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 = 30; // set arr[2] to 30

while (ptr >= arr)

{

cout << \*ptr << endl; // print values 30, 20, 10 in that order

ptr--;

}

}

1b. This function does not work because the original pointer passed in (ptr), is not actually changed in the function. To fix this, I changed the parameter of int\*pToMax to pass by reference through the & symbol. Now the function will actually change the address contained in ptr.

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

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

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

{

if (arr[i] > \* pToMax)

pToMax = arr + i;

}

}

int main()

{

int nums[4] = { 5, 3, 15, 6 };

int\* ptr;

findMax(nums, 4, ptr);

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

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

cout << "Its value is " << \*ptr << endl;

}

1c. Before, \*ptr did not actually access the value at an address of a variable. By creating an integer variable such as a and making ptr contain the address of it, the function computeCube can work since \*ncubed will change the value at the address ptr is assigned to.

void computeCube(int n, int\* ncubed)

{

\*ncubed = n \* n \* n;

}

int main()

{

int\* ptr;

int a;

ptr = &a;

computeCube(5, ptr);

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

}

1d. Notably, str1 and str2 refer to the addresses of str1[0] and str2[0]. In order to check the individual characters of the C strings, we need to look at the values in the addresses of str1 and str2. The dereference operator serves this purpose. Lastly in the return line, the last character of str1 and str2 should be the zero byte which can be checked again through \*.

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?

}

int main()

{

char a[15] = "Chen";

char b[15] = "Chen";

if (strequal(a, b))

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

}

1e. The program is returning the address of an array considered garbage value after the function has finished. This is because the array anArray[100] is declared locally and outside of the function it does not exist.

2.

a) double\* cat;

b) double mouse[5];

c) cat = &mouse[4];

d) \*cat = 25;

e) \*(mouse + 3) = 54;

f) cat -= 3;

g) cat[1] = 27;

h) cat[0] = 42;

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

j) bool d = (cat == mouse);

3a.

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

{

const double\* ptr = scores;

double tot = 0;

int count = 0;

while (ptr + count != scores + numScores)

{

tot += \*(ptr + count);

count++;

}

return tot / numScores;

}

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. First, \*ptr contains the value at the address &array[0] since 5 is greater than 4 in the maxwell function. Then the value at array[0] is changed to -1. (-1, 3, 4, 17, 22, 19)

Next, ptr is shifted to the address of &array[2]. Then ptr[1] is the same as \*(ptr + 1) which is set to 9, making array[3] equal to 9. (-1, 3, 4, 9, 22, 19) Then \*(array + 1) sets the value at array[1] equal to 79. (-1, 79, 4, 9, 22, 19)

The first line that is printed out takes the difference in address positions of &array[5] and the ptr which is at &array[2], a difference of 3 elements. 3 is printed.

Swap1 does not actually do anything since it merely changes the addresses of the local pointers within the function. Swap2 swaps the values of array[0] and array[2] in the actual array. (4, 79, -1, 9, 22, 19)

Each element of the array is then printed out.

5. void removeS(char\* chr) {

while (\*chr != 0) {

if (\*chr == 's' || \*chr == 'S') {

char\* inc = chr;

while (\*inc != 0) {

inc++;

\*(inc - 1) = \*inc;

}

continue; //in case another S is present

}

chr++;

}

}