Today's plan

- Remainder:
  - Find your mid-term project partner and register your team on Piazza.
  - Midterm project introduction on Friday (3/5).
- Review
- Ex 6-2

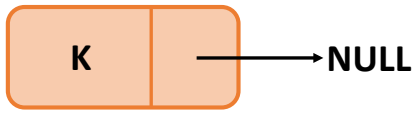
Ex 6-1 - linked list

- `add_after(Node* cur, char data);`
  - Crate new node with `data`.
  - Update new node's next pointer first using `cur->next`.
  - Update `cur->next` to point to the new node.
- `length(const Node* cur);`
  - Set a counter, advance it while iterating the list until the end (using the next pointer). Or
  - recursion on `cur->next`. i.e. `length(cur) = length(cur->next) + 1`.
- `print(const Node* cur);`
  - Print `data` while iterating the list until the end. Or
  - recursion on `cur->next`. i.e. print out `cur->data`, then call `print(cur->next)`.
- `reverse_print(const Node* cur);`
  - Get the length, allocate an array, iterate the list and store the data in an array. Then do the reverse print. Or
  - recursion on `cur->next`. i.e. call `print(cur->next)`, then print out `cur->data`.

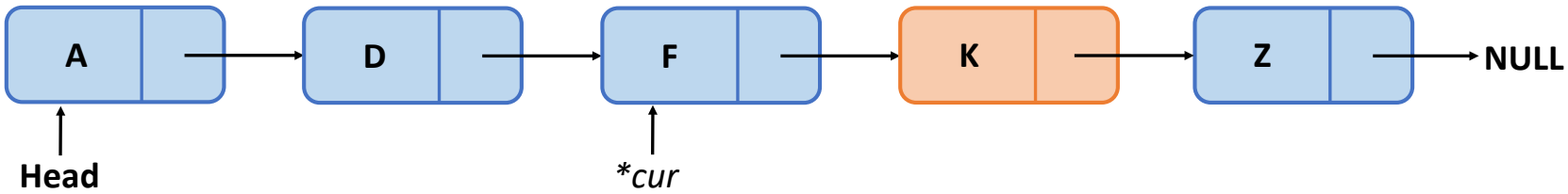
Linked lists

add_after(*cur*, 'K');

1) Create a new node

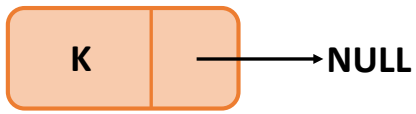


2) Add the new node after **cur*

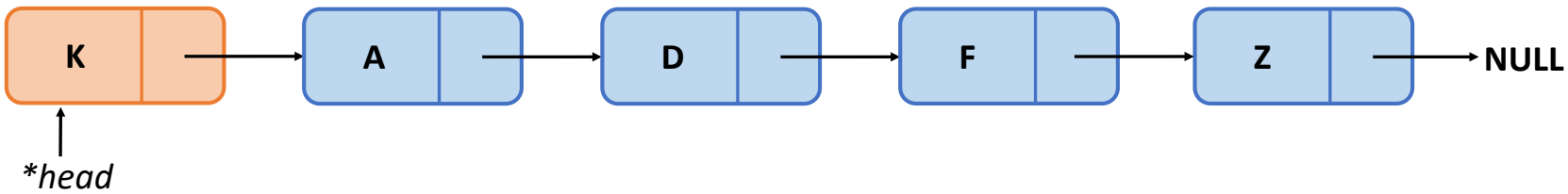


add_front(&*head*, 'K');

1) Create a new node

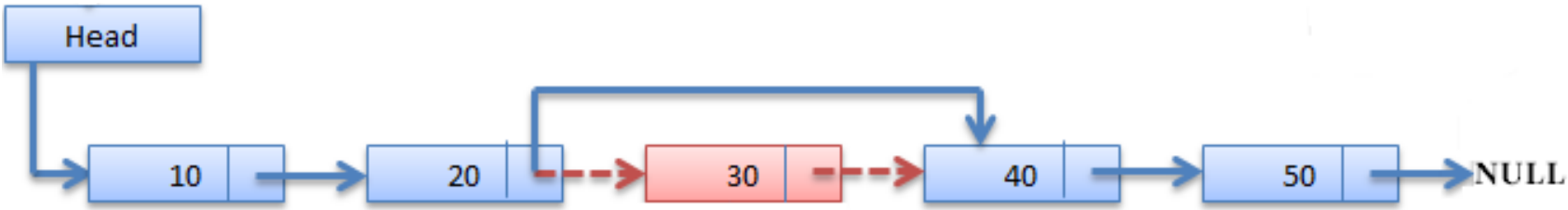


2) Add the new node in the front

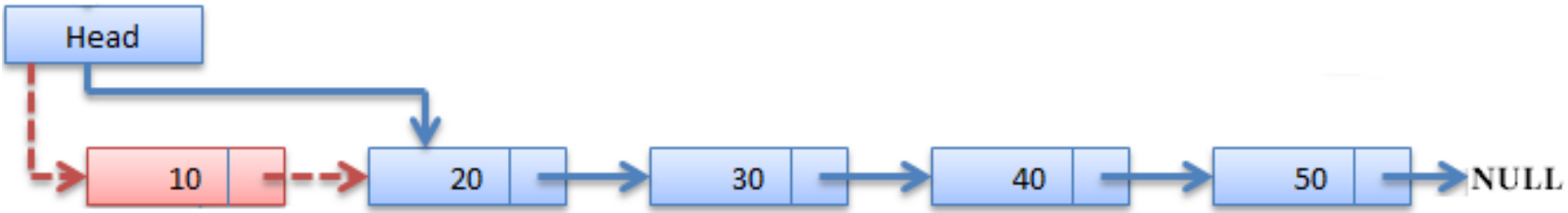


Linked lists

```
char remove_after(Node* node);
```




```
char remove_front(Node** head_ptr);
```



Linked lists

- Doubly linked list: i.e. bi-directional
  - Add a new pointer (e.g. `prev`) that point to the previous node similar to the next pointer.
  - `prev` is `NULL` if it has no previous node.
- Circular linked list: i.e. tail node's next pointer is pointing to the head node.
  - tail node's next pointer is set to the head.
- Multiply linked list: i.e. one node has multiple links to other nodes
  - the next pointer is changed to an array of pointers to node or pointer to pointer (if it is dynamically allocated.)
  - a field to keep track of the number of child nodes.

Linked lists

Definition of a Node data type:

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

Consider the following function:

```
void mystery(Node **head_ptr) {
    Node* head = *head_ptr;
    *head_ptr = (*head_ptr)->next;
    free(head);
}
```

What does this function do?

A Correctly removes the first node for any list

B Correctly removes the first node of any non-empty list

C Has no effect

D The code does not compile

E None of the above

Class exercises

Ex 6-2