DOUBLY LINKED LISTS

Doubly Linked List

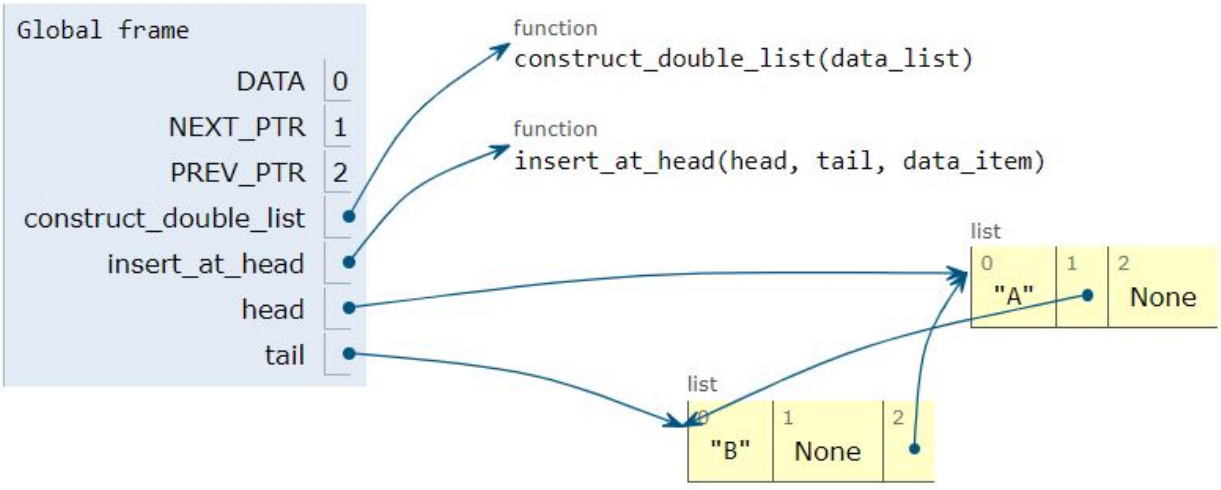
- linear collection of nodes
- each node contains:
  1. DATA field
  2. NEXT_PTR to the next node
  3. PREV_PTR to previous node
- *head* points to start
- *tail* points to end
- no random access
- efficient insert/delete

Constructing Doubly Linked List

```
DATA     = 0
NEXT_PTR = 1
PREV_PTR = 2

def construct_double_list(data_list):
    head = None
    tail = None
    for e in data_list:
        new_node = [e, None, None]
        if head is None:
            head = new_node
            tail = new_node
        else:
            tail[NEXT_PTR] = new_node;
            new_node[PREV_PTR] = tail
            tail = new_node
    return head, tail

data_items = ['A', 'B']
head, tail = construct_double_list(data_items)
```

Constructing Doubly Linked List

```
head,tail = construct_double_list(['A', 'B'])
```



List Traversal

```
def forward_traverse(head):
    print('\n forward traversal: ', end = ' ')
    next_node = head
        while next_node is not None:
            print(next_node[DATA], end = ' ')
                next_node = next_node[NEXT_PTR]
        return

def reverse_traverse(tail):
    print('\n reverse traversal: ', end = ' ')
    next_node = tail
        while next_node is not None:
            print(next_node[DATA], end = ' ')
                next_node = next_node[PREV_PTR]
        return
```

