NWEN 241
Systems Programming

Alvin C. Valera

alvin.valera@ecs.vuw.ac.nz

Content

- More on Linked Lists (not to be assessed)

Recap: Singly-Linked List Example

- Node type definition

```
typedef struct node
{ char data;
  struct node *next;
} Node;
```

- Node variables declaration and initialization

```
Node node4 = {'t', NULL};
Node node3 = {'s', &node4};
Node node2 = {'i', &node3};
Node node1 = {'l', &node2};
Node *head = &node1;
```

Problems with Previous Example

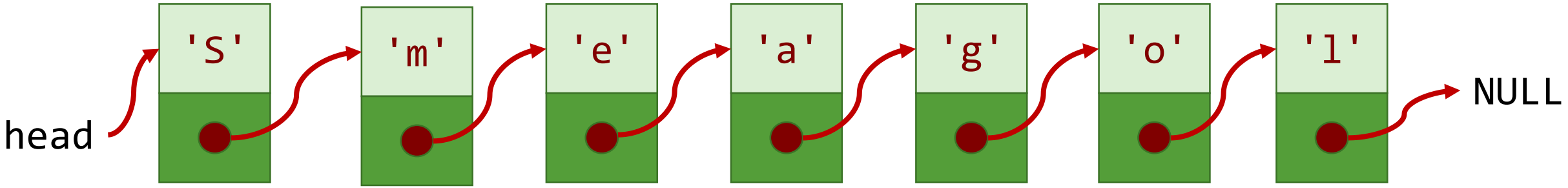
- Need to know list elements during coding
- What if the list elements are not known prior to program execution?
- The right way: **use dynamic memory allocation**

Motivating Example

- Ask user to input arbitrary string
- Convert string to a singly-linked list, with each node containing a character
- User Input:

```
Smeagol
```

- Linked List:



Preliminaries

- Node type definition

```
typedef struct node
{ char data;
  struct node *next;
} Node;
```

- Node variables declaration and initialization

```
Node *head = NULL;
```

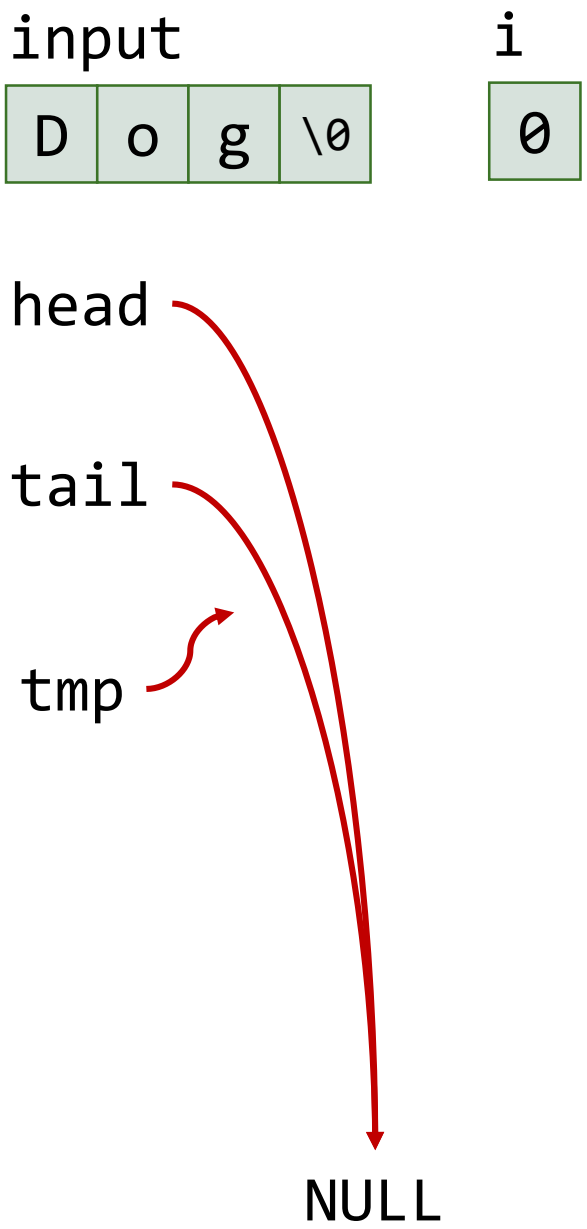
The Rest of The Code

```
char input[100];
int i = 0;
Node *tail = NULL, *tmp;

scanf("%s", input);

while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) { head = tmp; tail = head; }
    else { tail->next = tmp; tail = tmp; }
    i++;
}
```

Walkthrough

- Suppose user input is Dog

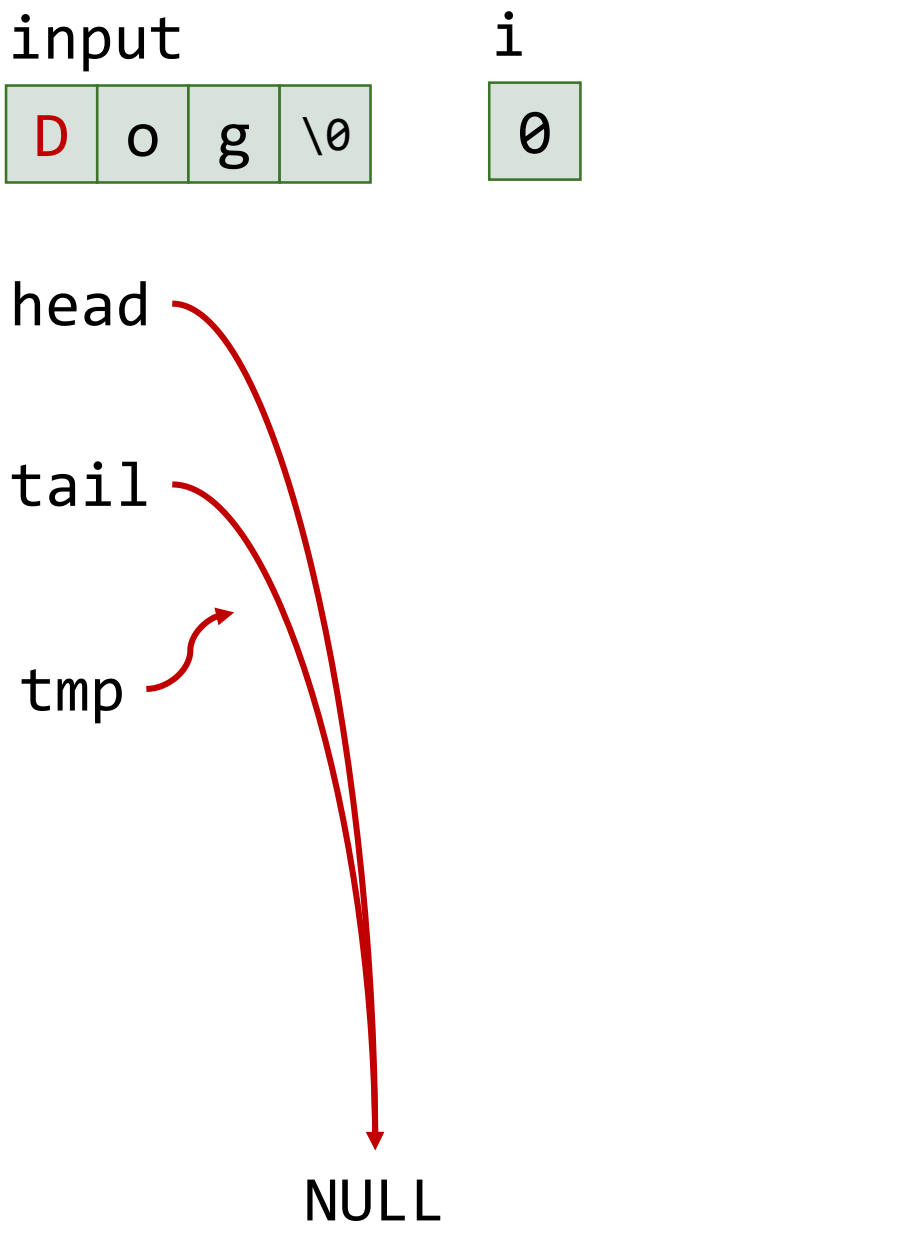
```
char input[100];
int i = 0;
Node *tmp, *tail = NULL;

scanf("%s", input);

…
```



