**Project 6**

1. **Question 1:** 
   1. **Part A**
      1. The first error is that in the line *\*ptr + 1 = 20;* the programmer wants to add one to move the pointer to the next value in the array, but is actually adding one to the value that *ptr* points to, and essentially says *30 + 1 = 20;* which will give an error.
      2. The second error is in the line *ptr[0] = 10;* as the programmer wants to set the 3rd element to 10, but *ptr* is not pointing at the third element of the array
      3. The last error is that the while loop starts at the end of the array and attempts to print the array out backwards, which, if the rest of the code was correct, would print out 10 (endl) 20 (endl) 30 (endl), which is not what the program is supposed to do
         1. int main()
         2. {
         3. int arr[3] = { 5, 10, 15 };
         4. int\* ptr = arr;
         5. \*ptr = 30; // set arr[0] to 30
         6. ptr += 1;
         7. \*ptr= 20; // set arr[1] to 20
         8. ptr += 1;
         9. \*ptr = 10; // set arr[2] to 10
         10. ptr = arr; //sets ptr back to the 0th position
         11. while (ptr <= (arr+2))
         12. {
         13. cout << \*ptr << endl; // print values
         14. ptr++;
         15. }
         16. }
   2. **Part B**
      1. This function won’t allow the caller to know the location of the greatest item because the function itself is a void function and therefore cannot pass the pointer back. The only solution to this would be to pass the pointer back by reference so that the value of ptr would be updated in the main function. Because the creator of this program did not do this, the value of ptr in the main function will remain the same before and after calling the function, as the pointer is passed by value and not by reference from the function.
      2. The solution to this problem would be to pass the pointer back by reference by setting the parameters of the void findMax function to:
         1. void findMax(int arr[], int n, int\* &pToMax)
   3. **Part C**
      1. The main function will not work because the pointer is uninitialized and thus in the computeCube function you cannot follow the pointer to the value it points to.
      2. The solution to this problem is to do the following: initialize the value of pointer to the memory address of the number that needs to be cubed, which can later be followed and changed by the computeCube function.
         1. void computeCube(int n, int\* ncubed)
         2. {
         3. \*ncubed = n \* n \* n;
         4. }
         5. int main()
         6. {
         7. int value = 5;
         8. int\* ptr;
         9. ptr = &value;
         10. computeCube(value, ptr);
         11. cout << "Five cubed is " << \*ptr << endl;
         12. }
   4. **Part D**
      1. The problem with this implementation is that the bool function compares the pointers with each other rather than the values that the pointers point to, thus comparing the memory addresses of the values that the pointers point to, rather than the actual values of the locations where the pointers point to
      2. The bool function also has a condition within the while loop that checks if str1 or str 2 is equal to a zero byte or not, except in this implementation, it checks if the memory address of the object that the str1 and str2 points to isn’t a zero byte, rather than if the value at each spot in the array is equal to a zero byte or not
      3. The solution to this problem is to dereference each pointer so that the bool function will compare the values at each of the spots rather than the memory addresses.
         1. bool strequal(const char str1[], const char str2[])
         2. {
         3. int i = 0;
         4. while (\*(str1 + i) != 0 && \*(str2 + i) != 0)
         5. {
         6. if (\*(str1 + i) != \*(str2 + i)) //compare characters
         7. {
         8. return false;
         9. }
         10. i++; //advance to the next character
         11. }
         12. return \*(str1 + i) == \*(str2 + i); //both ended at same time?
         13. }
   5. Part e
      1. The thing that the program is doing that is incorrect is creating a local array in the scope of the getPtrToArray function and trying to print values from it out, an array that is not available for the main function to access as it is removed when the function is done running.
