*Thomas Baratta*

***C++ Practice Sheet Solutions***

***Arrays and Pointers***

**1.1)**

**#include <iostream>**

**int main()**

**{**

**int arr[5];**

**for (int number = 0; number < 5; number++)**

**{**

**arr[number] = number \* number;**

**}**

**for (int counter = 1; counter < 5; counter++)**

**{**

**std::cout << arr[counter] << " ";**

**}**

**return 0;**

**}**

**The bug is found in the second for loop. The loop tries to print the 5th index of the array which is out of bounds. The array contains an index 0 – 4.**

**1.2)**

**Output:**

**Enter a value for index 0: 1**

**Enter a value for index 1: 2**

**Enter a value for index 2: 3**

**Enter a value for index 3: 4**

**Enter a value for index 4: 5**

**Enter a value for index 5: 6**

**6 5 4 3 2 1**

**Process finished with exit code 0**

**Manual Execution:**

**Step 1: Declaration and Input**

**int arr[6]; — A 6-element integer array arr is declared.**

**First for loop (input loop):**

**This loop prompts the user to input 6 values, one at a time, and stores them in the array.**

**Let's assume the user enters the following values for the array:**

**Input 1: 1**

**Input 2: 2**

**Input 3: 3**

**Input 4: 4**

**Input 5: 5**

**Input 6: 6**

**arr = {1, 2, 3, 4, 5, 6}**

**Step 2: Output the Array in Reverse Order**

**Second for loop (output loop):**

**This loop starts at the last index (index 5) and moves backwards to the first index (index 0), printing each value.**

**Here's the detailed execution of this loop:**

**Iteration 1 (reverseIndex = 5):**

**Output: arr[5] = 6**

**Iteration 2 (reverseIndex = 4):**

**Output: arr[4] = 5**

**Iteration 3 (reverseIndex = 3):**

**Output: arr[3] = 4**

**Iteration 4 (reverseIndex = 2):**

**Output: arr[2] = 3**

**Iteration 5 (reverseIndex = 1):**

**Output: arr[1] = 2**

**Iteration 6 (reverseIndex = 0):**

**Output: arr[0] = 1**

**1.3)**

**Output:**

**4 + 5 = 9**

**Process finished with exit code 0**

**Manual Execution:**

**Step 1: Setting Up Variables in main()**

**arr[5] = {1, 2, 3, 4, 5} — The array contains five elements.