Week 1 Assignment on C Programming

Name: Alain Niganze
Email: niganzealain@gmail.com

Date: 13 june 2024

Summary of Activities:

1. **Exercise 1: Change value via a pointer**
   - **Objective:** The goal of this exercise was to demonstrate how pointers can be used to change and read the value of an integer. Specifically, it involved defining an integer `x` and two pointers to that integer, modifying `x` via one pointer, and then reading the modified value via the other pointer.
   - **Explanation:** In the `pointer.c` program, I initialized an integer `x` and two pointers `p` and `q`. Both pointers were set to point to `x`. by then changed the value of `x` through the pointer `p` and printed the updated value using pointer `q`. This exercise highlighted how multiple pointers can reference the same memory location and how changes through one pointer are reflected when accessed via another.

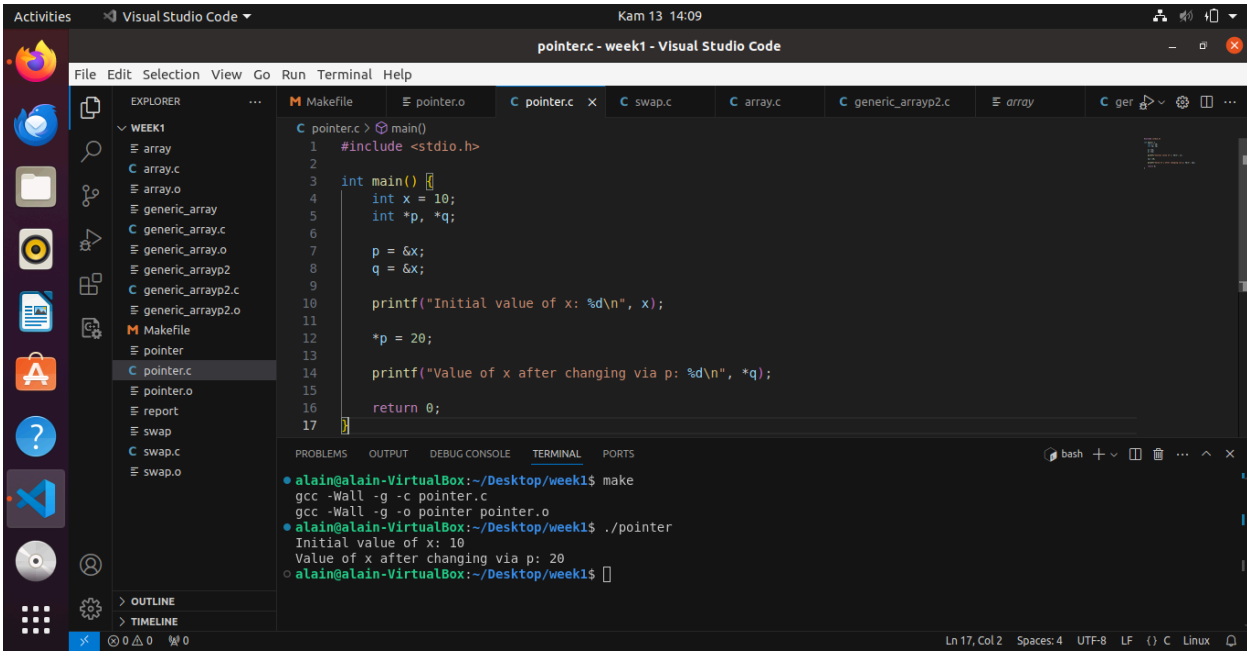
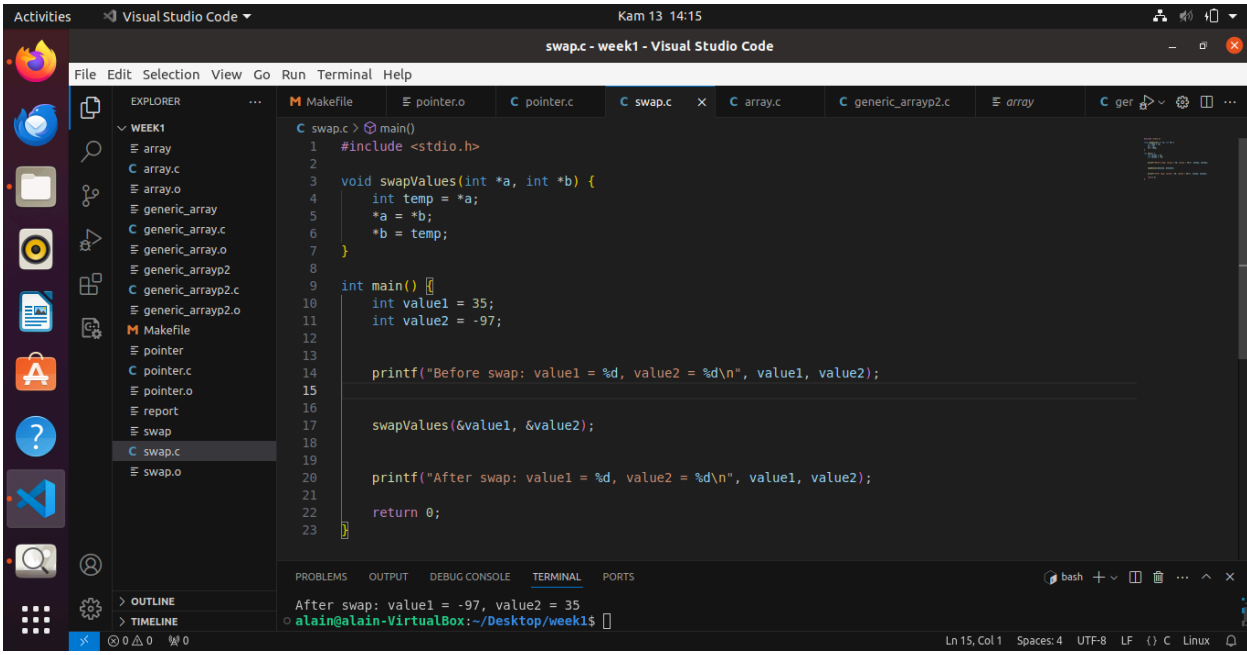


