PARADOX TECHNOLOGY HUB

ASSIGNMENT 1 FOR WEEK1

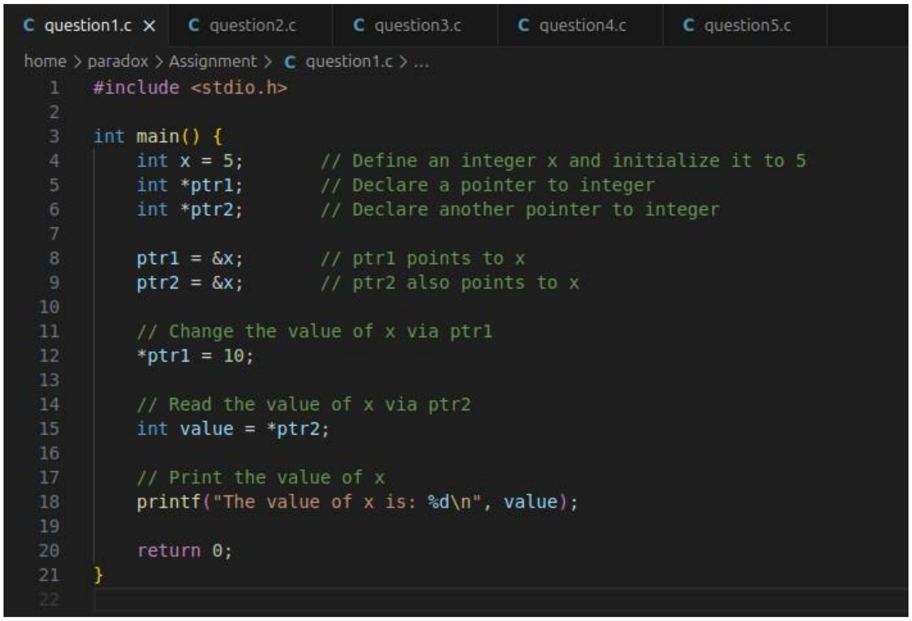
WEEK1 REPORT

Robotic Team

Question1

Write a program to define an integer x and two pointers to integer. Both pointers should point to the integer x. Change the integer value via one pointer and read it back via the other pointer.

Answer;



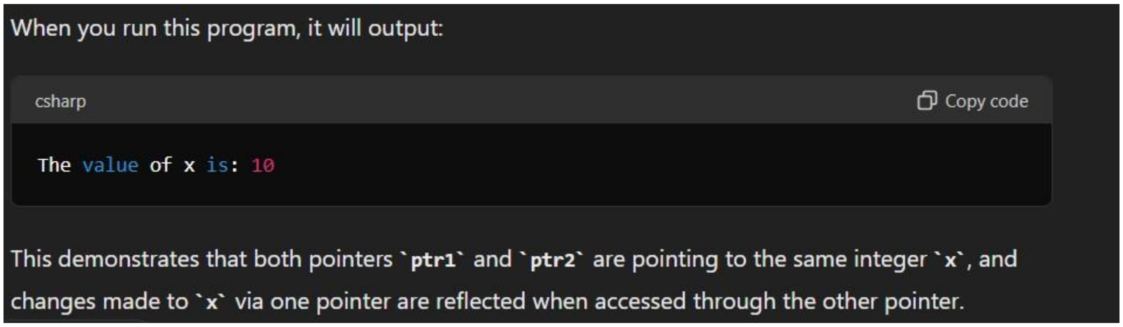
```c
#include <stdio.h>

int main() {
    int x = 5;       // Define an integer x and initialize it to 5
    int *ptr1;       // Declare a pointer to integer
    int *ptr2;       // Declare another pointer to integer

    ptr1 = &x;       // ptr1 points to x
    ptr2 = &x;       // ptr2 also points to x

    // Change the value of x via ptr1
    *ptr1 = 10;

    // Read the value of x via ptr2
    int value = *ptr2;

    // Print the value of x
    printf("The value of x is: %d\n", value);

    return 0;
}

