1. **Usestaticarray.c**

**Submission: Give this a try and submit screen shot (Window user)**

**A screenshot of a computer program

AI-generated content may be incorrect.**

1. **Mallocprinter.c**

**NOTHING TO DO OR SUBMIT (Understand how malloc works)**

1. **Dynamically-allocating.c**

**Submission: Modify the code and explain the approach and why it solves the problem.**

My approach leaned towards following the guidelines. I left parts of the program alone and changed the for-loop into an infinite while loop, which we utilize the leftover parts to work with the new loop. I made it so entering strings will end the loop (essential ending the program). I then set up a new variable (int count) that will count the number of numbers in the array and set it so if it reaches the current size (num\_elements), it’ll resize using the method implemented in 7-managing allocation. Lastly, it’ll print out the array size and current numbers in it. This approach works because we’re increasing the size of the array by 10 (random number, could resize it to whatever is needed) which ensures that there will be space for more numbers. This was a common strategy in CSCI 1933 when we were resizing arrays in java, just in this case stuff like realloc was used.

1. **Simple-dynamic-example.c**

**Submission: Submit screenshot of running the program.**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Python-lists-are-objects-not-arrays.py**

**Submission: WRITE A SHORT PARAGRAPH ->**

**Define what an Object is in terms of the Object-Oriented Programming paradigm.**

From the knowledge of CSCI 1933, in Object Oriented Programming, an object is an instance of a class that includes variety of data (types) and has methods (getters and setters) that interacts with the data. Essential, it’s a way for software to reuse, build, and represent actual real objects like say car class -> car object or -> person class -> person object. We can also look at the example as it’s a big example of an object. The python list is an object because it stores in elements and comes in with built in methods like append and insert. This design takes away the more complex details of memory management and allocation that C had us do which makes it easier to work with this data structure. Basically, what Copilot or any AI is to us, we can put in something complex, and they’ll spit the results out (Either as an answer or giving us an easier way to understand).

1. **Linkedlist.c**

**Submission: WRITE A SHORT PARAGRAPH ->**

**How does using a linked list solve some of the same problems we are trying to overcome when using dynamic memory?**

With arrays, memory is allocated already in its position. When resizing it, that means it requires allocation to a new position and copying over data which could be inefficient and could cause errors (especially in big structures). LinkedList however allocates memory via nodes, each separately. This means that data and memory doesn’t have to be moved around at all, which creates a more efficient and smooth transition of inserting and deleting data. Essentially when it comes to resizing, inserting, or deleting data, all a Linked list must do is update its pointers in adjustment to its adjacent nodes. This specific data structure provides a way to deal with the challenges that come with dynamic memory.



1. **Managing-allocation-with-struct.c**

**Submission: Nothing to submit, just review the code.**

1. **The work should be in the folder/directory: myheader, next to the header-examples.**