**Chelsea Biala**

**CS31 Smallberg**

**Fall 2020**

(Errors and fixes are in bold and pink)

**1.** a. 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 **// Won’t compile**

ptr += 2;

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

while (ptr >= arr) **// This will print the array backward**

{

ptr--; **// This will avoid printing the final value (10)**

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

}

}

*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 -= 2;

while (ptr <= (arr + 2))

{

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

ptr++;

}

}

b. The function doesn’t work properly because the pointer is passed by value into the function. That means that even when the pointer is assigned within the function, in the main routine it will remain uninitialized. That means that we can fix it by passing the pointer by reference into the function.

*Fixed version:*

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;

}

}

c. The program doesn’t execute correctly because in the main routine, ptr has not been initialized. Therefore, it doesn’t point to anything, so the function will not alter any value and printing out the value at ptr will yield nothing. We can fix this by creating an int variable for ptr to point to before calling the function.

*Fixed version:*

int main()

{

int i;

int\* ptr = &i;

computeCube(5, ptr);

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

}

d. The function doesn’t work properly because first, the loop doesn’t iterate correctly through the strings. It will not execute at all as is, because both str1 and str2 will begin at index 0 and the loop won’t execute at all. We can fix this by checking if the pointers are at the end of the string, checking if they’re equal to the zero byte. Second, the loop compares the two pointers, not the characters contained in the pointers. We can fix this by dereferencing the pointers, putting a \* in front of them. Finally, at the end, we can’t compare the pointers to see if both strings have ended; we must check at each one to see if they point to the zero byte.

Fixed version:

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++;

}

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

}

e. First, when getPtrToArray is called, anArray is declared within the function. That means it’s a local variable and will get thrown out when the function is over. Even if ptr was assigned to that memory location, the memory can be reused and it will now point to a completely arbitrary value (probably filled by f()).

**2.** a. double\* cat;

b. double mouse[5];

c. cat = &mouse[4];

d. \*cat = 25;

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

f. cat -= 3;

g. cat[1] = 42;

h. cat[0] = 27;

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

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

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

{

double tot = 0;

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

{

tot += \*(scores + i);

}

return tot/numScores;

}

b. const char\* findTheChar(char\* c, char chr)

{

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

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

return c+k;

return nullptr;

}

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

{

while(\*c != ‘\0')

{

if (\*c == chr)

return c;

c++;

}

return nullptr;

}

**4.** The program will print:

3

4

79

-1

9

22

19

The first line of output, 3, is printed because:

* The pointer ptr points to array, or array[0], because the maxwell function returns the pointer which points to a greater value out of the parameters (in this case, array[0] and array[2], 5 > 4 so pointer to array[0] is returned)
* The pointer is incremented by two, so it then points to array[2]
* The first output statement prints the difference between the address of array[5] and ptr, which is still array[2]. 2 - 5 = 3, so 3 is printed

The next 6 lines of output come from a for loop that iterates through array and prints each element in order:

* 4 is printed first because array[0] = 4. This is because the swap2 method swapped the values pointed to by array and &array[2], which were respectively array[0], -1, and array[2], 4.
* 79 is printed next because array[1] = 79. It was originally 3, but it was set to 79 with \*(array+1) = 79. This expression modified array[2] because array is a pointer to array[0], incrementing it by one made it temporarily point to array[1], and \* allowed alteration of the value it pointed to. The swap1 method did not affect array[1], because it swapped the pointers to the values, not the values themselves. Since the for loop iterated through the array using square brackets, it had no effect.
* -1 is printed next because array[2] = -1. This value was originally in array[0] because we changed it from 5; when the pointer pointed to array[0] because of maxwell we changed the value at pointer to -1. The swap2 method then swapped the values pointed to by array and &array[2] which were respectively array[0], -1, and array[2], 4.
* 9 is printed next because array[3] = 9. This is because when ptr pointed to the value at array[2], we set ptr[1] to 9. This changed array[3] from 17 to 9, because ptr[1] sets the value at the position 1 greater than pointer, which was array[3].
* 22 is printed next because array[4] = 22. This value remained unchanged throughout the program and is still equal to the original value when the array was declared.
* 19 is printed next because array[5] = 19. This value remained unchanged throughout the program and is still equal to the original value when the array was declared.

**5.** void removeS(char\* c)

{

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

{

char\* j = c;

if(tolower(\*c) == 's')

{

while(\*(j+1) != '\0')

{

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

j++;

}

\*j = '\0';

}

else

c++;

}

}