**1a.**

One source of error was the order of precedence. \*ptr + 1 is not the same as \*(ptr + 1). \*ptr + 1 is equivalent to ptr[0] + 1 which is 31 in this case. 31 is an integer and cannot be assigned a value.

Another source of error was that the loop was making the array go out of bounds. I fixed this by incrementing at the end of the loop rather than the beginning.

The final source of error was that we were looping the array backwards. In order to fix this, I created a new pointer that stayed at the start of array until we started looping. (I did not use array because generally, you don’t want to completely lose the position of the first element of the array)

int main()

{

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

int\* ptr = arr;

int\* ptr2 = arr; // Created so we can do a forward loop without modifying 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 (ptr2 <= ptr)

{

cout << \*ptr2 << endl; // print values

ptr2++; // Increment at the bottom to avoid going out of bounds

}

}

**1b.**

Before, ptr was being passed by value. Because of this, a copy of ptr was made called pToMax which goes away after the function is done. In order to fix this, we pass ptr by reference by adding the ‘&’ symbol in the pointer parameter.

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, an uninitialized pointer was being passed into the computeCube function. However passing an uninitialized pointer into a function causes an error. Even if this were to work, if we dereference an uninitialized pointer, as done in the 3rd line, another error would occur because we are dereferencing a pointer that points to nothing. This can be fixed by adding an int variable and generating a pointer that points to that int.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int x = 0;

int\* ptr=&x;

computeCube(5, ptr);

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

}

**1d.**

The problem was that str1 and str2 were not dereferenced when we were looking for the 0 byte and comparing the char they point to. This is fixed by simply adding a ‘\*’ before str1 and str2. Modifications were made on lines 3, 5, and 10.

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] = "Tang";

char b[15] = "Zhang";

if (strequal(a, b))

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

}

**1e.**

The storage for anArray goes away when we exit the function getPtrToArray but we are left with the pointer pointing at the start of that array. Since no space was reserved for the array, the compiler is free to replace those values. Now when we dereference the pointer, we yield undefined behavior.

**2.**

// 2a

double\* cat;

// 2b

double mouse[5];

// 2c

cat = &mouse[4];

// 2d

\*cat = 42;

// 2e

\*(mouse + 3) = 17;

// 2f

cat -= 3;

// 2g

cat[1] = 25;

// 2h

cat[0] = 54;

// 2i

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

// 2j

bool d = \*cat == mouse[0];

**3a.**

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

{

const double\* ptr = scores;

double tot = 0;

int currentIndex = 0;

while (currentIndex != numScores)

{

tot += \*(ptr + currentIndex);

currentIndex++;

}

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;

}

**4.**

The program prints:

3

4

79

-1

9

22

19

ptr was assigned to array because the function maxwell calculated that the element at array was pointing to is greater than the element at array[2]. We then replace the element ptr is pointing to with -1 and move ptr up 2 elements.

At this point, array is {-1, 3, 4, 17, 22, 19}.

Then we set the element one index to the right of the one ptr is at to 9.

At this point, array is {-1, 3, 4, 9, 22, 19}.

We not set the index one to the right of array to become 79.

At this point, array is {-1, 79, 4, 9, 22, 19}.

We now print out the difference of elements between &array[5] and ptr. ptr is currently at array[2] so the difference is 3, which we print.

Our output is now…

3

We now call swap1 which essentially does nothing since it is being passed by value.

We call swap2 which switches the elements between the one array is pointing to and the one &array[2] is pointing to.

At this point, array is {4, 79, -1, 9, 22, 19}.

We now print out the elements of the array, one per line.

Our output is now…

3

4

79

-1

9

22

19

**5.**

void removeS(char\* original){

char\* ptr = original;

for (; \*original != '\0'; original++){

while (\*ptr == 'S' || \*ptr == 's'){

ptr++;

}

\*original = \*ptr;

original++;

ptr++;

}

\*original = '\0';

}