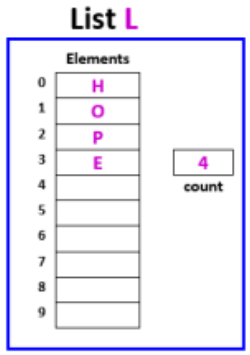
Cursor-based Implementation of List

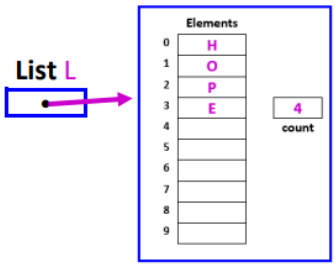
Array and Linked List Implementations of List (A Recall)

A. Array Implementation

In this implementation of list, the elements of the List L are stored in contiguous cells of the array. If an element is deleted, succeeding elements will be shifted to 1 index lower to close the gap.



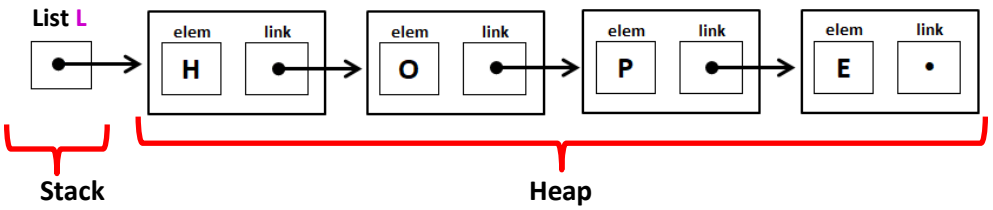
- Array Implementation Version #1
- In this array implementation variation, variable L is allocated storage in the stack.



- Array Implementation Version #2
- In this array implementation variation, variable L is allocated storage in the stack and the variable that L is pointing to is dynamically allocated and stored in the heap.

B. Linked List Implementation

In this implementation, variable L is stored in the stack and the elements of the List L are stored in dynamically allocated nodes that are stored in random areas in the heap. Management of the heap is handled by the Operating System. In C language, the functions malloc(), calloc(), realloc(), and free() are can be invoked when program tasks involve dynamically allocated spaces. Definitions of these functions are found in stdlib.h library.

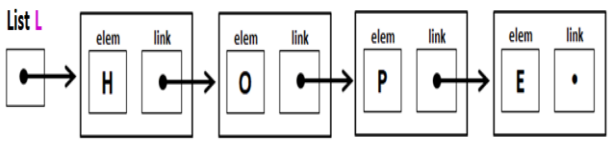
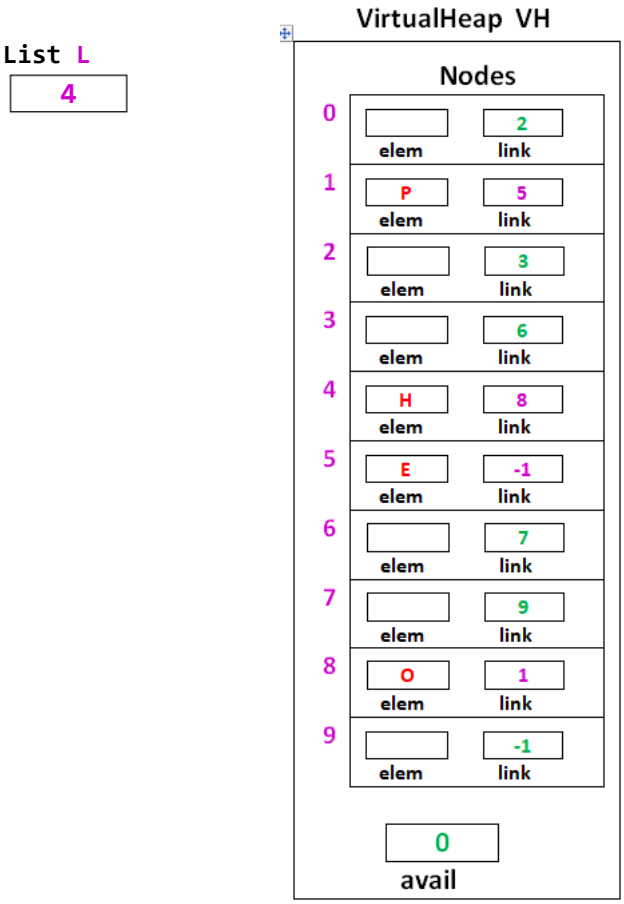


Cursor-based Implementation

This implementation uses both array and linked list. It simulates linked list implementation, but uses array nodes/cells instead of dynamically allocated nodes.

In the illustration below, the array variable Nodes and variable avail, are group together in a structure call the VirtualHeap. Like in linked list, the cursor-based list has also 4 elements: H, O, P, and E. List L has a value of 4 since the index of element H is 4. In effect L points to the array node containing element H. The link field of index 4 is 8, pointing to the 2nd element which is O. The link field of index 8 is 1, pointing to the 3rd element which is P. The link field of index 1 is 5, pointing to the 4th element which is E. The link field of index 5 is -1, a sentinel value which indicates that it is not pointing to any node, i.e. end of the list.

Available nodes in the array have indexes: 0, 2, 3, 6, 7, and 9. Variable avail holds the index of the 1st available node. The sentinel value -1 is used to indicate the end of the list of available nodes.