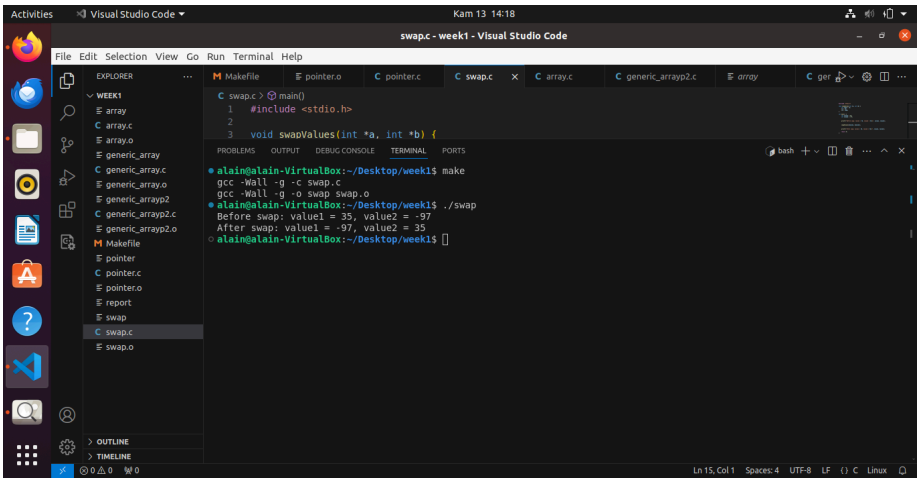
Figure1: Shows with combined the code and result on pointer function see in terminal.

2. **Exercise 2: Value swap**

- **Objective:** The objective here was to create a function that swaps the values of two integers using pointers. This approach is necessary when a function needs to return more than one value.
- **Explanation:** The `swap.c` program defined a function `swapValues` that takes two integer pointers as parameters. Within the function, the values pointed to by these pointers are swapped using a temporary variable. This technique is essential for in-place modification of the variables passed to the function. The main function demonstrates this by swapping the values of two integers and printing the results before and after the swap.

