COMP1511: Recursion, Linked List with Recursion

Session 2, 2018

Recursion

- **Recursion** is a programming pattern where a function calls ***itself***

- For example, we define *factorial* as below,

  n! = 1*2*3* … *(n-1)*n

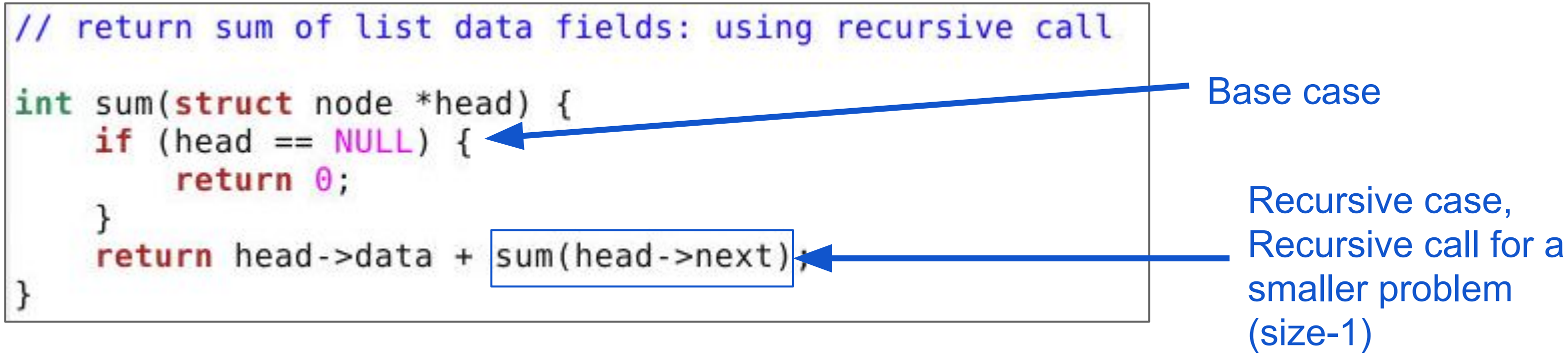
- We can ***recursively*** define *factorial* function as below,

  f(n) = 1 , if (n=0)

  f(n) = n * f(n-1) , for others

Pattern for a Recursive function

- Base case(s)
  - Situations when we **do not** call the same function (no recursive call), because the problem can be solved easily without a recursion.
  - All recursive calls eventually lead to one of the base cases.
- Recursive Case
  - We **call** the **same function** for a problem with ***smaller size***.
  - Decrease in a problem size eventually leads to one of the base cases.

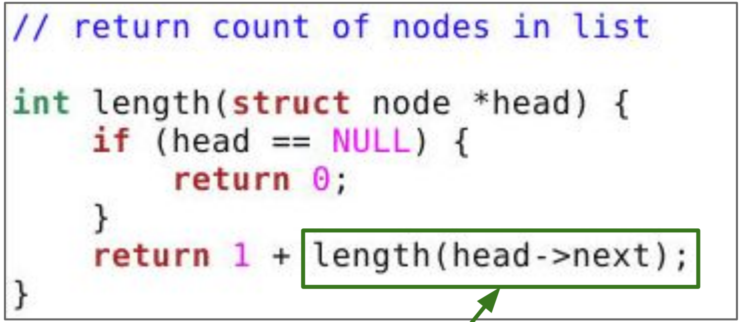


```
// return sum of list data fields: using recursive call

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

Base case

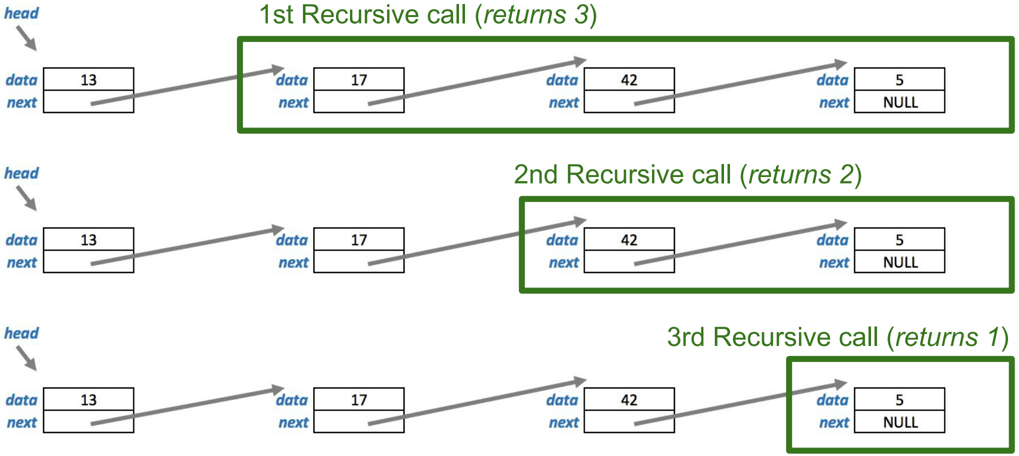
Recursive case,
Recursive call for a
smaller problem
(size-1)

Linked List with Recursion