**

**target = 9 — We want to find pairs of elements in the array that sum to 9.**

**size = sizeof(arr) / sizeof(arr[0]) calculates the size of the array:**

**sizeof(arr) gives the total size of the array, which is 5 × 4 = 20 bytes.**

**sizeof(arr[0]) gives the size of the first element, which is 4 bytes.**

**So, size = 20 / 4 = 5.**

**Step 2: Execution of twoSum(arr, size, target)**

**We now call twoSum(arr, size, target) to check for pairs of elements that add up to the target value (9).**

**Initial State:**

**foundPair = false — Initially, no pairs have been found.**

**We will start iterating over the array using nested for loops.**

**Step 3: Nested for Loops Execution**

**The nested loops will iterate as follows:**

**Outer Loop Iterations (i):**

**i = 0 (The outer loop starts at the first element arr[0] = 1).**

**Inner Loop Iterations (j):**

**For i = 0, the inner loop j starts at j = 1:**

**arr[0] + arr[1] = 1 + 2 = 3 → Not equal to 9.**

**arr[0] + arr[2] = 1 + 3 = 4 → Not equal to 9.**

**arr[0] + arr[3] = 1 + 4 = 5 → Not equal to 9.**

**arr[0] + arr[4] = 1 + 5 = 6 → Not equal to 9.**

**i = 1 (The outer loop now moves to arr[1] = 2).**

**For i = 1, the inner loop j starts at j = 2:**

**arr[1] + arr[2] = 2 + 3 = 5 → Not equal to 9.**

**arr[1] + arr[3] = 2 + 4 = 6 → Not equal to 9.**

**arr[1] + arr[4] = 2 + 5 = 7 → Not equal to 9.**

**i = 2 (The outer loop now moves to arr[2] = 3).**

**For i = 2, the inner loop j starts at j = 3:**

**arr[2] + arr[3] = 3 + 4 = 7 → Not equal to 9.**

**arr[2] + arr[4] = 3 + 5 = 8 → Not equal to 9.**

**i = 3 (The outer loop now moves to arr[3] = 4).**

**For i = 3, the inner loop j starts at j = 4:**

**arr[3] + arr[4] = 4 + 5 = 9 → This equals 9, so the program prints:**

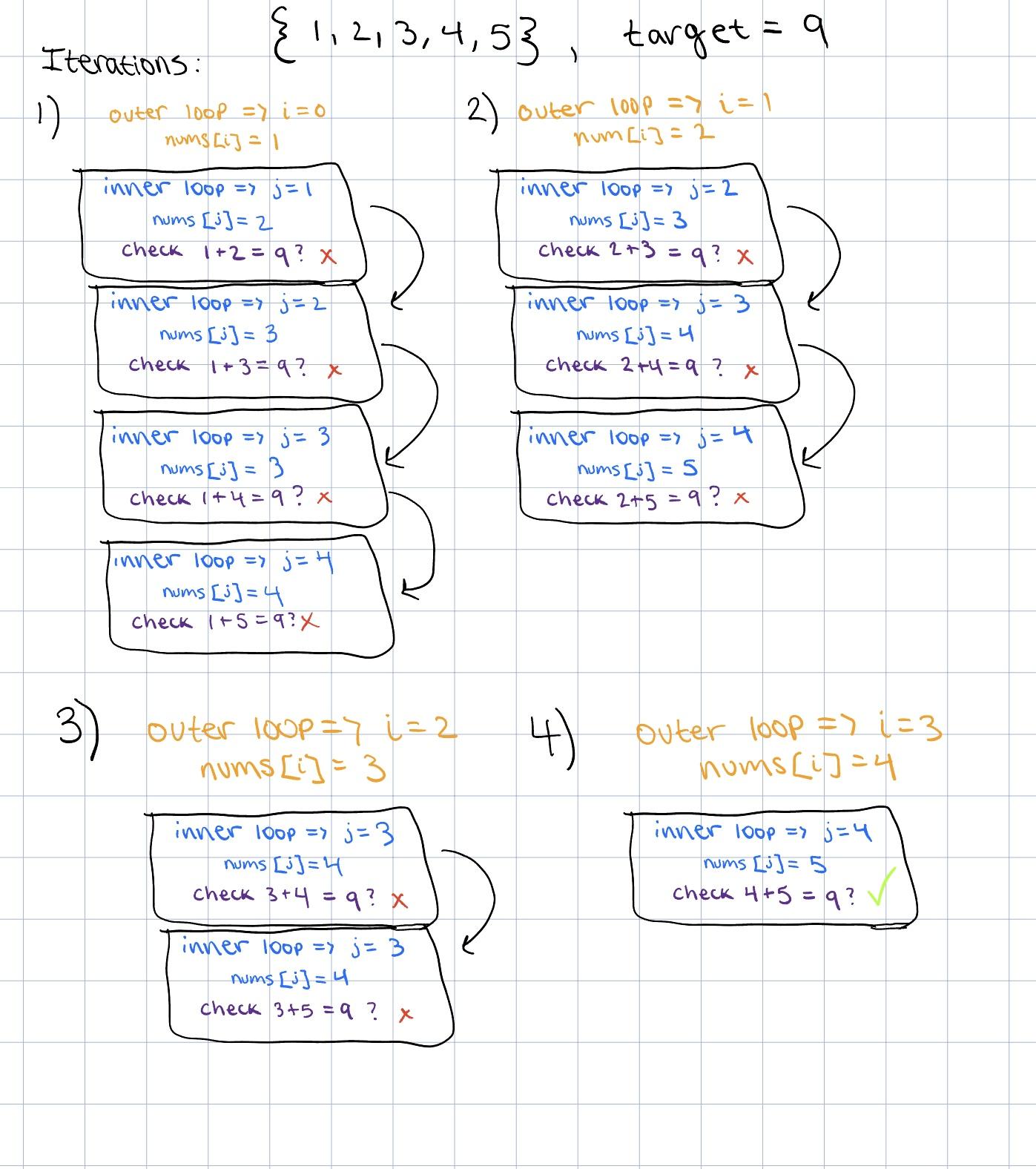
**4 + 5 = 9**

**foundPair = true is set to true.**

**i = 4 (The outer loop now moves to arr[4] = 5). The inner loop doesn't execute because j = i + 1 = 5, and the inner loop stops when j reaches the size of the array.**