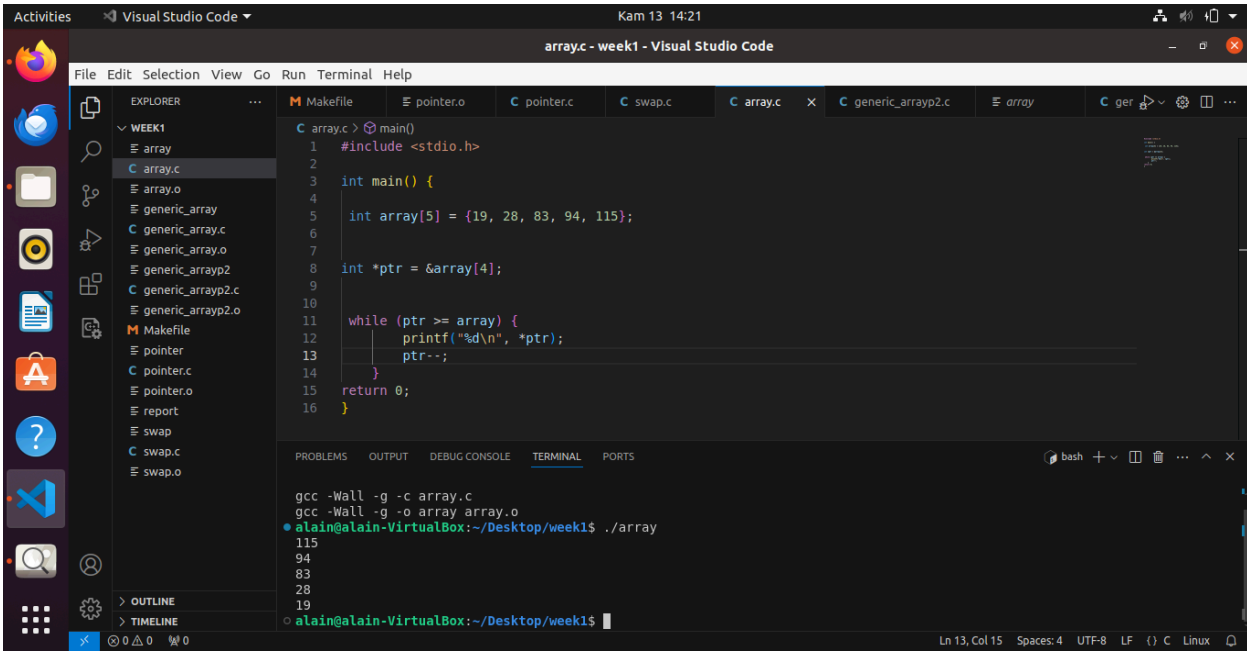


- Figure2:shows the code snipped on swap function
  And here is the screenshot thst show the output after testing our code from make file , see…..



