10.3-5.

Let L be a doubly linked list of length n stored in arrays *key*, *prev*, and *next* of length m. Suppose that these arrays are managed by ALLOCATE-OBJECT and FREE-OBJECT procedures that keep a doubly linked free list F. Suppose further that of the m items, excatly n are on list L and $m-n$ are on the free list. Write a procedure COMPACTIFY-LIST(L, F) that, given the list L and the free list F, moves the items in L so that they occupy array positions $1, 2, ..., n$ and adjusts the free list F so that it remains correct, occupying array positions $n+1, n+2, ..., m$. The running time of your procedure should be $\Theta(n)$, and it should use only a constant amount of extra space. Argue that your procedure is correct.

Answer.

We represent the combination of arrays *key*, *prev*, and *next* by a multible-array A. Each object of A's is either in list L or in the free list F, but not in both. The procedure COMPACTIFY-LIST transposes the first object in L with the first object in A, the second objects, ... until the list L is exhausted.

```
COMPACTIFY-LIST(L, F)
 1  TRANSPOSE(A[L.head], A[1])
 2  if F.head == 1
 3      F.head = L.head
 4  L.head = 1
 5  l = A[L.head].next
 6  i = 2
 7  while l ≠ NIL
 8      TRANSPOSE(A[l], A[i])
 9      if F == i
10          F = l
11      l = A[l].next
12      i = i + 1
```

```
TRANSPOSE(e_1, e_2)
    SWAP(e_1.prev.next, e_2.prev.next)
    SWAP(e_1.prev, e_2.prev)
    SWAP(e_1.next.prev, e_2.next.prev)
    SWAP(e_1.next, e_2.next)
```

```
SWAP(x, y)
    temp = x
    x = y
    y = temp
```

This COMPACTIFY-LIST procedure takes time $\Theta(n)$, and it use only a constant amount of extra space.

To prove that this algorithm is correct, observe that at any moment, we can devide A into two subarrays: $A[1..i-1]$ of compact leading objects in L and $A[i..m]$ of objects to be compactify. This reveals the following loop invariant:

> At the start of each iteration of the **while** loop of line 7–12, the objects $A[1..i-1]$ are the first $i-1$ compact objects in list L.

Email address: informlarry@gmail.com

This loop invariant should help us prove the correctness of COMPACTIFY-LIST. For convenience, we denote the sequence of objects in L by $L_1, L_2, ..., L_n$.

Initialization. Before entering the iteration of the **while** loop, the procedure transpose L_1 with the first one in A, making L's first object takes up $A[1]$. It then increments i by 1 to be $i=2$, so that $A[1\,..\,i-1]=A[1]$ is the only compact object in list L. If the free list F happens to be started at $A[1]$, we update its head after this transposition.

Maintenance. The loop continues as long as the list L has not been exhausted. It goes by transposing the remaining incompact objects L_2, L_3, ..., L_n of L and reallocates them at $A[2]$, $A[3], ..., A[n]$. The compact sublist of L expands from $A[1\,..\,i-1]$ to $A[1\,..\,i]$. Incrementing i for the next iteration of the **while** loop then preserves the loop invariant. Still, if F's head is encountered in this process, we update its pointer to the new position after transposition.

Termination. The condition cause the **while** loop terminate is that l = NIL. Because each loop iteration increase i by 1 and list L has a length n, we must have $i=n+1$ at that moment. Substituting $n+1$ for i in the statement of the loop invariant, we have that the objects $A[1\,..\,n]$ are the first n compact objects in list L. Since each object in array A is either in list L or in the free list F, the rest $m-n$ continuous objects $A[n+1]$, $A[n+2]$, ..., $A[m]$ must all belongs to the free list F. Hence, the algorithm is correct.