```

Figure 1: Question one codes in vs

Explanation:

1. **Initialization**:
    - int x = 5;: Define an integer x and initialize it to 5.
    - int *ptr1;: Declare a pointer to an integer.
    - int *ptr2;: Declare another pointer to an integer.
2. **Pointer Assignment**:
    - ptr1 = &x;: Assign the address of x to ptr1.
    - ptr2 = &x;: Assign the address of x to ptr2. Now both ptr1 and ptr2 point to x.
3. **Modify the Value**:
    - *ptr1 = 10;: Dereference ptr1 and assign the value 10 to x. This changes the value of x via ptr1.
4. **Read the Value**:
    - int value = *ptr2;: Dereference ptr2 to read the value of x. The value is stored in the variable value.

5. **Output**:
    - printf("The value of x is: %d\n", value);: Print the value of x.



When you run this program, it will output:

```
The value of x is: 10
```

This demonstrates that both pointers `ptr1` and `ptr2` are pointing to the same integer `x`, and changes made to `x` via one pointer are reflected when accessed through the other pointer.

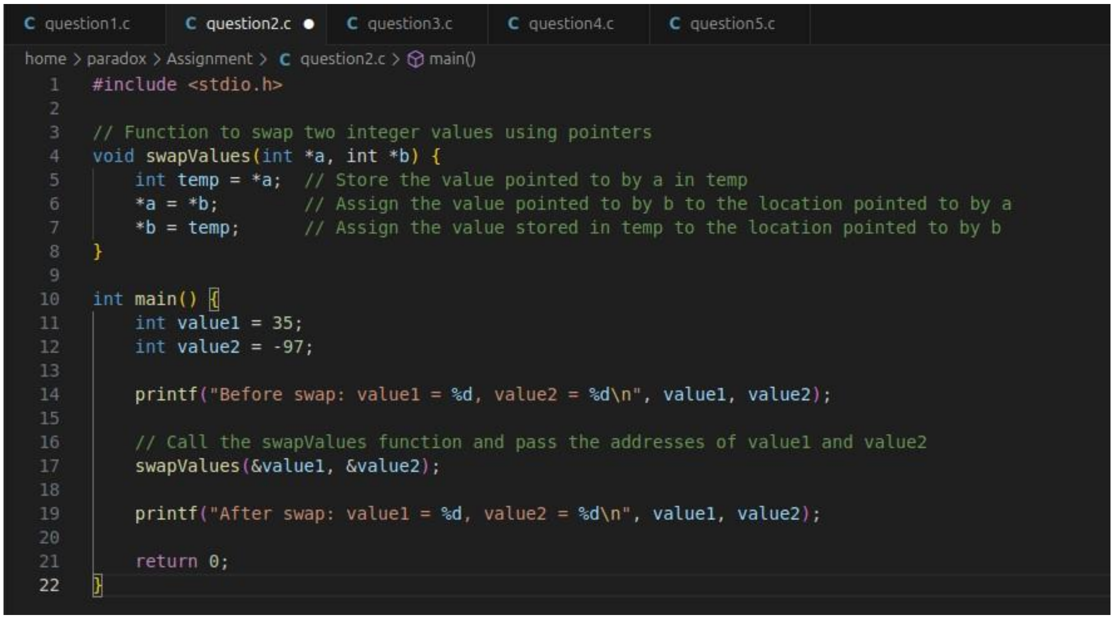
Figure 2: Output for code 1

Question2

When you want to return a value from a function, you can simply return that value. What happens when you need to return more than one value? In that case you can use a pointer type parameter. Write a function that can swap two integer values. The function is called swapValues(), the following code snippet explains what it does:

int value1 = 35;
int value2 = -97;
swapValues(….,…….);
// now value1 equals -97 and value2 equals 35.

