CS31 Professor Smallberg Leo Gretzinger

Project 6 Writeup 11/26/18

1. The subparts to this problem involve errors in the use of pointers.
   1. int main()

{

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

int\* ptr = arr;

\*ptr = 10; // set arr[0] to 30. Switch assignments around s

// they work with the while loop

\*(ptr + 1) = 20; // set arr[1] to 20. Need (ptr + 1) in

//parantheses, otherwise adding one to the value

ptr += 2;

\*ptr = 30; // set arr[2] to 10. If incrementing by two in

// previous line, need \* operator

while (ptr >= arr)

{

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

ptr--; // Switch decrement to after print

// statement

}

}

* 1. The findMax function returns void, so in the main statement, calling the function using ptr doesn’t actually do anything to ptr, which is being printed later, and an error occurs. To work, the pToMax parameter in the findMax function needs to be passed by reference. Here is the fixed function:

void findMax(int arr[], int n, int\*& pToMax) // pToMax param contains ‘&’

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

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

{

if (arr[i] > \*pToMax)

pToMax = arr + i;

}

}

* 1. The main statement does not work, because the call of computeCube utilizes an uninitialized variable (ptr). To fix this, arbitrarily initialize ptr:

int main()

{

int a;

int\* ptr = &a; // Initializes ptr

computeCube(5, ptr);

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

}

* 1. The first problem is that the function doesn’t check if the two strings are the same length. In order to be exactly the same they have to be the same length. Also, the function is comparing the location of each corresponding character, not the actual value of the corresponding characters (both within the while loop and in the statement outside it). Lastly, the while loop needs to go until the value of str1 or str2 equal 0 (or the null character) not until the addresses of them do. Here is the fixed function:

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

{

if (strlen(str1) != strlen(str2))

return false;

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

}

* 1. The noChange() and the f() functions work fine. The thing that creates the problem is that in the getPtrToArray() function, a local array is initialized, then returned back into the main function. This copies the array over, but it also makes the array susceptible to being overwritten. In general, when attempting to call functions with the array created in getPtrToArray(), the array is out of scope because it is just a local variable in the function. As a result, the main statement ends up printing garbage values instead of the intended 100 99 98 3 2 1.

1. Each line of the code is a part of question 2:

double\* cat;

double mouse[5];

cat = &mouse[4];

\*cat = 25;

\*(mouse + 3) = 42;

cat -= 3;

cat[1] = 27;

cat[0] = 54;

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

bool d = (cat == mouse);

1. These are the fixed functions:
   1. double mean(const double\* scores, int numScores)

{

double tot = 0;

int i = 0;

while (i < numScores)

{

tot += \*(scores + i);

i++;

}

return tot / numScores;

}

* 1. 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;

}

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

{

while (\*str != 0) {

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

1. int main()

{

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

// Points to index 0 of array because maxwell chooses the greater of the

// two inputted pointer values

int\* ptr = maxwell(array, &array[2]);

// Sets index 0 of the array to -1

\*ptr = -1;

// Moves the pointer up to index 2 of array

ptr += 2;

// Sets index 3 of array to 9 (because ptr[1] adds 1 to where ptr points

// to)

ptr[1] = 9;

// Sets index 1 of array to 79 because 'array' points to index 0, and '+1'

// adds 1

\*(array + 1) = 79;

// ptr is currently pointing to index 2 and the difference between the

// pointer to index 5 and the pointer to index 2 is essentially the

// distance between the two: 3. This line prints "3" and enters.

cout << &array[5] - ptr << endl;

// The swap1 function swaps the memory addresses of two pointers and

// returns nothing, so the call to this function doesn't have any effect on

// the array.

swap1(&array[0], &array[1]);

// The swap2 function swaps the values of two pointers, which ends of

//swapping the two values in the array. Swapping index 0 and index 2 of

// array makes array now 4 79 -1 9 22 19

swap2(array, &array[2]);

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

// This for loop cycles through array and prints every value, so

// this line prints the values of array, each on a new line:

// 4 79 -1 9 22 19

cout << array[i] << endl;

}

1. void removeS(char\* str) {

while (\*str != 0) {

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

char\* ptr = str;

while (\*ptr != 0) {

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

ptr++;

}

}

else {

str++;

}

}

}