1a.

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;

}

1b.

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

{

if(n<=0)

return;

for (int k = 1; k < n; k++)

{

if (arr[k] < arr[k-1])

{

p = arr + k;

return;

}

}

p = nullptr;

}

**The problem with this function was that the pointer was not passed by reference and just returned the random pointer’s address, index, and value that was initialized in the main function.**

1c.

void hypotenuse(double leg1, double leg2, double\* resultPtr)

{

\*resultPtr = sqrt(leg1\*leg1 + leg2\*leg2);

}

int main()

{

double\* p = new double;

hypotenuse(1.5, 2.0, p);

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

}

**OR**

int main()

{

double x = 0.0;

double \*p = &x;

hypotenuse(1.5, 2.0, p);

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

}

**The problem was that the initialized pointer in the main function had no empty memory space to hold the result of the function.**

1d.

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;

if (\*str1 == \*str2) {

str1++; // advance to the next character

str2++;

}

}

return true; // 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";

}

**The problem here was that the original function returned if str1 ==str2, which is always false because they are 2 different set places in memory. Also, what the function was comparing was str1 and str2’s elements’ memory location, not their actual char values.**

1e.

**The problem is that the computeSquares returns a pointer to an int, but instead, the function is trying to return the whole array of elements. Furthermore, ptr is thus only initialized to the FIRST element of this array and all of the rest of arr’s values are ignored. This might cause the compiler to allocate the rest of the array’s un-pointed elements’ memory addresses to be allocated to some other values(such as those of the array junk).**

2a. string\* fp;

2b. string fish[5];

2c. fp = fish + 4;

2d. \*fp = “yellowtail”;

2e. \*(fish+3) = “salmon”;

2f. fp-=3;

2g. fp[2] = “cod”;

2h. fp[0] = “eel”;

2i. bool d = (fp==fish);

2j. bool b = (\*fp == \*(fp+1));

3a.

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

{

double tot = 0;

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

{

tot += \*(scores+i);

}

return tot/nScores;

}

3b.

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

{

int k = 0;

while(\*(str+k) != '\0'){

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

return (str+k);

}

else{

k++;

}

}

return nullptr;

}

3c.

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

{

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

if (\*(str) == chr){

return (str);

}

else{

str++;

}

}

return nullptr;

}

4.

#include <iostream>

using namespace std;

int\* minimart(int\* a, int\* b)//Declares a function that returns a pointer to an int

{

if (\*a < \*b)//If the value pointed to by a is less than the value b points to

return a;//Return the address of a

else

return b;

}

void swap1(int\* a, int \*b)//Swaps two pointers

{

int\* temp = a;//Initialzes a temporary pointer

a = b;//Make pointer a's address = b's address

b = temp;//Make b = temp address(a)

}

void swap2(int\* a, int \*b)//Swaps values of two pointers

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

int array[6] = { 5, 3, 4, 17, 22, 19 };//Initializes an array of 6 ints

int\* ptr = minimart(array, &array[2]);//Initialize a pointer, ptr, to the array's third element's address(&array[2])

ptr[1] = 9; //Sets the element after the third element's address to 9.

ptr += 2;//Moves to the fourth element of array (&array[4])

\*ptr = -1;//Sets the value of the fourth element to -1

\*(array+1) = 79;//Sets the second element of array (3) to 79.

cout << "diff=" << &array[5] - ptr << endl;//Prints the difference between the blocks of MEMORY of array's fifth element (&array[5]) and ptr, which is now (&array[4]). Since they are 1 index of memory apart, their difference equals 1.

swap1(&array[0], &array[1]);//Swaps the first and second elements' ADDRESSES of array.

swap2(array, &array[2]);//Swaps the first and third element's VALUES of the array.

for (int i = 0; i < 6; i++)

cout << array[i] << endl;//Prints the new array

}

5.

void deleteG(char\* string){

char\* p = string;

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

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

p++;

}else{

\*string = \*p;

string++;

p++;

}

}