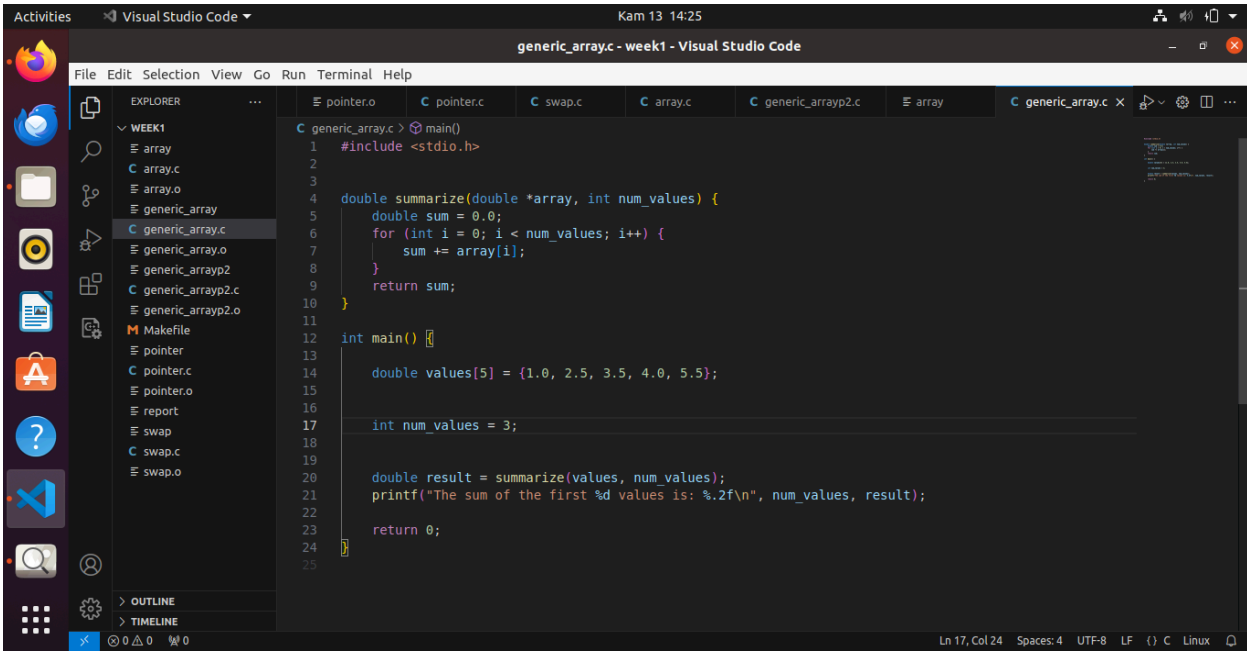
- Figure3:after making the compilation we see that on this screen we achieve our desirable output.
-

3. **Exercise 3: Pointer arithmetic on array**
   - **Objective:** The task was to traverse an array in reverse order using a pointer without employing a loop counter. This exercise aimed to practice pointer arithmetic.
   - **Explanation:** In `array.c`, an array of 5 integers was declared along with a pointer that initially pointed to the last element of the array. A while loop was then used to traverse the array in reverse by decrementing the pointer. Each value was printed as the pointer moved from the end to the beginning of the array. This exercise demonstrated how to manipulate and navigate arrays using pointers.

