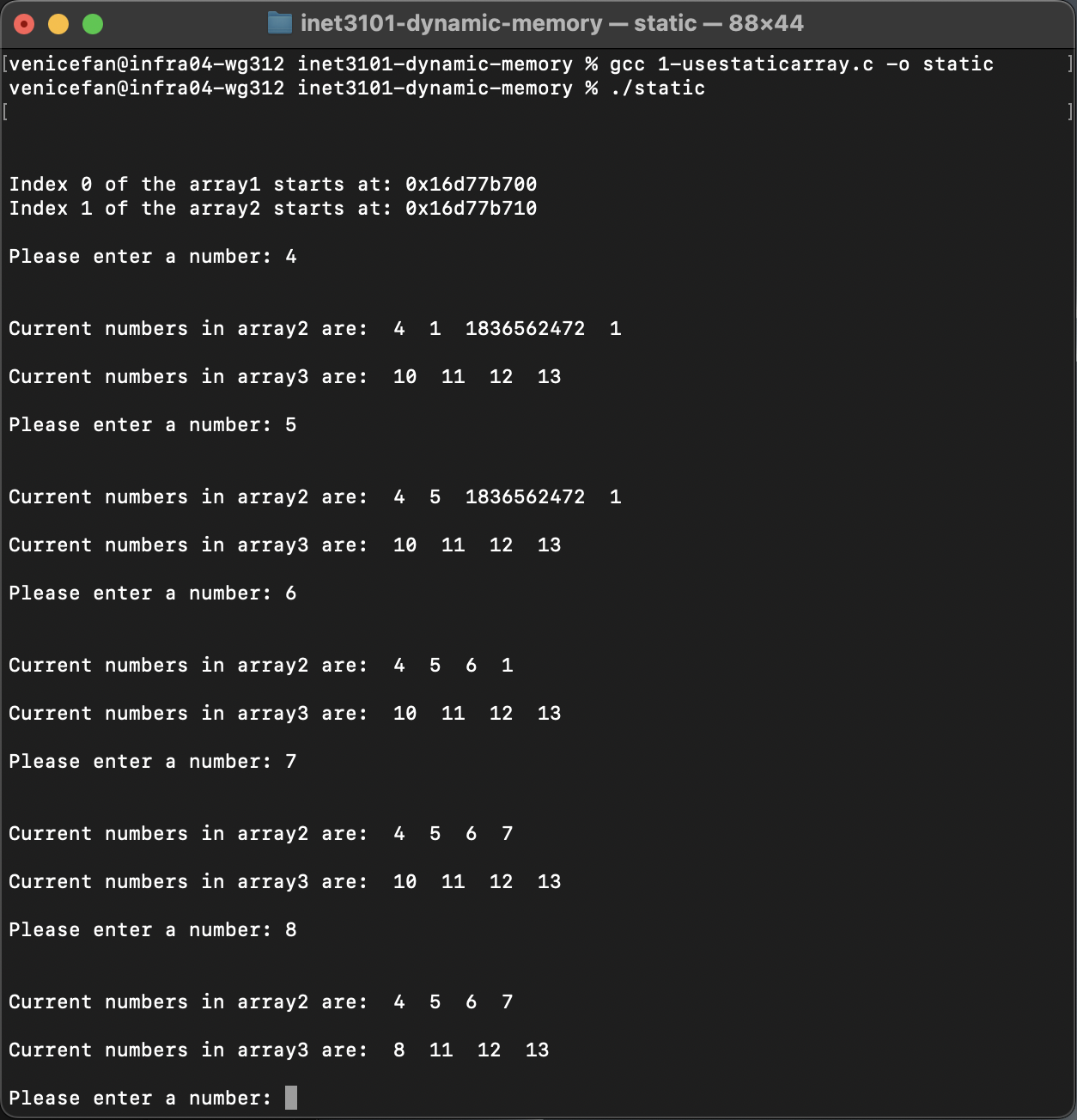
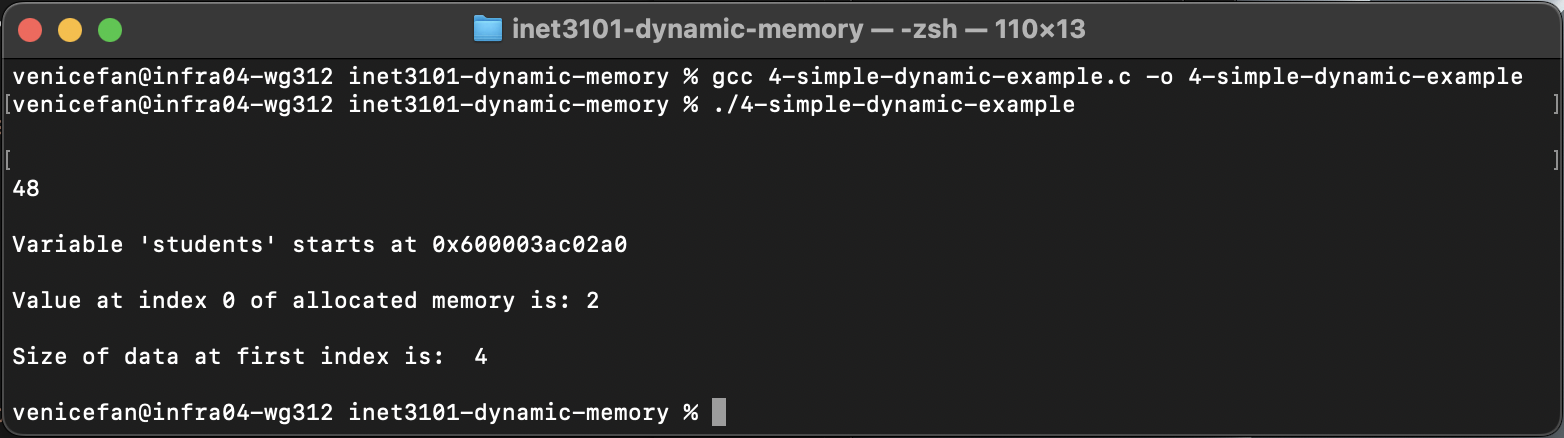
**1-usestaticarray.c**

