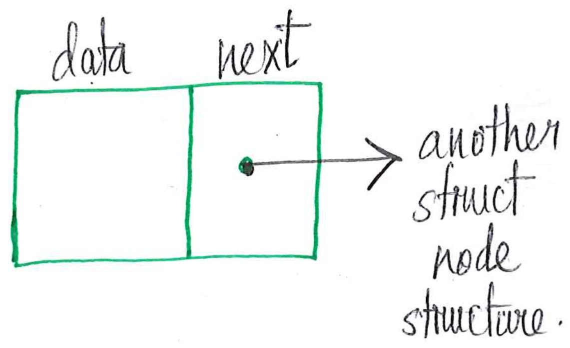
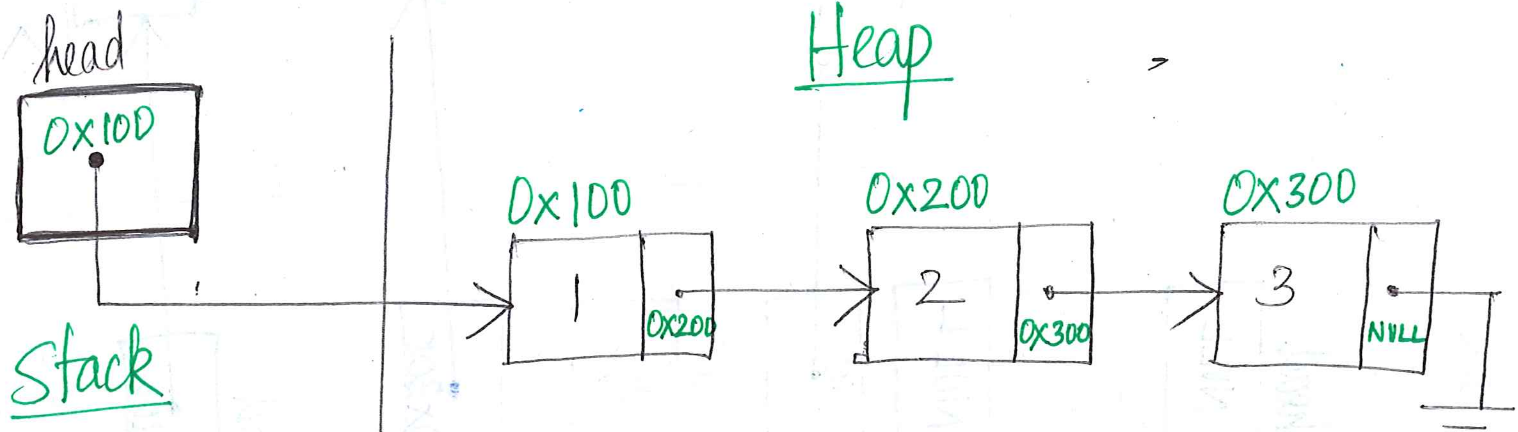
Feb 3, 2015 Lecture-6 (Linked Lists)

Self-referential structures ①

```
struct node {
    int data;
    struct node *next;
};
```





~~Stack~~

```
main()
{   int len;
    struct node *head;
head = build_three_nodes();
len = length(head);
head = insert_at_end(head, 4);
pop(&head);
}
```

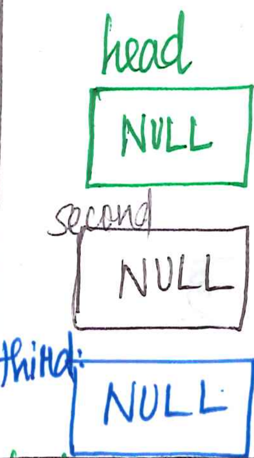
Code ~~Heap~~

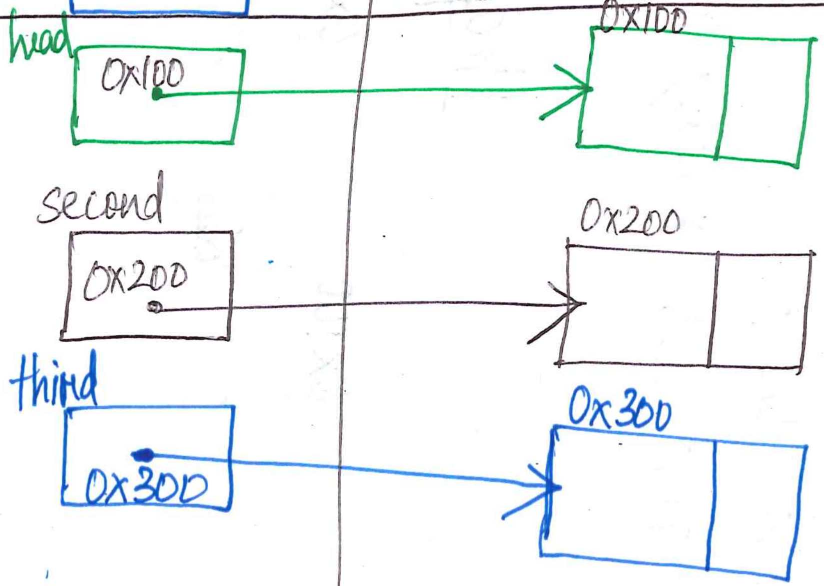
```
struct node * build_three_nodes()
{
    // code here.
}
```