List Traversal

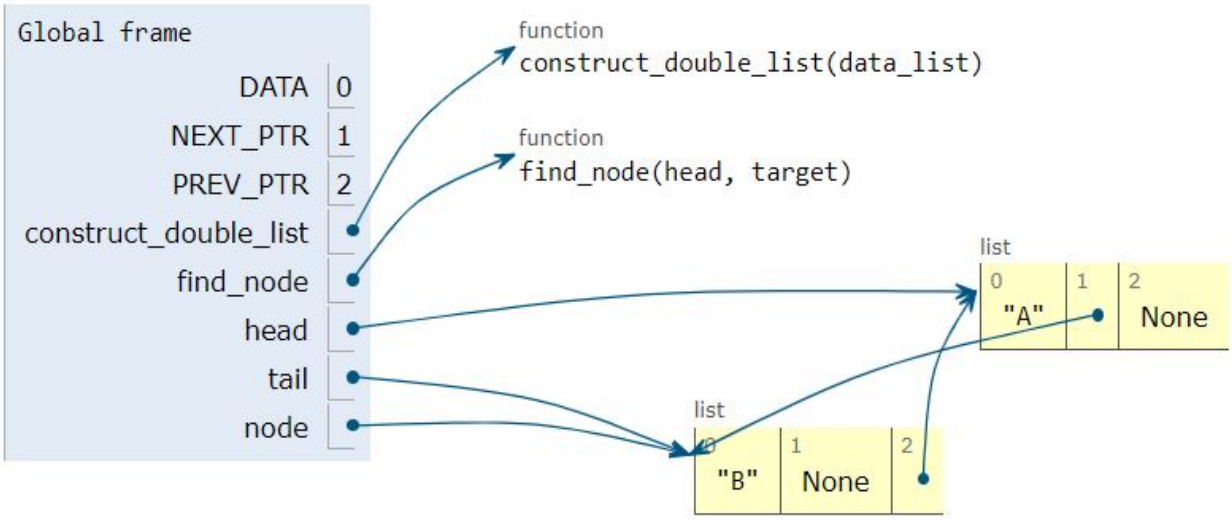
```
head,tail = construct_double_list(['A','B','C'])
forward_traverse(head)
reverse_traverse(tail)
```

Print output (drag lower right corner to resize)

```
forward traversal:  A B C
reverse traversal:  C B A
```

Finding an Element

```
def find_node(head, target):
    node = head
    while node is not None:
        if node[DATA] == target:
            return node
        else:
            node = node[NEXT_PTR]
    return node
head, tail = construct_double_list(['A', 'B'])
node = find_node(head, 'B')
```

