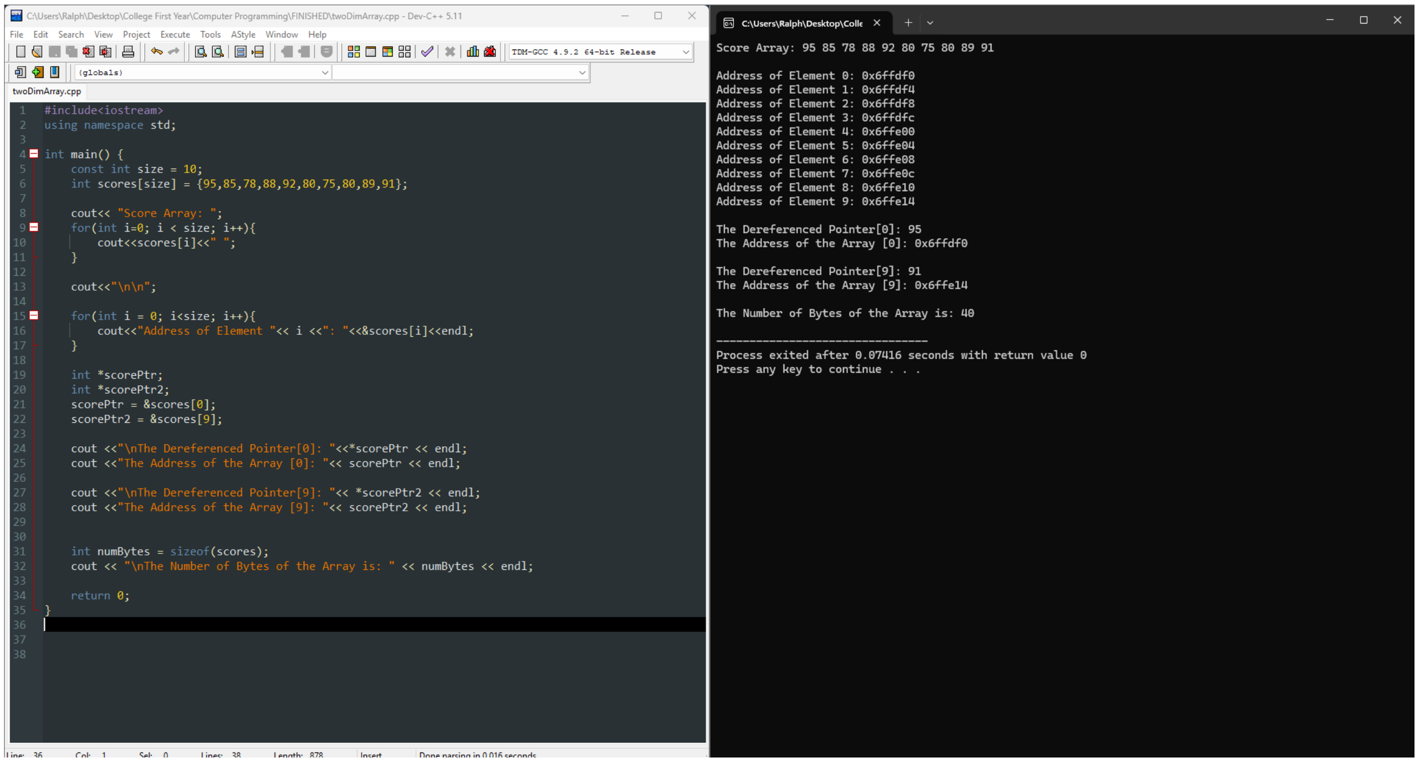
Seatwork 4.2	
Pointers	
Course Code: CPE007	**Program:** Computer Engineering
Course Title: Programming Logic & Design	**Date Performed: 9/18/2025**
Section: CPE11S1	**Date Submitted: 9/18/2025**
Name(s): Ralph Angelov F. Braganza	**Instructor: Engr. Jimlord M. Quejado**
Output	

CODE:

```
#include<iostream>
using namespace std;

int main() {
   const int size = 10;
   int scores[size] = {95,85,78,88,92,80,75,80,89,91};

   cout<< "Score Array: ";
   for(int i=0; i < size; i++){
      cout<<scores[i]<<" ";
   }

   cout<<"\n\n";

   for(int i = 0; i<size; i++){
      cout<<"Address of Element "<< i <<": "<<&scores[i]<<endl;
   }

   int *scorePtr;
   int *scorePtr2;
         scorePtr = &scores[0];
         scorePtr2 = &scores[9];

         cout <<"\nThe Dereferenced Pointer[0]: "<<*scorePtr << endl;
         cout <<"The Address of the Array [0]: "<< scorePtr << endl;

         cout <<"\nThe Dereferenced Pointer[9]: "<< *scorePtr2 << endl;
         cout <<"The Address of the Array [9]: "<< scorePtr2 << endl;


         int numBytes = sizeof(scores);
         cout << "\nThe Number of Bytes of the Array is: " << numBytes << endl;

         return 0;
}
```