Answer;

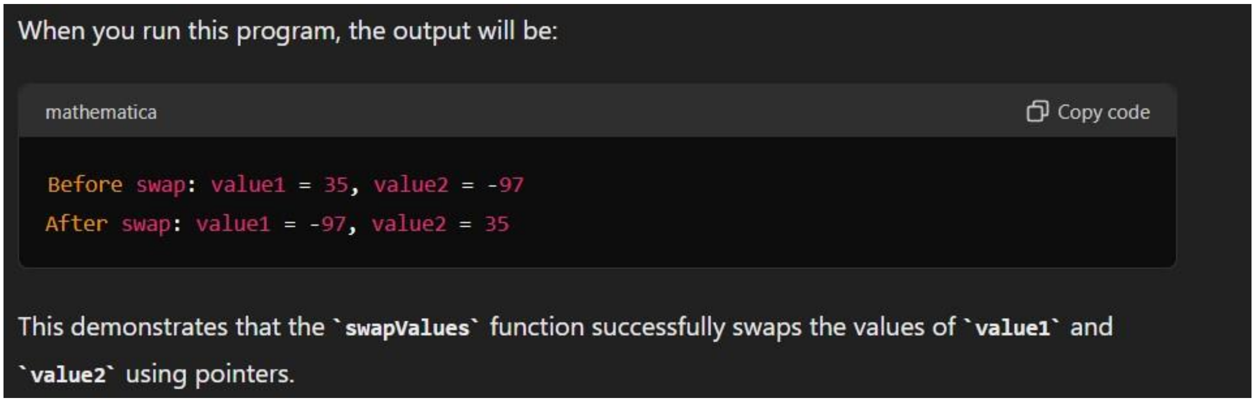


```c
#include <stdio.h>

// Function to swap two integer values using pointers
void swapValues(int *a, int *b) {
    int temp = *a;  // Store the value pointed to by a in temp
    *a = *b;        // Assign the value pointed to by b to the location pointed to by a
    *b = temp;      // Assign the value stored in temp to the location pointed to by b
}

int main() {
    int value1 = 35;
    int value2 = -97;

    printf("Before swap: value1 = %d, value2 = %d\n", value1, value2);

    // Call the swapValues function and pass the addresses of value1 and value2
    swapValues(&value1, &value2);

    printf("After swap: value1 = %d, value2 = %d\n", value1, value2);

    return 0;
}
```

Figure 3: Question two codes in vs

Explanation:

1. **Function Definition**:
    - void swapValues(int *a, int *b): This function takes two pointers to integers as parameters.
    - int temp = *a;: Store the value pointed to by a in a temporary variable temp.
    - *a = *b;: Assign the value pointed to by b to the location pointed to by a.
    - *b = temp;: Assign the value stored in temp to the location pointed to by b.
2. **Main Function**:
    - int value1 = 35;: Initialize value1 to 35.
    - int value2 = -97;: Initialize value2 to -97.
    - printf("Before swap: value1 = %d, value2 = %d\n", value1, value2);: Print the values of value1 and value2 before swapping.
    - swapValues(&value1, &value2);: Call the swapValues function, passing the addresses of value1 and value2 as arguments.
    - printf("After swap: value1 = %d, value2 = %d\n", value1, value2);: Print the values of value1 and value2 after swapping.
3. **Output**



When you run this program, the output will be:

```
Before swap: value1 = 35, value2 = -97
After swap: value1 = -97, value2 = 35
```

This demonstrates that the `swapValues` function successfully swaps the values of `value1` and `value2` using pointers.