**Step 4: No More Loops**

**Since foundPair = true (because we found one pair, 4 + 5 = 9), the final if (!foundPair) block does not execute, and the message "No pairs of indices add to the target value" is not printed.**



**2.1)**

**Output:**

**10**

**10**

**0x7ff7b9fd54a8**

**Process finished with exit code 0**

**Manual Execution:**

**Step 1: Variable Declaration and Initialization**

**int number = 10;**

**A variable number is declared and initialized with the value 10.**

**int \*ptr = &number;**

**A pointer variable ptr is declared, which holds the address of number. The & operator is used to get the memory address of number.**

**At this point:**

**number holds the value 10.**

**ptr holds the address of number (let’s assume the address is 0x7ff7b9fd54a8 for this example).**

**Step 2: Output Statements**

**First Output Statement: std::cout << number << std::endl;**

**This prints the value of number.**

**Output: 10**

**Second Output Statement: std::cout << \*ptr << std::endl;**

**The \* operator is used to dereference the pointer ptr, which means it accesses the value at the address stored in ptr.**

**Since ptr points to number, this will print the value of number.**

**Output: 10**

**Third Output Statement: std::cout << ptr << std::endl;**

**This prints the address stored in ptr, which is the address of number.**

**2.2)**

**Output:**

**x = 5 and y = 10**

**x = 10 and y = 5**

**Process finished with exit code 0**

**Manual Execution:**

**Step 1: Variable Declaration and Initialization**

**int x = 5;**

**An integer variable x is declared and initialized with the value 5.**

**int y = 10;**

**An integer variable y is declared and initialized with the value 10.**

**Output Before Function Call:**

**The first output statement prints the initial values of x and y.**

**Step 2: Calling the mysteryFunction**

**mysteryFunction(&x, &y);**

**The addresses of x and y are passed to the function mysteryFunction. This means that x and y are passed by reference, allowing the function to modify their values directly.**