Linked List	Cursor-Based
	//Sample Illustration. Random placement of elements H,O,P,E in the virtual heap which is a possible scenario. 

Facts about the Virtual Heap

1) **It can be implemented using any of the 4 versions of array implementation. See Versions 1 and 2 of the Array implementation of List.**
2) **It can be shared by 2 or more lists containing the same element type.**
3) **Since the virtual heap is defined by the programmer, hence the virtual heap management functions will also be defined by the programmer.**

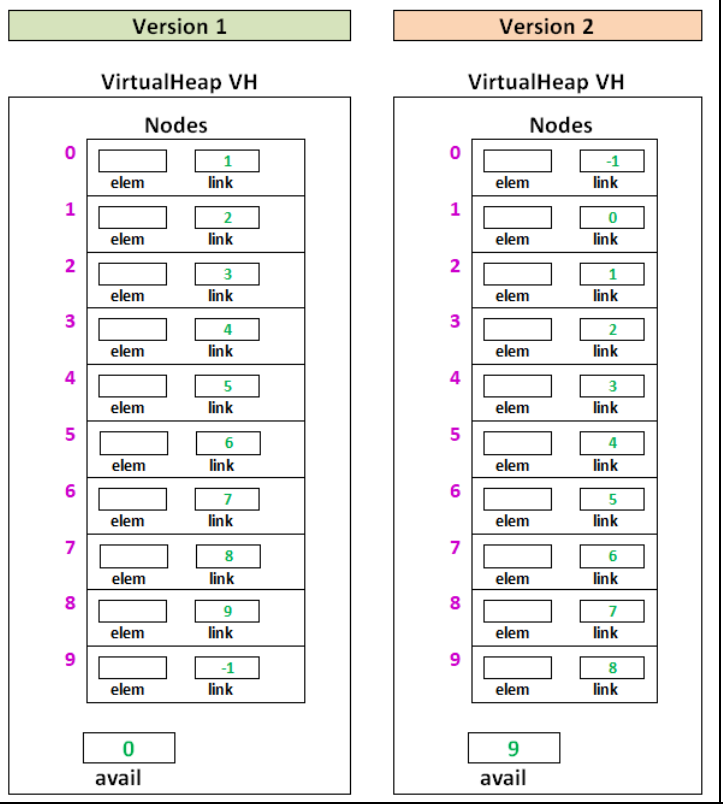
3 Virtual Heap Management Operations

a) **initializeVH().** Initializes the virtual heap by linking all the cells in the array and storing the 1st available node in the array in variable avail. See illustration below.

b) **allocSpace().** This is equivalent to malloc() . This function will remove the first available node in the virtual heap if there is still space in the virtual heap and returns the index of that node to the calling function. It returns -1 if there is no more available space. The process of this function is similar to deleteFirst() in linked list.

c) **deallocSpace().** This is equivalent to free(). Given the index of a node in the array, the function will put back the node in the list of available nodes in the virtual heap. The process of this function is similar to insertFirst() in linked list.

Initializing the Virtual Heap



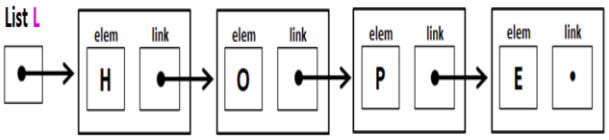
- The illustration on the left shows two different ways/versions in initializing a virtual heap.
- Whatever version will be implemented in function initializeVH(), the codes for allocSpace() and deallocSpace() will not be affected.

Implementation of List: Array vs. Linked List vs. Cursor-based

	Linked List	Array	Cursor-based
Location of elements	Heap	Stack and/or Heap	Stack and/or Heap
Access of elements	Sequential	Sequential or direct access	Sequential

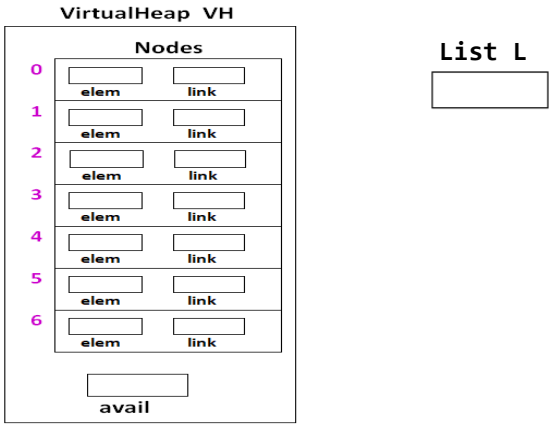
Practice Exercise:

Linked List	Illustration

typedef struct node {
  char data;
  struct node *link;
}*List;
 	
//The real heap is managed by the OS, and the compiler will just make system call to OS when it needs heap space.
	

Write the code of function insert. The function will insert a new character element at the first position of the given list. Note: Include a check to determine if malloc() is successful.	

Data Structure Definition	Illustration of the Virtual heap and cursor-based List

//Definition of Datatype VirtualHeap
#define SIZE 7
typedef struct {
  char elem;
  int link;
}Nodetype;
 	
typedef struct {
  Nodetype Nodes[SIZE];
  int avail; //holds the index of the 1st available node
}VirtualHeap;
 	
//Definition of Datatype List
  typedef int List;
	

Write the code of function insert. The function will insert a new character element at the first position of the given list. Write the function prototype of allocSpace() and assume the that definition of the function exists.	
Compare the codes of the function insert() using linked list and cursor-based implementations.	