1.

a.

int main()

{

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

int\* ptr = arr;

\*ptr = 30; // set arr[0] to 30

\*ptr + 1 = 20; // set arr[1] to 20 – invalid syntax

ptr += 2;

ptr[0] = 10; // set arr[2] to 10 – invalid syntax

while (ptr >= arr)

{

ptr--;

cout << \*ptr << endl; // print values – logic error; swap with previous line

}

}

int main()

{

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

int\* ptr = arr;

\*ptr = 30; // set arr[0] to 30

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

ptr += 2;

\*ptr = 10; // set arr[2] to 10

while (ptr >= arr)

{

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

ptr--;

}

}

b.

The function asks for a the pointer pToMax but only the pointer’s value is passed. Instead, the function should ask for a reference to a pointer so it can modify the actual pointer passed. The function header should look like:

void findMax(int arr[], int n, int\* &pToMax)

c.

The pointer ptr isn’t initialized to anything (ie it doesn’t point to anything) so it can’t be dereferenced/nothing can be stored to where it points. An int where the value of n^3 should be declared and the pointer should be initialized to it. The main function should look like:

int main()

{

int cubed;

int\* ptr = &cubed;

computeCube(5, ptr);

cout << "Five cubed is " << \*ptr << endl;

}

d.

The problem with the current implementation is that str1 and str2 are pointers and therefore must be dereferenced to be compared, to have their values checked against 0, and to increment their position. The correct implementation of strequal() is:

bool strequal(const char str1[], const char str2[])

{

while (\*str1 != 0 && \*str2 != 0)

{

if (\*str1 != \*str2) // compare corresponding characters

return false;

str1++; // advance to the next character

str2++;

}

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

}

e.

getPtrToArray() returns a pointer to a local variable created and destroyed within its scope, so outside of the function the array pointed to does not exist.

2.

a.

double \*cat;

b.

double mouse[5];

c.

cat = mouse + 4;

d.

\*cat = 25;

e.

\*(mouse+3) = 42;

f.

cat -= 3;

g.

cat[1] = 27;

h.

cat[0] = 54;

i.

bool b = \*cat == \*(cat + 1);

j.

bool d = cat == mouse;

3.

a.

double mean(const double \*scores, int numScores)

{

double tot = 0;

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

{

tot += \*(scores + i);

}

return tot / numScores;

}

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 nullptr;

}

c.

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

{

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

{

if (\*str == chr)

return str;

}

return nullptr;

}

4.

#include <iostream>

using namespace std;

int \*maxwell(int \*a, int \*b) // returns pointer to greater value pointed to by a or b (if equal, returns b)

{

if (\*a > \*b)

return a;

else

return b;

}

void swap1(int \*a, int \*b) // swaps local pointers but doesn't affect anything outside the function scope

{

int \*temp = a;

a = b;

b = temp;

}

void swap2(int \*a, int \*b) // correctly swaps a and b

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

int array[6] = {5, 3, 4, 17, 22, 19}; // declares and initializes array of ints

int \*ptr = maxwell(array, &array[2]); // declares and initializes a pointer to &array[0]

\*ptr = -1; // sets array[0] to -1

ptr += 2; // moves ptr to &array[2]

ptr[1] = 9; // sets array[3] to 0

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

cout << &array[5] - ptr << endl; // prints &array[5] - &array[2] which is 3 because it's the same as (array + 5) - (array + 2)

swap1(&array[0], &array[1]); // doesn't actually do anything because the function doesn't work

swap2(array, &array[2]); // swaps the values of array[0] and array[2]

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

cout << array[i] << endl; // prints out each value of the array, which is now: 4, 79, -1, 9, 22, 19

}

Output:

3

4

79

-1

9

22

19

5.

void removeS(char \*str)

{

char \*result = str;

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

{

if (\*str != 'S' && \*str != 's')

{

\*result = \*str;

result++;

}

}

\*result = 0;

}