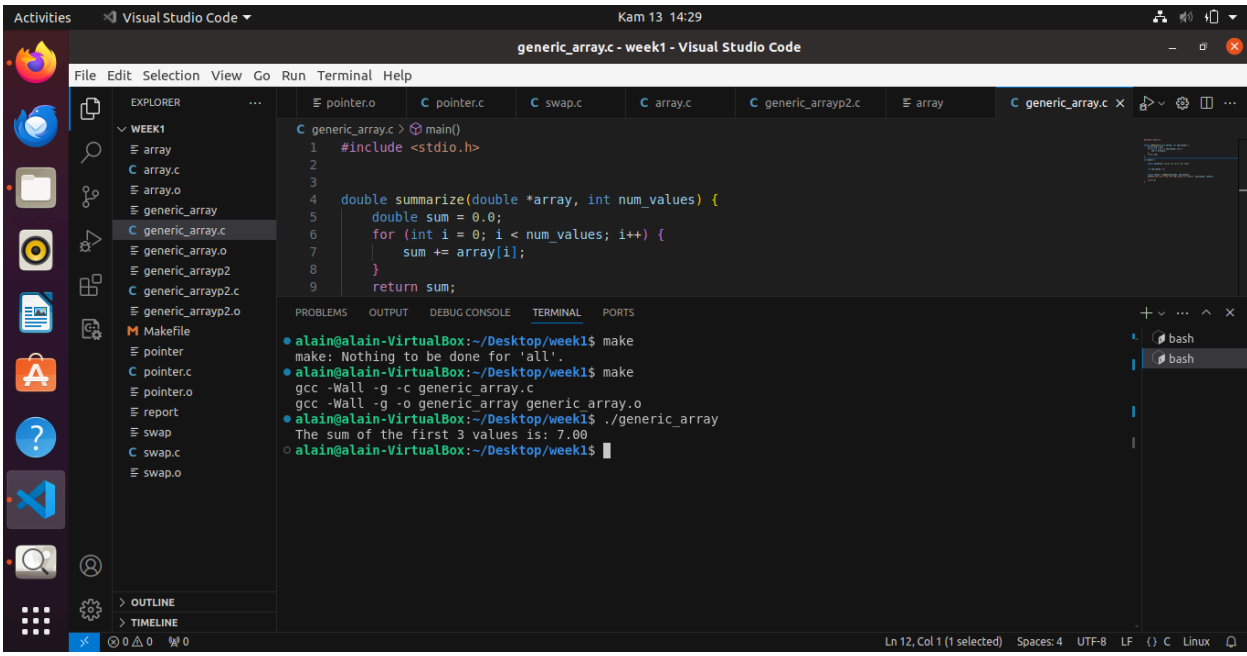


   - Figure4:shows the both code and result from the terminal.

4. **Exercise 4: Generic array add**
   - **Objective:** This exercise involved writing a function to sum a specified number of values in an array of doubles and return the result. This task was meant to showcase basic array manipulation and function return values.
   - **Explanation:** In `generic_array.c`, a function `summarize` was defined to take an array of doubles and the number of values to sum. The function iterates through the specified number of elements, calculating the total sum, and returns this sum. The main function provides an example array and calls the `summarize` function, printing the result.



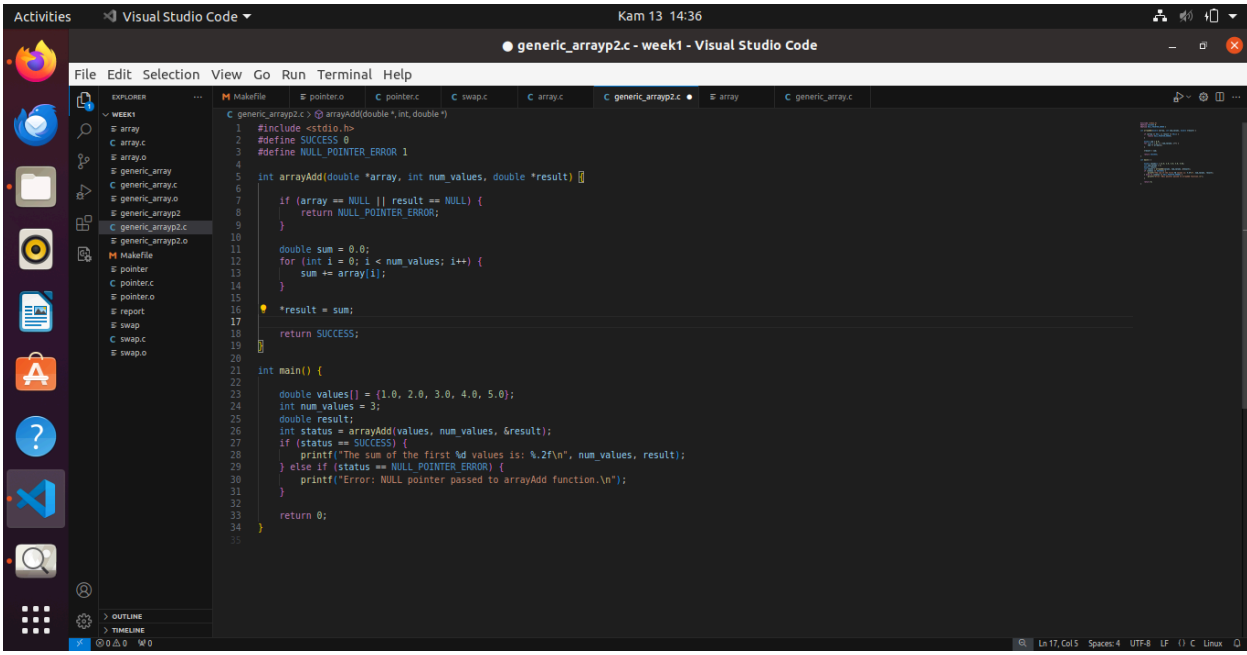
- Figure5:show the code the array of doubles and return the result
  So now let us drop down the result to see is this code above is really works .



