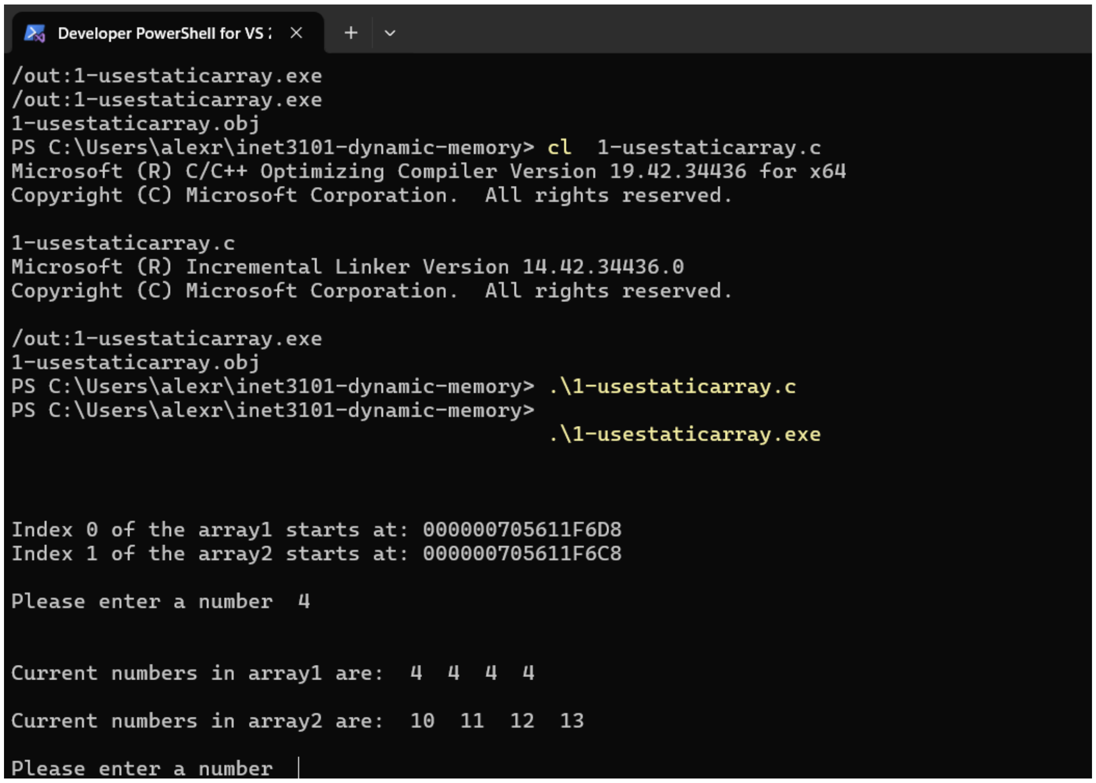
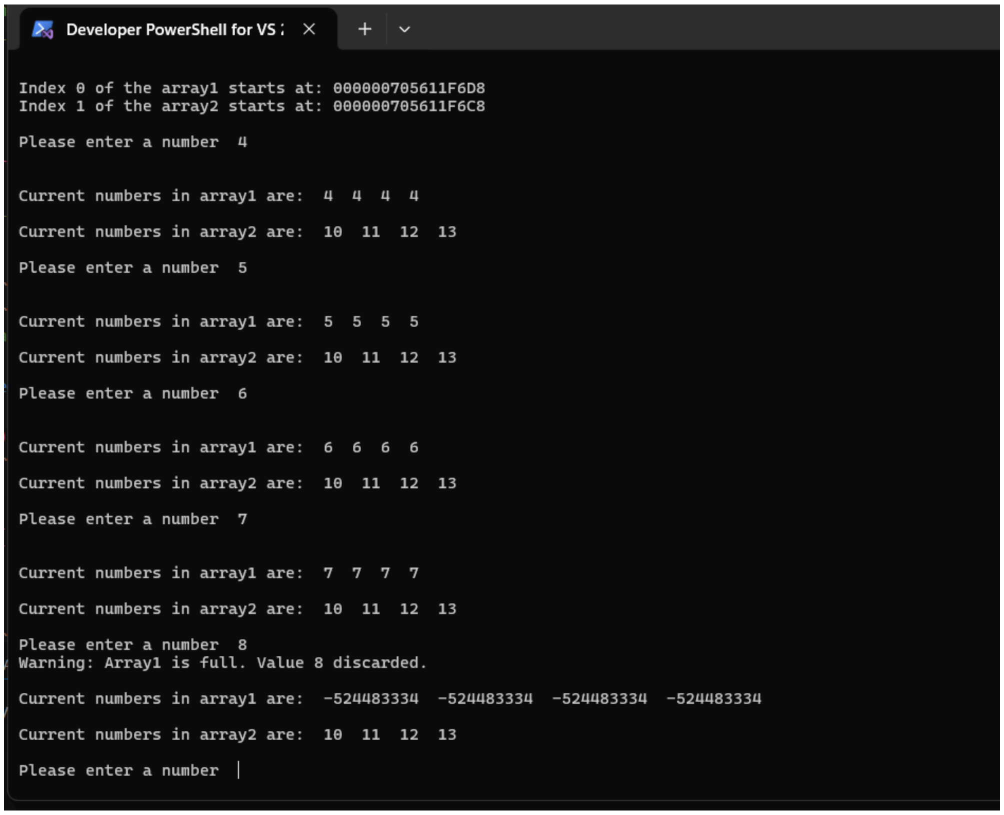
Question 1