Figure 4:: Output for code 2

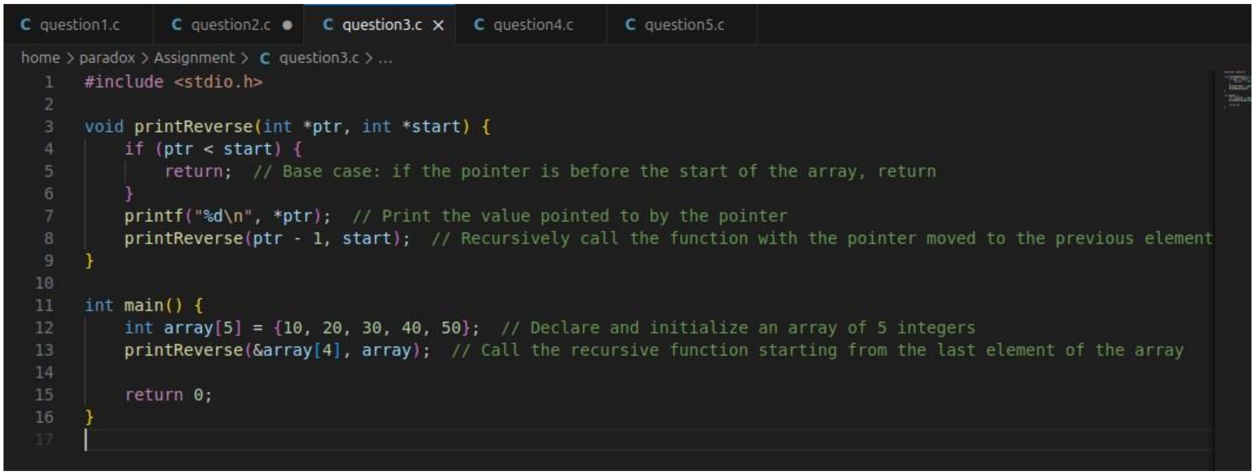
Question3

Write a small program that declares an array of 5 integers and declare a pointer that points to the last element of the array. Write a loop that traverses the array via the pointer in reverse direction.This means that a loop such as this is not allowed:

```
for (int i = 4; i >= 0; i--)

{

 printf("%d\n", array[i]);

}
```

In other words: you are not allowed to use a loop counter, you can only use the pointer you have.

Answer;



```c
#include <stdio.h>

void printReverse(int *ptr, int *start) {
    if (ptr < start) {
        return;  // Base case: if the pointer is before the start of the array, return
    }
    printf("%d\n", *ptr);  // Print the value pointed to by the pointer
    printReverse(ptr - 1, start);  // Recursively call the function with the pointer moved to the previous element
}

int main() {
    int array[5] = {10, 20, 30, 40, 50};  // Declare and initialize an array of 5 integers
    printReverse(&array[4], array);  // Call the recursive function starting from the last element of the array

    return 0;
}
```

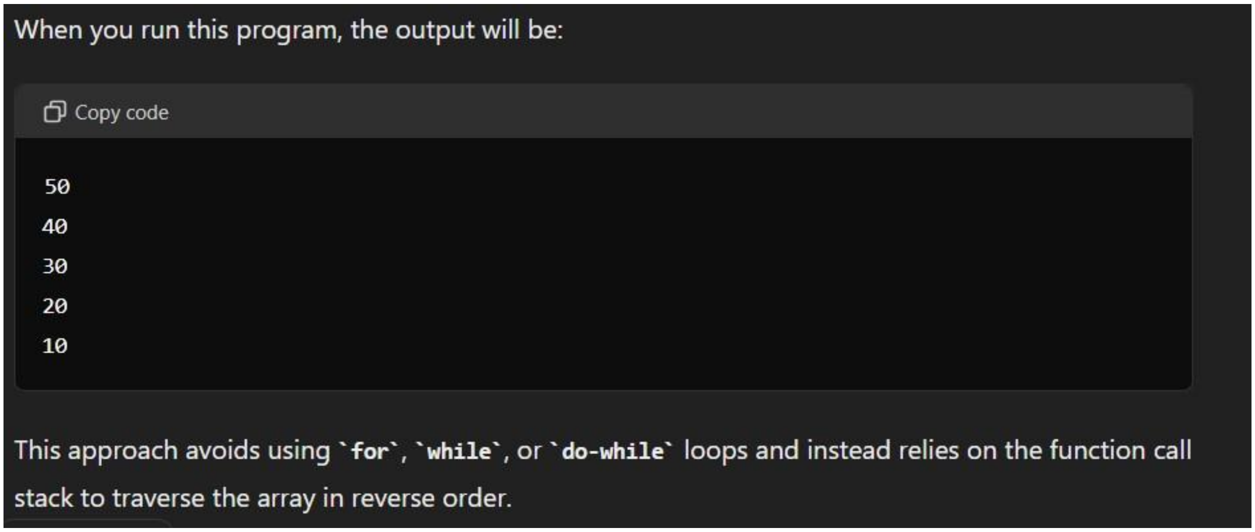
Figure 5: Question three codes in vs

Explanation:

1. **Recursive Function Definition**:
   - void printReverse(int *ptr, int *start): This function takes two pointers as parameters. ptr points to the current element to be printed, and start points to the first element of the array.
   - if (ptr < start) { return; }: This is the base case for the recursion. If the pointer ptr is before the start of the array, the function returns without doing anything.
   - printf("%d\n", *ptr);: This prints the value currently pointed to by ptr.
   - printReverse(ptr - 1, start);: This is the recursive call that moves the pointer to the previous element.
2. **Main Function**:

- int array[5] = {10, 20, 30, 40, 50};: Declare and initialize an array of 5 integers.
- printReverse(&array[4], array);: Call the printReverse function, starting from the last element of the array (&array[4]) and passing the pointer to the first element of the array (array).

3. **Output**



When you run this program, the output will be:

```
50
40
30
20
10
```

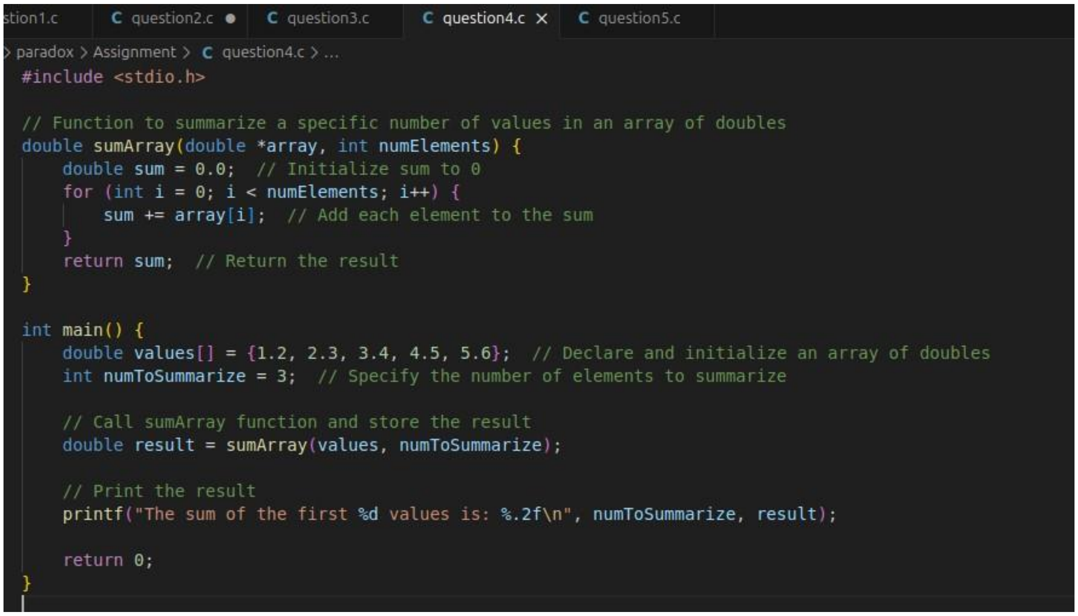
This approach avoids using `for`, `while`, or `do-while` loops and instead relies on the function call stack to traverse the array in reverse order.

Figure 6: : Output for code 3

Question4

Write a function that can summarize a specific number of values in an array of doubles and return the result.

