Exercise Session n. 7

Algorithms and Data Structures

We practice a bit with linked-lists. All the algorithms we want to develop must be *in-place,* meaning that they may not create any new `List` object nor any other data structure (such as arrays) to store list elements, even if only temporarily.

Invert the Order of a Single-Link List

Consider a single-link list defined by the following Python class:

```python
class List:
    def __init__(self,v):
        self.value = v;
        self.next = None
```

You may also use the following *print* function:

```python
def print_list(l):
    while l != None:
        print(l.value)
        l = l.next
```

Write a function `reverse_list(l)` that, given a linked list l returns a linked-list that contains exactly the same elements initially in l, but in reverse order. `reverse_list(l)` must work *in-place*, meaning that it may not create new `List` objects or other data structures such as arrays to store list elements, even if only temporarily. In other words, `reverse_list(l)` must simply rearrange the `List` objects from l.

Examples

```
>>> l = List(7)
>>> l.next = List(10)
>>> l.next.next = List(31)
>>> print_list(l)
7
10
```

```
31
>>> l = reverse_list(l)
>>> print_list(l)
31
10
7
```

Concatenating Two Single-Link Lists

Write a function `concatenate_lists(l1,l2)` that, given two lists l_1 and l_2 returns a list that contains all the elements of l_1, in the original order, followed by all the elements of l_2 also in the original order. `reverse_list(l)` must work *in-place.* Also, analyze the complexity of your solution.

Examples

```
>>> l1 = List(7)
>>> l1.next = List(10)
>>> l1.next.next = List(31)
>>> l2 = List(67)
>>> l2.next = List(99)
>>> print_list(l1)
7
10
31
>>> print_list(l2)
67
99
>>> l = concatenate_list(l1,l2)
>>> print_list(l)
31
10
7
67
99
```

Invert the Order of a Doubly-Linked List

Consider a doubly-link list with sentinel defined by the following Python class and functions:

```
class List:
    def __init__(self):
        self.value = None;
        self.next = self
        self.prev = self

def list_append (l, v):
    n = List()
    n.prev = l.prev
    n.next = l
    n.prev.next = n
    n.next.prev = n

def print_list(sentinel):
    l = sentinel.next
    while l != sentinel:
        print(l.value)
        l = l.next
```

Write a function `reverse_list(l)` that, given a doubly-linked list l, returns a doubly-linked list that contains exactly the same elements initially in l in reverse order. `reverse_list(l)` must work *in-place*, meaning that it may not create new `List` objects or other objects (arrays, etc.) to store the elements, even if only temporarily.

Examples

```
>>> l = List()
>>> list_append(l, 7)
>>> list_append(l, 10)
>>> list_append(l, 31)
>>> print_list(l)
7
10
31
>>> l = reverse_list(l)
>>> print_list(l)
31
10
7
```

Concatenating Two Doubly-Linked Lists

Write a function `concatenate_lists(l1,l2)` that, given two doubly-linked lists (with sentinel) l_1 and l_2, returns a doubly-linked list that contains all the elements of l_1, in the original order, followed by all the elements of l_2 also in the original order. `reverse_list(l)` must work in-place, meaning that it may not create new `List` objects or other objects (arrays, etc.) to store the elements, even if only temporarily. Also, analyze the complexity of your solution.

Examples

```
>>> a = List()
>>> for v in [1,2,3,4]:
...     list_append(a,v)
...
>>> print_list(a)
1
2
3
4
>>> b = List()
>>> for v in [10,20,30]:
...     list_append(b,v)
...
>>> print_list(b)
10
20
30
>>> c = concatenate_list(a,b)
>>> print_list(c)
1
2
3
4
10
20
30
```

Concatenating Two Doubly-Linked Lists in Constant Time

If you haven't done that already with the previous exercise, write a function `concatenate_lists(l1,l2)` that concatenates two lists l_1 and l_2 in constant time (and also *in-place* exactly as specified in the previous exercise).