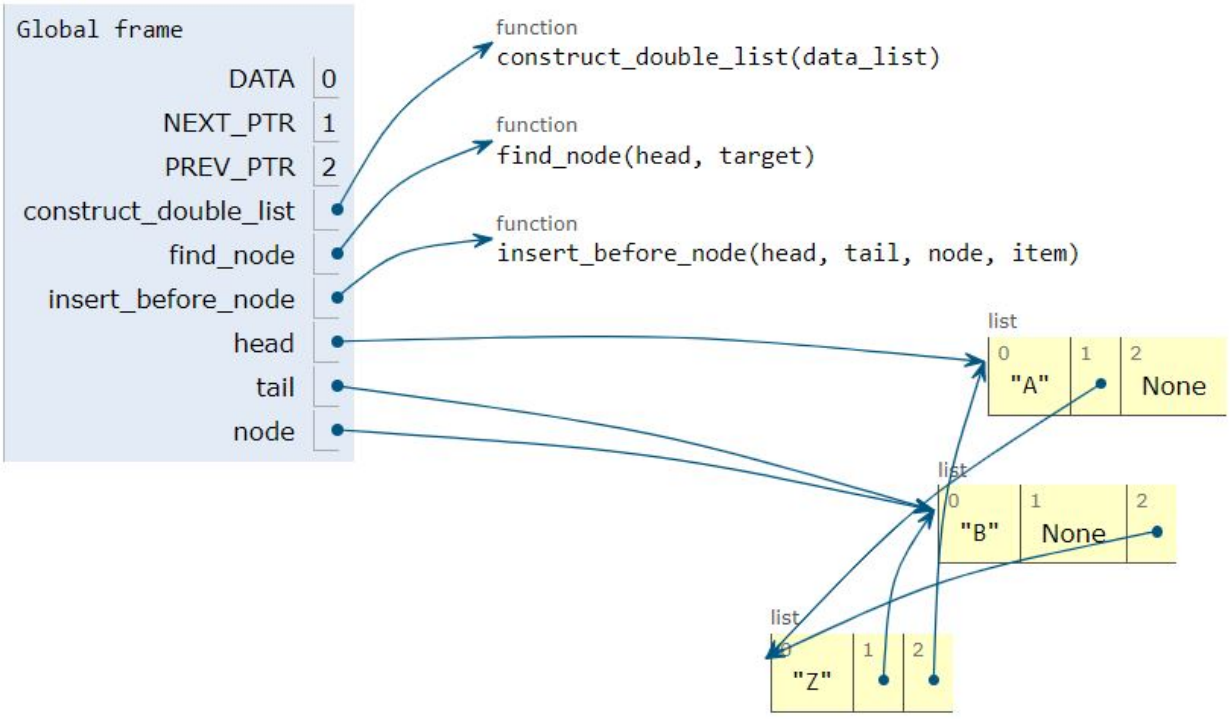


Inserting an Element

```
def insert_before_node(head, tail, node, item):
    new_node = [item, None, None]
    if node is None:
        return new_node, new_node
    else:
        prev_node = node[PREV_PTR]
        if prev_node is None:  # insert at front
            new_node[NEXT_PTR] = node
            node[PREV_PTR] = new_node
            head = new_node
            return head, tail
        else:
            prev_node[NEXT_PTR] = new_node
            new_node[PREV_PTR] = prev_node
            new_node[NEXT_PTR] = node
            node[PREV_PTR] = new_node
            return head, tail
```

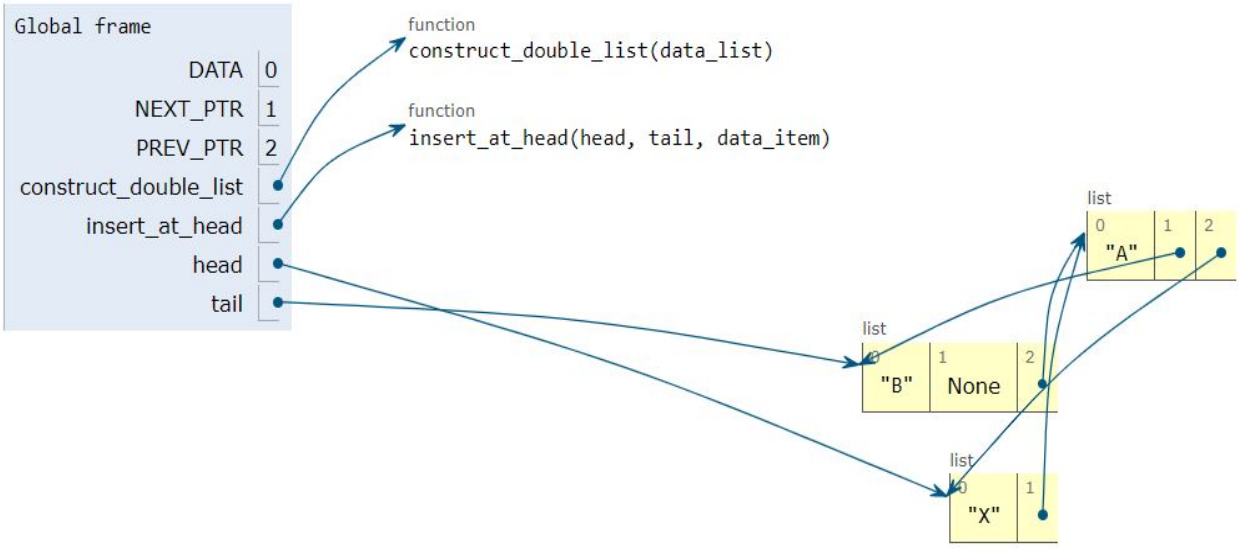
Inserting an Element

```
head, tail = construct_double_list(['A', 'B'])
node = find_node(head, 'B')
head, tail = insert_before_node(head,tail,
                                node,'Z')
```



