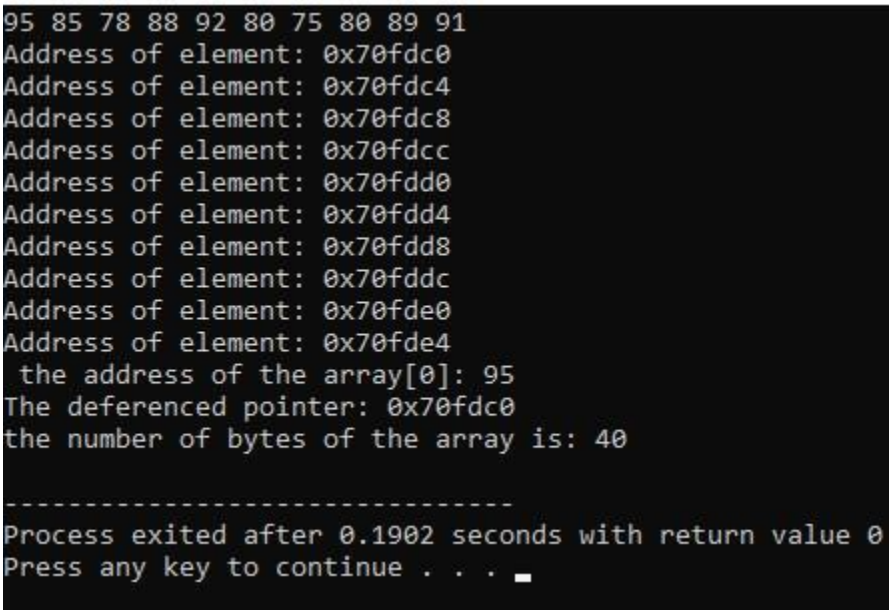
Seatwork 4.2	
Pointers	
Course Code: CPE007	**Program:** Computer Engineering
Course Title: Programming Logic and Design	**Date Performed: 09/18/25**
Section: CPE11S1	**Date Submitted: 09/18/25**
Name(s): James Daniel M Verano	**Instructor: Engr. Jimlord M. Quejado**
6. Output	

Code Snippet:

```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};

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

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

    int *scorePtr;
    scorePtr = &scores[0];
    cout << " the address of the array[0]: "<<  *scorePtr << endl;
    cout << "The deferenced pointer: " << scorePtr << endl;

    int numBytes = sizeof(scores);
    cout << "the number of bytes of the array is: " << numBytes << endl;

    return 0;
}
```

Outputs:



7. Supplementary Activity

Analysis:

Based on the recent discussion, we know that pointers are functions that take the address of a specific array. Every array element has each of their unique addresses, and pointers take them to point them to their element value. The character "&" is what is defined as the character between the variable to be able to take the address of the array element. As you can see in the code snippet, we used a for loop function where, once we printassign and index to their respective array element values, we created an output function where we will be printing the address of the array elements using the "&scores[i]"

for visuals:

&scores[i] = &scores[0] = Takes the address of the element value in array[0] / index 0.

Now, in the second code, we can see that we declared another variable, "int *scorePtr", According to the definition of Pointers in the PPT, "a pointer is a special variable that stores the memory address of another variable." in this line code, line 18 to 21, is where we are storing the address and pointing out the array's element value and printing both the address and element array. Using the dereference Operator.

for visuals:

int *scorePtr = points out the address's array's value
scorePtr = &scores[0] = takes the address of the array's value

In the last line code function, line 23 -24, which is the using of formula in identifying of storage consumption of arrays. In order to find the memory bytes of an array, we can use this formula: Memory (bytes) = Number of elements x Size of each element. This function is also known as the (sizeof).

for visuals:

int numBytes = sizeof(scores); = sizeof(int = 4) * 10 ⇒ scores[10].
⇒ 40 bytes.

8. Conclusion

In this lesson, I was able to implement the function of the pointers in terms of storing an address, address operator, and dereference; And also the type of procedure to get the storage consumption or arrays. As given in the program above, I was able to successfully present the intended outputs in all pointers. Especially in the dereference function, wherein we take the address and point out the array's value of the taken address to its value of the array element. I think I was able to attain the intended learning outcomes, and the areas that I still need to improve is that, to implement the pointer's structure into a real program wherein, it will be more focused into a direct take and point of array values for a more accurate data storing and pointing.