```
// return count of nodes in list

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

Recursive call



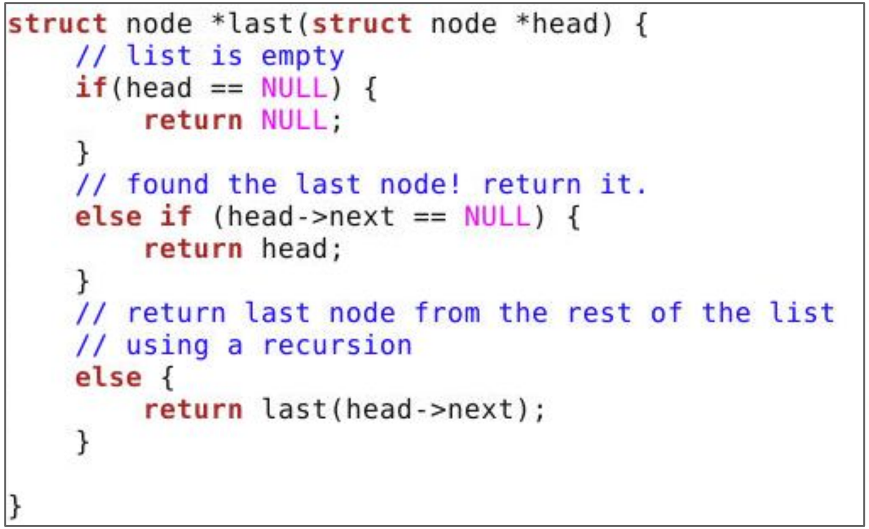
1st Recursive call (*returns 3*)

2nd Recursive call (*returns 2*)

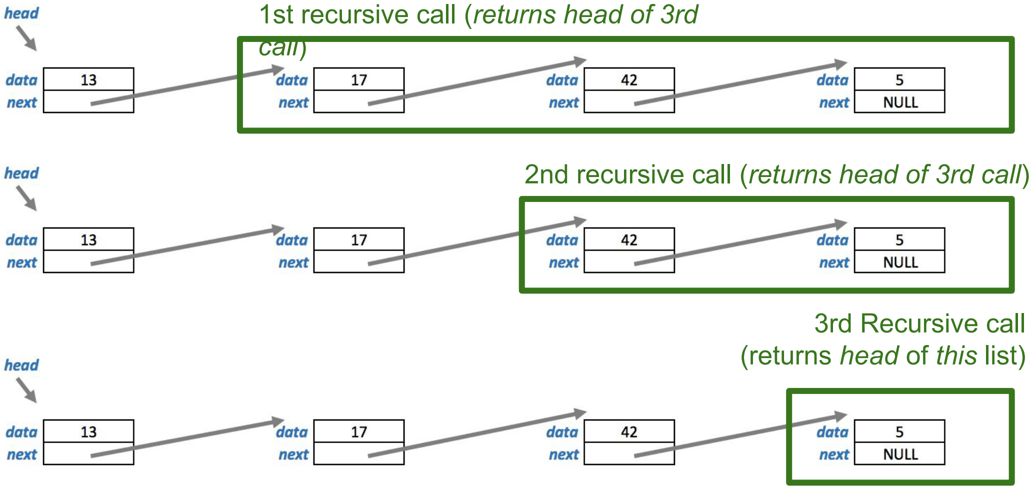
3rd Recursive call (*returns 1*)

Base case
Head == NULL
returns 0

Last Node using Recursion



```
struct node *last(struct node *head) {
    // list is empty
    if(head == NULL) {
        return NULL;
    }
    // found the last node! return it.
    else if (head->next == NULL) {
        return head;
    }
    // return last node from the rest of the list
    // using a recursion
    else {
        return last(head->next);
    }

}
```