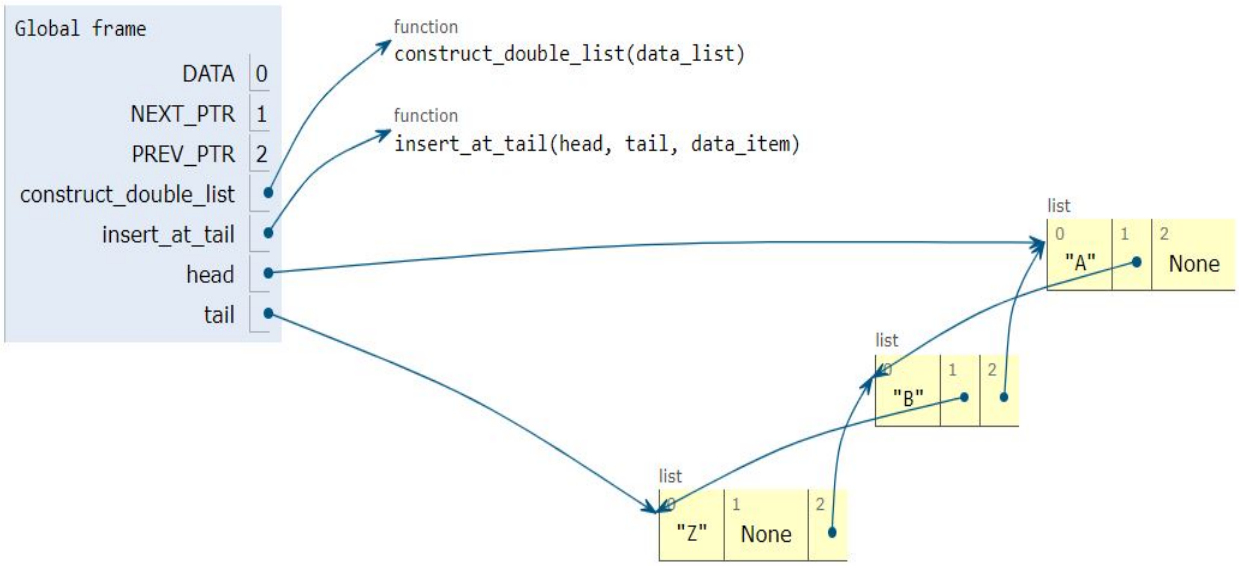
Inserting at *head*

```
def insert_at_head(head, tail, data_item):
    new_node = [data_item, None]
    if head is None:
        head = new_node; teil = new_node
    else:
        head[PREV_PTR] = new_node
        new_node[NEXT_PTR] = head
        head = new_node
    return head, tail
head, tail = insert_at_head(head,tail,'X')
```



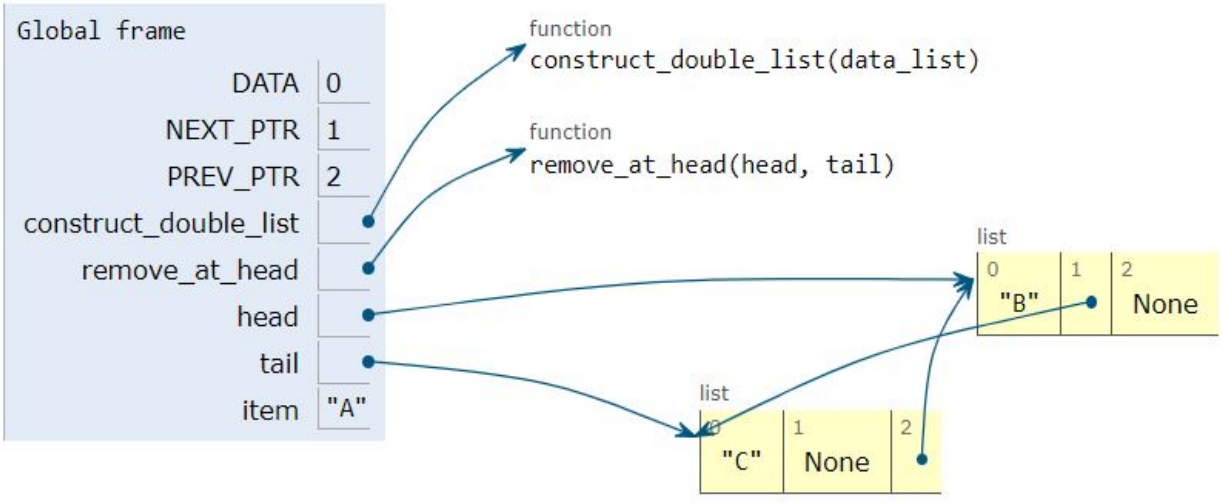
Inserting at *tail*

```
def insert_at_tail(head, tail, data_item):
    new_node = [data_item, None, None]
    if head is None:
        head = new_node; teil = new_node
    else:
        new_node[PREV_PTR] = tail
        tail[NEXT_PTR] = new_node
        tail = new_node
    return head, tail
head, tail = construct_double_list(['A', 'B'])
head, tail = insert_at_tail(head, tail,'Z')
```