RESULT:



C:\Users\Ralph\Desktop\College First Year\Computer Programming\FINISHED\twoDimArray.cpp - Dev-C++ 5.11

```cpp
#include<iostream>
using namespace std;

int main() {
    const int size = 10;
    int scores[size] = {95,85,78,88,92,80,75,80,89,91};

    cout<< "Score Array: ";
    for(int i=0; i < size; i++){
        cout<<scores[i]<<" ";
    }

    cout<<"\n\n";

    for(int i = 0; i<size; i++){
        cout<<"Address of Element "<< i <<": "<<&scores[i]<<endl;
    }

    int *scorePtr;
    int *scorePtr2;
    scorePtr = &scores[0];
    scorePtr2 = &scores[9];

    cout <<"\nThe Dereferenced Pointer[0]: "<<*scorePtr << endl;
    cout <<"The Address of the Array [0]: "<< scorePtr << endl;

    cout <<"\nThe Dereferenced Pointer[9]: "<< *scorePtr2 << endl;
    cout <<"The Address of the Array [9]: "<< scorePtr2 << endl;


    int numBytes = sizeof(scores);
    cout << "\nThe Number of Bytes of the Array is: " << numBytes << endl;

    return 0;
}
```

```
Score Array: 95 85 78 88 92 80 75 80 89 91

Address of Element 0: 0x6ffdf0
Address of Element 1: 0x6ffdf4
Address of Element 2: 0x6ffdf8
Address of Element 3: 0x6ffdfc
Address of Element 4: 0x6ffe00
Address of Element 5: 0x6ffe04
Address of Element 6: 0x6ffe08
Address of Element 7: 0x6ffe0c
Address of Element 8: 0x6ffe10
Address of Element 9: 0x6ffe14

The Dereferenced Pointer[0]: 95
The Address of the Array [0]: 0x6ffdf0

The Dereferenced Pointer[9]: 91
The Address of the Array [9]: 0x6ffe14

The Number of Bytes of the Array is: 40

--------------------------------
Process exited after 0.07416 seconds with return value 0
Press any key to continue . . .
```

Supplementary Activity

Analysis:

We have all the regular functions so the code could run #include <iostream> then int main (), we made a constant integer called size that is equal to 10 so when don't have to keep typing 10 everywhere and better readability of the code. The array has 10 elements because scores[size] (size = 10) then the elements are indexed from 0 to 9 (the count always starts at 0). The first for-loop will print out the score array which we have tackled many times in the past but I will explain it anyways, int i = 0 and that will print out the first element value of the array which is 95 then i++ increments it by 1 to print the next element value of the array until i < size. This is where new territory is discovered (at least for me) wherein our second for-loop will print the element number i and the memory address of each element in the array (&scores[i]). The & operator before scores[i] is the memory address operator where it will give you the address of the variable. It won't give you the same output every single time, it depends on the PC and what you have configured on it. The next lines of code will be about pointers, we initialized that int *scorePtr and *scorePtr2 and the " * "before scorePtr and scorePtr is the dereference operator that shows the value of that address. Let's say that the address is 0x6ffdf0 and instead of printing "0x6ffdf0" it will print out the value of that address which in my case it is 95.

That is how we get the following output:

The Dereferenced Pointer[0]: 95
The Address of the Array [0]: 0x6ffdf0

The Dereferenced Pointer[9]: 91
The Address of the Array [9]: 0x6ffe14

Just to emphasize
scorePtr = the address
*scorePtr = the value at that address

The following line of code will be checking the array size, numBytes will be the name of the number of bytes we have in our array and it is declared as an integer. Essentially what sizeof does is retrieve the size in bytes, our scores is declared as an integer meaning it has 4 bytes (I got that from the lecture) and that will be multiplied by 10 because we have 10 elements in the array (this is basically the total size of the whole array).

Here is the formula (I also got this from the lecture)

sizeof(scores)=number of elements × sizeof(element_type)

Then we will print out the text “"\nThe Number of Bytes of the Array is: " (\n is for spacing) then print the total size of bytes in the array which is 40.

Conclusion

During this seatwork I learned a few new things from this, which are the & operator and the * operator. It took me a while to realize that most of us have different addresses of outputs when we print it, which made me confused when others print their outputs. I’ve also learned about byte sizes for different types and how it works when we use sizeof() to be able to identify/print out the total size of bytes that are in the array. It took me quite a while to understand and analyze how the code works because I never knew that the values in the arrays also have addresses and then convert that address into its value. The procedure was effective; it allowed me to practice and understand these new operators that were introduced to me to move on to printing the memory address and apply pointers to see the connection between the variable's address and values. I didn’t really modify the code, but I just fixed the spacing between everything to make sure everything looks clean and neat for the output of it. Overall, I believe I did somewhat okay in trying to understand how this works; of course, I still need to practice improving myself to a more advanced level. This was a good roadmap for me to get the general concept of what we are trying to do with this code, and I hope I’ll be able to improve more in the future as I move forward to more advanced ways to use these operators and functions.