



Linked Lists

The Archnemesis of Arrays

Tim Ngo - tingo@student.42.us.org
42 Staff - pedago@42.fr

*Summary: This is an introduction to **Linked Lists**.*

Contents

I	Foreword	2
II	Introduction	3
III	Goals	4
IV	General instructions	5
IV.1	Node Class	6
IV.2	SinglyList Class	6
V	Exercise 00 : Print All Nodes in a List	7
VI	Exercise 01 : Add an Item to the End of a List	8
VII	Exercise 02 : Remove an Item From a List	9
VIII	Exercise 03 : Cycle Detection	10
IX	Exercise 04 : Merge Two Lists	11
X	Exercise 05 : Sort a Linked List	12
XI	Bonus part	13
XII	Exercise 06 : Trainyard Revisited	14
XIII	Turn-in and peer-evaluation	15

Chapter I

Foreword

The forewords section of a 42 subject is usually not related in any way to the actual topic of the subject. The idea is to share some jokes (often questionable) or something that the community might be interested in.

TIM YOU WILL NEED TO FIGURE THIS OUT

As a consequence, let's use the forewords section of this sample 42 subject to introduce the contents of this document and its goals. In particular, the formatting of a trivial **LaTeX** document and the normalized chaptering of our subjects. If you read this from the pdf, don't forget to open the source file (file **sample.en.tex**) next to this pdf, in order to see behind the scenes, and to understand which command generates which result. Otherwise, if you have started with the sources, congrats, that's the spirit ! But open the pdf (file **sample.en.pdf**) anyway.

What to do if the file **sample.en.pdf** is not available ? Easy, just compile the source file **sample.en.tex** using the shell command **make**. Please refer to the documentation to set up **LaTeX** on your system if needed.

If you're not familiar with **LaTeX**'s syntax, here is a fairly exhaustive list of everything you'll need to write your subject.

Chapter II

Introduction

Today's lecture gave us a general overview of linked lists and their uses in general coding practice. Linked lists are the foundation for more advanced data structures and are commonly used in place of arrays for data storage due to their ability to store more information in non-contiguous blocks of memory. This makes linked lists a very powerful tool for parallel computation.

There are many applications to the different variations of linked lists. Many games will store cycling character animations in a circular linked list or a circular array. Many operating systems line up processes and jobs in a queue. The desire to simulate genetic mutation and give people superpowers by removing and swapping base pairs in DNA strands can best be organized by a doubly linked list.

Within the 42 curriculum, the entire graphics branch frequently utilizes linked lists to store coordinates of points for an object in a map or rendering. In many Unix projects, making a process and job queue for operating systems is essential to not making a computer crash.

Chapter III

Goals

This exercise set contains 7 exercises which include adding and removing items, detecting infinite cycles, and sorting a list. I hope you read this Tim because I have no idea what I want them to accomplish from doing these exercises.

FIX THIS TIM FIX THIS TIM FIX THIS TIM FIX THIS TIM FIX THIS TIM FIX THIS TIM
THIS TIM

Chapter IV

General instructions

The following exercises are designed such that written functions are to be turned in as standalone functions; they will not be included in the class for `LinkedLists` but will still perform standardized operations. All exercises should be written in Python and no external function calls should be made except for the first question; only the functions supplied in the classes provided to you should be needed.

Both a `Node` and `SinglyList` class have been given to you for use. The `SinglyList` class has an in-class iterator defined to traverse a list, but you are free to iterate through the list yourself :)

If a function requires the use of the `Node` or `SinglyList` class, do not include that code in your file submission. If you use a previous solution please make sure to include that function in the file submission.

IV.1 Node Class

```
class Node(object):
    def __init__(self, content):
        if content is None:
            raise ValueError('Node must have content')
        self.c = content
        self.n = None

    @property
    def content(self):
        return self.c

    @content.setter
    def content(self, val):
        self.c = val

    @property
    def next(self):
        return self.n

    @next.setter
    def next(self, val):
        self.n = val
```

IV.2 SinglyList Class

```
class SinglyList(object):
    def __init__(self):
        self.h = None

    def __iter__(self):
        current = self.head
        while current:
            yield current
            current = current.next

    @property
    def head(self):
        return self.h


    @head.setter
    def head(self, val):
        self.h = val

    def isEmpty(self):
        return self.head == None

    def add_head(self, node):
        if self.isEmpty():
            self.head = node
        else:
            node.next = self.head
            self.head = node
```

Chapter V

Exercise 00 : Print All Nodes in a List

	Exercise 00
Print All Nodes in a List	
Turn-in directory : <i>ex00/</i>	
Files to turn in : <code>print_list.py</code>	
Forbidden functions : Everything except <code>print()</code> :D	
Notes : n/a	

Understanding how to traverse a linked list is important to mastering its concept. Given the head node of a singly linked list, print out all the items in the list.

