STEVENS INSTITUTE of TECHNOLOGY

Schaefer School of Engineering & Science

CPE 360

Computational Data Structures & Algorithms

Zhanfu Yang
Teaching Assistant
Electrical and Computer Engineering

For academic use only.

Struct

```
struct data {
    int value;
    struct data * next;
};
```

Int a[4];

a[0]	a[1]	a[2]	a[3]
int	int	int	int

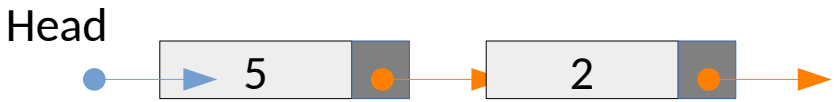
(1) Unlike an array, a structure can contain many different data types (int, string, bool, etc.).

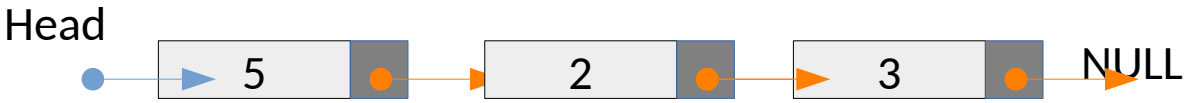
Tailed link list

```c
struct data * tail_linked_list(struct data *head_linked_list)
{
    struct data *ptr = head_linked_list;
    while (ptr->next != NULL) ptr = ptr->next;
//    printf("== Tail value = %d\n", ptr->value);

    return ptr;
}
```


```c
    insert_element_tail(my_linked_list, 2);
    insert_element_tail(my_linked_list, 3);
```





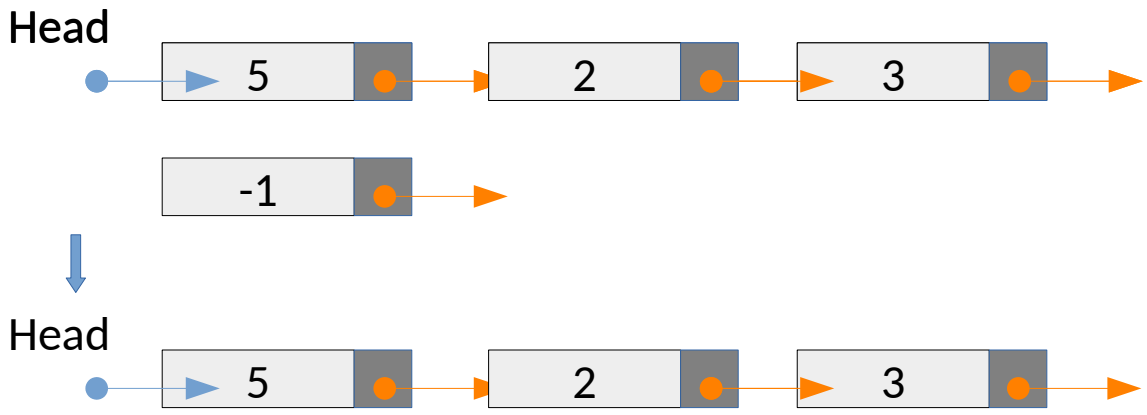
Insert Element

```
void insert_element(struct data *head_linked_list, int val_before_insertion, int val_to_insert)
{
    //Create a container to store the to-be-inserted value;
    struct data *tmp = new struct data;
    tmp -> value = val_to_insert;

    //Search through the linked list to find "val_before_insertion" or
    //- if we didn't find it, it returns the tail
    struct data *ptr = head_linked_list;
    while ((ptr->value != val_before_insertion) && (ptr->next != NULL)) ptr = ptr->next;

    //Let the created container store the next of "value_before_insertion"
    tmp->next = ptr->next;

    //Let the value_before_insertion's next store the created container
    ptr->next = tmp;
}
```



