Stewart Dulaney

Project 6 HW: What's the point?

**1.**

**a)**

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[0] = 30; // set arr[2] to 30

for(ptr = arr; ptr < &arr[3]; ptr++)

{

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

}

}

**b)**

The "findmax" function did not work because of the last parameter called by the function. The original parameter "int\* pToMax" called for the value of a pointer variable of type int. However, it is an error to follow the pointer "ptr" (the parameter passed by the main routine) because it was not initialized in the main routine and does not point to a variable. Therefore, the uninitialized pointer "ptr" needs to be passed by reference to "findMax", so that it can be initialized and changed in the function.

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 index " << ptr - nums << endl;

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

}

**c)**

The error in the main function was that it tried to pass the pointer "ptr" to the function "computeCube" without initializing it. An uninitialized pointer has no pointer value, so when "computeCube" calls for a pointer value in its second parameter ("int\* ncubed"), there is an error. The solution is to initialize a variable of type int ("int k;") and assign "ptr" to point to it ("int\* ptr = &k). As a result, "computeCube" is passed the parameter of type pointer that it calls for.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int k;

int\* ptr = &k;

computeCube(5, ptr);

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

}

**d)**

The "strequal" function uses pointers to move through the C strings, which are arrays of characters. One problem in the implementation of the "strequal" function was the condition in the while loop it contained. It initially tested the condition "(str1 != 0 && str2 != 0)" which is wrong, because it tests if the address of the characters is zero, which is irrelevant. The correct while statement tests for the zero byte at the end of the C strings and is written "while (\*str1 != '\0' && \*str2 != '\0')". An if-else statement with the condition "(\*str1 == '\0' && \*str2 == '\0')" needed to be added to test if both strings ended at the same time, returning true if they did and false if they didn't. Also, two dereferencing operators needed to be added in the if statement nested within the while statement so that characters of the strings were compared instead of their addresses in memory. The correct if statement is written "if (\*str1 != \*str2)".

// return true if two C strings are equal

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++;

}

if(\*str1 == '\0' && \*str2 == '\0') // both ended at same time?

return true;

else

return false;

}

int main()

{

char a[10] = "Bryan";

char b[10] = "Bryant";

if (strequal(a,b))

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

}

**e)**

The problem with this program is that the array "anArray" is local to the getPtrToArray function. Therefore, when the program tries to output values from anArray in the main routine, garbage values from memory are outputted.

**2.**

a) double\* cat;

b) double mouse[5];

c) cat = &mouse[4];

d) \*cat = 17;

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

f) cat -= 3;

g) cat[1] = 33;

h) cat[0] = 25;

i) bool b;

if(\*cat == \*(cat + 1))

b = true;

else

b = false;

j) bool d;

if(cat == mouse)

d = true;

else

d = false;

**3.**

**a)**

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

{

const double\* ptr = scores;

double tot = 0;

for(int k = 0; k < numScores; k++)

{

tot += \*(ptr + k);

}

return tot/numScores;

}

**b)**

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

{

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

if (\*(str + k) == chr)

return str + k;

return NULL;

}

**c)**

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

{

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

{

if(\*str == chr)

return str;

str++;

}

return NULL;

}

**4.**

Output of program:

3

4

79

-1

9

22

19

Why each line of output prints out the way it does:

The original array is initialized as {5, 3, 4, 17, 22, 19}. Then ptr is set to point to array[0] because maxwell determined that the dereferenced value of a pointer to array[0] is greater than the dereferenced value of a pointer to array[2]. The element that ptr points to, array[0], is then assigned the value -1 using the dereferencing operator. Next the integer 2 is added to ptr, causing it to move two elements forward and point array[2]. Then using square brackets, the element following the element ptr points to, array[3], is assigned the value 9. Next, the dereferencing operator is used to assign the value 79 to the element after the zeroth element of array, array[1]. The array now contains the values {-1, 79, 4, 9, 22, 19}. The program then outputs the value (&array[5] - ptr). This outputs 3 because ptr points to array[2], and elements in an array have memory addresses that increment by 1, so 5 - 2 = 3. Next the swap1 function exchanges the pointer values of &array[0] and &array[1] within the function, but the dereferenced values of the pointers are not swapped so array still contains the values {-1, 79, 4, 9, 22, 19}. Then the swap2 function exchanges the dereferenced values of array (which is &array[0]) and &array[2], which swaps -1 and 4. Now array contains {4, 79, -1, 9, 22, 19}. The program then uses a for loop to output each element of the array in order, starting at the element at index zero, which outputs 4 79 -1 9 22 19, one per line.

**5.**

void removeS(char\* string)

{

char\* ptr = string;

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

{

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

{

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

{

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

ptr++;

}

ptr = string;

}

ptr++;

}

}