**Step 3: Inside the mysteryFunction**

**Inside mysteryFunction:**

**int temp = \*x;**

**The value pointed to by x (which is 5) is stored in the temporary variable temp.**

**\*x = \*y;**

**The value pointed to by y (which is 10) is assigned to the location pointed to by x. So now, x holds the value 10.**

**\*y = temp;**

**The value stored in temp (which is 5) is assigned to the location pointed to by y. So now, y holds the value 5.**

**After executing the function, the values of x and y are swapped:**

**x becomes 10**

**y becomes 5**

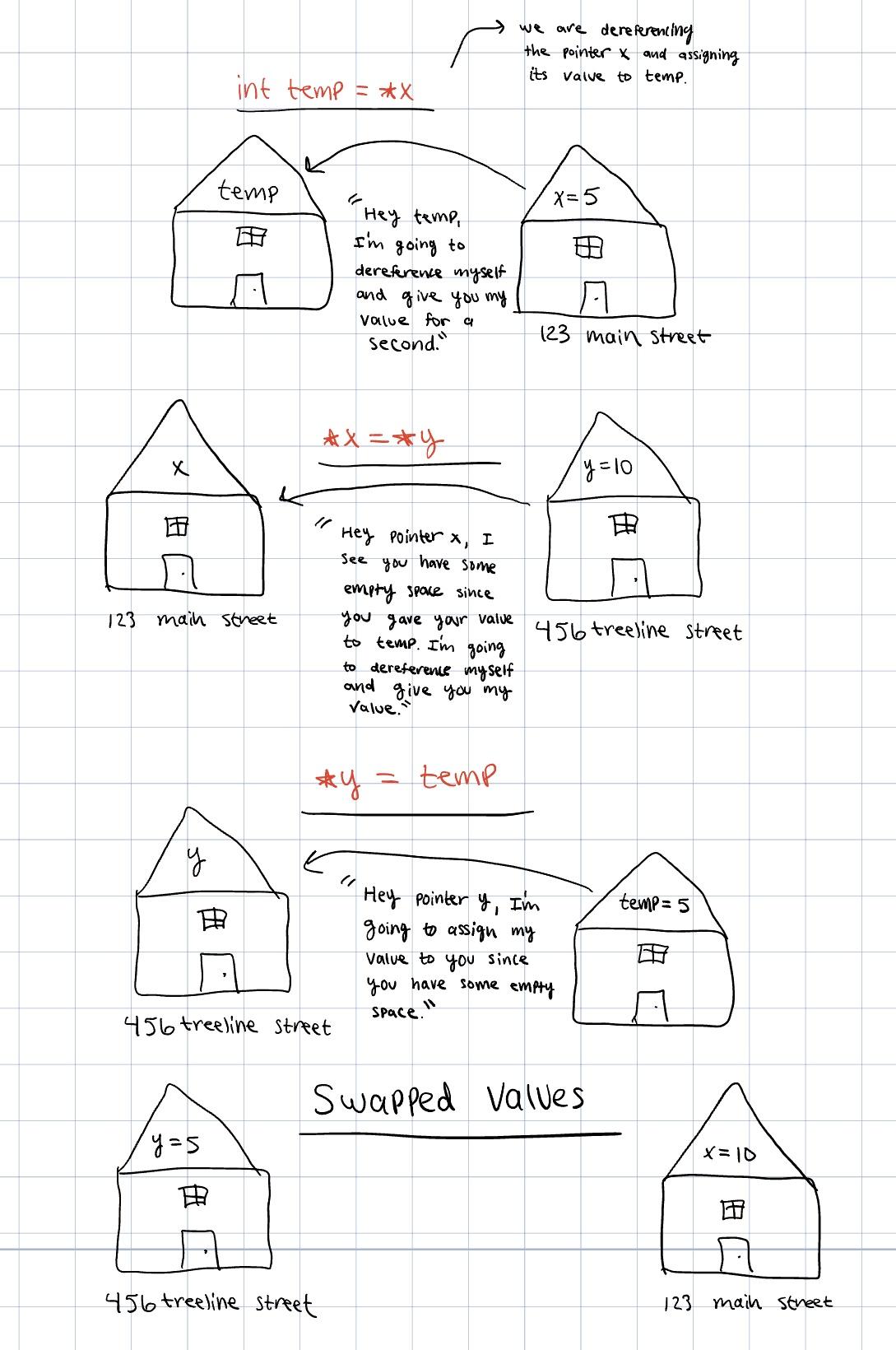
**Step 4: Output After Function Call**

**Output After Function Call:**

**The second output statement prints the new values of x and y after the function call.**

**Output:**

**x = 10 and y = 5**



**2.3)**

**Output:**

**Array elements are: 10 20 30 40 50**

**Modified array elements are: 10 20 100 40 50**

**Process finished with exit code 0**

**Manual Execution:**

**Step 1: Variable Initialization**

**Declare and Initialize Array:**

**arr is initialized with the values {10, 20, 30, 40, 50}.**

**Step 2: Declare Pointer**

**ptr is declared and initialized to point to the first element of arr (ptr = arr;).**

**State:**

**arr: [10, 20, 30, 40, 50]**

**ptr: points to 10 (the first element of arr)**

**Step 3: First Print Statement**

**Output: "Array elements are: ".**

**First for Loop Execution for (int i = 0; i < 5; i++):**

**Iteration 1 (i = 0):**

**\*(ptr + 0) = 10**

**Output: "10 ".**

**Iteration 2 (i = 1):**

**\*(ptr + 1) = 20**

**Output: "20 ".**

**Iteration 3 (i = 2):**

**\*(ptr + 2) = 30**

**Output: "30 ".**

**Iteration 4 (i = 3):**

**\*(ptr + 3) = 40**

**Output: "40 ".**

**Iteration 5 (i = 4):**

**\*(ptr + 4) = 50**

**Output: "50 ".**

**Output So Far:**

**Array elements are: 10 20 30 40 50**

**Step 4: Modify the Array**

**The value at index 2 is modified to 100: \*(ptr + 2) = 100;**

**The array now contains: {10, 20, 100, 40, 50}.**

**State:**

**arr: [10, 20, 100, 40, 50]**

**Step 5: Second Print Statement**

**Output: "Modified array elements are: ".**

**Second for Loop Execution for (int j = 0; j < 5; j++):**

**Iteration 1 (j = 0):**

**\*(ptr + 0) = 10**

**Output: "10 ".**

**Iteration 2 (j = 1):**

**\*(ptr + 1) = 20**

**Output: "20 ".**

**Iteration 3 (j = 2):**

**\*(ptr + 2) = 100**

**Output: "100 ".