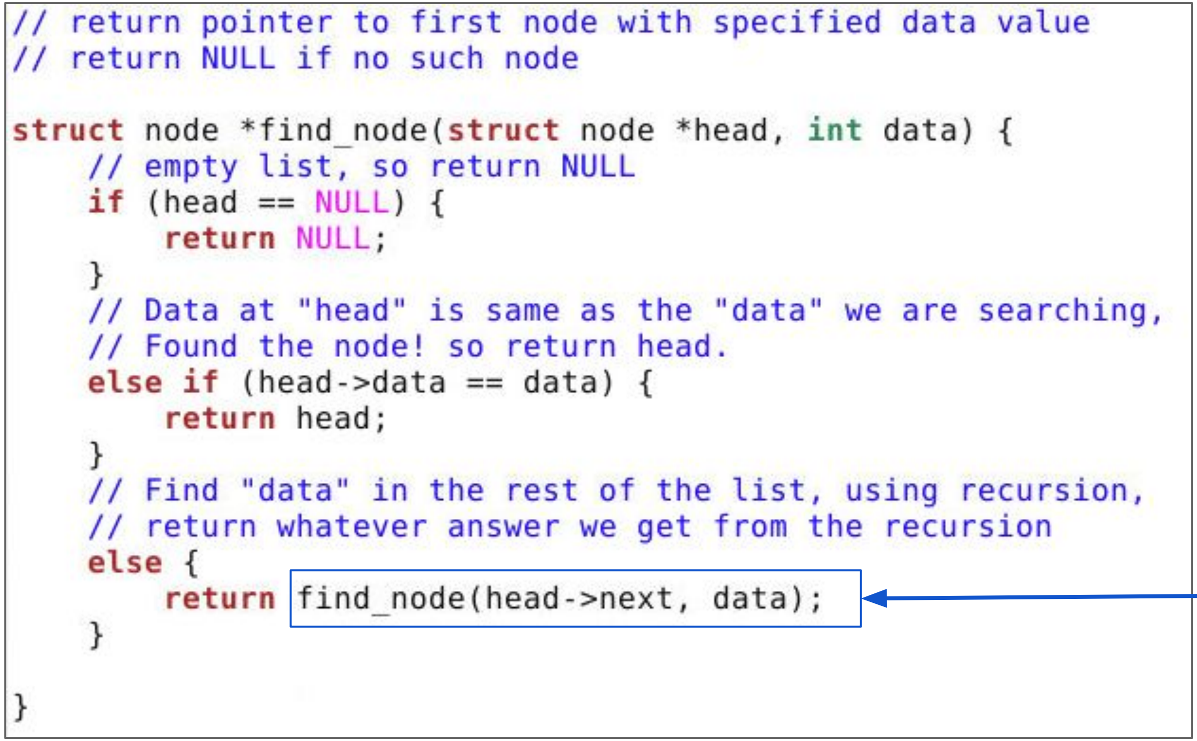
1st recursive call (*returns head of 3rd call*)

2nd recursive call (*returns head of 3rd call*)

3rd Recursive call (returns *head* of *this* list)