- Figure6: shows actually result from array with functionality of pointer.

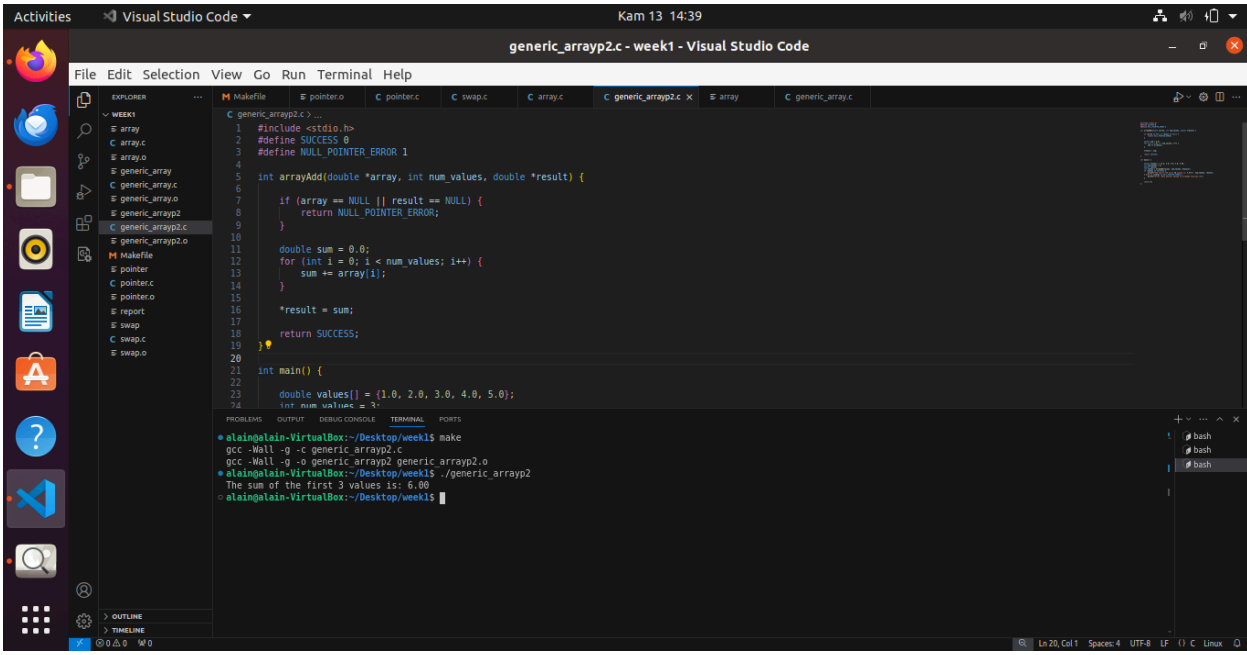
5. **Exercise 5: Generic array add (part 2)**
   - **Objective:** This exercise aimed to modify the previous function to return the result via a parameter and handle errors like NULL pointers. This approach is useful when a function needs to return multiple results or indicate errors.

- **Explanation:** The `generic_arrayp2.c` program introduced an enhanced version of the previous function called `arrayAdd`. This function accepts an additional parameter to store the result and checks for NULL pointers to handle potential errors. If no errors are found, it sums the specified number of values in the array and stores the result in the provided memory location. The main function demonstrates the usage of `arrayAdd`, checking the returned status and printing the result or an error message accordingly.



- Figure 7 shows the enhancement of code from exercise number 4 with passed the result as third parameter.

  Let us now see what we can in terminal as the result



○

Reflection:

- **Challenges Faced:** Some challenges included understanding and correctly implementing pointer arithmetic and ensuring proper handling of edge cases like NULL pointers.
- **Learnings:** Through these exercises, I gained a deeper understanding of how pointers work in C, how to manipulate arrays using pointers, and how to handle multiple return values and errors in functions.
- **Next Steps:** I plan to continue exploring more advanced topics in C programming, such as dynamic memory allocation, linked lists, and file handling, to build a stronger foundation.