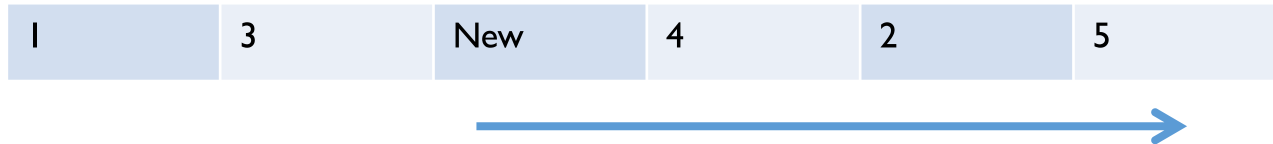
Linked Lists

Prof. Michael Tsai

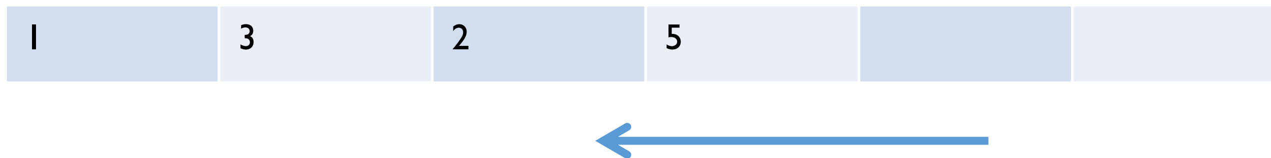
2017/3/14

What's wrong with Arrays?

- Inserting a new element



- Deleting an existing element



- Time complexity= O(??)

Complexity for the array implementation

	Array	Dynamic Array (Expand to twice of the original space when full)	Linked List (What we are learning today)
Indexing (Get a particular element)			??
Insert/Delete at the head			??
Insert/Delete at the tail			??
Insert/Delete in the middle			??
Wasted space			??

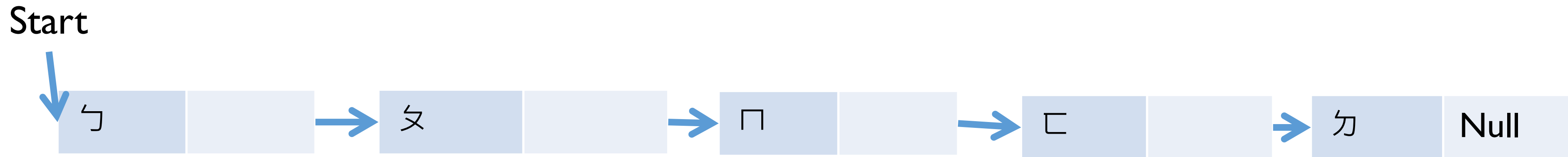
New friend: Linked List

- How do we arrange the data, such
  1. We can arbitrarily change their order,
  2. But we still keep a record of their order?

- Answer:
- The order of the elements can be arbitrary,
- But, we store "which is the next" in addition to the data.

index	[0]	[1]	[2]	[3]	[4]
data	ㄇ	ㄆ	ㄅ	ㄉ	ㄈ
Which is the next	4	0	1	-1	3
head	1				

Conceptually, it looks like this:



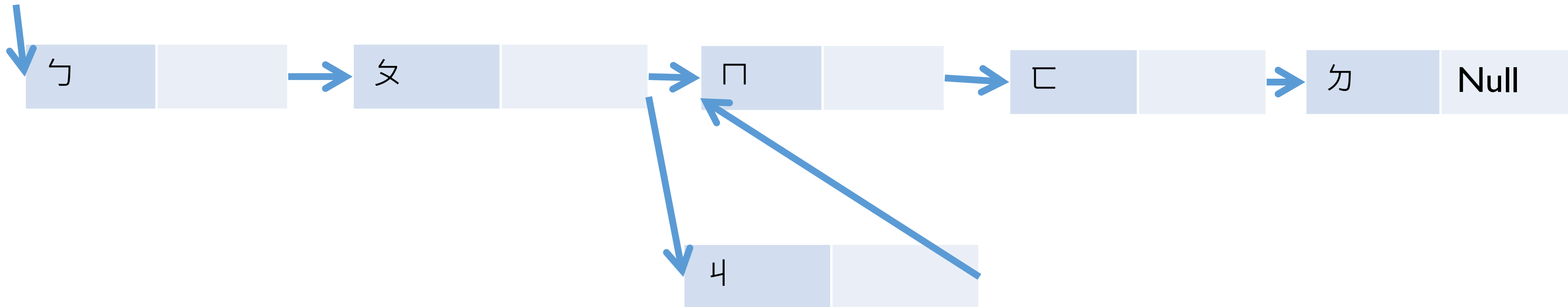
Actual implementation

index	[0]	[1]	[2]	[3]	[4]
data	ㄇ	ㄆ	ㄅ	ㄉ	ㄈ
Which is the next	4	0	1	-1	3

head	2

Add an additional element?