Walkthrough

- Suppose user input is Dog

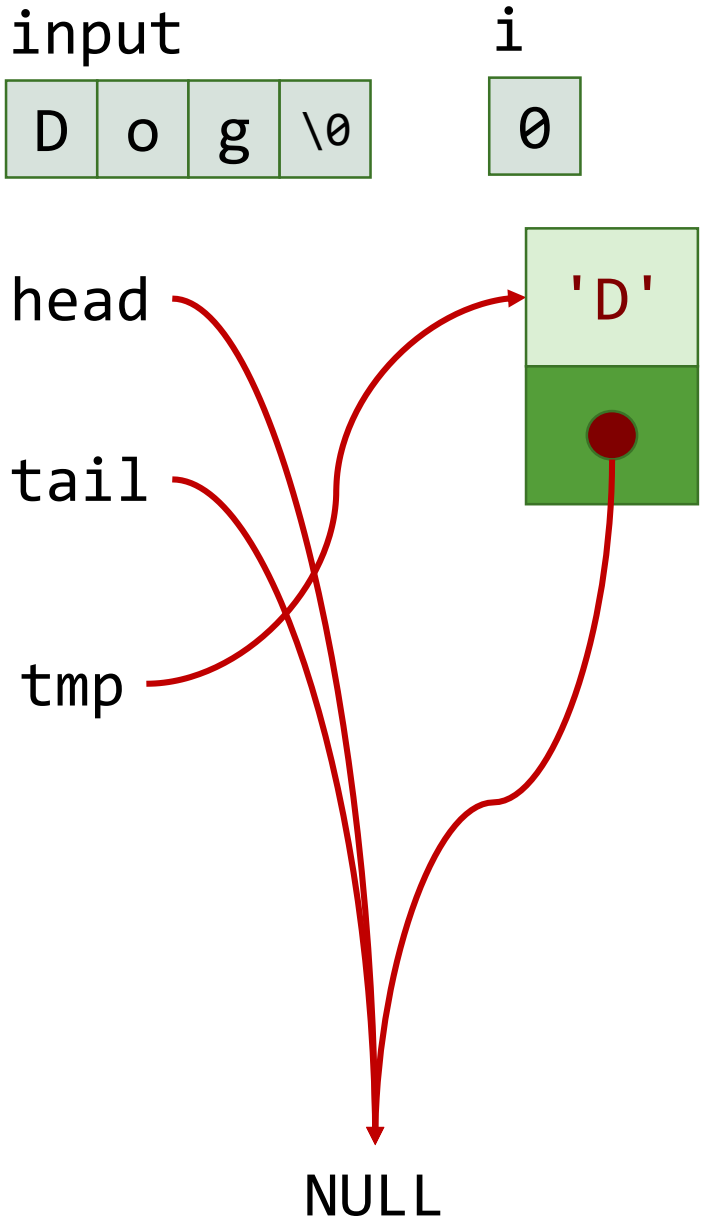
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

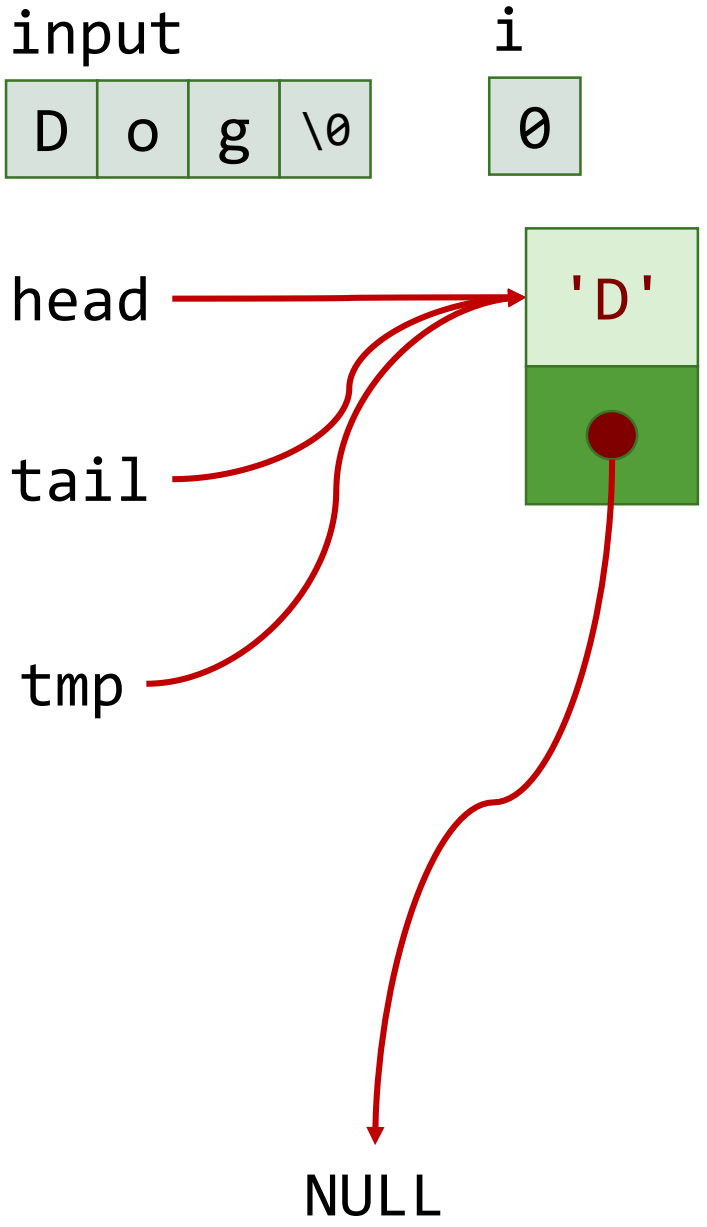
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

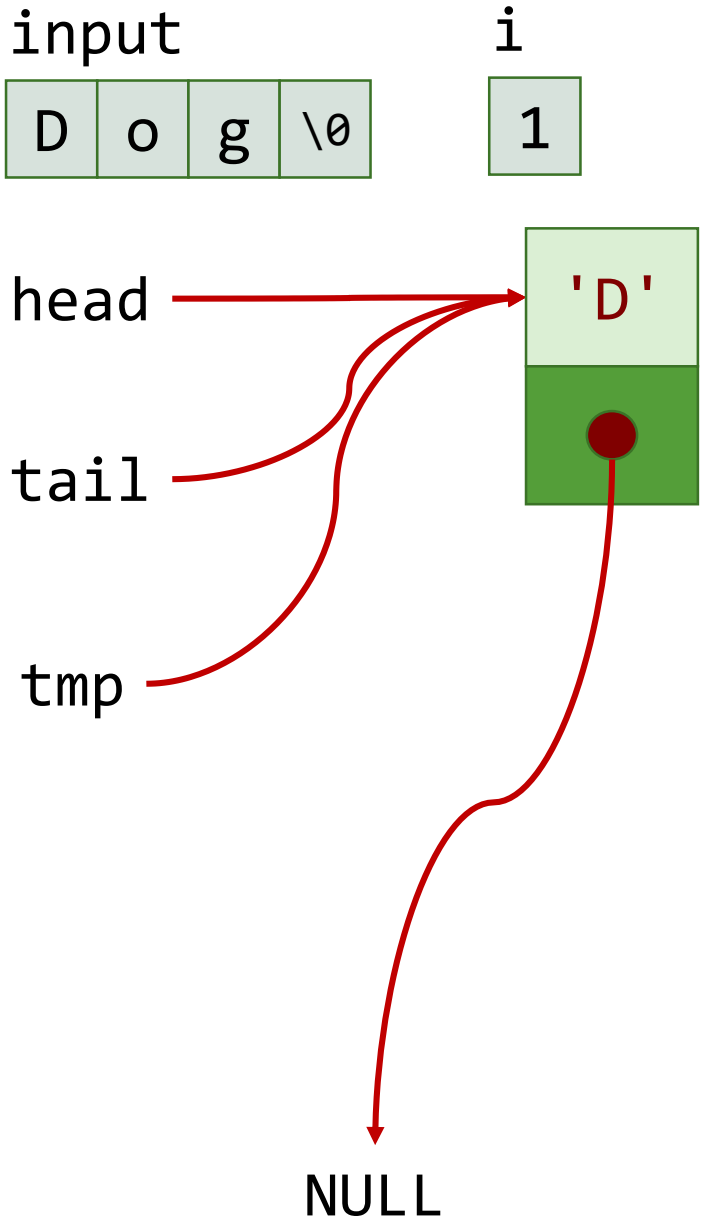
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

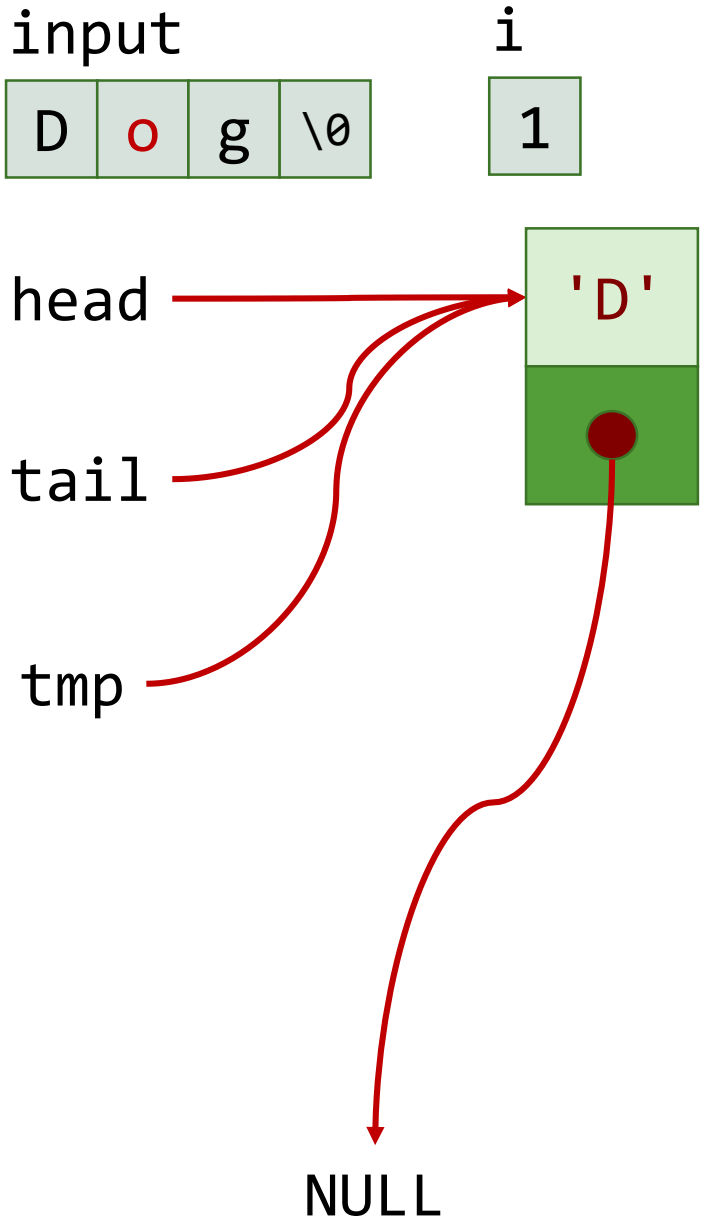
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

