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);
```

2. **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;
	}
 }
```

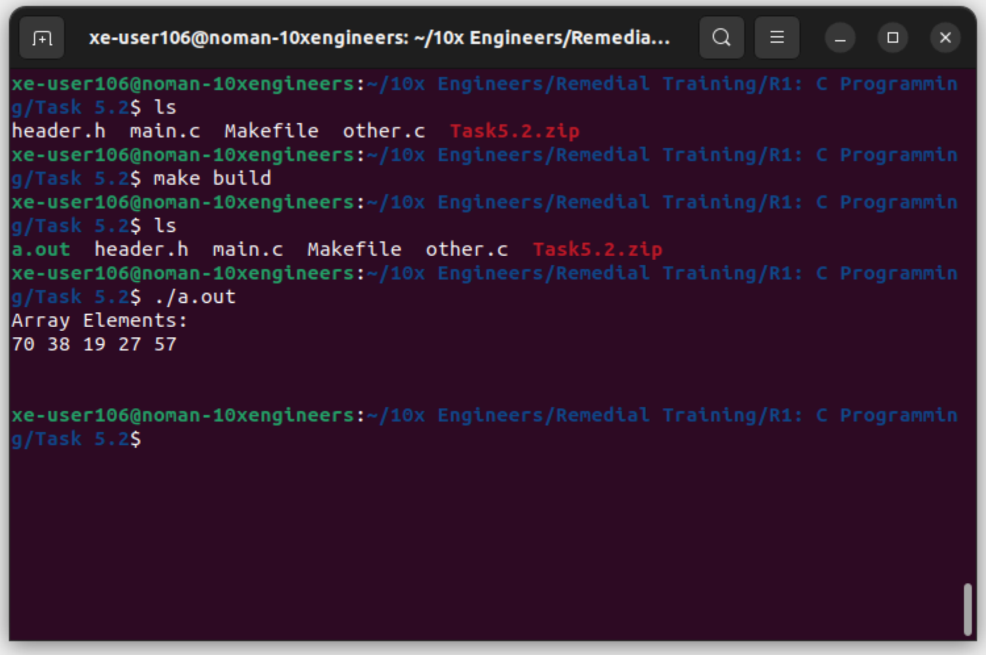
3. **Showing Array:**
   - **Code Snippet:**
     `main.c`

```
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");
}
```



```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$ ls
header.h  main.c  Makefile  other.c  Task5.2.zip
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$ make build
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$ ls
a.out  header.h  main.c  Makefile  other.c  Task5.2.zip
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$ ./a.out
Array Elements:
70 38 19 27 57


xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$
```

4. **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");
```

```
    }
}
```

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

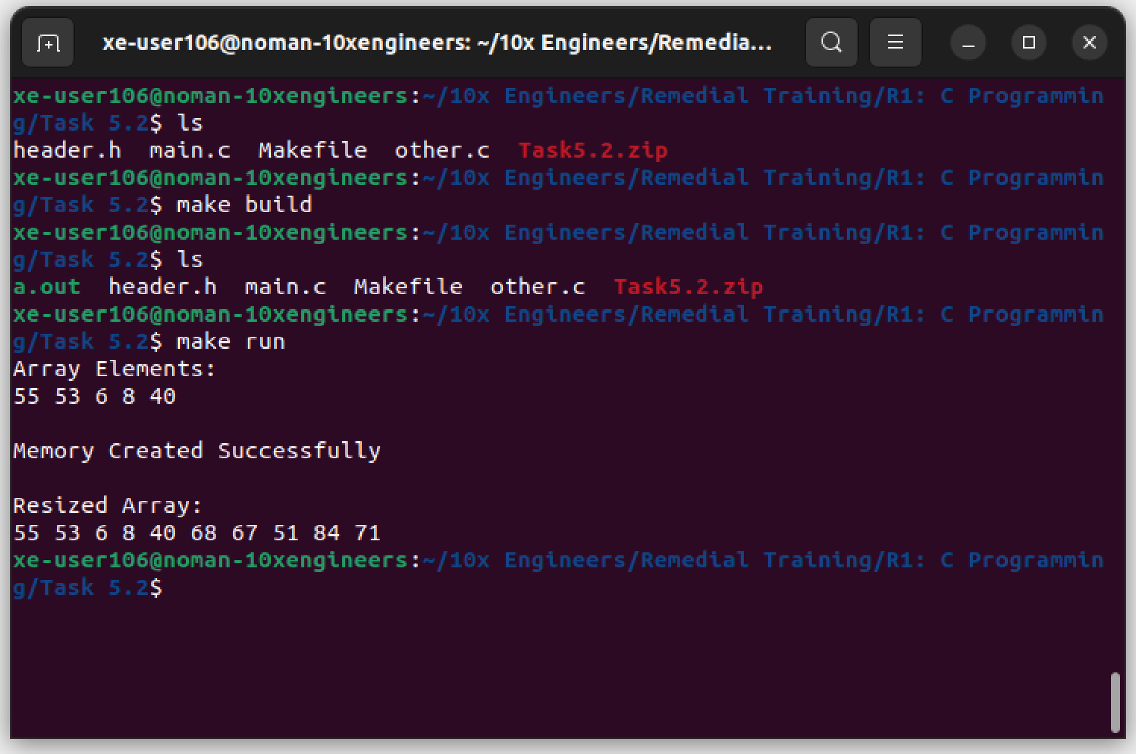
     `main.c`

```
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;
    }
}
```

6. **Output:**



```
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$ ls
header.h  main.c  Makefile  other.c  Task5.2.zip
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$ make build
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$ ls
a.out  header.h  main.c  Makefile  other.c  Task5.2.zip
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$ make run
Array Elements:
55 53 6 8 40

Memory Created Successfully

Resized Array:
55 53 6 8 40 68 67 51 84 71
xe-user106@noman-10xengineers:~/10x Engineers/Remedial Training/R1: C Programmin
g/Task 5.2$
```

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.