Base case
head->next == NULL
returns head of 3rd call

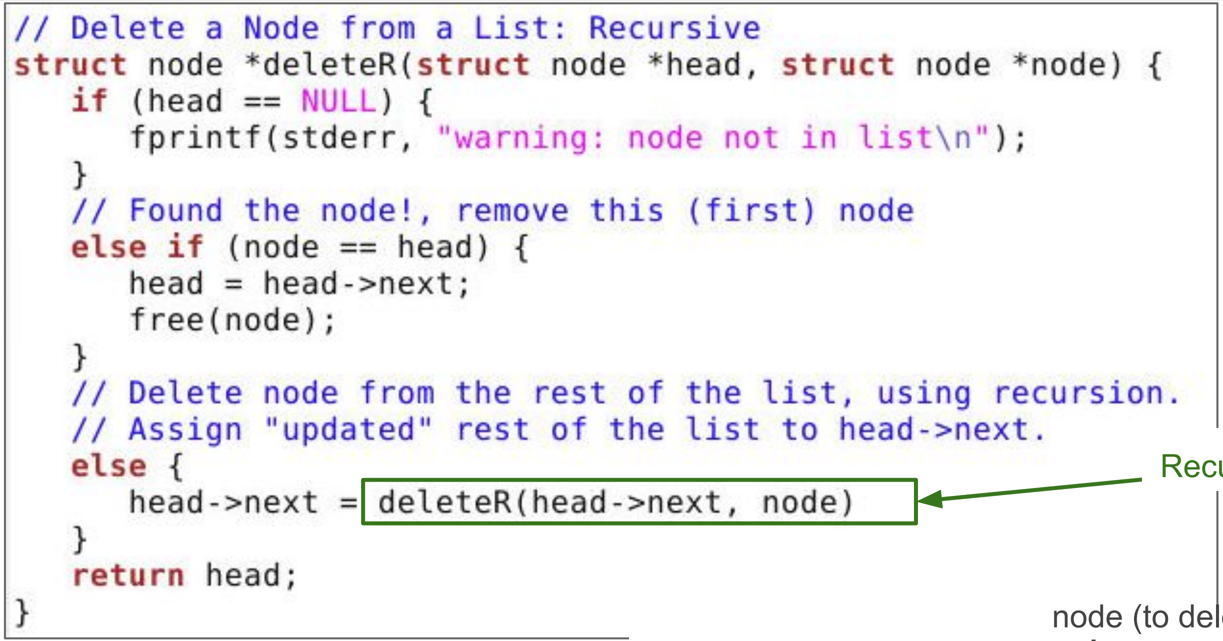
Find Node using Recursion



```
// return pointer to first node with specified data value
// return NULL if no such node

struct node *find_node(struct node *head, int data) {
    // empty list, so return NULL
    if (head == NULL) {
        return NULL;
    }
    // Data at "head" is same as the "data" we are searching,
    // Found the node! so return head.
    else if (head->data == data) {
        return head;
    }
    // Find "data" in the rest of the list, using recursion,
    // return whatever answer we get from the recursion
    else {
        return find_node(head->next, data);
    }

}
```

Recursive call

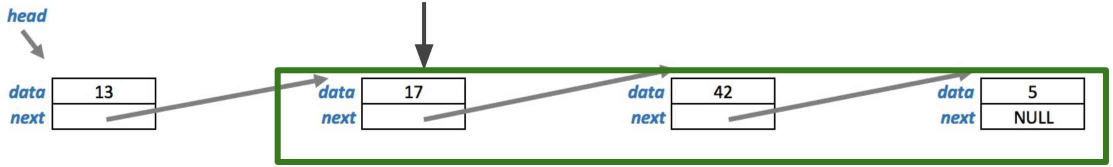
Delete From List using Recursion



```
// Delete a Node from a List: Recursive
struct node *deleteR(struct node *head, struct node *node) {
    if (head == NULL) {
        fprintf(stderr, "warning: node not in list\n");
    }
    // Found the node!, remove this (first) node
    else if (node == head) {
        head = head->next;
        free(node);
    }
    // Delete node from the rest of the list, using recursion.
    // Assign "updated" rest of the list to head->next.
    else {
        head->next = deleteR(head->next, node)
    }
    return head;
}
```

