Programming Exercises

Counting the nodes in a List

- Recursive version:

```
int count (struct node * head)
{
    if (head==NULL)
        return 0;
    else
        return(1 + count(head->next));
}
```

- Iterative version:

```
int count(struct node *head)
{
    struct node * p;
    int c = 0;

    p = head;
    while (p != NULL){
        c = c + 1;
        p = p->next;
    }
    return c;
}
```

Look up an item in the list pointed by head

```
/* Given the item and the pointer to the head of
the list, the function returns NULL if the item is
not found; or returns a pointer to the node which
matches the item
*/
struct node * lookup(int item, struct node *head)
{
    if (head == NULL)
        return NULL;
    else if (item == head->data)
        return head;
    else
        return(lookup(item, head->next));
}
```

Creating a List

- Recursive version:

```
// Copies the contents of an array into a
// dynamically growing list.

struct node *array_to_list(int a[], int j, int n)
{
      struct node *head;

      if (j >= n)  //base case
          return NULL;
      else {
          head = (struct node *)
                   malloc(sizeof(struct node));
          head->data = a[j];
          head->next = array_to_list(a, j+1, n);
          return head;
      }
}
```

Calling the function:

```
int numbers[] = {1, 2, 3, 4};
struct node *my_list;

my_list = array_to_list(numbers, 0, 4);
```

- Iterative version:

```
struct node *array_to_list(int a[], int n)
{
    struct node *head, *current;
    int j;

    if (n == 0)     //the array is empty
        return NULL;

    else {          //create the head node for
                    //the first element
        head = (struct node *)
                malloc(sizeof(struct node));
        current = head;
        current->data = a[0];

        //create nodes for the other elements
        j = 1;
        while (j < n){
            current->next = (struct node *)
                     malloc(sizeof(struct node));
            current = current->next;
            current->data = a[j];
            j = j +1;
        }
        current->next = NULL; //finish the list

        return head;
    }
}
```

Calling the function:

```
my_list = array_to_list(numbers, 4);
```

Freeing Memory

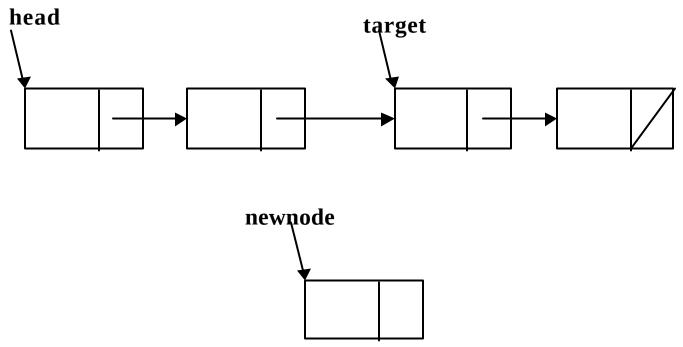
The following function frees the memory of an entire list once we don't need to use that list.

```
void delete_list(struct node *p) {

    if (p !=NULL) {
        delete_list(p->next);
        free(p);
    }
}
```

Insertion before a node

Problem: Insert the node pointed by **newnode** *before* the node pointed by **target**. This is not so easy. We need to find the previous node so that we can update its pointer.



Inserting a new node before a given node:

Attention:
this is a pointer to pointer

```
void insert_before(struct node **head,
                   struct node *target,
                   struct node *newnode)
{
    struct node *p, *prep;

    // search target pointer and keep track of the
    // previous pointer
```

we have to use the dereferencing operator

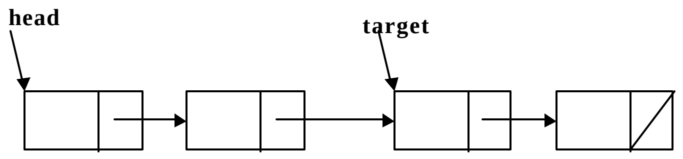
```
    p = *head;

    while (p != NULL && p != target){
        prep = p;
        p = p->next;
    }

    //insert it
    newnode->next = p;
    if (*head == p)
        *head = newnode;
    else
        prep ->next = newnode;
}
```

Deletion of a node from the list

Problem: Delete the node pointed by **target**.
Since we don't have a pointer to the previous node, we have to search the list to find it first:



```
//delete the target node. If the target
//node is not in the list do nothing.

void delete (struct node **head,
             struct node *target)
{
    struct node *p,*prep;

    //find previous node
    p = *head;
    while (p != NULL && p != target){
        prep = p;
        p = p->next;
    }

    //delete the target
    if (p != NULL){    //target is in the list
        if (p == *head)  //deleting the first node
            *head = p->next;
        else
            prep ->next = p->next;
        free(p);
    }
}
```

Dynamic Arrays

We can also dynamically allocate space for an array. We can do this through pointers.

```
int main() {

    int *x, size;

    printf("Enter the size of your array.\n");
    scanf("%d", &size);

    if (size > 0)
        x = (int *)malloc(size * sizeof(int));

    for (i = 0; i < size; i++) {
        printf("Enter the next array value.\n");
        scanf("%d", &x[i]);
    }
}
```

Class Exercise

1) Write a function that will split a linked list into two lists L1 and L2 such that, successive nodes go to different lists. (The first, third and all the other odd numbered nodes go to the first list, the second, fourth and all the other even numbered nodes go to the second list).

2) Write a function that will reverse the pointers of a linked list while traversing it only once.