Zachary Chang

UID: 805289445

Discussion 1G

Lecture 1

Programming Assignment 6

**1a.** (below)

The bug is that the pointer did not reset to point back to the first value. It should print left to right through the array.

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 -=2; //reset pointer back to first position

while (ptr <= arr + 2)

{

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

ptr++;

}

}

**1b.** (below)

The original program does not modify the address that the pointer points to. An ampersand is needed so that the function modifies the pointer address in the main function (pass-by-reference).

void findMax(int arr[], int n, int\*& pToMax) //need to declare a reference to a pointer

{

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. (below)**

The original function does not provide an address for the pointer to point to. By assigning the number that is to be cubed to a variable, it opens up memory space that the pointer can point to. Thus, the value at that address can now be modified.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int\* ptr;

int num = 5;

ptr = &num;

computeCube(num, ptr);

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

}

**1d. (below)**

In the original function, all of the comparisons are comparing pointers, not comparing the values at those addresses. To make the function work as intended, the values must be compared, so dereferencing symbols must be placed before each array variable.

// return true if two C strings are equal

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

{

while (\*str1 != 0 && \*str2 != 0) //should be comparing values not pointers

{

if (\*str1 != \*str2) // compare corresponding characters, again should be comparing values

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time? Compare values not pointers

}

int main()

{

char a[15] = "Chen";

char b[15] = "Cheng";

if (strequal(a,b))

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

}

**1e. (below)**

The initialization of the variable *ptr* involves accessing the memory of an array that was declared INSIDER the *getPtrToArray* function. By the time the address of array *anArray* is returned back to initialize *ptr*, the memory that previously held the array no longer holds the array: it has been scrapped after leaving the function.

**2. (below)**

int main()

{

double\* cat;

double mouse[5];

cat = mouse + 4;

\*cat = 25;

\*(mouse+3) = 54;

cat -= 3;

cat[1] = 27; //changed

cat[0] = 42;

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

bool d = (cat == mouse);

}

**3a. (below)**

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

{

double tot = 0;

int i = 0;

while (i < numScores)

{

tot += \*(scores + i);

i++;

}

return tot/numScores;

}

**3b. (below)**

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

{

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

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

{

str = str + k;

return str;

}

return nullptr;

}

**3c. (below)**

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

{

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

{

if (\*(str) == chr)

{

return str;

}

str = str + 1;

}

return nullptr;

}

**4.** The program prints the following:

3

4

79

-1

9

22

19

(Explained below, chronologically)

***main***

The main method first declares an array of 6 integers. After this, the program calls the function *maxwell*,passing the pointer to the first and third elements of the array as parameters.

***maxwell Function***

The function compares the integers at the first and third positions of the array. It determines that the integer at the first array is larger and returns the pointer to the first element of the array.

***main (continued)***

The variable *ptr* is currently pointing at the first element of the array. The next statement changes the value at the first position in the array to -1. After this, the pointer is incremented 2 positions to the right, now pointing at the third element of the array. Setting this new position as the “first element of the pointer’s array,” the value at index 1 is 17. This value is changed to 9. The array pointer is still pointing at the first element, so incrementing it by 1 and setting the dereferenced pointer to 79 changes 3 to 79. Our array is currently as follows: {-1, 79, 4, 9, 22, 19}. The *ptr* variable is still pointing to the third element of the array (index 2). The *cout* line takes the address that the *ptr* variable holds and subtracts it from the address of the last element of the array (index 5). The resulting value is the difference in index values which is 3. Next, the addresses of the first and second elements of the array are passed as parameters to the swap1 function.

***swap1***

The function declares two pointer variables *a* and *b* and has them point to the addresses of the first and second elements of the array. The function then swaps the values in the array that the pointers point to, so now *a* points to 79 and *b* points to -1.

***main (continued)***

The main method passes the addresses of the first and third elements of the array to the function *swap2*.

***swap2***

The function declares two pointer variables a and b and has them point to the addresses of the first and third elements of the array. The function then swaps the VALUES in the array that the pointers point to, so now the first element is 4 and the third element is now -1.

***main (continued)***

The current state of the array is as follows: {4, 79, -1, 9, 22, 19}. The for loop at the end prints out each value of the array on a separate line.

**5. (below)**

void removeS(char\* word)

{

char\* ptr = word;

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

{

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

{

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

{

ptr++;

}

}

\*word = \*ptr;

word++;

ptr++;

}

\*word = '\0';

}