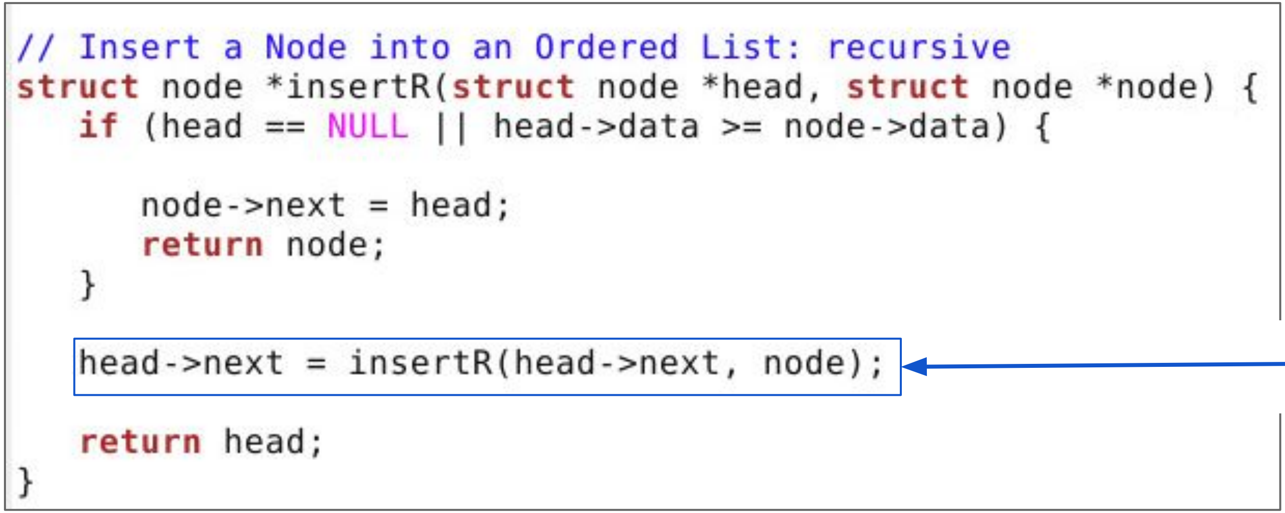
Recursive call

node (to delete)



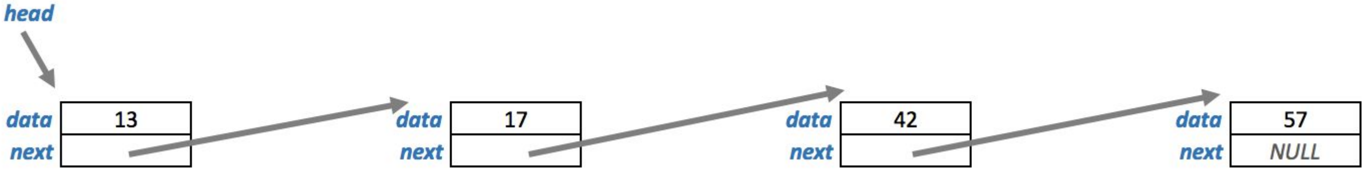
1st recursive call (node to delete is same as “head” of this call, *returns updated list, pointing to node with 42)*

Linked List with Recursion

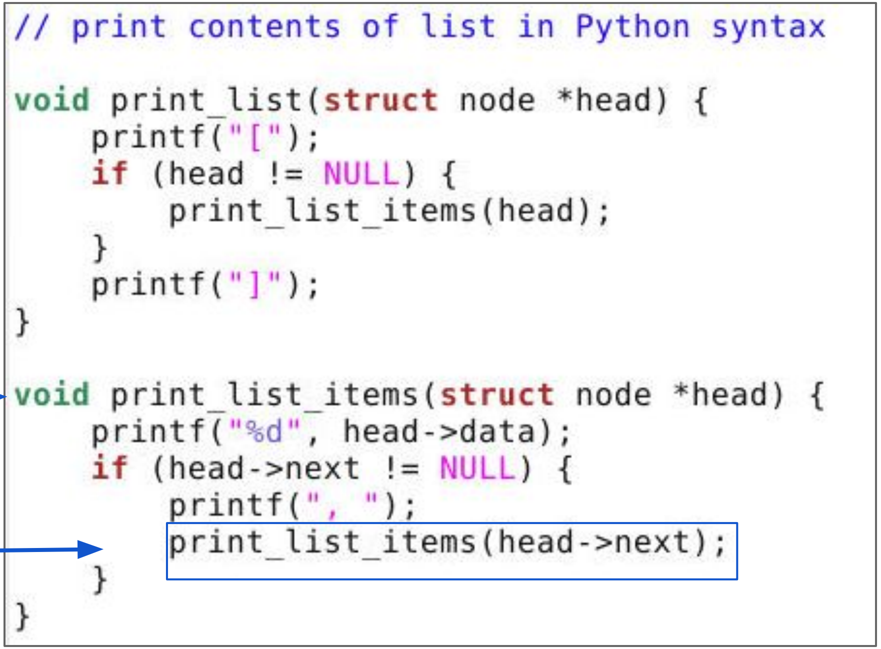


```
// Insert a Node into an Ordered List: recursive
struct node *insertR(struct node *head, struct node *node) {
   if (head == NULL || head->data >= node->data) {

      node->next = head;
      return node;
   }

   head->next = insertR(head->next, node);

   return head;
}
```

Recursive call



Print Python List using Recursion



```
// print contents of list in Python syntax

void print_list(struct node *head) {
    printf("[");
    if (head != NULL) {
        print_list_items(head);
    }
    printf("]");
}

void print_list_items(struct node *head) {
    printf("%d", head->data);
    if (head->next != NULL) {
        printf(", ");
        print_list_items(head->next);
    }
}
```

Recursive function

Recursive call