Answer;

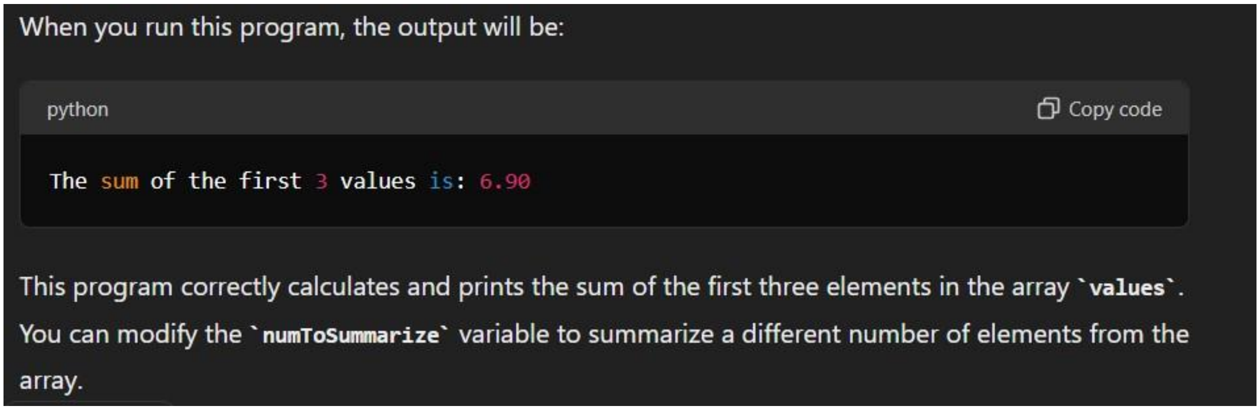


```c
#include <stdio.h>

// Function to summarize a specific number of values in an array of doubles
double sumArray(double *array, int numElements) {
    double sum = 0.0;  // Initialize sum to 0
    for (int i = 0; i < numElements; i++) {
        sum += array[i];  // Add each element to the sum
    }
    return sum;  // Return the result
}

int main() {
    double values[] = {1.2, 2.3, 3.4, 4.5, 5.6};  // Declare and initialize an array of doubles
    int numToSummarize = 3;  // Specify the number of elements to summarize

    // Call sumArray function and store the result
    double result = sumArray(values, numToSummarize);

    // Print the result
    printf("The sum of the first %d values is: %.2f\n", numToSummarize, result);

    return 0;
}
```

Figure 7: Question four codes in vs

Explanation:

1. **Function Definition** (sumArray):
    - double sumArray(double *array, int numElements): This function takes a pointer to an array of doubles and the number of elements to summarize.
    - double sum = 0.0;: Initialize the sum to 0.
    - for (int i = 0; i < numElements; i++) { sum += array[i]; }: Loop through the first numElements elements of the array, adding each element to the sum.
    - return sum;: Return the calculated sum.
2. **Main Function**:
    - double values[] = {1.2, 2.3, 3.4, 4.5, 5.6};: Declare and initialize an array of doubles.
    - int numToSummarize = 3;: Specify the number of elements to summarize.
    - double result = sumArray(values, numToSummarize);: Call the sumArray function and store the result in result.
    - printf("The sum of the first %d values is: %.2f\n", numToSummarize, result);: Print the result.
3. **Output**



When you run this program, the output will be:

```
The sum of the first 3 values is: 6.90
```