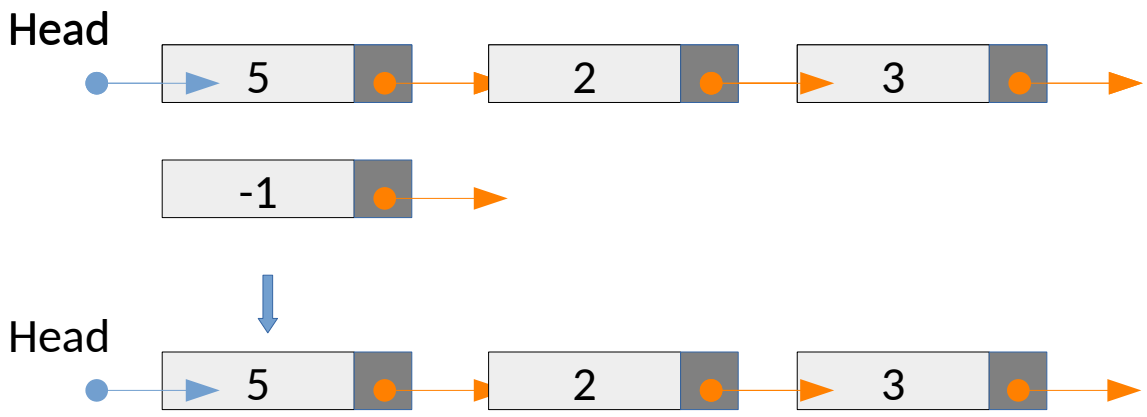

```
insert_element(my_linked_list, 2, -1);
```

```
void insert_element(struct data *head_linked_list, int val_before_insertion, int val_to_insert)
{
    //Create a container to store the to-be-inserted value;
    struct data *tmp = new struct data;
    tmp -> value = val_to_insert;

    //Search through the linked list to find "val_before_insertion" or
    //- if we didn't find it, it returns the tail
    struct data *ptr = head_linked_list;
    while ((ptr->value != val_before_insertion) && (ptr->next != NULL)) ptr = ptr->next;

    //Let the created container store the next of "value_before_insertion"
    tmp->next = ptr->next;

    //Let the value_before_insertion's next store the created container
    ptr->next = tmp;
}
```



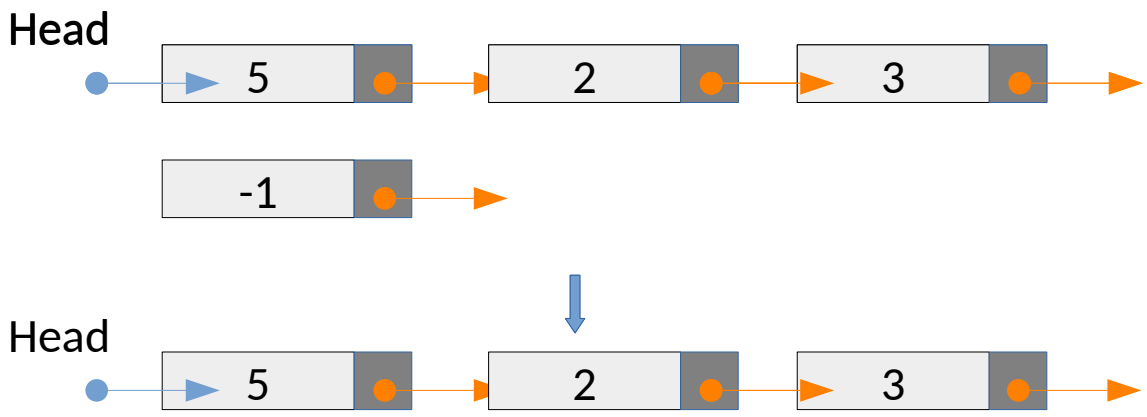

```
insert_element(my_linked_list, 2, -1);
```

```
void insert_element(struct data *head_linked_list, int val_before_insertion, int val_to_insert)
{
    //Create a container to store the to-be-inserted value;
    struct data *tmp = new struct data;
    tmp -> value = val_to_insert;

    //Search through the linked list to find "val_before_insertion" or
    //- if we didn't find it, it returns the tail
    struct data *ptr = head_linked_list;
    while ((ptr->value != val_before_insertion) && (ptr->next != NULL)) ptr = ptr->next;

    //Let the created container store the next of "value_before_insertion"
    tmp->next = ptr->next;

    //Let the value_before_insertion's next store the created container
    ptr->next = tmp;
}
```



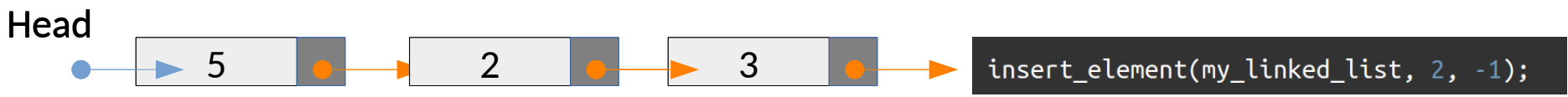

```
insert_element(my_linked_list, 2, -1);
```

