**Project 6**

**1A.**

int main()

{

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

int\* ptr = arr;

\*ptr = 10;

\*ptr + 1 = 20; **// not correct, invalid**

ptr += 2;

ptr[0] = 30;

while (ptr >= arr)

{

ptr--;

cout << ' ' << \*ptr;

}

cout << endl;

}

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 10

ptr+=2;

ptr[0] = 30;

ptr++;

while (ptr > arr)

{

ptr--;

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

}

cout << endl;

}

**1B.**

The pointer is not being referenced; changes are happening, but they do not roll over to the pointer. In changing that, the loop should also be looked at in terms of pointer location/value instead of array location/value.

void findDisorder(int arr[], int n, int\* &p)

{

for (p = arr+1; p < arr + n; p++)

{

if (\*p < \*(p-1))

{

return;

}

}

p = nullptr;

}

**1C.**

It might not work because the pointer doesn’t have a value (0) to start from. Some sort of “initialization” needs to take place in order for the pointer to have a value stored there(have a place to point to). It seems to act as a null pointer before it is initialized (points to nothing beforehand).

int main()

{

double initialize = 0.0; //needs to be assigned to something

double\* p = &initialize;

hypotenuse(1.5, 2.0, p);

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

}

**1D.**

The entire function considers pointers and not the values that the pointers point to (need asterisks).

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

{

while (\*str1 != 0 && \*str2 != 0) //if the value is 0

{

if (\*str1 != \*str2) //if the values are equal

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; //if the values are equal

}

**1E.**

The pointer is just pointing to the first element of the arr array. This results in the loop being unsuccessful because it will not run through the arr array. It doesn’t get a chance to look at the other elements in the array because the area dedicated to the array is taken up by junk. The array basically disappears with the function.

**2. //assuming parts before have been executed**

**a.** string\* fp;

**b.** string fish[5];

**c.** fp = fish+4;

**d.** \*fp = "yellowtail";

**e.** \*(fish+3) = "salmon";

**f.** fp = fp - 3;

**g.** fp[1] = "trout";

**h.** fp[0] = "eel";

**i.** bool d = (fp == &fish[0]);

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

**3A.**

double computeAverage(const double\* scores, int nScores)

{

int visitingDouble = 0;

double tot = 0;

while (visitingDouble < nScores)

{

tot += \*(scores + visitingDouble);

visitingDouble++;

}

return tot/nScores;

}

**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.**

**//note: whether the \* is attached to the int or the variable doesn’t matter (i.e int\* a vs. int \*a)**

int\* minimart(int\* a, int\* b) //returns the pointer of the smaller number of the numbers the pointers point to

{

if (\*a < \*b)

return a;

else

return b;

}

void swap1(int\* a, int \*b) //attempts to swap ADDRESSES (pointers also do not change because they are not passed by reference)

{

int\* temp = a;

a = b;

b = temp;

}

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

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = minimart(array, &array[2]); //ptr starts at array[2]

ptr[1] = 9; //array[3] == 9, ptr still points to array[2]

ptr += 2; //points to array[4]

\*ptr = -1; //array[4] == -1

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

cout << "diff=" << &array[5] - ptr << endl; //5-4

swap1(&array[0], &array[1]); //nothing really happens

swap2(array, &array[2]); //swaps 4 and 5

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

cout << array[i] << endl;

}

//the following will print:

//diff = 1 -- comes from the subtraction of pointer spaces, 5-4

//4 - after the swap1 not changing the array, this value gets swapped with the value in array[2]

//79 - comes from \*(array+1) -- sets the number at array[1] to 79

//5 - after swap1 not changing the array, it gets swapped with the value in array[0]

//9 - comes after the minimart -- the value next to array[2] is changed (array[3])

//-1 - the pointer moves 2 positions from position array[2] and sets array[4] to -1

//19 - never changed

**5.**

void deleteG(char\* ptr)

{

//make a separate array

char\* newMsg = ptr;

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

{

if(\*ptr == 'g' || \*ptr == 'G')

{

ptr++;

}

else

{

\*newMsg = \*ptr;

newMsg++;

ptr++;

}

}

ptr = newMsg;

\*newMsg = '\0';

}