1a)

Bugs:

\*ptr+1 = 20; // \*ptr is an int that is initialized to 10, so this is like saying 10+1 = 11 = 20, which is nonsensical.

ptr—should not be before “cout…”. Otherwise, 30 will not be printed out.

Fixed version:

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) Pointer is pass by value as opposed to pass by reference, so changes in function will not show up in main function

void findDisorder(int arr[], int n, int \* &p ) //add ‘&’ before ‘p’

{

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

{

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

{

p = arr + k;

return;

}

}

p = nullptr;

}

1c) Pointer is not initialized. This is undefined behavior.

int main()

{

double\* p ;

double x = 0;

p = &x;

hypotenuse(1.5, 2.0, p);

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

}

1d) str1 != 0 and str2 != 0 mean we’re comparing a full array to a 0 character, which is not possible..

C strings, which are like arrays with characters, cannot be compared with == or != operators. We can only compare character by character within each array.

Also, the “return str1 == str2” is a logic and compiler error. The == operator doesn’t apply for c strings. What if one string is longer than another? The return statement is true, but a longer string is never equal to a shorter string.

We should be comparing the pointers to these characters.

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?

}

1e) arr[10] is not declared within the scope of main function,

//so arr becomes uninitialized after program leaves function computeSquares, and arr has just garbage //values.

2a) string\* fp;

b) string fish[5];

c) fp = &fish[4];

d) \*fp = “yellowtail”;

e) \*(fish + 3) = “salmon”;

f) fp-=3;

g) fp[1]= ”loach”;

h) fp[0] = “eel”;

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

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

//3a) double computeAverage(const double\* scores, int nScores)

{

const double\* ptr = scores; //initialized to beginning

double tot = 0;

int k = 0;

while (k < nScores)

{

tot += \*(ptr+k);

k++;

}

return tot / nScores;

}

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

{

int k = 0;

char\* ptr = str;

for (; \*(ptr + k) != '\0'; k++) //pointer must not point to null character

{

if (\*(ptr + k) == chr)

return ptr + k;

}

return nullptr;

}

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

{

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

{

if (\*str == chr)

return str;

}

return nullptr;

}

//4)

Output:

diff=1

4

79

5

9

-1

19

Explanation:

diff=1 (difference between two pointers by comparing their positions)

The function minimart took two pointers and compared the values they pointed to. The pointer that pointed to a smaller integer was returned, and ptr was initialized to the returned pointer. The program then moved one unit to the right of its initial position and changed the previous value to 9: ptr was at position 2 and then went to position 3 to change 17 to 9. Pointer ptr then went back to its original position.

Program then made ptr shift its starting position two units right so that ptr is at position 4. Program then changed the value at position 4 to -1. \*(array+1) means initial position of array +1, so program changed value at 3 to 79. Difference took the pointer at the last element, which is position 5, and ptr, which was at position 4. Since they are apart by 1 element, the diff = 1. New array order: {5, 79, 4, 9, -1, 19}

4 (first element of array, then newline)

79 (second element of array, then newline)

5 (third element of array, then newline)

9 (fourth element of array, then newline)

-1 (fifth element of array, then newline)

19 (last element of array, then newline)

swap1 switched the pointers pointing at positions 0 and 1 of array, but it did not change the values the pointers pointed to. But swap2 switched the values at position 0 and position 2, so at position 0, new value is 4 and at position 2, new value is 5. Final array order: {4, 79, 5, 9, -1, 19}

Each value was printed until position of element equaled the size of array.

//5) void deleteG(char\* string)

{

char\* ptr = string;

for (; (\*string) != '\0'; string++)

{

if (\*string != 'G' && \*string != 'g')

{

\*ptr = \*string;

ptr++;

}

}

\*ptr = '\0';

}