Joshua Liu

105136031

November 22, 2018

1a.

// Original

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 + 1 needs to be in parentheses to correctly call for the value at the second position in the array which we then set 20

ptr += 2;

ptr[0] = 10; // set arr[2] to 10 <== Values cannot be changed this way. ptr is already pointing at the third position in the array, so we can say \*ptr = 10;

while (ptr >= arr) <== Causes values to be printed out in the opposite order. Instead, we point the pointer back to the first position then print out the value at the pointer location, incrementing after printing until the last number in the array is printed, represented by &arr[2], the address of the last the last position of the array.

{

ptr--;

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

}

}

// Fixed

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 <== Added parentheses

ptr += 2;

\*ptr = 10; // set arr[2] to 10 <== Set value at point equal to 10

ptr -= 2; <== Point back to first position

while (ptr <= &arr[2]) <== We increase ptr after printing out the value at the pointer each time

{

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

ptr++;

}

}

1b.

The function was written incorrectly in that the pointer was not pass-by-reference. pToMax needs to be a pass-by-reference value in order to change the value of the pointer. Adding an ampersand to int\* for pToMax in the function fixes the program and allows it to work.

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.

The main function is incorrect in that pointer ptr does not point to any value, making it uninitialized. Pointing it to any address causes the program to work as we want it to.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int i = 0; <== Make point ptr point to i to give it an address to follow

int\* ptr = &i;

computeCube(5, ptr);

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

}

1d.

In the function, when str1 and str2 are compared, we need to add an asterisk before str1 and str2 in order to call the values at that point in the memory to compare. Otherwise, we would be comparing different memory values which are never the same for two different variables. Also, in the while statement, we compare str1 and str2 to the null character to determine when the C string ends. Lastly, in the return statement, asterisks are added to str1 and str2 in order to see if the remain characters are the same, which is if they are both null byte characters.

// return true if two C strings are equal

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

{

while (\*str1 != '\0' && \*str2 != '\0') <== Add \* before str1 and str2 to compare values

{

if (\*str1 != \*str2) // compare corresponding characters <== Add \* before str1 and str2 to compare values

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time? <== Add \* before str1 and str2 to compare values

}

int main()

{

char a[15] = "Zhou";

char b[15] = "Zou";

if (strequal(a, b))

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

}

1e.

We cannot declare the array anArray in the getPtrToArray function since it will not exist in the main function after running through the getPtrToArray function. The scope of the array is incorrect and as a result leads to an undesired output.

2a. double\* cat;

2b. double mouse[5];

2c. cat = mouse + 4;

2d. \*cat = 25;

2e. \*(mouse + 3) = 42;

2f. cat -= 3;

2g. cat[1] = 27;

2h. cat[0] = 54;

2i. bool b = (\*cat == \*(cat + 1));

2j. bool d = (cat == mouse);

3a.

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

{

double tot = 0;

int count = 0;

while (count < numScores)

{

tot += \*(scores + 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;

return nullptr;

}

3c.

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

{

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

{

if (\*(str) == chr)

return str;

str += 1;

}

return nullptr;

}

4a.

The program begins by initializing an int array array[6] with 6 integers inside.

Then, pointer ptr is initialized and set equal to the return value of maxwell(array, &array[2]);

The maxwell function returns the address of the greater value between the array passed in at position 0 and position 2. In this case, the address at position zero is returned since 5 is greater than 4. This means the pointer is pointing at the first element in the array.

\*ptr = -1; ==> Then, the value of the pointer(at the first position of the array) is changed to -1, represented by the asterisk before variable ptr.

ptr += 2; ==> Next, the pointer value is changed so that it is pointing to the third element of the array, or the value at array[2].

ptr[1] = 9; ==> The next statement sets the value of array[3] to 9 since ptr[0] is currently array[2].

\*(array + 1) = 79; ==> Now, the value of the second element of the array, array[1], is set to 79 since there is an asterisk refereing to the value at that point.

cout << &array[5] - ptr << endl; ==> This cout statement prints out the difference between the memory locations between array[5] and the pointer ptr. ptr is currently pointing to the third element in the array, and &array[5] is currently the address of the 6th element, so subtracting those two results in the value 3(the difference between the locations of the two pointers(since an array is a pointer).

swap1(&array[0], &array[1]); ==> Next, this swap function basically does not change the values of any items. a and b, pointers to the first and second element in the array, are changed within the function but is not affected in the main because the values are pass-by-value.

swap2(array, &array[2]); ==> This function, unlike the first one, changes the values that the pointers are pointing to. a is pointing to the first element, and b is pointing to the third element(denoted by the &array[2] which gets the memory location of that spot). The asterisks call for the value that the pointers are pointing to and they are then swapped.

for (int i = 0; i < 6; i++) ==>> This last cout prints out the values in the array array, which has 6 elements. They are all printed on separate lines becasue of the endl.

cout << array[i] << endl;

OUTPUT:

3 ==> This three is from the first part of the program. It represents the difference between the memory values of position 5 and position 2, which is 3.

4 ==> The next 6 values printed are the values in the array. Before the array was changed by the swap functions, the elements were of the following values: { -1, 79, 4, 9, 22, 19 }. Then, the first and third elements are swapped, providing us the output as shown.

79

-1

9

22

19

A clearer representation:

{ -1, 79, 4, 9, 22, 19 } <== Before swap functions

{ 4, 79, -1, 9, 22, 19 } <== After swap functions

The first and third elements are swapped, and the latter is printed out.

5.

void removeS(char\* str)

{

while (\*str != '\0') // Keep looping through the pointer to look for 's' and 'S' to remove

{

if (\*str == 's' || \*str == 'S')

{

char\* temp = str; // Create a temporary pointer that points to the start of the original pointer in order to set the original pointer back to the zero position after changing the pointer's position

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

{

\*str = \*(str + 1); // Copy the characters onto the previous one to remove a 's' or 'S'

str++;

}

str = temp; // Set str back to the zeroth position

continue; // Do not increment str after this if-statement since there can be two s's in a row

}

str++;

}

}

int main()

{

char msg[50] = "She'll be a massless princess";

removeS(msg);

cout << msg; // prints he'll be a male prince.

}