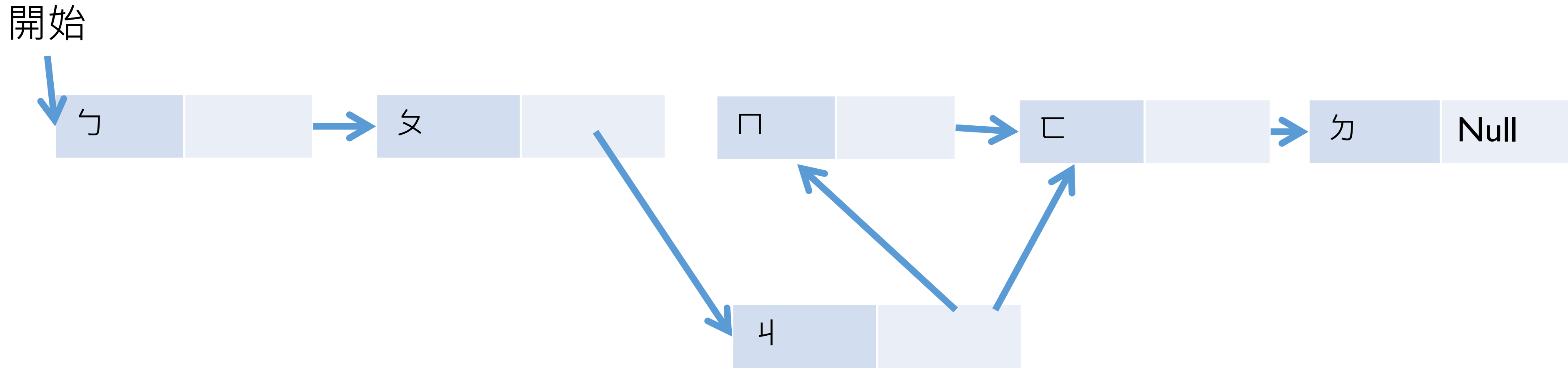
Start



index	[0]	[1]	[2]	[3]	[4]	[5]
data	ㄇ	ㄆ	ㄅ	ㄉ	ㄈ	ㄐ
Which is the next	4	5	1	-1	3	0

head	2

Remove an existing element?



index	[0]	[1]	[2]	[3]	[4]	[5]
資料	ㄇ	ㄆ	ㄅ	ㄉ	ㄈ	ㄐ
下一個是誰	4	5	1	-1	3	4

開始	2

Code segments:
Struct and create a new node.

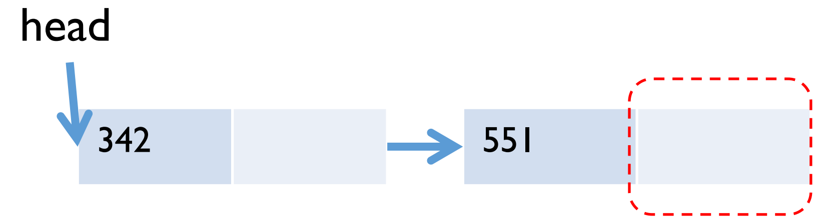
```
1  //Structure declaration for the list node
2  struct ListNode {
3  int data;
4  struct ListNode *next;
5  };
6
7  //Create a new node
8  struct ListNode *new;
9  new=(struct ListNode*)malloc(sizeof(struct listNode));
```

Code segments: accessing the structure members

- `new` is a pointer , pointing at a variable of type `struct listNode`.
- How do we obtain the member `data` in this variable?
- Answer: by
- `(*new).data`
- Or,
- `new->data`

