Q.1 NOTE: The program itself is correct but Debug in such a manner that would provide an optimal solution in case the program became large.

C Program: Calculate Average of Array

```c
#include <stdio.h>
double calculateAverage(int numbers[], int size) {
int sum = 0;
for (int i = 0; i <= size; i++) {
sum += numbers[i];
}
return sum / size;
}
int main() {
int numbers[] = {5, 10, 15, 20, 25};
int size = sizeof(numbers) / sizeof(numbers[0]);
double average = calculateAverage(numbers, size);
printf("Average: %f\n", average);
return 0;
}
```

Debugging Questions:

Question 1: The program is supposed to calculate the average of the numbers in the array, but the output is not what is expected. What is the issue causing this incorrect output?

- Hint: Think about how the loop in `calculateAverage` function iterates through the array elements.

Question 2: After fixing the loop issue, the program still doesn't output the correct average. It seems to be lower than expected. What could be causing this problem?

- Hint: Consider the data types used in the calculation of the average.

Question 3:The program uses `int` for the sum variable in `calculateAverage`. What potential issue could arise from this choice if the array contains very large numbers?

- Hint: Think about the maximum value an `int` variable can hold.

Question 4: How would you modify the program to ensure it can handle an empty array without resulting in an undefined behavior or crash?

- Hint: What happens when you divide by zero?

Ans : -

The debugged C code

```c
 1. #include <stdio.h>
 2. double calculateAverage(int numbers[], int size) {
 3.   int sum = 0;
 4.   for (int i = 0; i < size; i++) {
 5.     sum += numbers[i];
 6.   }
 7.   return (double)sum / size;
 8. }
 9.
10. int main() {
11.   int numbers[] = {5, 10, 15, 20, 25};
12.   int size = sizeof(numbers) / sizeof(numbers[0]);
13.   double average = calculateAverage(numbers, size);
14.   printf("Average: %.2f\n", average);
15.   return 0;
16. }
```

I have made changes to the code as follows:

1. **(In line no. 3)** Change the data type to `long long int` to avoid overflow and handle very large numbers.
2. **(In line no. 4)** Looping condition in `calculateAverage` function:
   In the given question the for loop in the `calculateAverage` function iterates from `i=0` to `i<=size` , this includes the element at index size which is outside the valid bounds of the array. To fix this, the loop condition should be `i<size`.
3. **(In line no. 7)** Casting the sum to double for accurate floating-point division.
4. **(In line no. 14)** Print average with 2 decimal places.

## Answers for debugging questions:

Q1. Ans : -

The issue lies in the loop condition of the calculateAverage function of the original code.

The issue is that :

1. The loop iterates from `i = 0` to `i <= size`. This includes the element at index `size`.
2. Arrays in C are zero-indexed, meaning the first element has index 0 and the last element has index `size - 1`.
3. Accessing elements beyond the valid range (i.e., `numbers[size]`) leads to undefined behaviour, which might explain the unexpected output.

To fix this, we need to change the loop condition to iterate only over valid elements within the array.

The fixed code is

```c
for (int i = 0; i < size; i++)
{
  sum += numbers[i];
}
```

Q2 . Ans : -

The issue is the data type used in the calculation of the average . In the original code the sum is declared as int .

For eg. (int array[3,5,9]) and their sum is 17 , and if we take the average of the array ie 17/3 = 5.666667 , since the sum is an integer the result of average will be 5 < 5.67 . That's the reason why the output of the original code is wrong.

So , the corrected code is

```
return (double)sum / size;
```

Q3. Ans :-

The size of the int data type in C is from -2^31 to 2^31 . If the sum of the elements in the array exceeds the maximum positive value an int can hold, an overflow occurs. If an overflow occurs during summation , the sum variable will not hold an incorrect value .

So, to avoid this we can use data type such as `long long int`

```
long long int sum = 0;
```

Q4. Ans :-

We can handle an empty array without resulting in an undefined behaviour or crash by using a case for exception handing in the `calculateAverage` function

The corrected code is changed such that it accepts user inputs for the array elements :

```
#include <stdio.h>
double calculateAverage(int numbers[], int size) {
  if (size == 0) {
    return -1.0;  // Handle empty array case
  }
  int i,sum = 0;
  for (i = 0; i < size; i++) {
    sum += numbers[i];
  }
  return (double)sum / size;
}
int main() {
  int i,numElements;

  printf("Enter the number of elements (positive integer): ");
  while (scanf("%d", &numElements) != 1 || numElements <= 0) {
    printf("Invalid input. Please enter a positive integer: ");
    scanf("%d"); // Discard invalid input
  }
  int numbers[numElements];
  printf("Enter %d numbers:\n", numElements);
  for (i = 0; i < numElements; i++) {
    scanf("%d", &numbers[i]);
  }
  double average = calculateAverage(numbers, numElements);
  if (average == -1.0) {
    printf("Error: Cannot calculate average for an empty array.\n");
  } else {
    printf("Average: %.2f\n", average);
  }
  return 0;
}
```