STATIC ARRAY

Insert n into arr at the next open position.

Length is the number of 'real' values in arr, and capacity

is the size (aka memory allocated for the fixed size array).

def insertEnd(arr, n, length, capacity):

#This line defines a function named insertEnd that takes four parameters:

‘’’

- **arr**: The array (or list) where we want to insert a new element.
- **n**: The new element to be inserted at the end of the array.
- **length**: The current number of elements in the array (i.e., the number of meaningful, initialized values in arr).(actual values excluding 0)
- **capacity**: The total size of the array (i.e., the maximum number of elements it can hold).’’’

if length < capacity:

‘’’ **if length < capacity:**

- This line checks if the array has space to insert a new element.
- **length < capacity** means that the current number of elements (length) is less than the total size of the array (capacity).
- If this condition is true, there is room to add a new element. If false, the array is full, and no insertion should be made.

’’’

arr[length] = n

‘’’ This line inserts the new element n at the position length in the array.

 Since arrays in Python are 0-indexed, length is the index where the new element should be placed.

 After insertion, the number of meaningful elements in the array should be increased by 1 (though the function itself doesn't do this—it should be handled outside the function).’’’

Ex:

- **Array arr**: [10, 20, 30, 0, 0] (initialized with a capacity of 5, where the last two 0s are placeholders for uninitialized elements)
- **length**: 3 (meaning there are 3 actual values: 10, 20, and 30)
- **capacity**: 5 (meaning the array can hold up to 5 elements)
- **n**: 40 (the new element we want to insert at the end)

Step-by-Step Execution:

1. **Condition Check (if length < capacity:)**
    - Current length is 3, and capacity is 5.
    - 3 < 5 is True, so the array has space to insert a new element.
2. **Insertion (arr[length] = n)**
    - The element n = 40 is inserted at index length = 3.
    - The array now looks like this: [10, 20, 30, 40, 0].
    - The length would now be updated to 4 (though this update is done outside the function).

So, after calling insertEnd(arr, 40, 3, 5), the array arr becomes [10, 20, 30, 40, 0], with the next length being 4.

Remove from the last position in the array if the array

is not empty (i.e. length is non-zero).

def removeEnd(arr, length):

‘’’ **def removeEnd(arr, length):**

- This line defines a function named removeEnd that takes two parameters:
    - **arr**: The array (or list) from which we want to remove the last element.
    - **length**: The current number of elements in the array (i.e., the number of meaningful, initialized values in arr).

’’’

if length > 0:

‘’’ **if length > 0:**

- This line checks whether the array has any elements (i.e., it is not empty).
- **length > 0** means that there is at least one element in the array.
- If this condition is true, the function proceeds to remove the last element. If false, the function does nothing because there is no element to remove.

’’’

Overwrite last element with some default value.

We would also the length to decreased by 1.

arr[length - 1] = 0

- ‘’’ **arr[length - 1] = 0**
    - This line removes the last element in the array by overwriting it with a default value (in this case, 0).
    - Since arrays in Python are 0-indexed, the last element is at index length - 1.
    - By setting arr[length - 1] to 0, the function effectively "removes" the element, although technically, it just resets it to a default value.
    - **Note**: The actual removal involves reducing the length by 1, but this step is expected to be handled outside the function.

Example

Let's say we have the following scenario:

- **Array arr**: [10, 20, 30, 40, 0] (initialized with a capacity of 5, with the last 0 as a placeholder for an uninitialized element)
- **length**: 4 (meaning there are 4 actual values: 10, 20, 30, and 40)

Step-by-Step Execution:

1. **Condition Check (if length > 0:)**
    - Current length is 4.
    - 4 > 0 is True, so the array has elements that can be removed.
2. **Overwrite the Last Element (arr[length - 1] = 0)**
    - The last element is at index length - 1 = 3.
    - The element at arr[3] is 40.
    - The function sets arr[3] to 0, so the array now looks like this: [10, 20, 30, 0, 0].

After calling removeEnd(arr, 4), the array arr becomes [10, 20, 30, 0, 0], with the last meaningful element (40) removed. The next length should now be updated to 3 (though this update should be done outside the function).

’’’

Insert n into index i after shifting elements to the right.

Assuming i is a valid index and arr is not full.

def insertMiddle(arr, i, n, length):

  # Shift starting from the end to i.

‘’’ **def insertMiddle(arr, i, n, length):**

- This line defines a function named insertMiddle that takes four parameters:

- **arr**: The array (or list) where we want to insert a new element.
- **i**: The index at which the new element n should be inserted.
- **n**: The new element to be inserted.
- **length**: The current number of elements in the array (i.e., the number of meaningful, initialized values in arr).