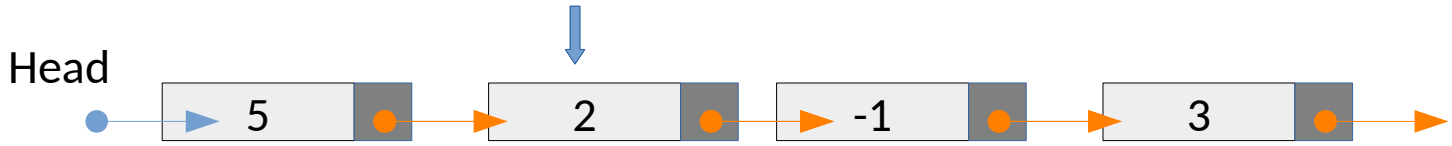
```
void insert_element(struct data *head_linked_list, int val_before_insertion, int val_to_insert)
{
    //Create a container to store the to-be-inserted value;
    struct data *tmp = new struct data;
    tmp -> value = val_to_insert;

    //Search through the linked list to find "val_before_insertion" or
    //- if we didn't find it, it returns the tail
    struct data *ptr = head_linked_list;
    while ((ptr->value != val_before_insertion) && (ptr->next != NULL)) ptr = ptr->next;

    //Let the created container store the next of "value_before_insertion"
    tmp->next = ptr->next;

    //Let the value_before_insertion's next store the created container
    ptr->next = tmp;
}
```





Delete Element

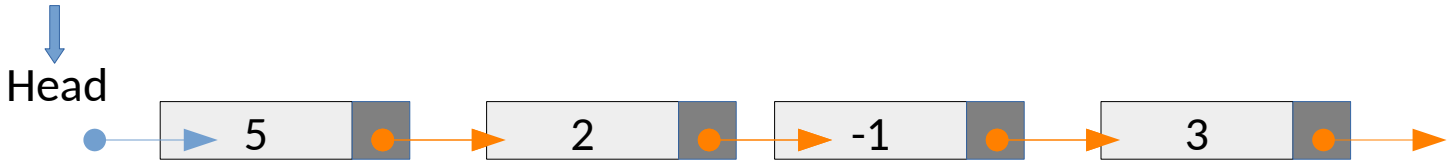
```
void delete_element(struct data *head_linked_list, int val_to_delete)
{
    //Search through the linked list to find "val_to_delete" or
    //- if we didn't find it, it returns the tail
    struct data *ptr_current = head_linked_list;
    struct data *ptr_prior = NULL;

    while ((ptr_current -> value != val_to_delete) && (ptr_current->next != NULL))
    {
        ptr_prior = ptr_current;
        ptr_current = ptr_current->next;

    }

    //Either we find ptr_current->value == val_to_delete
    if (ptr_current -> value == val_to_delete)
        ptr_prior->next = ptr_current->next;
}
```

```
delete_element(my_linked_list, 2);
```



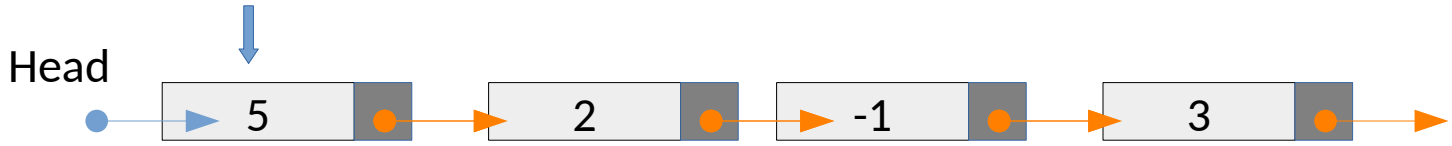
```c
void delete_element(struct data *head_linked_list, int val_to_delete)
{
    //Search through the linked list to find "val_to_delete" or
    //- if we didn't find it, it returns the tail
    struct data *ptr_current = head_linked_list;
    struct data *ptr_prior = NULL;

    while ((ptr_current -> value != val_to_delete) && (ptr_current->next != NULL))
    {
        ptr_prior = ptr_current;
        ptr_current = ptr_current->next;

    }

    //Either we find ptr_current->value == val_to_delete
    if (ptr_current -> value == val_to_delete)
        ptr_prior->next = ptr_current->next;
}
```

```c
delete_element(my_linked_list, 2);
```



```c
void delete_element(struct data *head_linked_list, int val_to_delete)
{
    //Search through the linked list to find "val_to_delete" or
    //- if we didn't find it, it returns the tail
    struct data *ptr_current = head_linked_list;
    struct data *ptr_prior = NULL;

    while ((ptr_current -> value != val_to_delete) && (ptr_current->next != NULL))
    {
        ptr_prior = ptr_current;
        ptr_current = ptr_current->next;

    }

    //Either we find ptr_current->value == val_to_delete
    if (ptr_current -> value == val_to_delete)
        ptr_prior->next = ptr_current->next;
}
```

```c
delete_element(my_linked_list, 2);
```