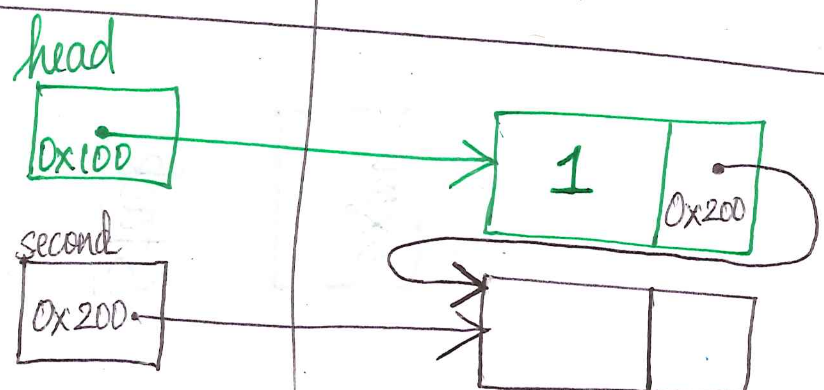
This function returns a pointer to a struct node.

Code	② Stack	Heap

```
struct node * build_three_nodes()
{
    struct node *head = NULL;
    struct node *second = NULL;
    struct node *third = NULL;

    head = malloc(sizeof(struct node));

    second = malloc(sizeof(struct node));

    third = malloc(sizeof(struct node));

    head -> data = 1;
    head -> next = second;
```

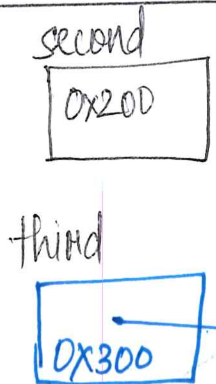


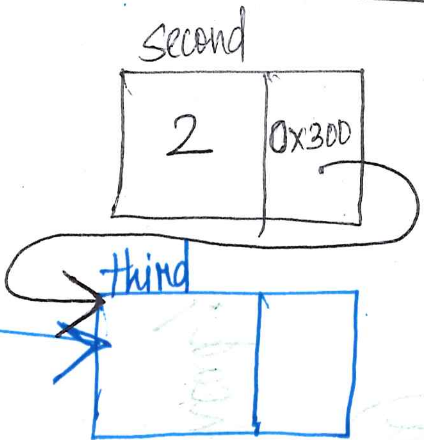




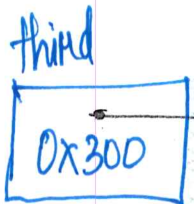
Code | ③ Stack | Heap

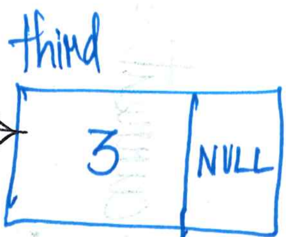
```
second -> data = 2;
second -> next = third
```



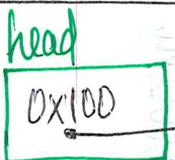


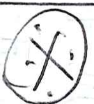

```
third -> data = 3;
third -> next = NULL;
```

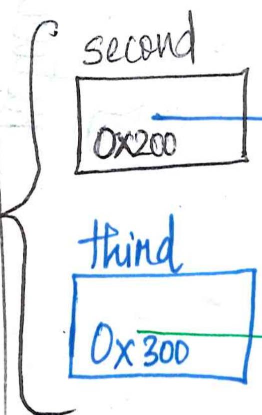


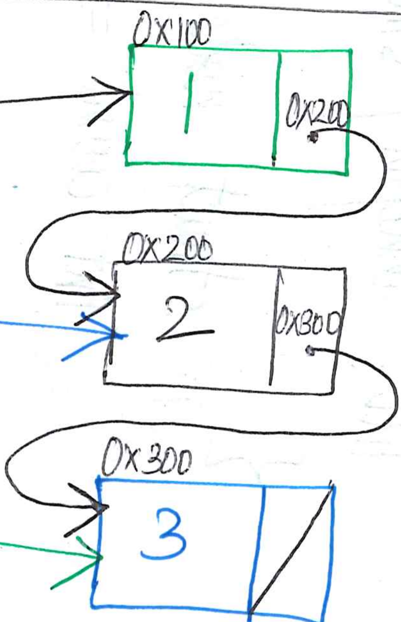



```
// state of the linked list
// return the address of the first node.
return head;
}
```