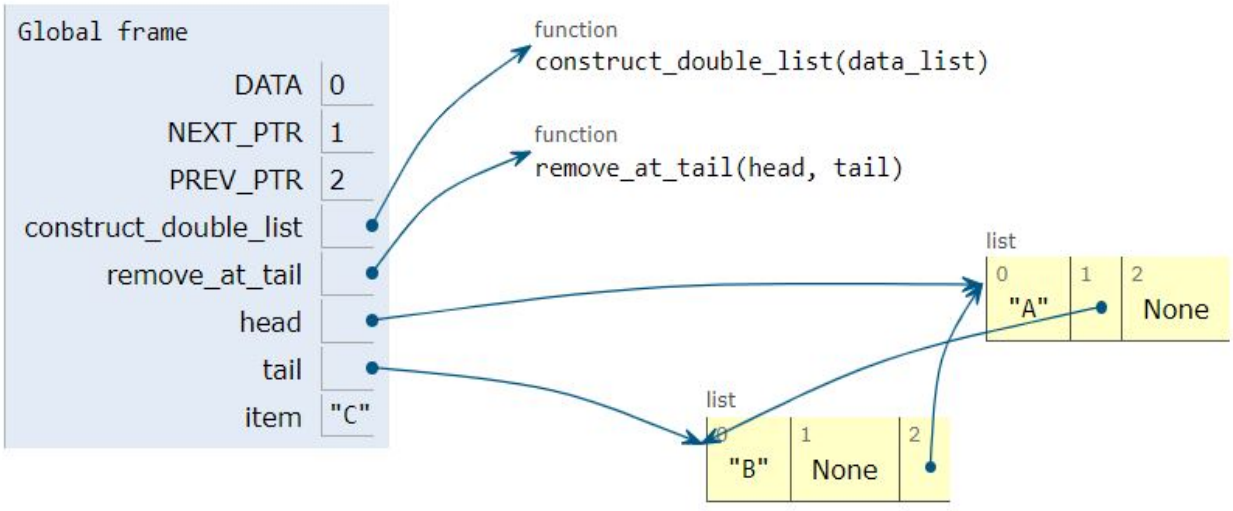
Remove at *head*

```
def remove_at_head(head, tail):
    if head is None:
        return None, None, None
    else:
        data = head[DATA]
        new_head = head[NEXT_PTR]
        new_head[PREV_PTR] = None
        return data, new_head, tail

head,tail = construct_double_list(['A','B','C'])
item, head, tail = remove_at_head(head, tail)
```



Remove at *tail*

```
def remove_at_tail(head, tail):
    if tail is None:
        return None, None, None
    else:
        data = tail[DATA]
        new_tail = tail[PREV_PTR]
        new_tail[NEXT_PTR] = None
        return data, head, new_tail

head,tail = construct_double_list(['A','B','C'])
item, head, tail = remove_at_tail(head, tail)
```



Removing an element

```
def remove_node(head, tail, node):
    if node is None:
        return head, tail
    else:
        prev_node = node[PREV_PTR]
        next_node = node[NEXT_PTR]

        if prev_node is None:
            new_head = head[NEXT_PTR]
            new_head[PREV_PTR] = None
            return new_head, tail

        elif next_node is None:
            new_tail = tail[PREV_PTR]
            new_tail[NEXT_PTR] = None
            return head, new_tail

        else:
            prev_node[NEXT_PTR] = next_node
            next_node[PREV_PTR] = prev_node
            return head, tail
```

Deleting an element

```
head, tail = construct_double_list(['A','B','C'])
node = find_node(head, 'B')
head, tail = remove_node(head, tail, node)
```