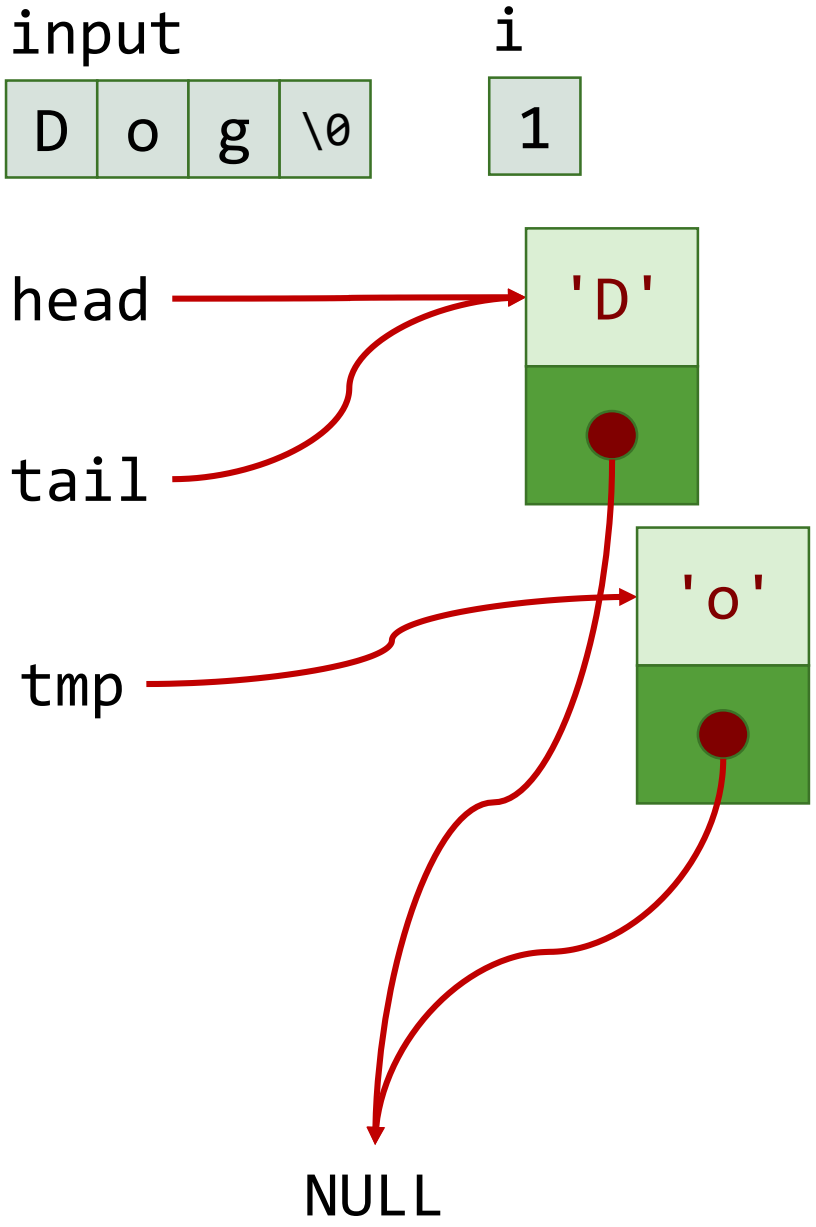
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

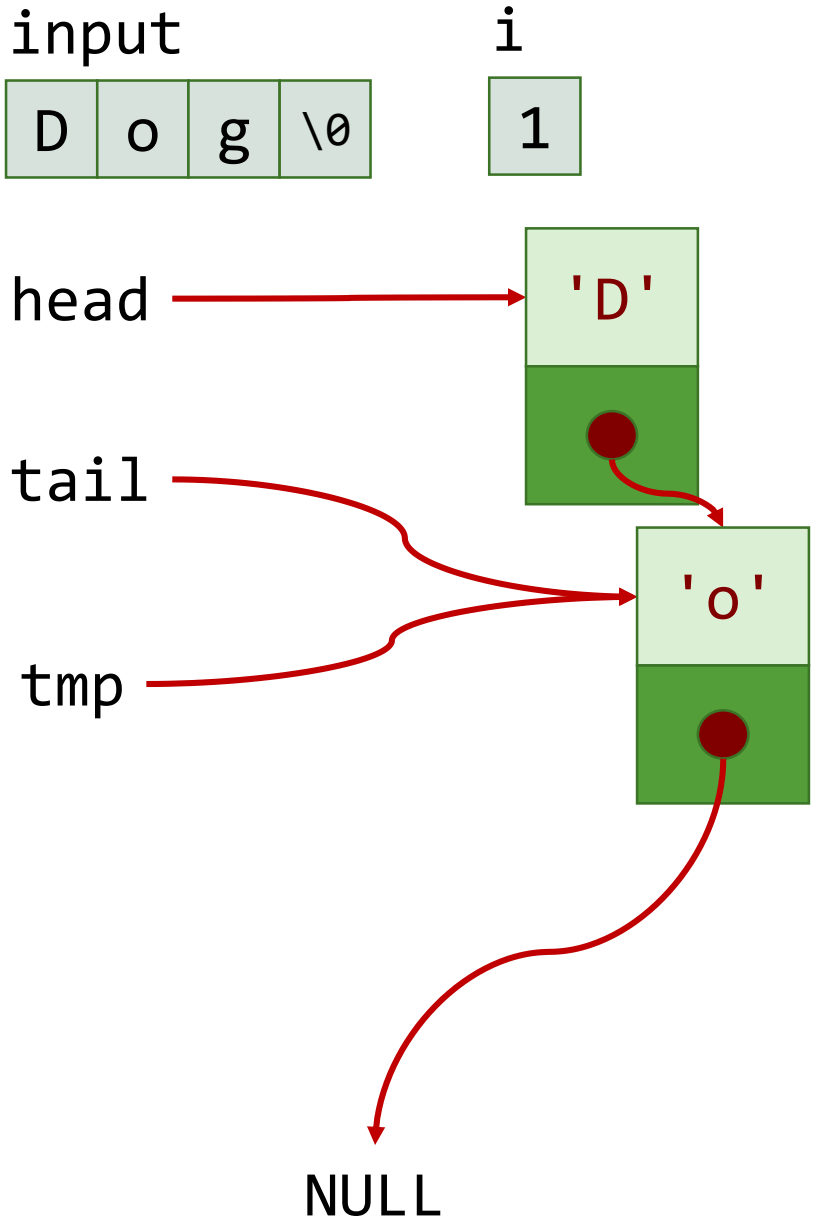
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

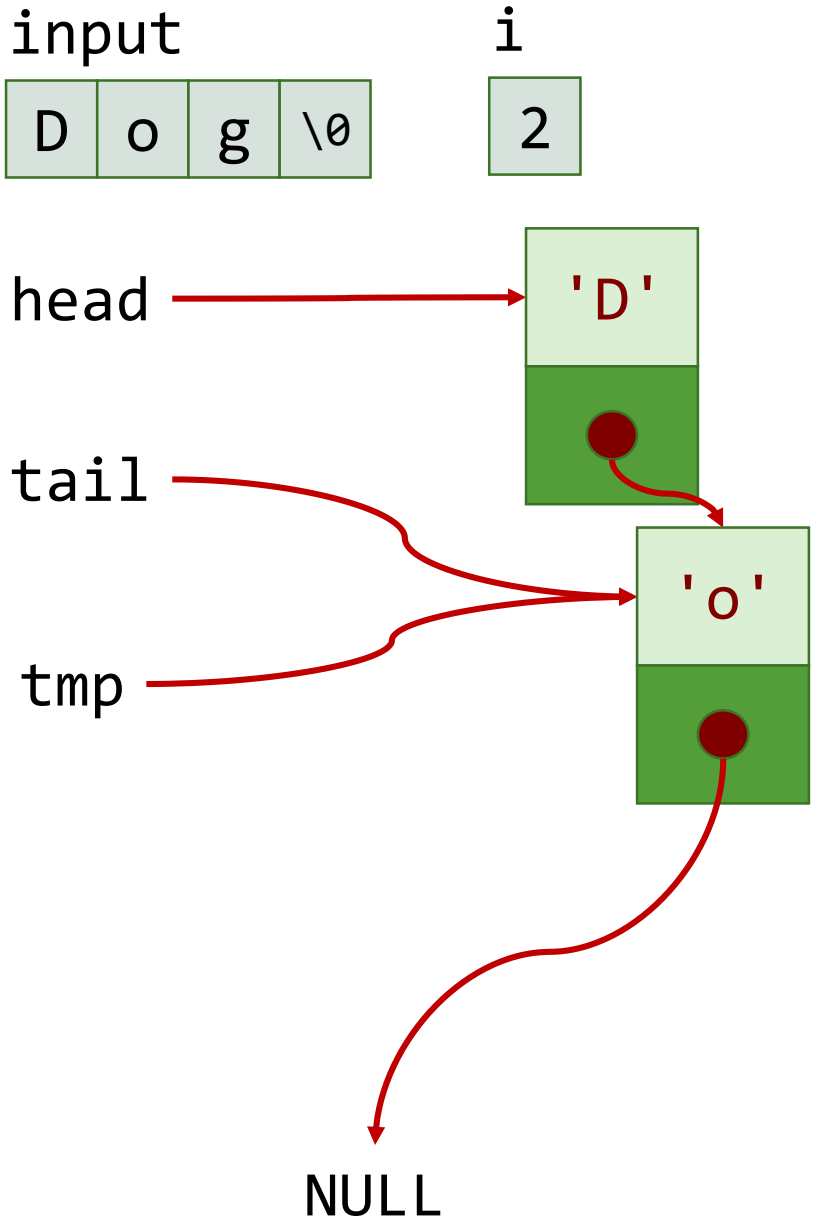
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