- How about `next`?
- `(*new).next`
- 或者,
- `new->next`

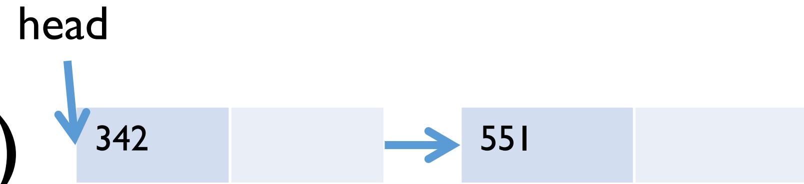
Code segments: accessing the next node



- Assume `head` points at the first node
- How do I get the value of `next` in the "551 node"?

```
1  struct ListNode {
2  int data;
3  struct ListNode *next;
4  };
```

Create two nodes (Insert from the head)



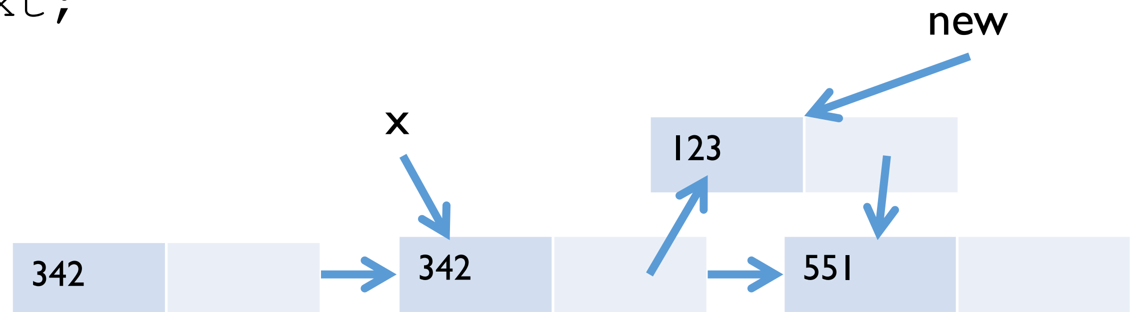

```
1   struct ListNode *head, *tmp;
2
3   tmp=(struct ListNode*)malloc(sizeof(struct ListNode));
4   if (tmp==NULL)
5           exit(-1); // exit program on error
6
7   tmp->data=551;
8   tmp->next=NULL;
9
10  head=tmp;
11  tmp=(struct ListNode*)malloc(sizeof(struct ListNode));
12
13  tmp->data=342;
14  tmp->next=head;
15
16  head=tmp;
```

Insert a new node after a certain node

```
1  struct ListNode *x;
2  //Pointing at the node before the location to
3  //be inserted
4
5  struct ListNode *new;
6
7  new=(struct ListNode*)malloc(sizeof(struct ListNode));
8  new->data=123;
```

Do we process `new->next` first or `x->next` first?

```
1  new->next=x->next;
2  x->next=new;
```

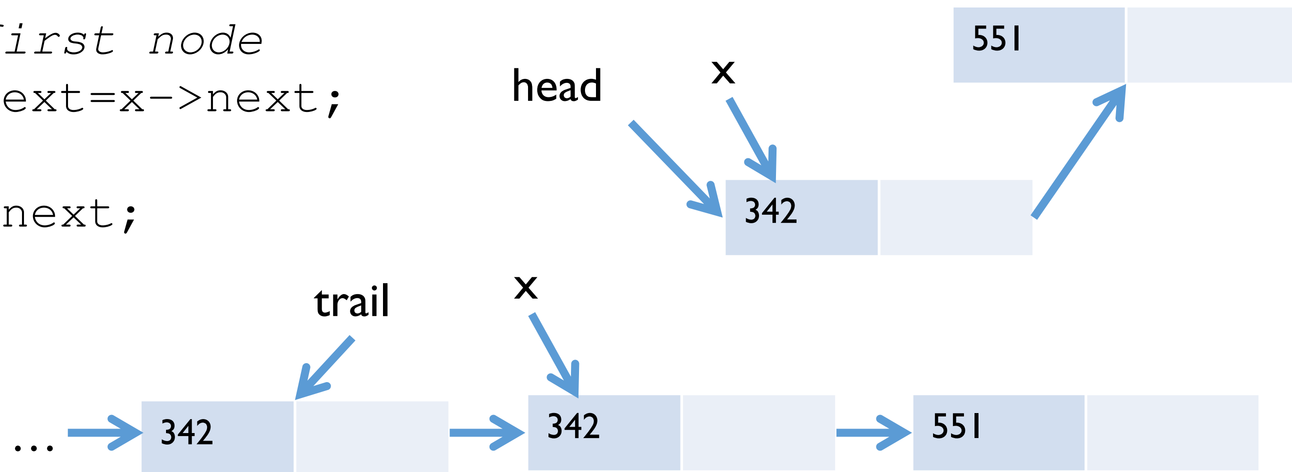


Deleting a node

```
1   struct ListNode *head; //Pointing at the head node
2   struct ListNode *x;
3   //Pointing at the node to be deleted
4   struct ListNode *trail;
5   //Pointing at the node before the node to be //deleted
```

