Amy Seo

CS 31

Professor Smallberg

25 November 2019

Project 6

1. a) Bugs:

* \*ptr + 1 = 20 is invalid because it’s setting an expression equal to a value.
* the while loop prints from the end of the array to the beginning and will print 20 30 then 0, one per line.

Fixed version:

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 += 2;

ptr[0] = 10; // set arr[2] to 10

ptr = arr;

while (ptr < arr + 3)

{

cout << \*ptr << endl;

ptr++;

}

}

b)

The function did not work the way as expected because the pointer is not being passed by reference and a copy of it is made in the findMax() function, so no changed are made to it. To fix it, I passed the pointer as a reference so that its value will be appropriately changed when printed out in the main 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 position " << ptr - nums << endl;

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

}

c)

The program does not work as intended because the ptr in the main function is not initialized to anything. It needs a location so that the value that it is pointing to can be set to n cubed. A way to fix this is to initialize ptr to point to an int:

int main()

{

int a = 1;

int\* ptr = &a;

computeCube(5, ptr);

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

}

d)

The original implementation of this function is wrong because you need to compare each character at every index with the use of the \* dereference character instead of comparing the pointers themselves. You can do this by creating char pointers in the function and iterating through the cstring with them:

// return true if two C strings are equal

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

{

const char\* ptr1 = str1;

const char\* ptr2 = str2;

while (\*ptr1 != 0 && \*ptr2 != 0)

{

// compare corresponding characters

if (\*ptr1 != \*ptr2)

return false;

ptr1++; // advance to the next character

ptr2++;

}

return \*ptr1 == \*ptr2; // both ended at same time?

}

int main()

{

char a[15] = "Chen";

char b[15] = "Cheng";

if (strequal(a,b))

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

}

e) When the main functions call for the pointer to the array, it only points to the memory location of the arrays without knowing if the intended array even exists.

In the getPtrToArray function, anArray[] is initialized and exists only within the function. When f() is called, junk[] takes up the memory location of anArray[]. But in the main function, both of these arrays do not exist anymore due to the scope of those variables.

1. a) double\* cat;

b) double mouse[5];

c) cat = &mouse[4];

d) \*cat = 25;

e) \*(mouse + 3) = 54;

f) cat -= 3;

g) cat[1] = 27;

h) cat[0] = 42;

i) bool b = (\*cat == \*(cat + 1));  
j) bool d =(cat == &mouse[0]);

1. a) double mean (const double\* scores, int numScores)

{

const double\* ptr = scores;

double tot = 0;

for (int i = 0; i < \*scores + numScores; i++)

{

tot += \*(ptr + i);

}

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 nullptr;

}

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

{

while (\*str != 0)

{

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

1. Output of the program:

3

4

79

-1

9

22

19

Part 1: The 3 is printed because ptr is initialized and declared to array as a result of the maxwell function; \*ptr = -1 sets the first element of array equal to -1; this pointer is then incremented by 2, making it point to the 3rd element in the array; when ptr[1] = 9 is called, it sets the 4th element equal to 9; \*(array+1) = 79 sets the value at the 1st index of array equal to 79. So when &array[5] - ptr is called to cout, ptr, which is equal to 2 because it currently points to the second index in the array, is subtracted from the reference to the value of the 5th element in the array, which is 5, and 5 - 2 = 3.

Part 2: The swap1 function call does not do anything to the values of the array even though the parameters are passed by reference because the pointers are simply passed as addresses without ever accessing the value it points to. However, the swap2 function uses the dereference operator when performing the swaps, which allows the values within the array to actually change. By swapping the element at index 0 with the element at index 2, array[0] is now set to 4 and array[2] is now set to -1.

Part 3: When the for loop is iterated through, it prints every element of the array on a new line. The first element printed is 4, as changed in swap2, the second element printed is 79 as changed in part 1, the third element printed is -1 as changed in swap2, the 4th element printed is 9 as changed in part 1, the 5th and 6th element printed are 22 and 19 respectively, as they are unchanged from the original array.

1. void removeS(char str[])

{

char\* ptr = str;

while (\*ptr != 0)

{

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

{

while(\*(ptr + 1) != 0)

{

if (\*(ptr + 1) != 's' && \*(ptr + 1) != 'S')

{

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

ptr++;

removeS(ptr);

\*ptr = 0;

}

else

removeS(ptr + 1);

}

}

ptr++;

}

}