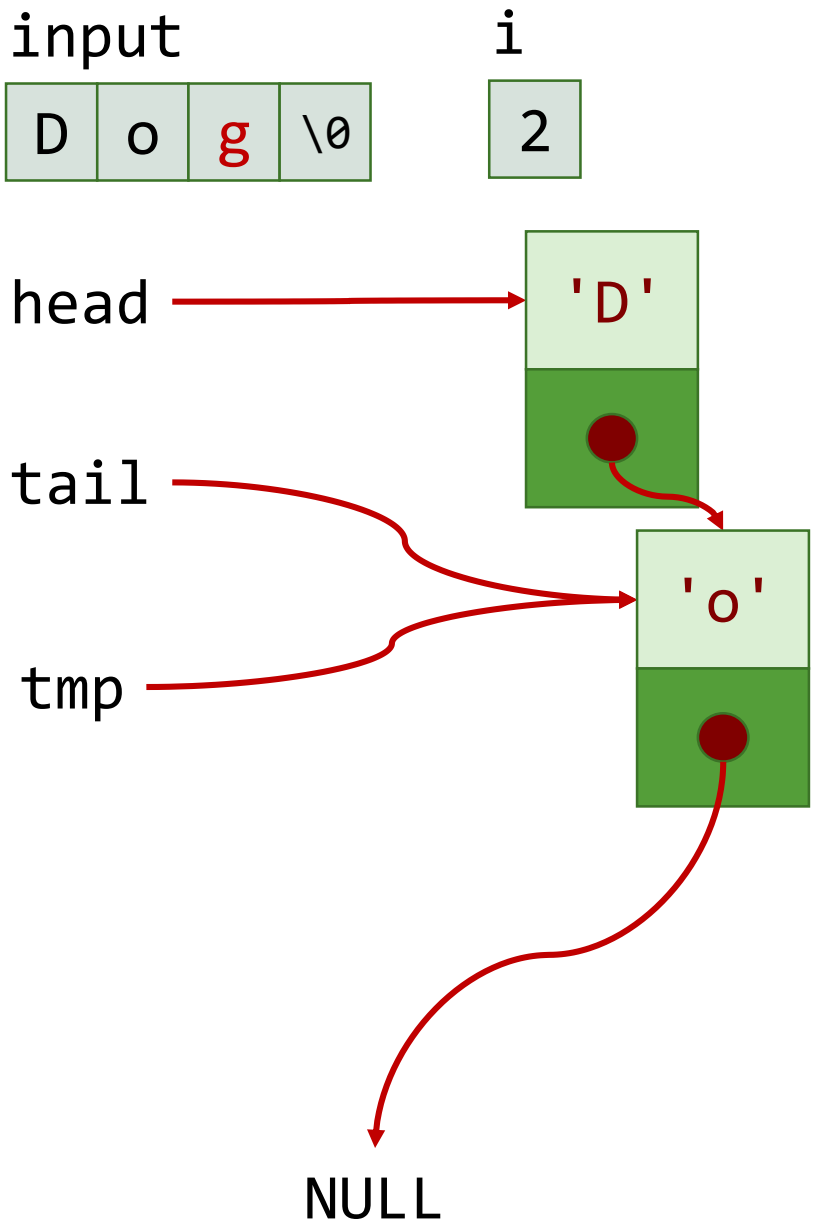
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

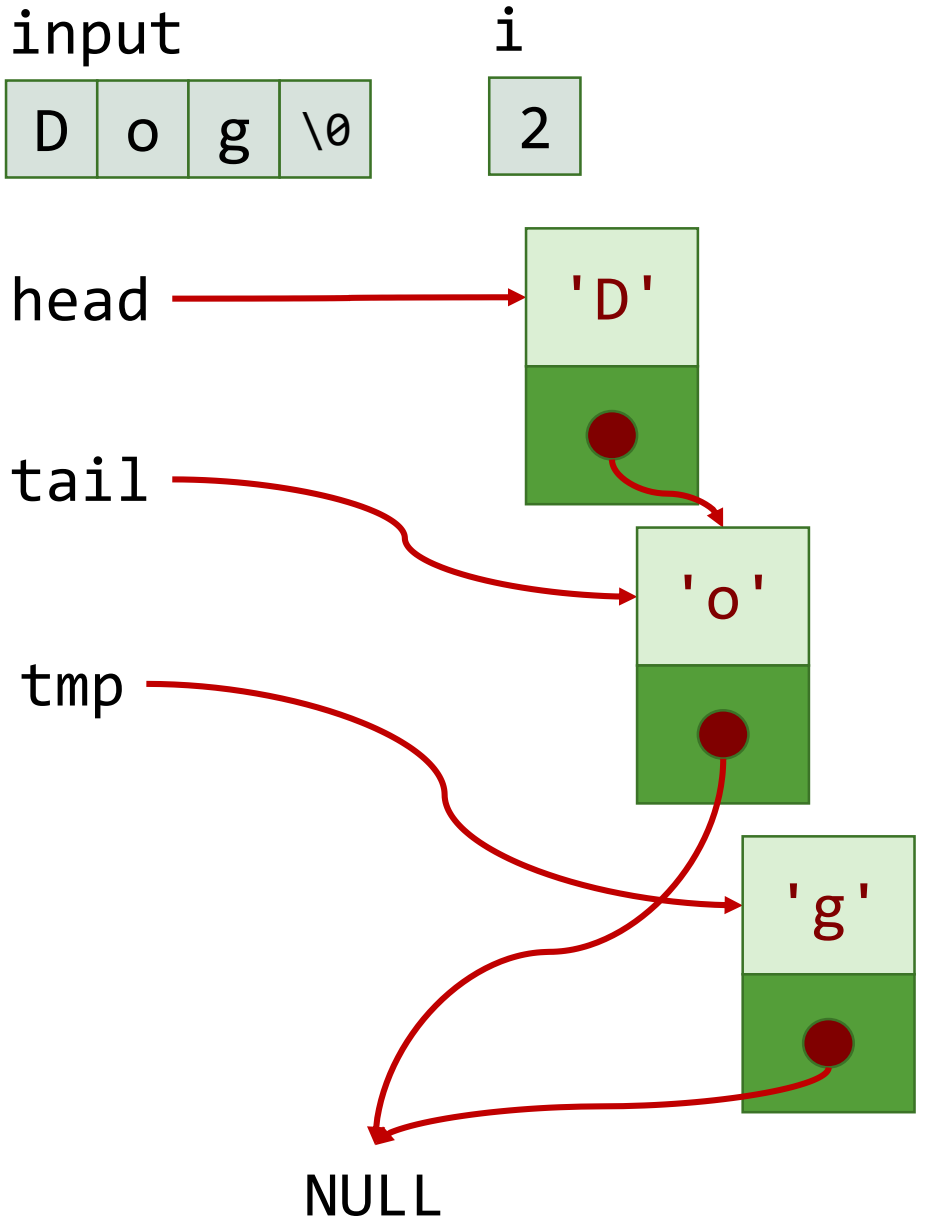
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