**

**Iteration 4 (j = 3):**

**\*(ptr + 3) = 40**

**Output: "40 ".**

**Iteration 5 (j = 4):**

**\*(ptr + 4) = 50**

**Output: "50 ".**

**2.4)**

**Output:**

**Enter 5 integers:**

**1**

**2**

**3**

**4**

**5**

**5 4 3 2 1**

**15**

**Process finished with exit code 0**

**Manual Execution:**

**Step 1) Initialization**

**An integer array arr of size 5 is declared.**

**A pointer ptr is set to point to the first element of arr.**

**An integer variable sum is initialized to 0.**

**int arr[5];**  
**int\* ptr = arr; // ptr points to first index in arr[]**  
**int sum = 0;**

**Step 2) Prompt User for Input**

**The program prompts the user to enter 5 integers.**

**Enter 5 integers:**

**Input Loop:**

**The program enters the first for loop, where it reads 5 integers into the array arr.**

**User inputs: 1, 2, 3, 4, 5**

**Execution of the loop:**

**Iteration 1: inputIndex = 0 → arr[0] gets 1**

**Iteration 2: inputIndex = 1 → arr[1] gets 2**

**Iteration 3: inputIndex = 2 → arr[2] gets 3**

**Iteration 4: inputIndex = 3 → arr[3] gets 4**

**Iteration 5: inputIndex = 4 → arr[4] gets 5**

**After this loop, arr contains: [10, 20, 30, 40, 50].**

**Step 3) Reverse Print Loop**

**The program enters the second for loop to print the elements of arr in reverse order using pointer arithmetic.**

**Execution of the loop:**

**Iteration 1: reverseIndex = 4 → Prints \*(ptr + 4) → 5**

**Iteration 2: reverseIndex = 3 → Prints \*(ptr + 3) → 4**

**Iteration 3: reverseIndex = 2 → Prints \*(ptr + 2) → 3**

**Iteration 4: reverseIndex = 1 → Prints \*(ptr + 1) → 2**

**Iteration 5: reverseIndex = 0 → Prints \*(ptr + 0) → 1**

**New Line Output:**

**The program outputs a new line after printing the reversed array.**

**5 4 3 2 1**

**Step 4) Sum Calculation Loop**

**The program enters the third for loop to calculate the sum of the integers in arr.**

**Execution of the loop:**

**Iteration 1: sumIndex = 0 → sum += \*(ptr + 0) → sum = 0 + 1 = 1**

**Iteration 2: sumIndex = 1 → sum += \*(ptr + 1) → sum = 1 + 2 = 3**

**Iteration 3: sumIndex = 2 → sum += \*(ptr + 2) → sum = 3 + 3 = 6**

**Iteration 4: sumIndex = 3 → sum += \*(ptr + 3) → sum = 6 + 4 = 10**

**Iteration 5: sumIndex = 4 → sum += \*(ptr + 4) → sum = 10 + 5 = 15**

**Final value of sum after the loop is 15.**

**Output the Sum:**

**The program outputs the total sum of the integers.**

**Sum: 150**