Q.1] Write a C program that implements a **singly linked list** using **your custom memory allocator (my_malloc() and my_free())** instead of malloc() and free().

Requirements:

1. Define a Node structure that stores an integer value and a pointer to the next node.
2. Implement the following functions:
    - insert_at_end(Node **head, int value): Allocates memory and inserts a node at the end.
    - delete_at_position(Node **head, int position): Deletes a node at a given index.
    - print_list(Node *head): Prints the list.
    - free_list(Node **head): Frees all nodes using my_free().
3. Test by inserting **10 elements**, deleting a few, and printing the final list.

Expected Output Example:

Initial List: 1 -> 2 -> 3 -> 4 -> 5

After Deleting at Position 2: 1 -> 2 -> 4 -> 5

Q.2] Write a C program that:

1. **Allocates a 2D matrix dynamically** using my_calloc(), ensuring all elements are **initialized to zero**.
2. Implements the following functions:
    - **int**** *my_calloc(size_t num, size_t size)`** – Allocates memory and sets it to zero.
    - **void my_free(void *ptr)** – Frees previously allocated memory.
    - **int**** **create_matrix(int rows, int cols)** – Allocates a matrix using my_calloc()`.
    - **void fill_matrix(int**** **matrix, int rows, int cols)`** – Fills the matrix with values.
    - **void print_matrix(int**** **matrix, int rows, int cols)`** – Displays the matrix.
    - **void free_matrix(int**** **matrix, int rows)** – Frees allocated memory using my_free()`.
3. **Frees allocated memory** after use to prevent memory leaks.

Functional Requirements

1. Implement **my_calloc()** that:
    - Uses my_malloc() to allocate memory.
    - Initializes all memory bytes to **zero**.
2. Implement **my_free()** that:
    - Ensures safe deallocation of memory.
3. Implement **dynamic memory allocation for a matrix**.
4. Ensure **proper memory deallocation** using my_free() after matrix usage.

#Expected Output :

Matrix Initialized with Zero Values:

0 0 0

0 0 0

0 0 0

Matrix After Filling with Values:

1 2 3

4 5 6

7 8 9

Memory Freed Successfully.

Q.3] You are required to implement a **dynamic array** in C that supports the following functionalities:

1. **Dynamically allocates and resizes memory** using my_realloc() instead of malloc().
2. Implements the following operations:
    - **init_array(DynamicArray *arr, int initial_size)** → Initializes the array with a given capacity.
    - **insert_element(DynamicArray *arr, int value)** → Inserts an element, resizing when the array is full.
    - **remove_element(DynamicArray *arr, int index)** → Removes an element at a given index and shifts elements left.
    - **resize_array(DynamicArray *arr, int new_size)** → Expands or shrinks the array dynamically using my_realloc().
    - **free_array(DynamicArray *arr)** → Frees all allocated memory using my_free().

Functional Requirements

1. **Define a DynamicArray structure** that contains:
    - int *data → A pointer to the dynamically allocated array.
    - int size → The current number of elements in the array.
    - int capacity → The total allocated size of the array.
2. **Implement and use custom memory management functions**:
    - **my_realloc(void *ptr, size_t new_size)** → Dynamically resizes memory for the array when required.
    - **my_free(void *ptr)** → Frees allocated memory safely.
3. **Ensure automatic resizing**:
    - When inserting elements, **double the capacity** if the array is full.
    - When deleting elements, **shrink the array** if the size is less than 25% of capacity.
4. **Ensure proper memory deallocation** using my_free() before the program exits.

#Input Commands :

```
DynamicArray arr;

init_array(&arr, 5);

insert_element(&arr, 10);

insert_element(&arr, 20);

insert_element(&arr, 30);

print_array(&arr);

remove_element(&arr, 1);

print_array(&arr);

resize_array(&arr, 10);

print_array(&arr);

free_array(&arr);
```

#Expected Output :

Initial array created with capacity: 5

Inserted 10

Inserted 20

Inserted 30

Array contents: [10, 20, 30]

Removing element at index 1

Array contents: [10, 30]

Resizing array to new capacity: 10

Array contents: [10, 30]

Memory Freed Successfully.