Developer PowerShell for VS

```
/out:1-usestaticarray.exe
/out:1-usestaticarray.exe
1-usestaticarray.obj
PS C:\Users\alexr\inet3101-dynamic-memory> cl  1-usestaticarray.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.42.34436 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

1-usestaticarray.c
Microsoft (R) Incremental Linker Version 14.42.34436.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:1-usestaticarray.exe
1-usestaticarray.obj
PS C:\Users\alexr\inet3101-dynamic-memory> .\1-usestaticarray.c
PS C:\Users\alexr\inet3101-dynamic-memory>
                                           .\1-usestaticarray.exe



Index 0 of the array1 starts at: 000000705611F6D8
Index 1 of the array2 starts at: 000000705611F6C8

Please enter a number  4


Current numbers in array1 are:  4  4  4  4

Current numbers in array2 are:  10  11  12  13

Please enter a number
```


Developer PowerShell for VS

```
Index 0 of the array1 starts at: 000000705611F6D8
Index 1 of the array2 starts at: 000000705611F6C8

Please enter a number  4


Current numbers in array1 are:  4  4  4  4

Current numbers in array2 are:  10  11  12  13

Please enter a number  5


Current numbers in array1 are:  5  5  5  5

Current numbers in array2 are:  10  11  12  13

Please enter a number  6


Current numbers in array1 are:  6  6  6  6

Current numbers in array2 are:  10  11  12  13

Please enter a number  7


Current numbers in array1 are:  7  7  7  7

Current numbers in array2 are:  10  11  12  13

Please enter a number  8
Warning: Array1 is full. Value 8 discarded.

Current numbers in array1 are:  -524483334  -524483334  -524483334  -524483334

Current numbers in array2 are:  10  11  12  13

Please enter a number
```

Question 3.

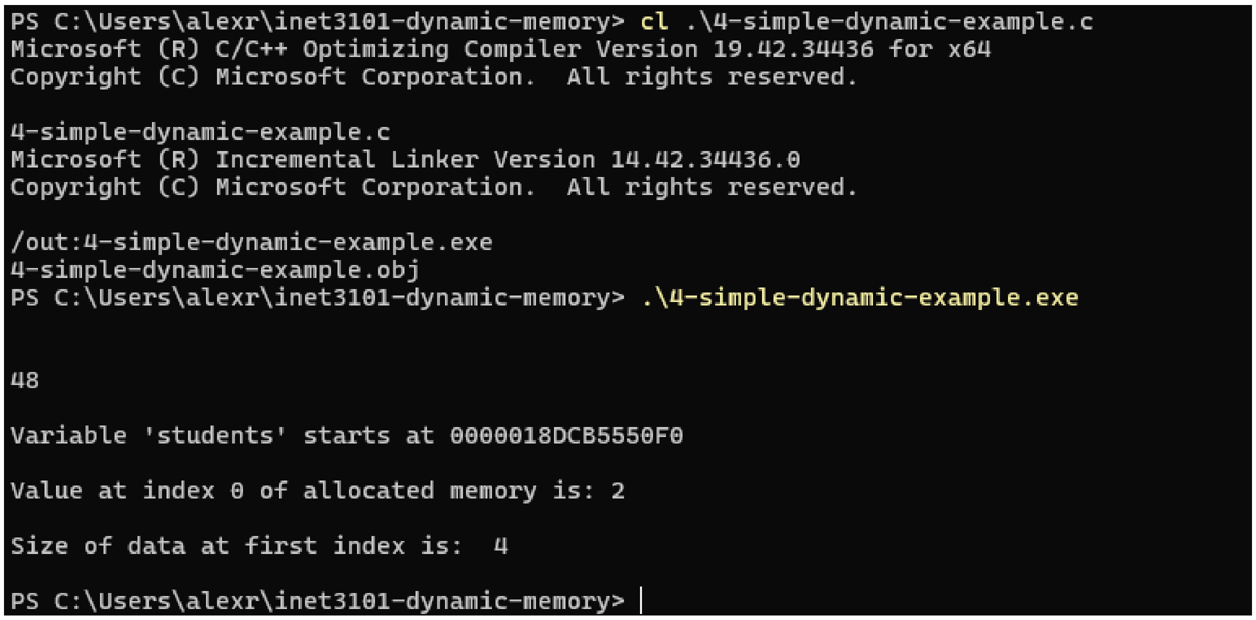
My Approach and Why It Works:

A dynamic resizing strategy was implemented using `realloc()` doubling the array's capacity upon its fullness . This approach:

1. Tracks both the current number of elements (`size`) and the total capacity of the allocation (`capacity`)
2. `realloc()` doubles the `capacity` when `size` gets to capacity.
3. In case that reallocation might fail we safely do it using a temporary pointer (`temp`)
4. The loop goes on forever until the user enters a non-integer value
5. Error handling is to prevent memory leak if allocation fails

Growth is performed geometrically (i.e. each array double in size as opposed to increment) so fewer expensive reallocations are needed. It's also memory safe since we check for allocation failures properly and it is also guaranteed that we free the memory. It is seamless to the user experience: the user can continue to enter numbers without being aware of the internal resizing operations (I did add a notification so you know when resizing occurs).

Question 4.



```
PS C:\Users\alexr\inet3101-dynamic-memory> cl .\4-simple-dynamic-example.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.42.34436 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

4-simple-dynamic-example.c
Microsoft (R) Incremental Linker Version 14.42.34436.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:4-simple-dynamic-example.exe
4-simple-dynamic-example.obj
PS C:\Users\alexr\inet3101-dynamic-memory> .\4-simple-dynamic-example.exe


48

Variable 'students' starts at 0000018DCB5550F0

Value at index 0 of allocated memory is: 2

Size of data at first index is:  4

PS C:\Users\alexr\inet3101-dynamic-memory>
```

Question 5:

Object Oriented Programming (OOP) is where we have objects which are basically self enclosed units that include data(attribute) and behavior(methods) into one entity. The classes are basically the blueprint of the objects and the objects are the instances of the classes. This is encapsulation of state and functionality in a way that abstractions for data are possible, where the implementation details are hidden and only a clean interface for interaction exists. The concept of OOP expressed above is exactly what a Python List constitutes. If we create a list with the value `[1,3,7,4]`when we do, we're not just allocating space for the data to be stored as we would with a C array. What we are actually doing is we are instructing the system to instantiate a list object which contains both the real data elements as well as many built in methods which work on that data.(`append()`, `insert()`, `sort()`, etc.) are these methods, which encapsulate the complex operation and memory management. For instance, suppose we call `list.append(10)`. list object addresses all the underlying memory reallocation by itself behind the scenes as the array grows. The partial abstraction shows that the internal implementation details (dynamic memory allocation for example) are completely hidden from us. Instead of dealing with manual pointer arithmetic as you would in C, we apply the defined interface of the list on that list. It shows that, like all Python lists, Python lists adhere to the basic OOP principles of encapsulation (chunking up data with methods), abstraction (concealing the details of the implementation), and offering a clean and straightforward interface to manipulate.

Question 6:

A linked list is used to manage the data to overcome the issues arising due to dynamic memory allocation by introducing a flexible and efficient method to store and operate on a group of data. A static array is fixed size predefined (the memory should be set aside), there is possibility of memory waste or overflow, on the other hand linked lists can grow dynamically as elements are inserted or removed. In a linked list, each node has a pointer to the next node. Thus, inserting and deleting elements in a linked list is efficient as compared to that of an array, where in the elements would have to shift. Moreover, this structure minimizes the memory fragmentation as nodes can be allocated and deallocated independently. Overall, linked lists serve as an example of how to use dynamic memory to make data structures that can accommodate data of different sizes and application needs.