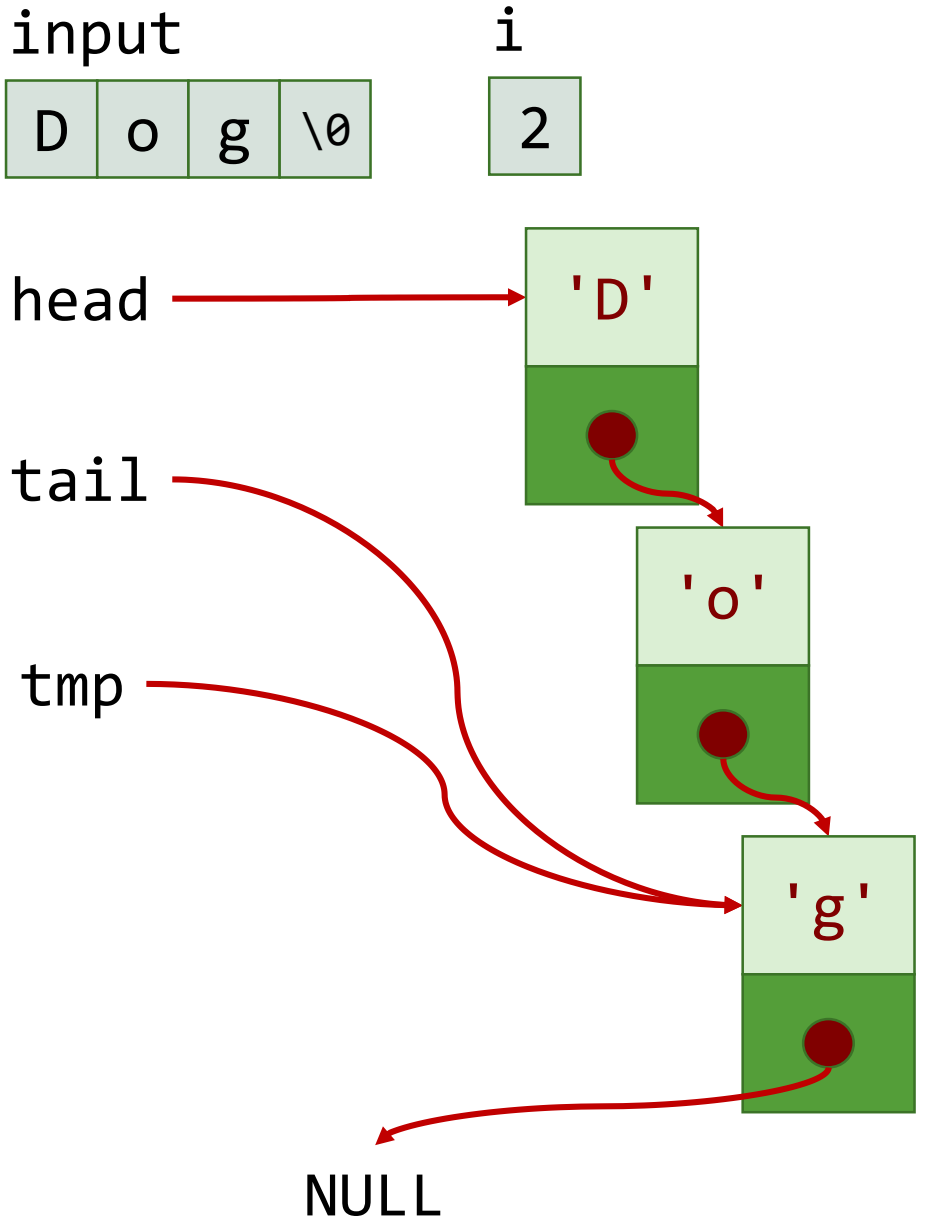
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

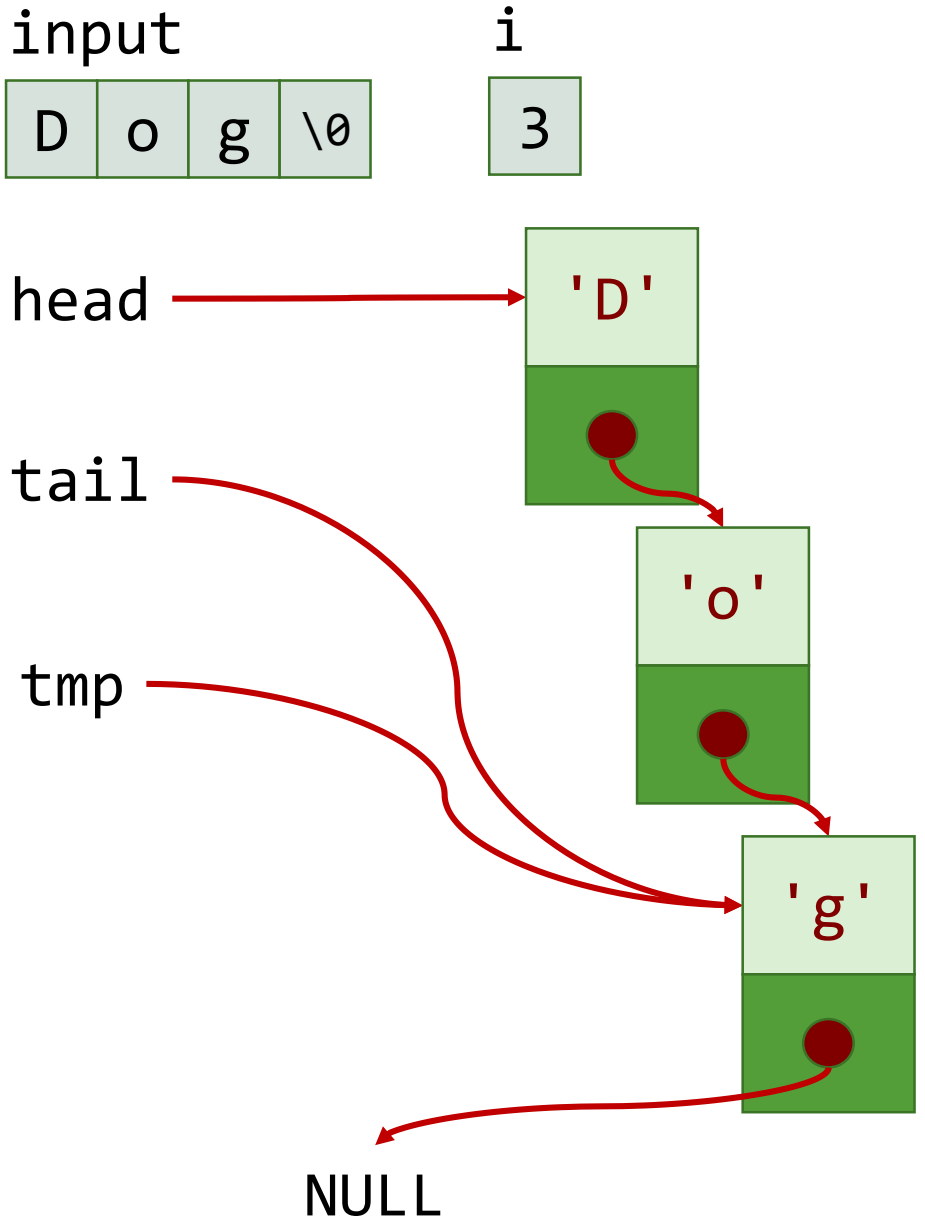
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