’’’

for index in range(length - 1, i - 1, -1):

’’’ **for index in range(length - 1, i - 1, -1):**

- This line sets up a loop that iterates backwards through the array from length - 1 down to i.
- **length - 1** is the index of the last element in the array.
- **i - 1** is the stopping point (the loop runs until index is equal to i).
- **-1** indicates that the loop decrements index by 1 in each iteration, meaning it moves from right to left.

’’’

arr[index + 1] = arr[index]

’’’ This line shifts the element at position index one position to the right (to index + 1).

 By shifting elements to the right, a space is created at index i where the new element can be inserted.’’’

Insert at i

arr[i] = n

- ’’’ **arr[i] = n**
  - This line inserts the new element n at the index i.
  - The space for this element was created by the previous shifting operation.
  - After insertion, the new element n occupies the index i.

Continuing the Example:

- The array after shifting is arr = [10, 20, 30, 30, 40].
- Inserting 25 at index = 2:
  - arr[2] = 25
- **Final Array**:
  - arr = [10, 20, 25, 30, 40]

’’’

‘’’ Consider an array arr = [10, 20, 30, 40, 0] with length = 4, and you want to insert 25 at index 2 (i = 2):

- **Before Shifting**:
  - arr = [10, 20, 30, 40, 0]
- **Shifting Process**:
  - Start from index = 3 (last element 40), and shift it to index = 4: arr = [10, 20, 30, 40, 40]
  - Move to index = 2 (element 30), and shift it to index = 3: arr = [10, 20, 30, 30, 40]

After the loop, the array looks like this: arr = [10, 20, 30, 30, 40], with space at index = 2 ready for the new element.

’’’

Remove value at index i before shifting elements to the left.

Assuming i is a valid index.

def removeMiddle(arr, i, length):

‘’’

def removeMiddle(arr, i, length):

- This line defines a function named removeMiddle that takes three parameters:
  - **arr**: The array (or list) from which we want to remove an element.
  - **i**: The index of the element that should be removed.
  - **length**: The current number of elements in the array (i.e., the number of meaningful, initialized values in arr).

’’’

```
    # Shift starting from i + 1 to end.
    for index in range(i + 1, length):
        arr[index - 1] = arr[index]
```

‘’’

 for index in range(i + 1, length):

- This line sets up a loop that iterates through the array starting from the index just after i (i + 1) to the last element (length - 1).
- **range(i + 1, length)** creates a range from i + 1 (the element immediately after the one being removed) up to but not including length.

 arr[index - 1] = arr[index]

- This line shifts the element at index one position to the left (to index - 1).
- This effectively overwrites the element at i, removing it by filling its space with the next element.
- The loop continues this process for all elements after i, shifting them leftward to close the gap created by the removed element.

’’’

 # No need to 'remove' arr[i], since we already shifted

’’’

Example of Shifting:

Consider an array arr = [10, 20, 30, 40, 50] with length = 5, and you want to remove the element at index 2 (i = 2).

- **Before Removal**:
  - arr = [10, 20, 30, 40, 50]
- **Shifting Process**:
  - Start from index = 3 (element 40), and shift it to index = 2: arr = [10, 20, 40, 40, 50]
  - Move to index = 4 (element 50), and shift it to index = 3: arr = [10, 20, 40, 50, 50]

’’’

def printArr(arr, capacity):

 ’’’

This line defines a function named printArr that takes two parameters:

- **arr**: The array (or list) that you want to print.
- **capacity**: The total size or capacity of the array, indicating how many elements to print.

’’’

 for i in range(capacity):

’’’

for i in range(capacity):

- This line sets up a loop that iterates capacity times.

- **range(capacity)** creates a sequence of numbers from 0 to capacity - 1.
- **i** is the loop variable that will take on values starting from 0 up to capacity - 1 during each iteration.

```
    print(arr[i])
'''
```

- **print(arr[i])**
    - This line prints the element at index i in the array arr.
    - During each iteration of the loop, arr[i] accesses the element at the current index i, and print(arr[i]) outputs it to the console.
    - Each element is printed on a new line because print in Python adds a newline character after each call by default.

Example

Let's say you have the following array and capacity:

- **Array arr**: [10, 20, 30, 40, 50]
- **capacity**: 5

Step-by-Step Execution:

1. **First Iteration (i = 0)**:
    - arr[0] is 10.
    - print(arr[0]) prints 10.
2. **Second Iteration (i = 1)**:
    - arr[1] is 20.
    - print(arr[1]) prints 20.
3. **Third Iteration (i = 2)**:
    - arr[2] is 30.
    - print(arr[2]) prints 30.
4. **Fourth Iteration (i = 3)**:
    - arr[3] is 40.
    - print(arr[3]) prints 40.
5. **Fifth Iteration (i = 4)**:
    - arr[4] is 50.
    - print(arr[4]) prints 50.

Output

The output in the console would be:

Copy code

10

20

30

40

50

Each number is printed on a new line, corresponding to each element in the array arr. The function prints all elements of the array up to the specified capacity.

,,,