****

**3-dynamically-allocating.c**

The program starts by allocating memory based on the user-specified starting size. Instead of using a for loop with a fixed number of elements, a while loop continuously prompts the user for numbers. If the allocated memory reaches its limit, realloc dynamically increases the size by 10 elements, similar to the addToList function. This prevents frequent reallocations, making the program more efficient while allowing unlimited user input. The program also includes proper memory checks to ensure allocation and reallocation succeed. By dynamically managing memory, this approach makes the array flexible, ensuring it can grow as needed without predefined constraints.

**4-simple-dynamic-example.c**

****

**5-python-lists-are-objects-not-arrays.py**

In Object-Oriented Programming (OOP), an **object** is an instance of a class that encapsulates both data (attributes) and behaviors (methods). Objects allow for modular, reusable, and organized code by bundling related functionality into self-contained units. A Python **list** embodies this principle by being more than just a simple data structure—it is an object that includes built-in methods such as .sort(), .append(), and .insert(). These methods allow for easy manipulation of the list without requiring direct memory management, showcasing how Python abstracts complexity and enables more intuitive programming.

**6-linkedlist.c**

Using a linked list helps address some of the same problems that dynamic memory allocation solves, particularly in scenarios where memory needs to be managed efficiently at runtime. Unlike arrays, which require a fixed size at allocation, linked lists allow for dynamic memory allocation, meaning elements can be added or removed without the need for resizing or shifting elements. This flexibility helps in situations where the size of the data structure is not known in advance. Additionally, since each node is allocated separately, linked lists can utilize fragmented memory more efficiently than arrays, which require a contiguous block of memory. However, linked lists introduce overhead due to additional pointers and can have slower access times compared to arrays due to the need for traversal.