- Suppose user input is Dog

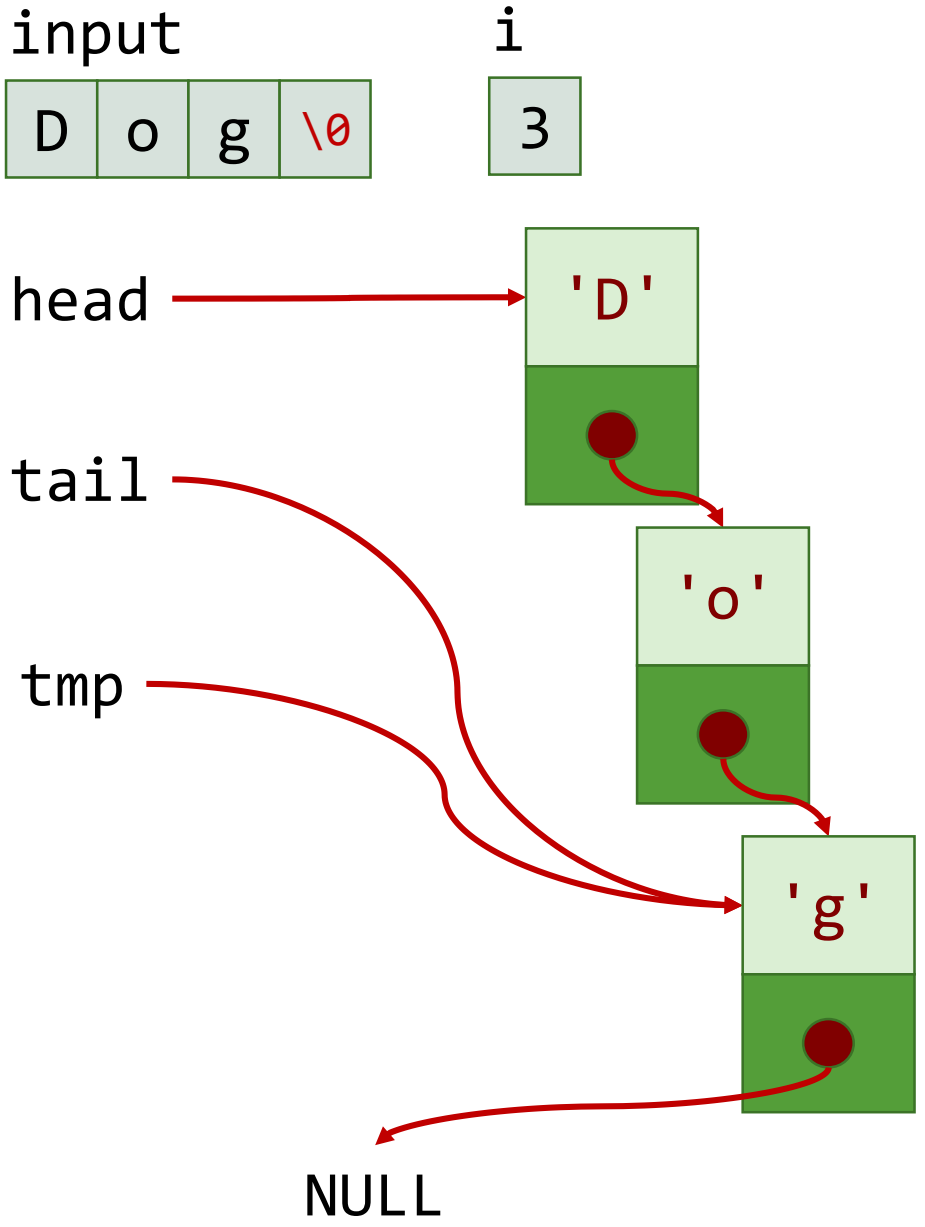
```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Walkthrough

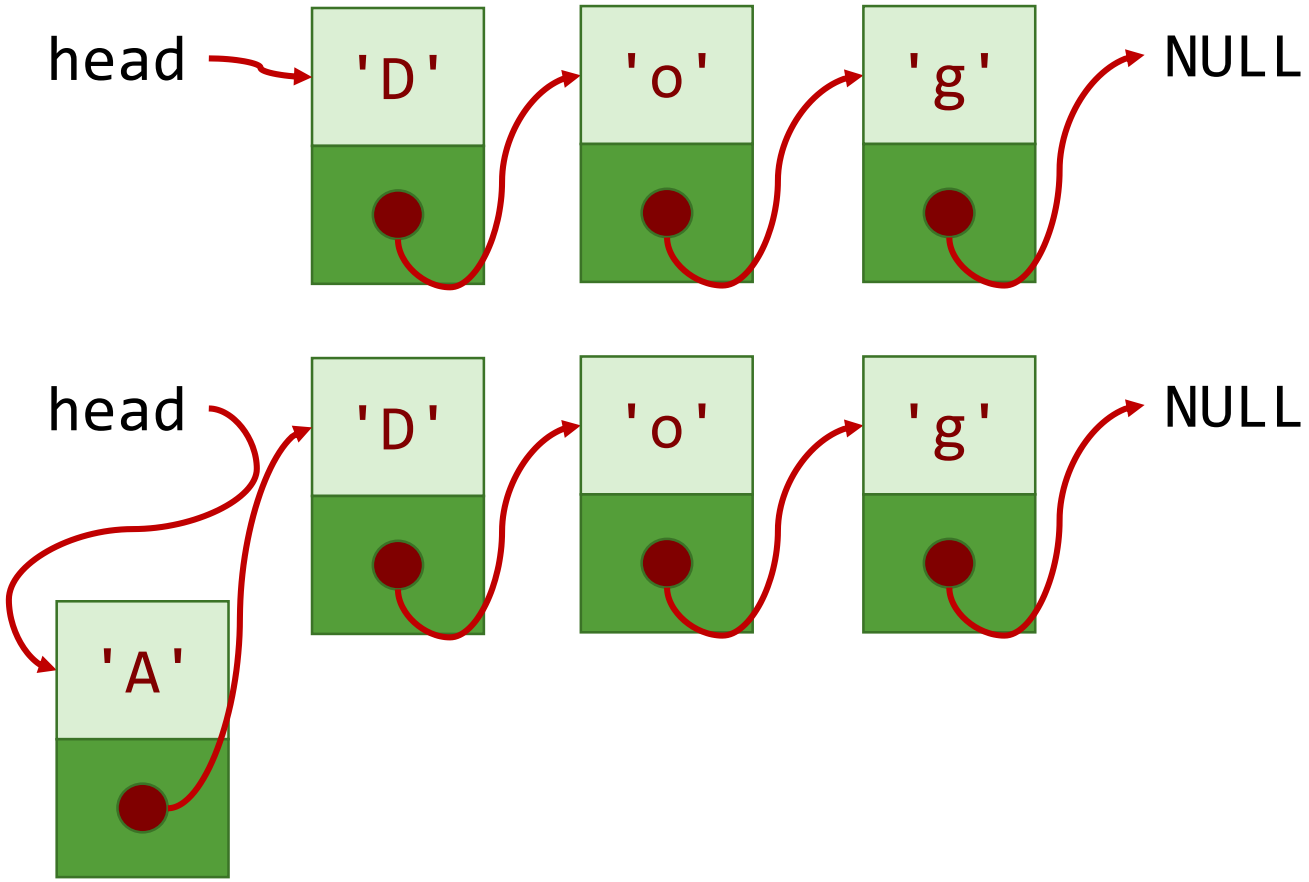
- Suppose user input is Dog

```
while(input[i] != '\0') {
    tmp = (Node *)malloc(sizeof(Node));
    tmp->data = input[i];
    tmp->next = NULL;
    if(head == NULL) {
        head = tmp;
        tail = head;
    } else {
        tail->next = tmp;
        tail = tmp;
    }
    i++;
}
```



Inserting a Node (At Head)

- Easy if insertion is before first node

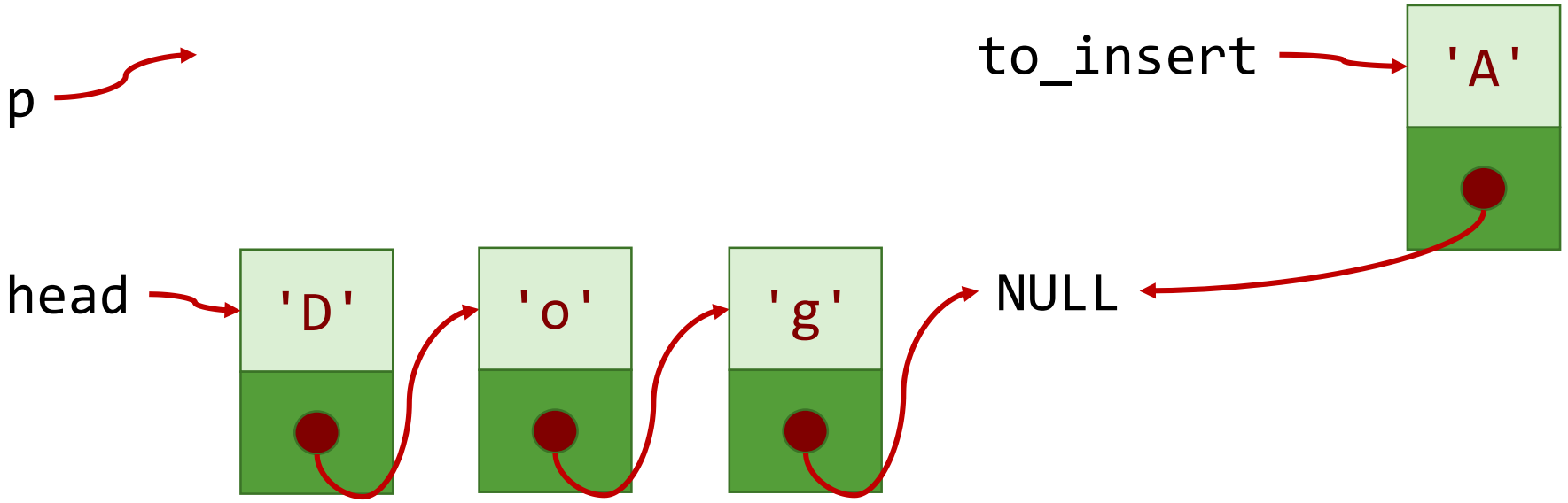



```
Node *to_insert;

