1a.

int main()

{

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

int\* ptr;

ptr = arr; //set ptr to arr instead of \*ptr

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

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

ptr += 2;

\*ptr = 30; // set arr[2] to 30 //do \*ptr instead of ptr[0]

while (ptr >= arr)

{

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

ptr--; //print value first before reducing the index value

}

cout << endl;

}

1b. The reason why the function didn’t work is because that it was passed by value instead of passing by reference to the main function. We fix it by adding & to int\*p to give int\* &p.

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 == nullptr)

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

else

{

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

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

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

}

}

1c. The function didn’t work because pointer p was not actually pointing to anything. Hence, I changed it to double p and changed the parameter p when calling the function to &p so that it points to p. When printing out the value of the hypotenuse, I also changed \*p to 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;

hypotenuse(1.5, 2.0, &p); //&p indicates pointer to p ie. address of p

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

}

1d. The addresses of the str1 and str2 are being compared rather than their values. Hence, I added \* to str and str2 in the while loop statement, if loop statement, and the return statement so that the values that str1 and str2 are pointing to 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?

}

int main()

{

char a[10] = "pointy";

char b[10] = "pointless";

if (match(a, b))

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

}

1e. The main function tries to access the output of the computeSquares function using a pointer to point to the elements in the array. However, the array does not exist outside of the computeSquares function since it is a local variable. Hence, using it in the main function gives random numbers.

2)

1. string\* fp;
2. string fish[5];
3. fp = &fish[4];
4. \*fp = "yellowtail";
5. \*(fish + 3) = "salmon";
6. fp -= 3;
7. fp[3] = "basa";
8. fp[0] = "sole";
9. bool d = (fp == fish);
10. bool b = (\*fp == \*(fp + 1));

3a.

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

{

double tot = 0;

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

{

tot = tot + \*(scores + count);

}

return tot / nScores;

}

3b.

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;

}

3c.

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

{

while (\*str != 0) {

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

4.

In the main function:

* Declares an array with 6 integers { 5, 3, 4, 17, 22, 19 };
* Declares pointer ptr and passes pointer to the first(5) and third (4) element of array to the minimart function. Minimart compares the two values (5 and 4) and returns the smaller value 4. Pointer now points to third element of array.
* (Ptr + 1) now points to fourth element in array and changes 17 to 9. array = { 5, 3, 4, 9, 22, 19 };
* Ptr + 2 means pointer advances two places down.
* Changes fifth element from 22 to -1. array[6] = { 5, 3, 4, 9, -1, 19 };
* \*(array + 1) points to second element of array and changes it from 3 to 79. array[6] = { 5, 79, 4, 9, -1, 19 };
* Prints the difference between address of the sixth element in the array and the pointer that contains the address of the fifth element. Since it’s just one position down, the difference is 1.
* Passes addresses of the first and second element to function swap1. Since a and b in swap is being passed by value, they do not affect the elements in the array.
* Passes addresses of the first and third element to function swap2. Location of 5 and 4 is swapped, which gives array[6] = { 4, 79, 5, 9, -1, 19 };
* Create a for loop to loop through each element in the array and prints the element out one by one, which gives:
* 4
* 79
* 5
* 9
* -1
* 19

5.

void deleteG(char\* cstring) {

if (NULL == cstring) {

return;

}

char\* temp = cstring;

while (\*cstring) {

if (\*cstring != 'G' && \*cstring != 'g') {

\*temp++ = \*cstring;

}

cstring++;

}

\*temp = '\0';

}

int main()

{

char msg[100] = "I recall the glass gate next to Gus in Lagos, near the gold bridge.";

deleteG(msg);

cout << msg; // prints I recall the lass ate next to us in Laos, near the old bride.

}