**Module: R1: C Programming**

**Section:** C Memory Management & Usage **Task:** Dynamic Memory

**Task 5.2**

**Dynamic Memory**

1. **Allocating Memory:**
   * **Code Snippet:**

main.c

//Allocating Memory

integer \*array = MALLOC(SIZE,integer);

1. **Populating Array:**
   * **Code Snippet:**

main.c

//Populating Array

initialize\_memory(array, SIZE);

other.c

void initialize\_memory(int \*arr, int n){

for (int i = 0; i < n; i++){

arr[i] = rand() % 100;

}

}

1. **Showing Array:**
   * **Code Snippet:**

main.c

printf("Array Elements:\n");

show\_memory(array, SIZE);

other.c

// Definition for show\_memory

void show\_memory(int \*arr, int n){

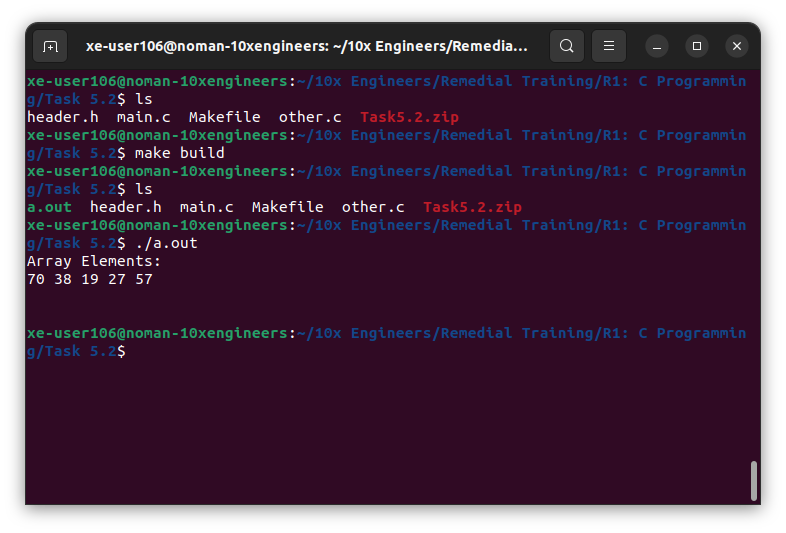
for (int i = 0; i < n; i++){

printf("%d ", arr[i]);

}

printf("\n");

}



1. **Re-Allocation:**
   * **Code Snippet:**

main.c

integer \*resized\_array = REALLOC(array, SIZE \* RESIZE\_FACTOR);

printf("\n");

other.c

void \*re\_allocate(void \*arr, size\_t size){

int \*ptr = malloc(size);

if (ptr != NULL){

printf("Memory Created Successfully\n");

memcpy(ptr, arr, size/2);

free(arr);

return ptr;

}

else {

printf("Out of Memory\n");

}

}

1. **Populating Re-Allocated Memory:**
   * **Code Snippet:**

main.c

populate\_resized(resized\_array, SIZE \* RESIZE\_FACTOR);

other.c

void populate\_resized(int \*arr, int n){

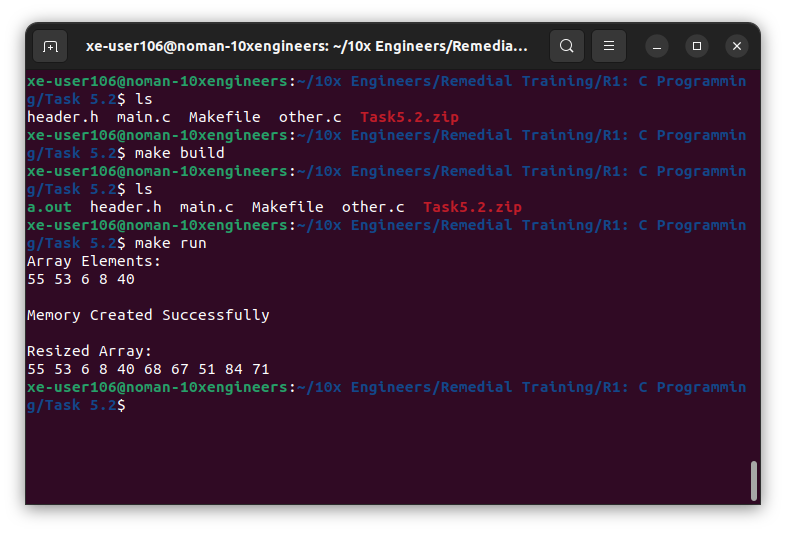
for (int i = n/2; i < n; i++){

arr[i] = rand() % 100;

}

}

1. **Output:**



‭

When resizing an array, if the new size is larger than the current size, additional memory‬ ‭needs to be allocated for the resized array. In this case, the additional elements beyond‬ ‭the current size of the array need to be populated with appropriate values.

‭On the other hand, if the new size is smaller than the current size, some elements of the‬ ‭array will be lost as they will no longer fit in the resized array. It's important to‬ understand that shrinking the array using **realloc** may result in data loss as the elements‬ ‭beyond the new size are discarded.‬