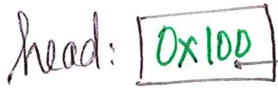


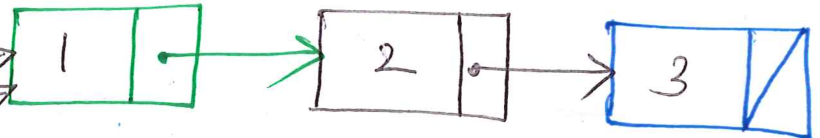
④

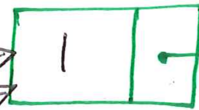
Length of a linked List.

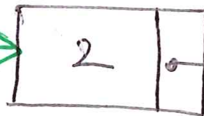
```
int length(struct node *head)
{
  ① struct node *current = head;
  ② int count = 0;
     while (current != NULL) {
         count ++;
         current = current->next;
     }
     return count;
}
```

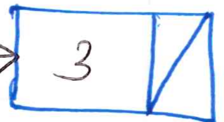
Stack | Heap











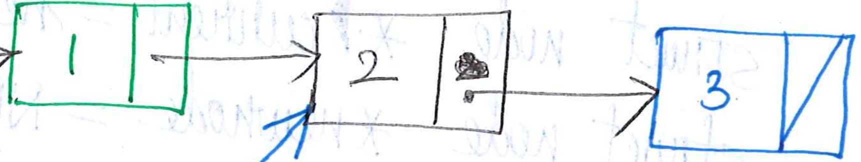
Stack	Heap

while loop after iteration 1

head: 0x100

current: 0x200

count: 1

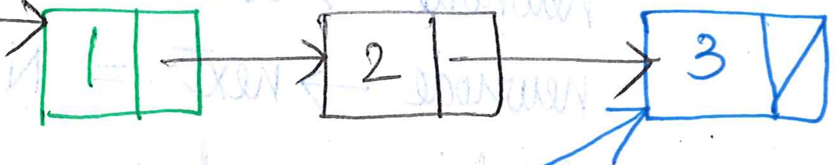


after iteration 2

head:

current: 0x300

count: 2

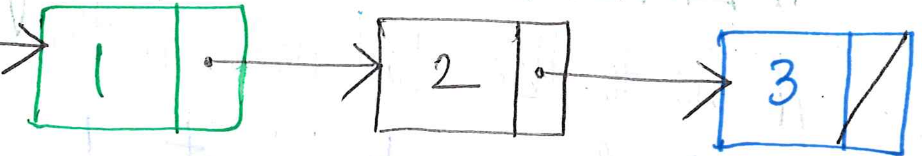


after iteration 3

head:

current: NULL

count: 3



End of while loop: count = 3 is returned to main.

```
struct node *insert_at_end(struct node *head, int data)
{
    struct node *current = head;
    struct node *newnode = NULL;

    //Linked List is empty.
    if(current == NULL) {
        newnode = malloc(sizeof(struct node));
        newnode -> data = data;
        newnode -> next = NULL;
        head = newnode;
        return head;
    }

    //Linked List is not empty.
    // Go to the last node in the list.
    while(current -> next != NULL) {
        current = current -> next;
    }
```

contd. on pg. 7.

```
// create a new node and add it to the end.
newnode = malloc (sizeof(struct node));
newnode -> data = data;
newnode -> next = NULL;
current -> next = newnode;

return head;
} // end of function insert_at_end.
```

NOTE: You can make the above code smaller by moving the part of creating the new node, and assigning its data and next to be before the if condition that checks if the list is empty.

Stacks and Queues

Stacks

struct node *push(struct node *head, int data);

push - adds element only at the beginning of the linked list.

int pop(struct node **phead);

pop - removes the first element in the linked list and returns its data part.

Why should we pass a pointer to a pointer to pop?

