Name: Sidharth Sudhir

UID: 805 759 182

CS31 Project 6

Question 1

*Part a:*

* The first bug I noticed was a syntactical error. When setting the second element of the array *arr*, to 20, the syntax *\*ptr + 1 = 20;* without using brackets around ptr + 1, which seemed to be an error to me.
  + Correction: I would correct this by writing \*(ptr + 1) = 20;. This would mean that the computer would change the value at 1 space ahead of ptr, which is the start of the array.
* The second bug was with printing out the integers:
  + The first issue is that ptr is decremented before the first integer is printed out. So, the first integer is skipped and the last integer printed will be a random value stored in the position in memory just before the array *arr* begins.
  + I fixed this by changing the logic of the while loop as shown below:

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

int x = 0;

while (ptr >= arr)

{

cout << arr[x] << endl;

x++;

ptr--; // print values

}

}

*Part b:*

* The first bug I noticed was that the for loop in the function sets i = 1. This means that it will completely ignore the first element in the array. So, the immediate first fix is to set this to 0.
* The second fix I made was not a bug but a slight modification made for my own readability. Under the if condition in the for loop, *pToMax* is set to *arr + i*. For my convenience, I rewrote that line as *pToMax = & ( arr[ i ] )*.
* The main bug of this program was that the pointer to the integer in the void function findMax was being lost after execution of the function. To fix this, I converted the pointer to call by reference in the function parameter. My modified code for the function is below:

void findMax(int arr[], int n, int\* &pToMax)

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

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

{

if (arr[i] > \*pToMax)

pToMax = &(arr[i]);

}

}

*Part c:*

* The primary bug in this code is that *ptr* is a pointer to an integer, but there is no integer that it is pointing to. To fix this, I declared a variable integer x with the line *int x;.* I then initialized the pointer to the address of x with the line *int\* ptr = &x;.* My final main function looks as shown below:

int main()

{

int x;

int\* ptr = &x;

computeCube(5, ptr);

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

}

*Part d:*

* The main error in this part was with how the function operated with C strings.
* Str1 and str2 can be used as either addresses to the first element of the arrays storing characters or the arrays themselves. So, there are two fixes to this problem:
  + **Solution 1**: You can add asterisks to str1 and str2 to start comparing the values at those points rather than the addresses, which is what was happening prior to the fix. As shown below:

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

{

while (\*str1 != 0 && \*str2 != 0)

{

if (\*str1 != \*str2)

return false;

position++;

}

return true;

}

* **Solution 2**: You can initialize an integer to 0 and use that to help loop through each element of the array and increment this integer. As shown below:

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

{

int position = 0;

while (str1[position] != 0 && str2[position] != 0)

{

if (str1[position] != str2[position])

return false;

position++;

}

return true;

}

*Part e:*

In the function getPtrToArray, a pointer to an array is the value that is returned. anArray, the array produced in this function, was only made and used locally within the function. The pointer which is returned is unable to access this array since it was only temporarily stored and discarded. So, when trying to access the value at that pointer, a random location in memory is accessed and erroneous values are seen.

Question 2

int main()

{

double\* cat; // a)

double mouse[5]; // b)

cat = &mouse[4]; // c)

\*cat = 25; // d)

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

cat = cat - 3; // f)

\*(cat + 1) = 17; // g)

cat[0] = 42; // h)

bool d = (cat == mouse); // i)

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

}

Question 3

*Part a:*

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

{

const double\* ptr = scores;

double tot = 0;

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

{

tot += \*(ptr + i);

}

return tot / numScores;

}

*Part 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;

}

*Part c:*

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

{

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

{

if (\*str == chr)

{

return str;

}

str++;

}

return nullptr;

}

Question 4

Line 1 3

Line 2 4

Line 3 79

Line 4 -1

Line 5 9

Line 6 22

Line 7 19

**Explanation for Line 1 of output**: Here we are subtracting two pointers, and so the compiler is checking the space between the two memory locations in the array of integers and dividing it by the number of bytes in each integer (4 bytes). The output of this computation is the integer 3. This means that the two addresses in memory being subtracted are 3 spaces apart.

**Explanation for Lines 2 - 7**: These are the elements of an array that were printed out. There was a reordering of the elements in this array caused by multiple lines of code. The line int\* ptr = maxwell(array, &array[2]); made the pointer *ptr* point towards the first element of the array, 5. The next line, \*ptr = -1;, changed the value at that pointer from 5 to -1. The line with code ptr += 2;, moved the pointer arrow to point two spaces down to index [2] of the array, pointing to 4. Through pointer arithmetic, ptr[1] = ptr[0+1] = ptr[0] + 1, so ptr[ 1 ] is the first element after the position that the pointer currently points towards. So, ptr[1] = 9; caused the first element after the pointer to change to 9. The line - \*(array+1) = 79; - changed the first element after the start of the array to 79 from 3.

At this stage the order of elements in the array is as follows:

**{ -1, 79, 4, 9, 22, 19 }**

After this, the swap1 and swap2 functions are executed. Swap1 results in the pointer arrows being switched between the two pointers passed in the function but this causes no change in the order of the array. Swap2 however, changes the two values that the two pointers passed point to. So, the integers in position 0 and position two were swapped. The final array’s order was:

**{ 4, 79, -1, 9, 22, 19 }**

A for loop was written to print this out line by line.

Question 5

void removeS(char\* s)

{

int i;

for(i = 0; \*(s+i) != '\0'; i++)

{

if(\*(s+i)== 's' || \*(s+i) == 'S')

{

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

{

\*(s+i+k) = \*(s+i+k+1);

}

i--;

}

}

}