Self-referential structures

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

Stack

Heap

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

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

This function returns a pointer to a struct node.
Heap

Stack

Code

2

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;
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;

not needed here after
Length of a linked list

```c
int length (struct node *head)
{
    struct node *current = head;
    int count = 0;
    while (current != NULL) {
        count ++;
        current = current -> next;
    }
    return count;
}
```
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: \[ \text{count} = 3 \] 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;
    }

    newnode->data = data;
    newnode->next = NULL;
    current->next = newnode;
}

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;

3. // 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 **head);
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
    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.

head

enqueue (&head, 1);
head

enqueue (&head, 2);

dequeue (&head);

[Diagram of queue operations with nodes and arrows]
Code

void insert_at_end_v2 (struct node **phead, int data)
{
    struct node *newnode = malloc (sizeof (*struct node));
    newnode -> data = data;
    newnode -> next = NULL;
    newnode = malloc (sizeof (struct node));
    newnode -> data = NULL;
    newnode -> next = NULL;

    phead -> next = newnode;
    newnode -> head = newnode;
    newnode -> next = head;
    head = newnode;
}

Stack

0x123

(in main)

phead: 0x123
head: 0x100

Heap

0x400

Pointers:

1. phead: 0x123
2. data: 0x1100
3. current: 0x123
4. newnode: 0x400
In `main()`:  
head: `0x100`  

1. `1`  
2. `2`  
3. `3`  
4. `4`  

Stack:  
- `0x100`  
- `0x200`  
- `0x300`  
- `0x400`  
- `0x300`  
- `0x400`  

Heap:  
- `0x100`  
- `0x200`  
- `0x300`  

Code:  
```c
if (current == NULL) {
    *phead = newnode;
}
```

Current is not null. If it was null, we should change the head in `main()`. After 3 iterations, the next part of `current` is the next node of `current`. This line changes the next part of `current`. if `current` is not null, the code changes the head.
```c
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;
    }
    current = current->next;
    //current is NULL
    newnode = (struct node) {
        data: 1
        next: NULL
    };
    phead -> next = newnode;
    //updates the head in main
    head: 0x100
    (in main)
    stack
    0x123
    head: NULL
    (in main)
    phead 0x123
    current NULL
    newnode 0x100
    phead 0x123
    head: 0x100
    (in main)
}