* Because we want to change the head of the linked list which is a pointer to struct node.
  ∴ We pass a pointer to the linked list's head.
* Also, we return the value of the element we just popped. So, we can't return the value of the updated head since we can return only value from functions.

```
int pop(struct node **phead)
{
    struct node *first = *phead;
    int data;
    assert(first != NULL); // include assert.h
    data = first->data;
    *phead = first->next;
    free(first);
    return data;
}
```

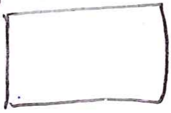
inside main() can be called as below.

```
int n;
//add some nodes to the linked list/ stack.
n = pop(&head);
```

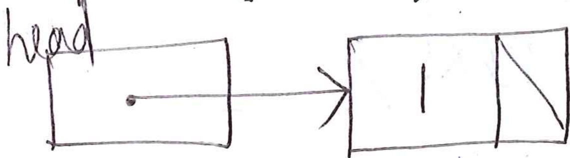
Queue

enqueue - add node only at the end of the linked list.

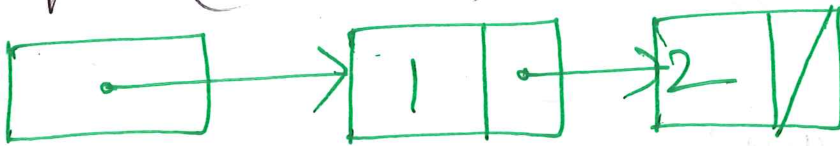
dequeue - remove node only at the beginning of the linked list



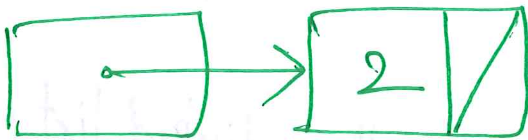
enqueue(&head, 1);



enqueue(&head, 2);



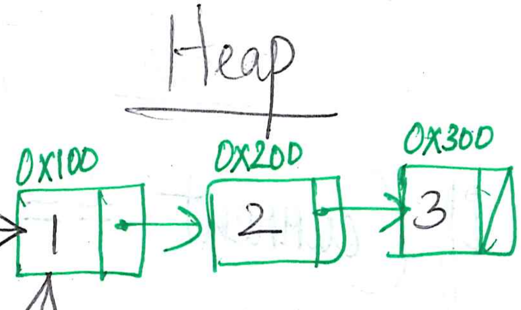
dequeue(&head);

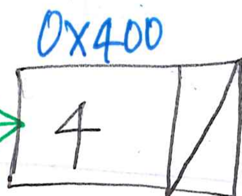


Code	Stack	Heap

Another version of insert_at_end.

```
void insert_at_end_v2(struct node **phead,
                      int data)
{

    struct node *current = *phead;

    struct node *newnode = NULL;

newnode = malloc(sizeof(struct node));
newnode->data = data;
newnode->next = NULL;
```





Code | ⑫ Stack | Heap

```
if (current == NULL) {
    *phead = newnode;
    return;
}
while (current->next != NULL)
{
    current = current->next;
}
current->next = newnode;
} //end of insert_at_end_v2.
```

current: 0x100

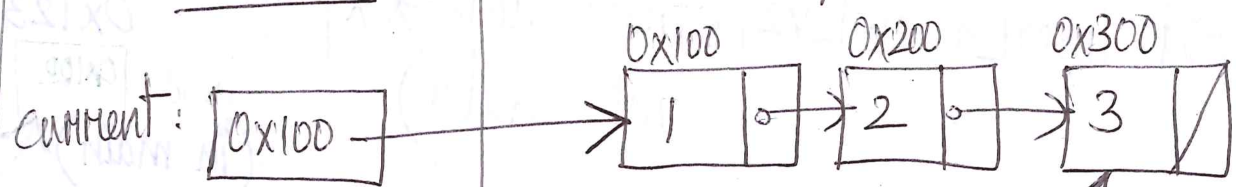
current is not null. if it was NULL we should change the head in main() using phead!

