1)

a)

Bugs:

- \*ptr + 1 should be \*(ptr + 1) since an addition cant b done on the left hand side

* in the while loop, the values compared at the beginning are &arr[2] and &arr[0], but since we want ot cout arr[0] first, the while loop should be while(ptr-2 <= arr+ 2
* since we changed the start point for the while loop, the cout must be changed to cout << \*(ptr - 2) << endl;
* Since we are increasing the storage of arr, in the loop it should be ptr++; at the end of the loop

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[0] = 10; // set arr[2] to 10

while (ptr-2 <= arr+2)

{

cout << \*(ptr-2) << endl; // print values

ptr++;

}

}

b) The function won't work because the value of ptr is not being changed when findMax is used. To fix it ptr must be referenced to the value of pToMax. To do this, make findMax the following:

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

With &pToMax changing the value of ptr.

c) The problem is that ptr wasn't initialized to any address, therefore when the value is changed in the function, there is no address to change the value at. To fix this ptr needs to be initialized with a new int.

int\* ptr = new int;

d) The problem is that what is being compared are address of arrays instead of the value of the arrays at the locations. Also for the while loop, it should stop when either string reaches '\0'.

To fix these problems the following needs to be changed:

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

{

while (\*str1 != '\0' && \*str2 != '\0') //Make the while stop when either string gets to '\0'

{

if (\*str1 != \*str2) // compare corresponding characters not addresses

return false;

str1++; // advance to the next character (change address)

str2++;

}

return \*str1 == \*str2; // Both Cstrings ended at '\0' at the same time

}

e) The problem is that ptr is assigned the address of an array in the function, but the array gets destroyed once the function ends. When f() is passed, the residual values of anArray might be replaced with the values of junk. At the end, when the main function reads the address of ptr, it's going to read the junk residual values instead of the intended values.

2)

1. double\* cat; //create cat pointer
2. double mouse[5]; //create five element array
3. cat = (mouse + 4); //cat points to &mouse[4]
4. \*cat = 42; //set the value of mouse[4] to 42
5. \*(mouse + 3) = 25; //set the value of mouse[3] to 25
6. cat -= 3; //cat now points to mouse[1]
7. \*(cat + 1) = 17; //set the value of mouse[2] to 17
8. cat[0] = 54; //cat[0] is the same as \*cat, therefore mouse[1] is set to 54
9. bool b = \*cat == \*(cat + 1); //if where cat is pointing and the following are equal
10. bool d = cat == mouse; //Compare address of cat and mouse[0]

3)

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

{

const double\* ptr = scores;

double tot = 0;

int x = 0;

while ((ptr + x) != scores + numScores)

{

tot += \*(ptr + x);

x++;

}

return tot/numScores;

}

b)

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

{

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

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) For this, I'm gonna pseudocode the main function:

int main(){

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

set ptr address to the biggest between array[0] and array[2], ptr = &array[0];

set array[0] to -1;

set ptr to &array[2];

set array[3] to 9;

set array[1] to 79

cout index difference between &array[5] - &array[2]; **OUTPUT IS 3**;

swap addresses of array[0] with array[1]; array[0] = -1 — array[1] = 79, doesn’t affect

swap values of array[0] and array[2]; array[0] = 4 —array[2] = -1

after all of these, array[6] = {4, -1, 79, 9, 22, 19};

couts are: **4, 79, -1, 9, 22, 19;**

}

Overall output:

3

4

79

-1

9

22

19

5)

void removeS (char \*letter)

{

char\* theAddress; //Placeholder address

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

{

if (\*letter == 's' || \*letter == 'S') //Letter equal to S || s

{

theAddress = letter;

while (\*theAddress != '\0') //Move every item left to the s, 1 to the right

{

\*theAddress = \*(theAddress + 1);

theAddress++;

}

letter—; //Check for double ss

}

letter++;

}

}