/* Allocate space for
   to_insert and
   initialize */
…

to_insert->next = head;
head = to_insert;
```

Inserting a Node (Not at Head)

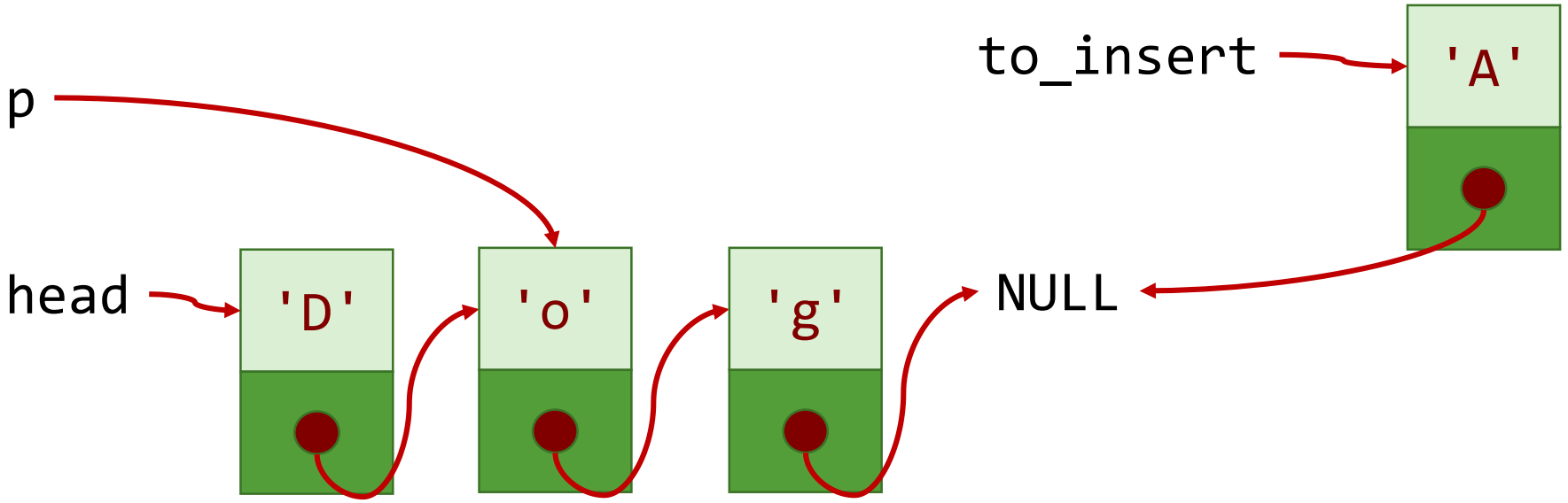
- Need to traverse list until insertion point

- Illustration: Insert `'A'` in between `'o'` and `'g'`.



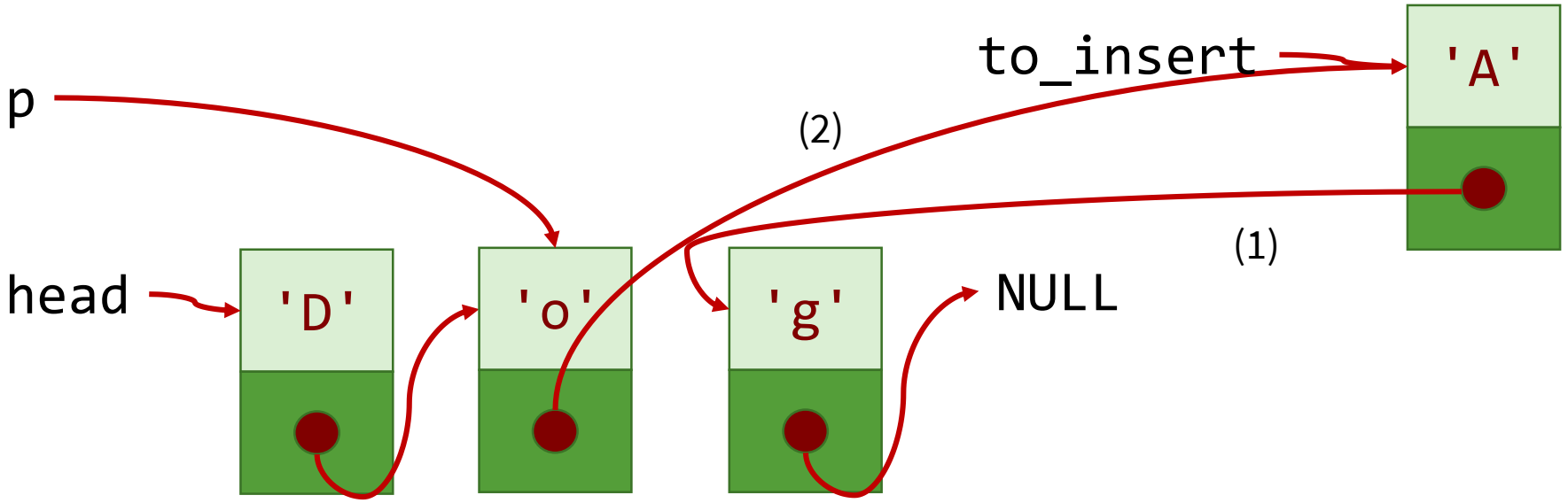
Inserting a Node (Not at Head)

- Illustration: Insert `'A'` in between `'o'` and `'g‘`
  - Traverse p until `'o'`



Inserting a Node (Not at Head)

- Illustration: Insert `'A'` in between `'o'` and `'g‘`
  - Traverse p until `'o'`
  - Insert node

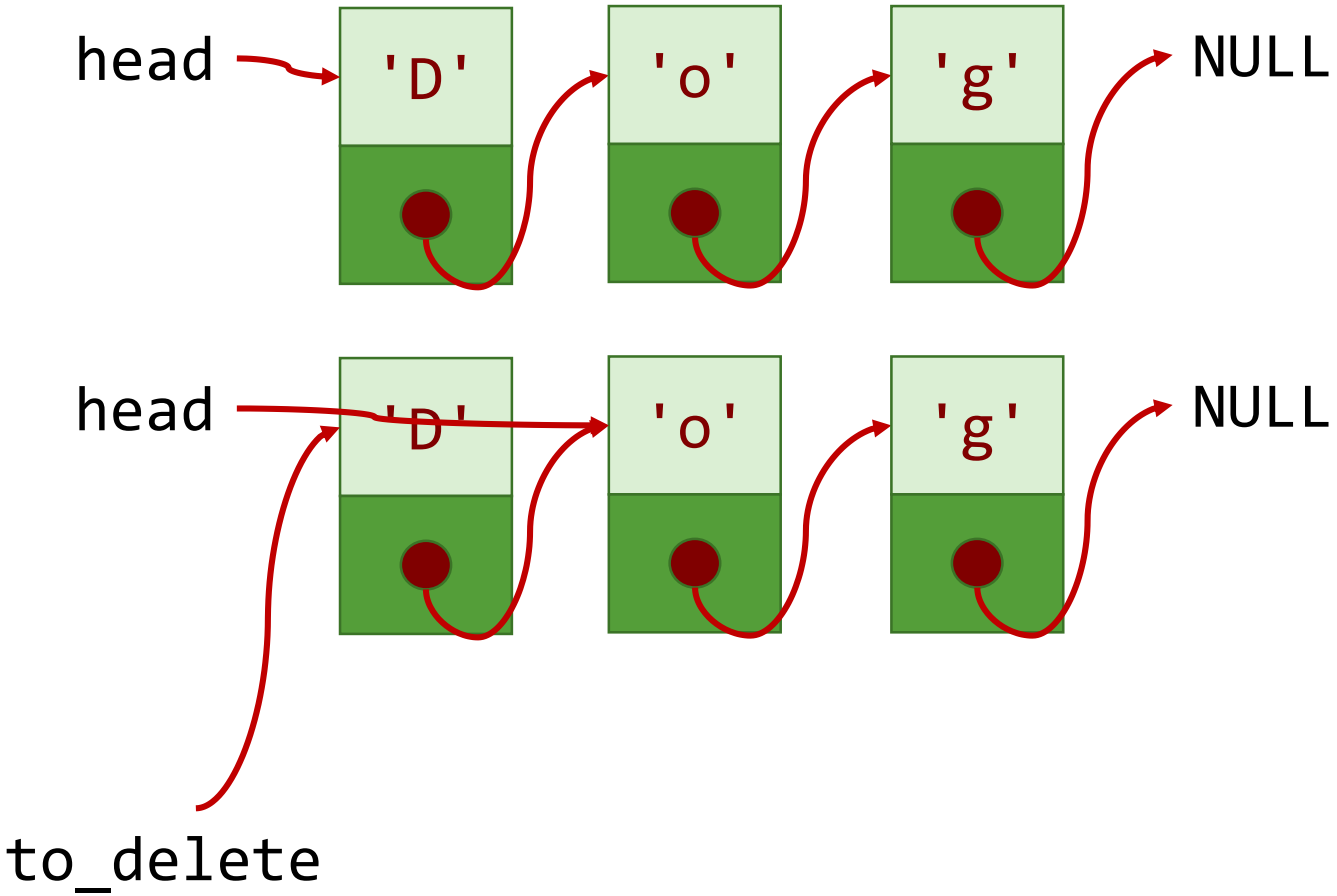


Inserting a Node (Not at Head)

```
Node *to_insert;

/* Allocate space for to_insert and initialize */
…

Node *p = head;

/* Traverse until desired node */
while(p != NULL && p->data != 'o')
      p = p->next;

/* Insert */
to_insert->next = p->next
p->next = to_insert;
```

Deleting a Node (At Head)

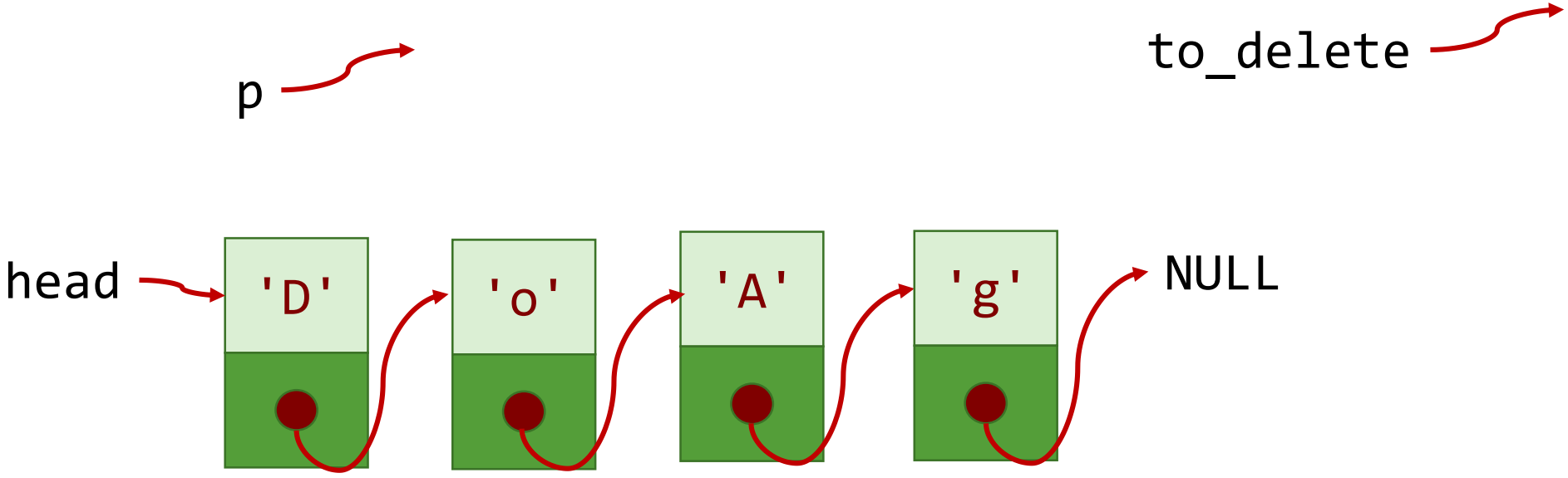
- Easy if node to delete is first node