- Two possible conditions: `x` is/is not the head node

```
1   if (trail)
2   //x is not the first node
3           trail->next=x->next;
4   else
5           head=x->next;
6   free(x);
```



Examples: Traverse and Print

- Traverse (and print) linked list

```
1  struct ListNode *tmp;
2  for(tmp=head; tmp!=NULL; tmp=tmp->next) {
3          printf("%d", tmp->data);
4  // you can do other processing here too
5  }
```

Correct the code below: Find

- Find the location before a node with a particular data value

```
1  int a=123; //123 is the data to look for
2  struct ListNode *tmp;
3  for(tmp=head; tmp!=NULL; tmp=tmp->next) {
4          if (tmp->next->data==a)
5                          break;
6             //when breaking, tmp is what we are looking for
7          }
8  }
```

- This code segment would crash in certain conditions. Correct it!

Comparison of complexity

	Array	Dynamic Array (Expand to twice of the original space when full)	Linked List
Indexing (Take a particular element)	O(1)	O(1)	
Insert/Delete at the head	O(n), only feasible if not full	O(n)	
Insert/Delete at the tail	O(1), only feasible if not full	O(1), if not full O(n), if full	
Insert/Delete in the middle	O(n), only feasible if not full	O(n)	
Wasted space	0 (when full)	O(n) (up to half of the space empty)	

Discussion

- When should we use array?
- When should we use linked list?
- Explain why.

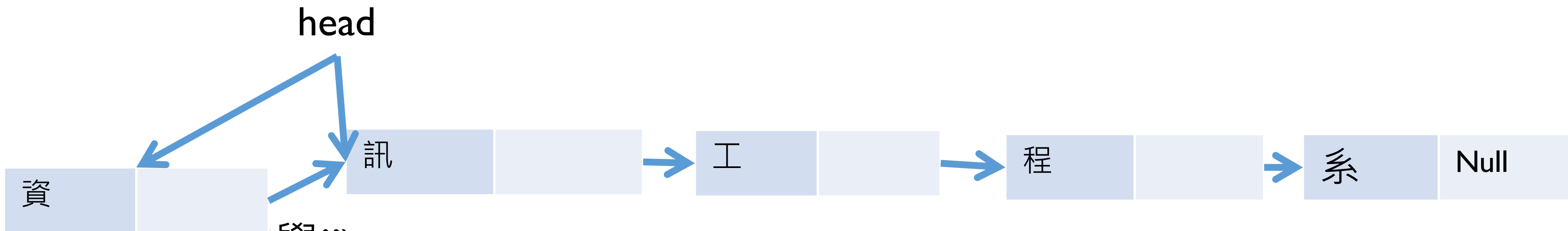
Example: Stacks & Queues

- 如果是一塊記憶體要放很多stack或queue
- 就很難做到很efficient
- 例如如果某一stack滿了, 就要把一堆資料往後擠
- 就不是O(1)了 T_T

- 解決: 跟Linked List當朋友

Stack

- 要怎麼拿來當stack呢? (想想怎麼做主要的operation)
- push & pop
- 請一位同學來講解☺



- 例: push("學")
- head當作stack top
- 怎麼寫code?
- 那pop呢?