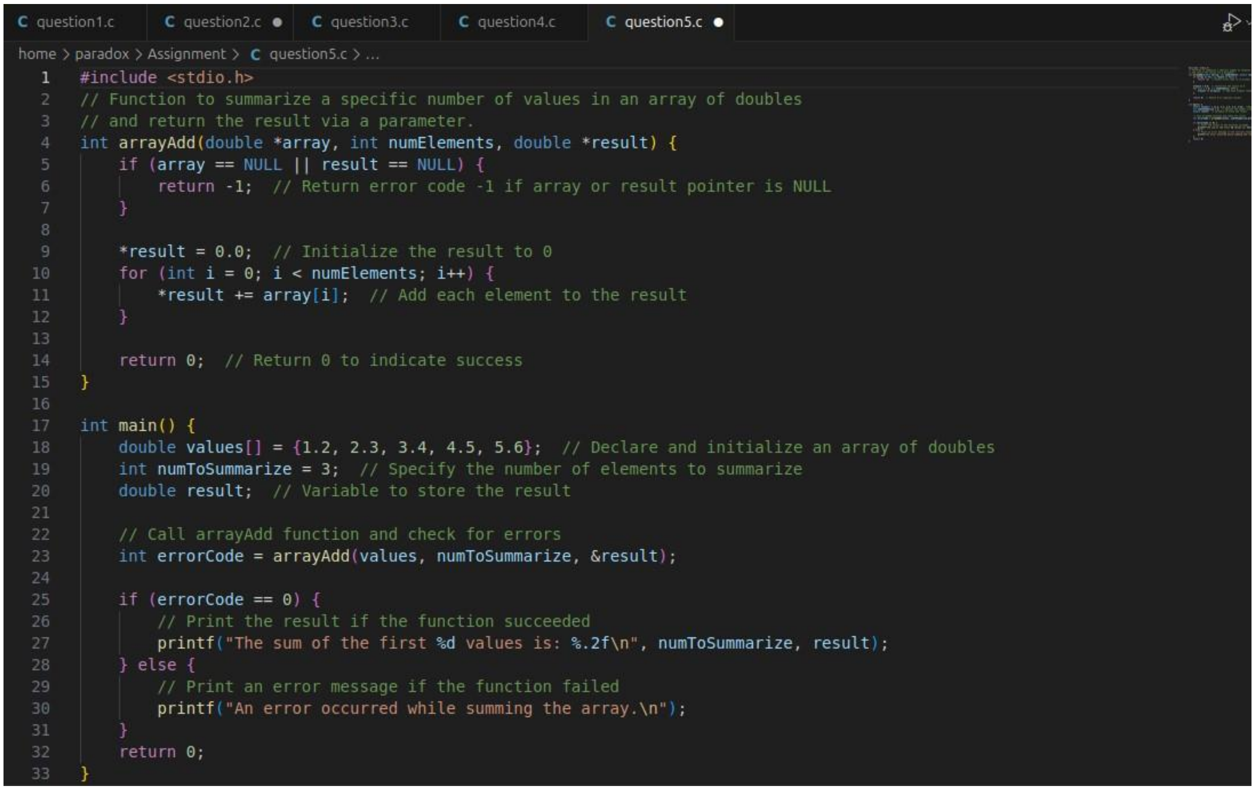
This program correctly calculates and prints the sum of the first three elements in the array `values`. You can modify the `numToSummarize` variable to summarize a different number of elements from the array.

Figure 8: : Output for code 4

Question5

Another way of writing the function in the previous assignment would be by returning the result via a parameter. A reason to do this is if you need to return more than one thing. For example: what would arrayAdd() do if it is called with a NULL pointer? It has no way of telling the caller that an error occurred. Please change the arrayAdd() function according this new specification.

Answer;



```c
#include <stdio.h>
// Function to summarize a specific number of values in an array of doubles
// and return the result via a parameter.
int arrayAdd(double *array, int numElements, double *result) {
    if (array == NULL || result == NULL) {
        return -1;  // Return error code -1 if array or result pointer is NULL
    }

    *result = 0.0;  // Initialize the result to 0
    for (int i = 0; i < numElements; i++) {
        *result += array[i];  // Add each element to the result
    }

    return 0;  // Return 0 to indicate success
}

int main() {
    double values[] = {1.2, 2.3, 3.4, 4.5, 5.6};  // Declare and initialize an array of doubles
    int numToSummarize = 3;  // Specify the number of elements to summarize
    double result;  // Variable to store the result

    // Call arrayAdd function and check for errors
    int errorCode = arrayAdd(values, numToSummarize, &result);

    if (errorCode == 0) {
        // Print the result if the function succeeded
        printf("The sum of the first %d values is: %.2f\n", numToSummarize, result);
    } else {
        // Print an error message if the function failed
        printf("An error occurred while summing the array.\n");
    }
    return 0;
}
```