```
Node *to_delete;

to_delete = head;
head = to_delete->next;

free(to_delete);
```

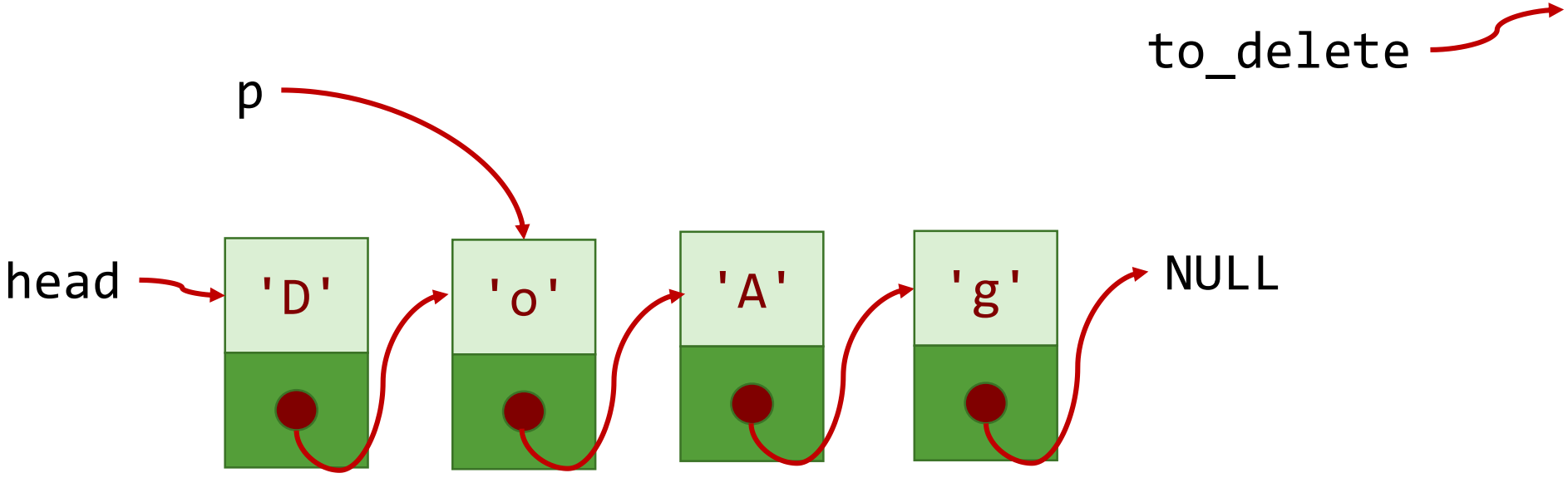
Deleting a Node (Not at Head)

- Need to traverse list until just before the node to be deleted
- Illustration: Delete `'A'` in between `'o'` and `'g'`.



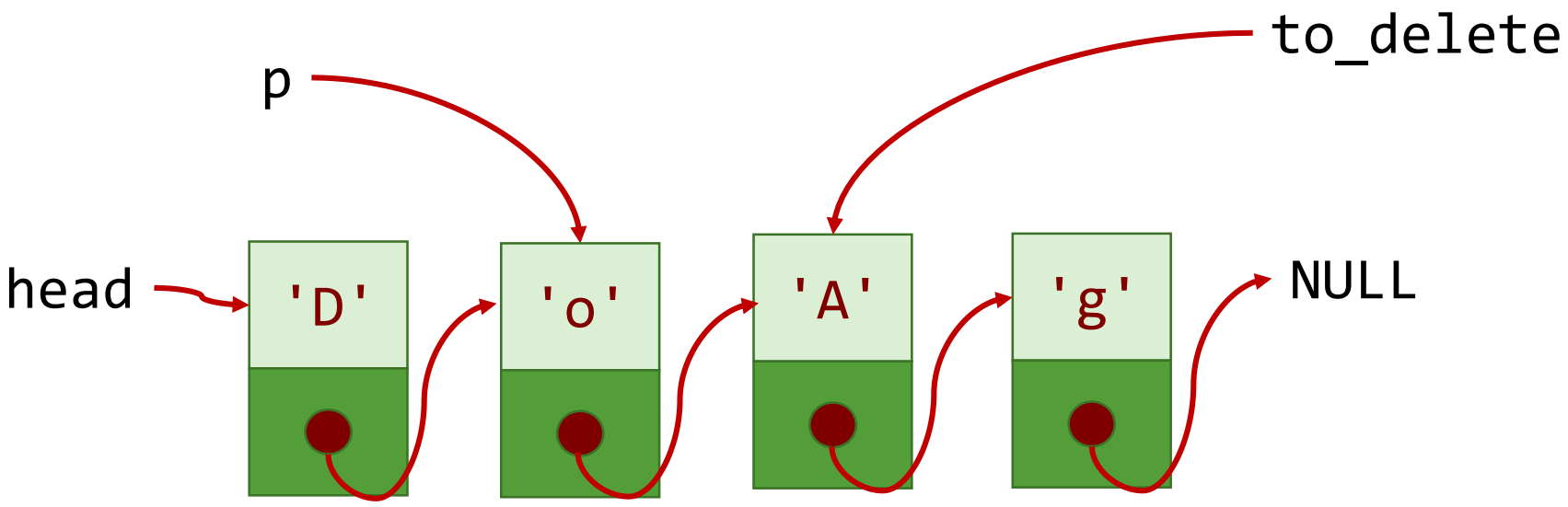
Deleting a Node (Not at Head)

- Illustration: Delete `'A'` in between `'o'` and `'g'`.
  - Traverse p until `'o'`



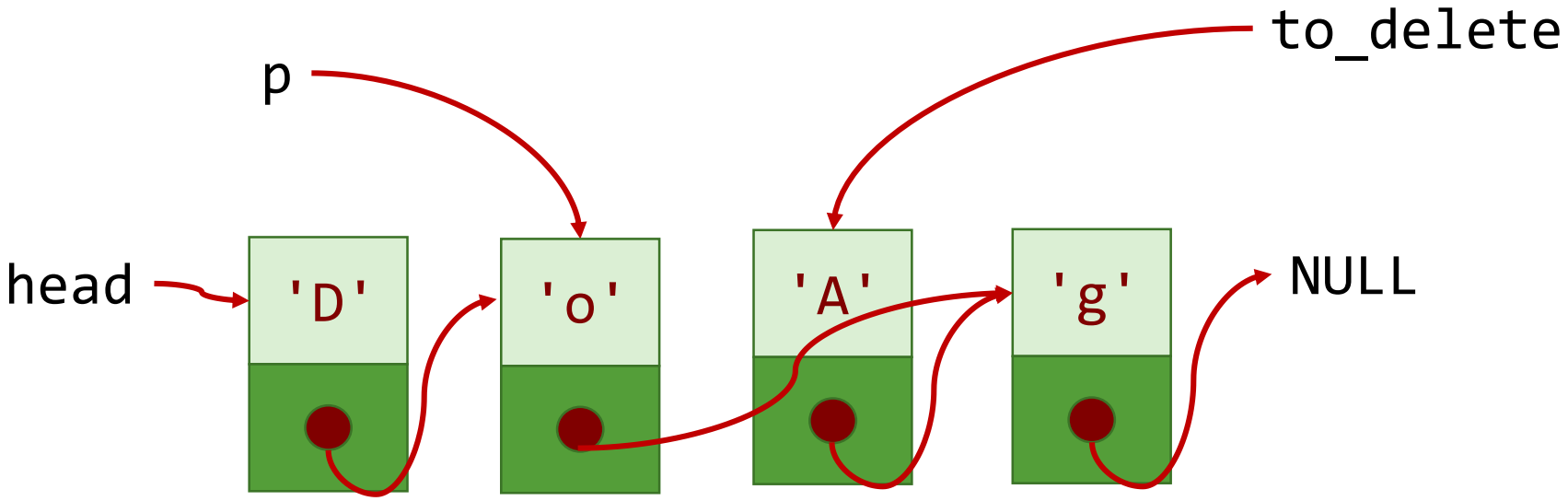
Deleting a Node (Not at Head)

- Illustration: Delete `'A'` in between `'o'` and `'g'`.
  - Traverse `p` until `'o'`
  - Let `to_delete` point to the next node (the node to be deleted)



Deleting a Node (Not at Head)

- Illustration: Delete `'A'` in between `'o'` and `'g'`.
  - Traverse p until `'o'`
  - Let `to_delete` point to the next node (the node to be deleted)
  - Delete node



Deleting a Node (Not at Head)

```
Node *to_delete;

Node *p = head;

/* Traverse until desired node */
while(p->next != NULL && p->next->data != 'A')
      p = p->next;

/* Delete */
if(p->next != NULL) {
      to_delete = p->next;
      p->next = to_delete->next;
      free(to_delete);
}
```

Things to Consider

- What if list is empty?

- What if traversal reaches end (NULL)?