TEST PLAN DOCUMENT

- This function rigoursly test possible inputs on the program and shows their output in different scenarios.

```
void demo()
{


        int *p1 = NULL;                         // To Store Integer Address
        float *c1 = NULL;                       // To Store Float Address
        double *f11 = NULL;                     // To Store Double Address
        float *z1 = NULL;                       // To Store Float Address
        int *p2 = NULL;                         // For Calloc
        int i;                                  // Loop Variable
        void *zero =NULL;

        printf("\nRequesting Integer Space ...\n");
        p1 = (int *)my_malloc(sizeof(int));
        // If no memory available
        if( !p1)
        {
                printf("Integer Malloc Failed.\nTerminating ...\n");
                exit(0);
        }
        printf("Integer Memory Allocation Successful: %p\n", p1);


        printf("\nRequesting Float Space ...\n");
        c1 = (float *)my_malloc(sizeof(float));
        // If no memory available
        if( !c1)
        {
                printf("Float Malloc Failed.\nTerminating ...\n");
                exit(0);
        }
        printf("Float Memory Allocation Successful: %p\n", c1);


        printf("\nRequesting Double Space ...\n");
        f11 = (double *)my_malloc(sizeof(double));
        // If no memory available
        if( !f11)
        {
                printf("Double Malloc Failed.\nTerminating ...\n");
                exit(0);
        }
        printf("Double Memory Allocation Successful: %p\n", f11);

        printf("\nRequesting Zero Bytes Space ...\n");
```

```
zero = my_malloc(0);
// If no memory available
if( !zero)
{
	printf("Can't Allocate Zero Byte.\n");
}
printf("Double Memory Allocation Successful: %p\n", f11);

printf("\nRe-alloc with NULL as Address - Behaves as Malloc ...\n");
zero = my_realloc(NULL,10);
// If no memory available
if( !zero)
{
	printf("Re-alloc Failed.\nTerminating ...\n");
	exit(0);
}
printf("Double Memory Allocation Successful: %p\n", f11);

printf("\nRemaining Size in Heap:%lu\n\n", free_space());

printf("\nReallocating Float Space ...\n");
z1 = (float *)my_realloc(c1, 5 * sizeof(float));
// If no memory available
if( !z1)
{
	printf("Re-alloc Failed.\nTerminating ...\n");
	exit(0);
}
printf("Float Re-Allocation Successful: %p\n", z1);

// Just for Address Computation
char *uv = (char *) z1;
uv = uv - BLOCK_SIZE;
struct block *uv1 = (struct block *)uv;

printf("Size of realloced space: %lu and address: %p", uv1 -> size, z1  );

printf("\n\nRemaining Size in Heap:%lu\n", free_space());

printf("\nRequesting Calloc Memory Allocation...\n");
p2 = (int *) my_calloc( (size_t)10, sizeof(int));
for( i = 1; i <= 10 ; i++)
	p2[i] = i;
for( i = 1; i <= 10 ; i++)
	printf("%d\t", p2[i]);

printf("\n\nStatus of Heap After Allocation of Memory:\n");
printf("Remaining Size in Heap:%lu\n\n", free_space());
struct block * tally = (struct block*)base;
while(tally)
```

```
	{
		printf("Meta Data Begins:%p\t", tally);
		printf("Size of Block: %ld\t", tally->size);
		printf("Free : %d\t", tally->free);
		printf("Next: %p\t", tally -> next);
		printf("Previous: %p\t", tally -> prev);
		tally = tally -> next;
		printf("\n");
	}

	printf("\nFree Up Double Space ...\n");
	free(f11);
	printf("After Free up of Double Space ...\n");
	printf("\n\nStatus of Heap after Free Up of Memory:\n");
	printf("Remaining Size in Heap:%lu\n\n", free_space());
	tally = (struct block*)base;
	while(tally)
	{
		printf("Meta Data Begins:%p\t", tally);
		printf("Size of Block: %ld\t", tally->size);
		printf("Free : %d\t", tally->free);
		printf("Next: %p\t", tally -> next);
		printf("Previous: %p\t", tally -> prev);
		tally = tally -> next;
		printf("\n");
	}
	printf("\nDefragmentation Started ...\n\n");
	defragment_my_heap();
	tally = (struct block*)base;
	printf("\nAfter Defragmenting the Heap: \n");
	while(tally)
	{
		printf("Meta Data Begins:%p\t", tally);
		printf("Size of Block: %ld\t", tally->size);
		printf("Free : %d\t", tally->free);
		printf("Next: %p\t", tally -> next);
		printf("Previous: %p\t", tally -> prev);
		tally = tally -> next;
		printf("\n");
	}
	printf("\n");

}
```