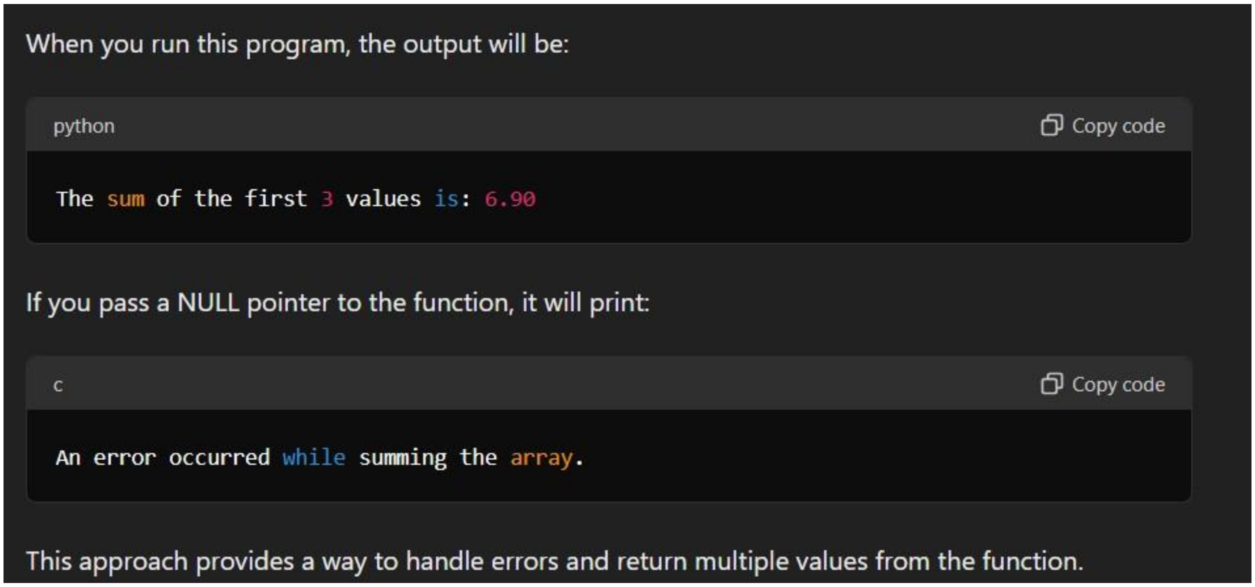
Figure 9: Question five codes in vs

Explanation:

1. **Function Definition** (arrayAdd):
   - int arrayAdd(double *array, int numElements, double *result): This function takes a pointer to an array of doubles, the number of elements to summarize, and a pointer to a double where the result will be stored. It returns an int error code.
   - if (array == NULL || result == NULL) { return -1; }: Check if either the array or result pointers are NULL. If so, return an error code -1.
   - *result = 0.0;: Initialize the result to 0.
   - for (int i = 0; i < numElements; i++) { *result += array[i]; }: Loop through the first numElements elements of the array, adding each element to the result.
   - return 0;: Return 0 to indicate success.

2. **Main Function**:
   - double values[] = {1.2, 2.3, 3.4, 4.5, 5.6};: Declare and initialize an array of doubles.
   - int numToSummarize = 3;: Specify the number of elements to summarize.
   - double result;: Declare a variable to store the result.
   - int errorCode = arrayAdd(values, numToSummarize, &result);: Call the arrayAdd function and store the error code.
   - if (errorCode == 0) { ... } else { ... }: Check the error code and print the result if successful, or an error message if there was an error.

3. **Output**



When you run this program, the output will be:

```python
The sum of the first 3 values is: 6.90
```

If you pass a NULL pointer to the function, it will print:

```c
An error occurred while summing the array.
```

This approach provides a way to handle errors and return multiple values from the function.

Figure 10: : Output for code 5