2. **Question 2:**
   1. double\* cat;
   2. double mouse[5];
   3. cat = &mouse[4];
   4. \*cat = 25;
   5. \*(mouse+3)= 54;
   6. cat -=3;
   7. cat[1] = 27;
   8. cat[0] = 42;
   9. bool b = (\*cat == \*(cat +1));
   10. bool d = (cat == &mouse[0]);
3. **Question 3:**
   1. **Part A**
      1. double mean(const double\* scores, int numScores)
      2. {
      3. const double\* ptr = scores;
      4. double tot = 0;
      5. int i = 0;
      6. while ((ptr+i)!= scores + numScores)
      7. {
      8. tot += \*(ptr+i);
      9. i++;
      10. }
      11. return tot / numScores;
      12. }
   2. **Part B**
      1. const char\* findTheChar(const char\* str, char chr)
      2. {
      3. for (int k = 0; \*(str + k) != 0; k++)
      4. {
      5. if (\*(str + k) == chr)
      6. return (str + k);
      7. }
      8. return nullptr;
      9. }
   3. **Part C**
      1. const char\* findTheChar(const char\* str, char chr)
      2. {
      3. while (\*str != 0)
      4. {
      5. if (\*str == chr)
      6. return str;
      7. else
      8. str++;
      9. }
      10. return nullptr;
      11. }
4. **Question 4**
   * + 1. int main()
       2. { 0 1 2 3 4 5
       3. int array[6] = { 5, 3, 4, 17, 22, 19 }; //Initializes int array of 6 values
       4. int\* ptr = maxwell(array, &array[2]); //maxwell compares the values of the 0th and 2nd positions of the array, and returns the pointer to the larger value of the two (5>4), initializing ptr to address of the 0th element in the array
       5. \*ptr = -1; //sets the value that ptr points to to -1 (0th position now = -1)
       6. ptr += 2; //moves ptr to the 2nd position in the array
       7. ptr[1] = 9; //sets the value that (ptr+1) points equal to 9, which sets the 3rd position of the array equal to the value 9 (3rd position = 9)
       8. \*(array+1) = 79; //temporarily moves array, which is a pointer that points to the 0th position of the array, ahead 1 spot and is pointing at the 1st position in the array, and sets that value equal to 79 (1st position now = 79)
       9. cout << &array[5] - ptr << endl; //outputs the difference between memory addresses of the last element in the array (position 5) and the memory address where ptr is currently at, which is equivalent to &array[2], and thus, according to the property, &a[i] - &a[j] ⇒ i-j, &array[5] - &array[2] 🡺 5 -2 == the output is:
          1. 3
       10. swap1(&array[0], &array[1]); //This call to swap1 does nothing in changing the main function as the swap1 function does not return anything, nor does it change the address values of the pointers, as the function uses call by value instead of call by reference
       11. swap2(array, &array[2]); //This function swap2 swaps the value that the pointer array points to (points to the value -1) with the value that is found when following the address of &array[2] (the value 4), I.E. the 0th position of the array now equals 4 and now the 2nd position of the array now equals -1
       12. for (int i = 0; i < 6; i++)
       13. cout << array[i] << endl;
       14. } //the array, after all these actions is the following: array[6] = { 4, 79, -1, 9, 22, 19}, thus the output at this line is:
           1. 4
           2. 79
           3. -1
           4. 9
           5. 22
           6. 19
     1. Thus, the total output of this program is:
        + 1. 3
          2. 4
          3. 79
          4. -1
          5. 9
          6. 22
          7. 19
5. **Question 5 //I added comments for myself, I know we didn’t need to**
   * + 1. void removeS(char\* cString)
       2. {
       3. char\* ptr = cString;
       4. while (\*ptr != '\0') //makes sure ptr isn’t at the zero byte
       5. {
       6. if (\*ptr == 'S' || \*ptr == 's') //if an s or S
       7. {
       8. for (; \*ptr != '\0'; ptr++) //loop through array starting from the s/S
       9. {
       10. \*ptr = \*(ptr + 1); //move all chars back 1, removing s/S
       11. }
       12. ptr = cString; //reset pointer to same spot & continue
       13. }
       14. else
       15. ptr++; //if its not an s/S at the spot, go to the next spot
       16. }
       17. }