Input Format


Complete the function `print_list(list_head)` which takes the head of a list.

Output Format

Just print out the items in the list in order. If a list is empty do not print anything. No return is needed for the function.

Chapter VI

Exercise 01 : Add an Item to the End of a List

	Exercise 01
Add an Item to the End of a List	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <code>add_tail.py</code>	
Forbidden functions : Everything :D	
Notes : There is a similar function in the <code>SinglyList</code> class to reference :)	

You're given the pointer to the head node of a singly linked list and the value of a node to add to the list. Create a new node with the given value. Insert this node at the tail of the linked list and return the head node after the insertion.

Input Format


Complete the function `add_tail(list_head, val)` which takes the head of a list and the value to add.

Output Format

Add the requested value into the back of the list as a node. No return is needed for the function.

Chapter VII

Exercise 02 : Remove an Item From a List

	Exercise 02
Remove an Item From a List	
Turn-in directory : <i>ex02/</i>	
Files to turn in : <code>remove.py</code>	
Forbidden functions : Everything :D	
Notes : If the list is empty, head will be null. The list will not contain duplicates.	

You're given the pointer to the head node of a singly linked list and the value of a node to delete from the list. Delete the node with the given value if it exists.

Input Format


Complete the function `remove(list_head, val)` which takes the head of a list and the value to delete. For simplicity, all node values will be numbers.

Output Format

If the node exists, delete that node and link its previous and next references together. If the node does not exist, do nothing. No return is needed for the function.

Chapter VIII

Exercise 03 : Cycle Detection

	Exercise 03
Cycle Detection	
Turn-in directory : <i>ex03/</i>	
Files to turn in : <code>has_cycle.py</code>	
Forbidden functions : Everything :D	
Notes : If the list is empty, head will be null.	

In a turn-based multiplayer game, a linked list can be used to cyclically repeat player order by having the last player's `next` reference the first player. Check that a given linked list cycles or not.

Input Format


Complete the function `has_cycle(list_head)` which takes the head of a list and determines whether the list cycles through the list repeatedly or not.

Output Format

If there is a cycle, return `True`; otherwise, return `False`.

Chapter IX

Exercise 04 : Merge Two Lists

	Exercise 04
Merge Two Lists	
Turn-in directory : <i>ex04/</i>	
Files to turn in : <i>merge.py</i>	
Forbidden functions : Everything :D	
Notes : All trains have at least one car, and no two cars have the same weight.	

Two cargo trains arrive in the trainyard and their cargo needs to be consolidated into a single train set before it can depart. Both trains were organized with the heaviest car in the front of the train descending to the lightest car in the back.

Input Format


Complete the function `merge(train1, train2)` which takes the head of two lists `train1` and `train2` and merges both train sets into a single train set. Consider reusing a function you've already written.

Output Format

Return the head of a new train list with all cars sorted by weight from heaviest to lightest.

Chapter X

Exercise 05 : Sort a Linked List

	Exercise 05
Sort a Linked List	
Turn-in directory : <i>ex05/</i>	
Files to turn in : <code>sort_asc.py</code>	
Forbidden functions : Everything :D	
Notes : There is no limitation on which sort to implement for this exercise. You will find some sorts are easier to work into a linked list than others...	

Understanding how to organize information in a list is important. Implement a sorting algorithm that organizes a linked list with numbers in **ascending** order.

Input Format

Complete the function `sort_asc(unsorted_list)` which takes the head of an unsorted list.

Output Format

Sort the list within the function and return nothing.


Chapter XI

Bonus part

Let's try to combine some of the concepts we've gone over and go back to a problem we've looked at with a little more difficulty. FIX THIS TIM FIX THIS TIM FIX THIS TIM FIX THIS TIM FIX THIS TIM FIX THIS TIM

Chapter XII

Exercise 06 : Trainyard Revisited

	Exercise 06
Trainyard Revisited	
Turn-in directory : <i>ex06/</i>	
Files to turn in : <code>trainyard.py</code>	
Forbidden functions : Everything :D	
Notes : All trains have at least one cart, and two carts may have the same weight.	

Two more cargo trains arrive in the trainyard and need to be merged before departure. These two trains are no longer organized by weight from heaviest to lightest prior to the merge. The two lightest cars must also stay behind at the trainyard for inspection and leave with the next incoming train.

Input Format

Complete the function `trainyard(train1, train2)` which takes the head of two lists `train1` and `train2`. Consider making more than one function to turn in.

Output Format

Return the head of a new train list with all train cars organized from heaviest to lightest. Remember to remove the two lightest trains from the set and to take into account cars with equal weights.

Chapter XIII

Turn-in and peer-evaluation

I'm not sure that they do peer evaluation so I'll leave this section in. FIX THIS TIM
FIX THIS TIM FIX THIS TIM FIX THIS TIM FIX THIS TIM FIX THIS TIM