b point to

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = minimart(array, &array[2]); //prt = &array[2]

ptr[1] = 9; //array[3] = 9

ptr += 2; //prt = &array[4]

\*ptr = -1; //array[4] = -1

\*(array+1) = 79; //array[1] = 79

cout << "diff=" << &array[5] - ptr << endl;

swap1(&array[0], &array[1]); //array does not change

swap2(array, &array[2]); //array[6] = { 4, 79, 5, 9, -1, 19 }

for (int i = 0; i < 6; i++) //prints every element of array

cout << array[i] << endl;

}

Now:

array[6] = {4, 79, 5, 9, -1, 19}

ptr = &array[4]

Output:

diff=1 //&array[5] – ptr = &array[5] - &array[4] = 5 – 4 = 1

4 //first element of array is 4

79 //second element of array is 79

5 //third element of array is 5

9 //fourth element of array is 9

-1 //fifth element of array is -1

19 //sixth element of array is 19

5.

void deleteG(char\* ch){

char\* result = ch;

for (; \*ch != 0; ch++){

if (\*ch != ‘g’ && \*ch != ‘G’){

\*result = \*ch;

result++;

}

}

\*result = 0;

}