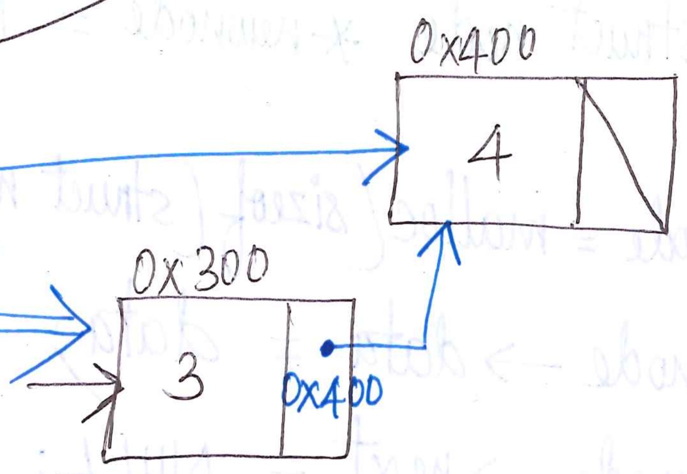
After 3 iterations

current: 0x300

newnode: 0x400

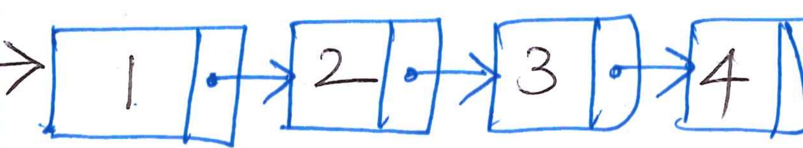
this line changes the next part of current

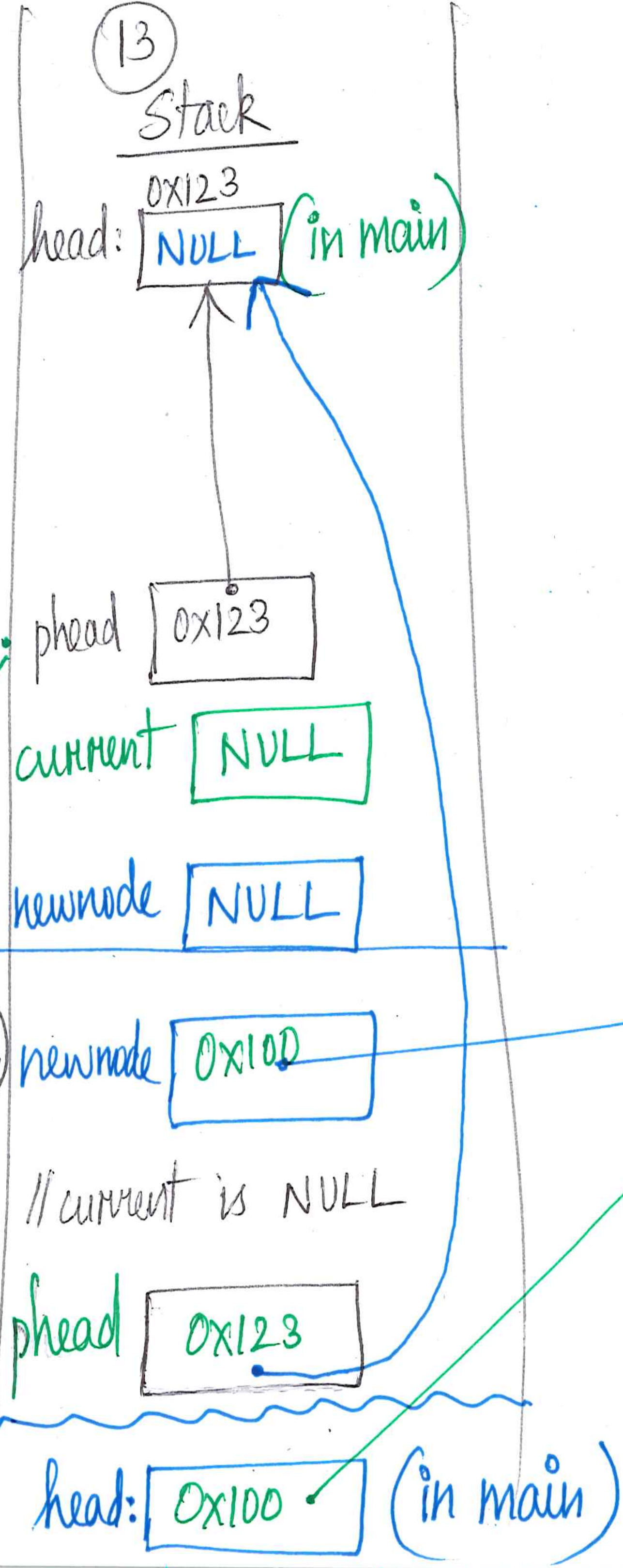




In main()

head: 0x100




13
Code
Stack
Heap
insert_at_end_v2(&head, 1);
// insert the first node.
void insert_at_end_v2(struct node **phead, int data)
{
struct node *current = *phead;
struct node *newnode = NULL;
//create the newnode and fill its data & next (see page 11)
if(current == NULL) {
*phead = newnode;
return;
}
modifies the head in main
0x123
head: NULL (in main)
phead 0x123
current NULL
newnode NULL
newnode 0x100
//current is NULL
phead 0x123
head: 0x100 (in main)
0x100
1