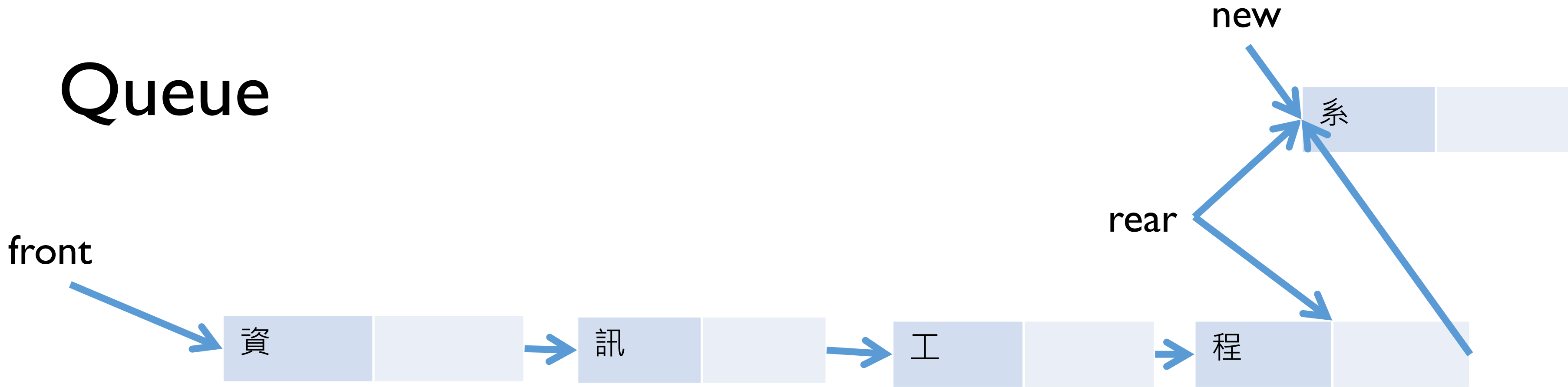
Queue

- 類似stack的作法
- 不過頭尾都要有一個指標
- 從頭拿, 從尾放

- 怎麼拿? (DeQueue)

```
struct ListNode* tmp;
tmp=front;
front=front->link;
tmp_data=tmp->data;
free(tmp);
return tmp_data;
```

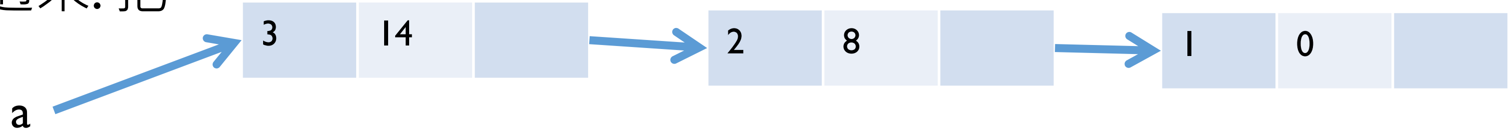
Queue



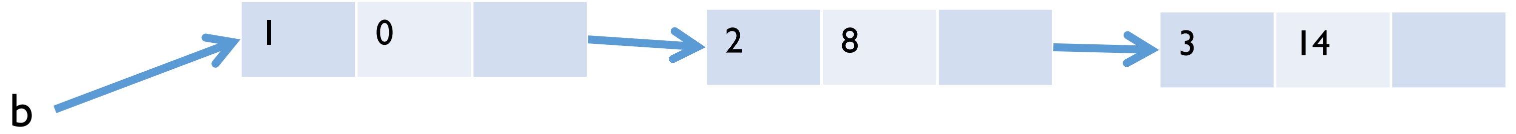
- 那怎麼放?
- 假設new是指到新的node
- rear->next=new;
- new->next=NULL;
- rear=new;

練習題2: 把linked list反過來

- 反過來: 把



- 變成

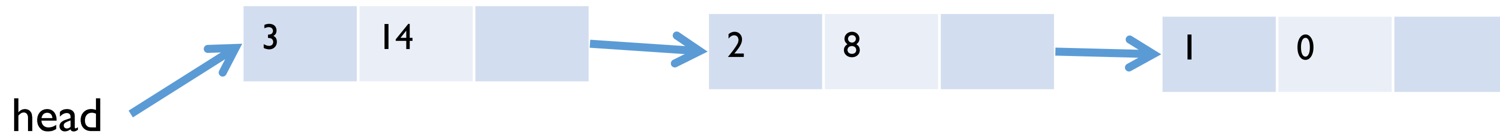


- 怎麼弄?

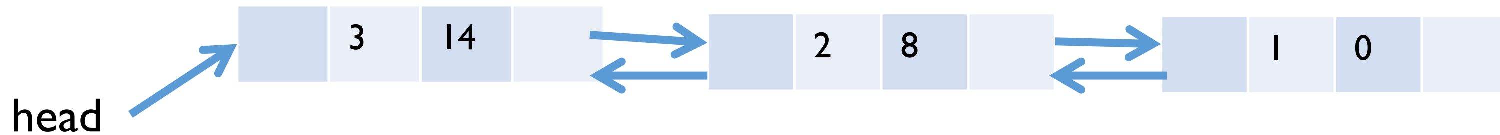
答案:Karumanchi 3.10 problem 16

Singly v.s. doubly linked list

Singly linked list:



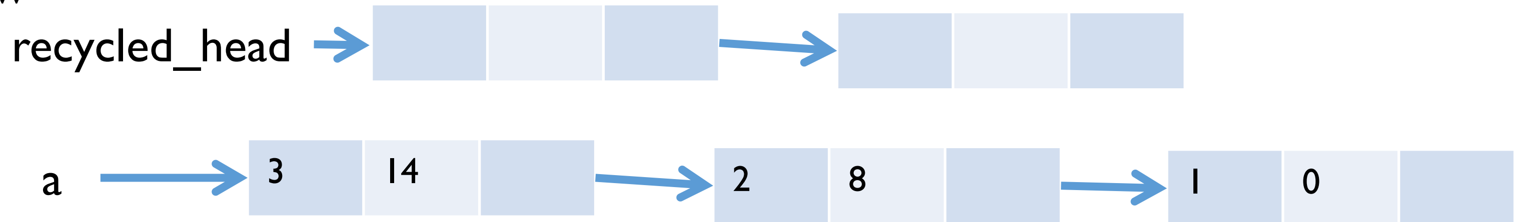
Doubly linked list:



- When do you need to use doubly linked list?
- Singly linked list: can only traverse forward, not backward
- (go all the way back to the head)
- When we need to frequently traverse backward: use doubly linked list
- Trade-offs:
  - Space for two pointers (instead of one) (see: http://goo.gl/qifrq2)
  - Additional time to process the pointers when inserting, deleting.

Recycling

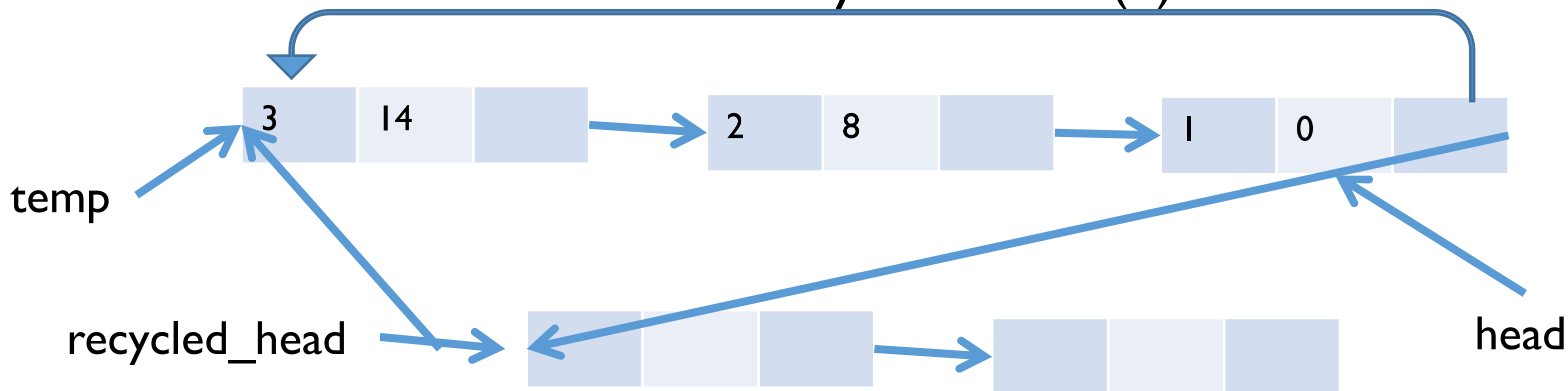
- Return the memory occupied by the nodes to the system when done. (Why?)
- O(n) time to return all the nodes
- Alternative: recycling! Collect all "deleted nodes", and use them when necessary.
- Goal: O(1) time for both delete and new (from recycled nodes)
- How



- Key: slow to find the tail (obviously, not O(1) time operation)
- Can we avoid using the tail pointer?

Sol: Circular List

- head pointer points at the tail.
- The tail node's next pointer points to the head node (instead of setting it as NULL).
- Easy to connect the entire list with another list!
  Place the entire list to the recycled list=O(1) !!



- We can also have doubly circular linked list!

Let's review!

- The types of linked lists that we introduced today:

- Singly linked list
  - Circular
  - Non-circular (chain)
- Doubly linked list
  - Circular
  - Non-circular (chain)

(If time permits) Practice Problems

- Given a (singly) linked list of unknown length, design an algorithm to find the n-th node from the tail of the linked list. Your algorithm is allowed to traverse the linked list only once.

- Reverse a given singly linked list using the original link nodes.