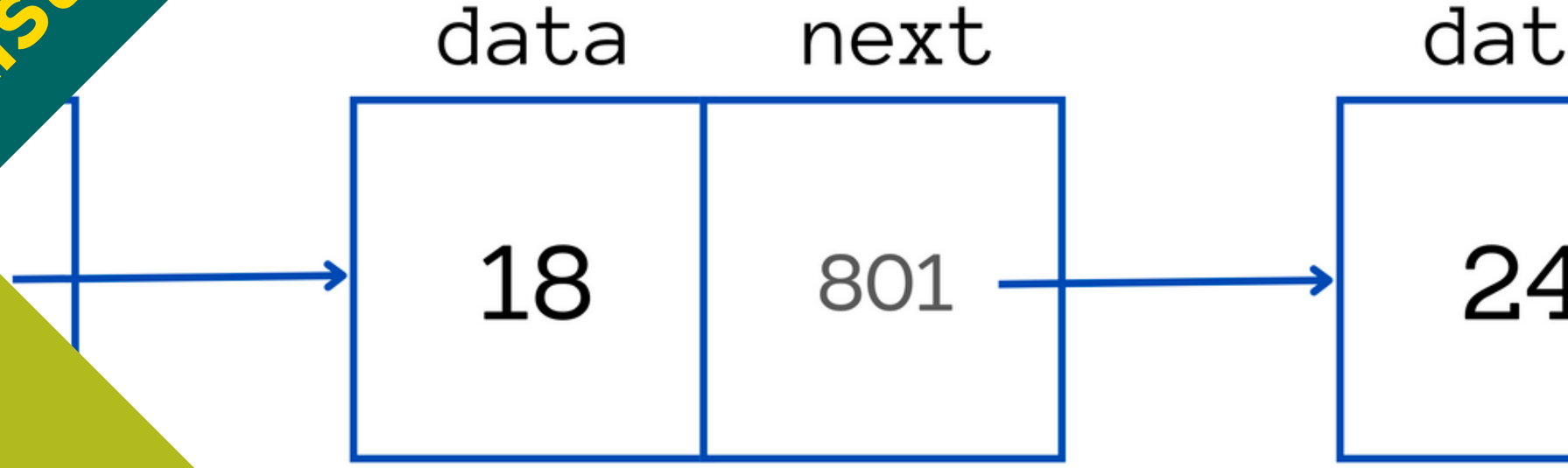
Linked List

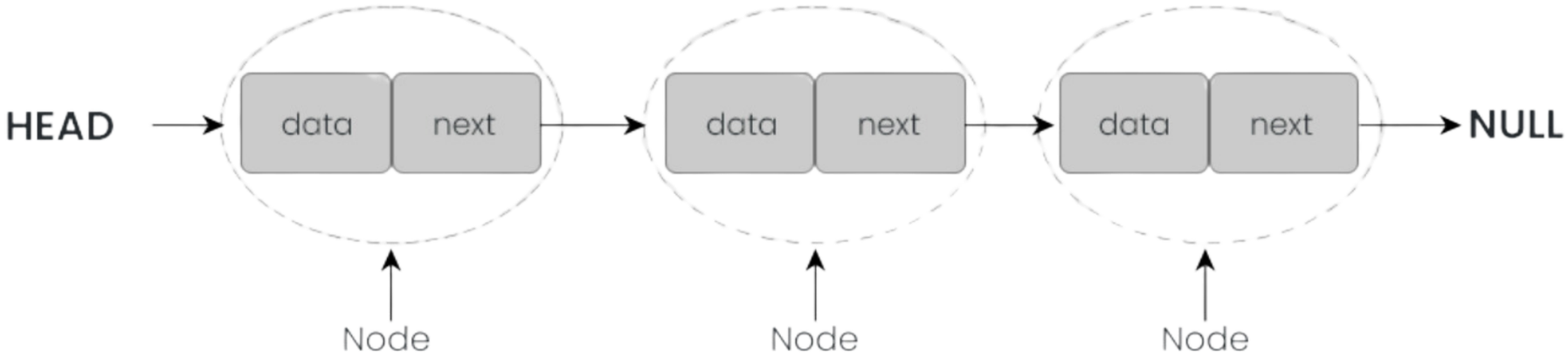


Linked List

- Linked List
- Operation on Linked List : Insertion
- Operation on Linked List : Iteration
- Operation on Linked List : Deletion

Linked List

Linked List is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers:



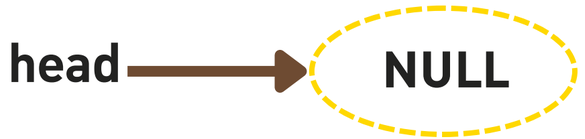
Node

```c
struct __Node {
    int data;
    struct __Node *next;
};

typedef struct __Node Node;
```

HEAD

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

typedef struct __Node Node;
typedef Node* List;
```

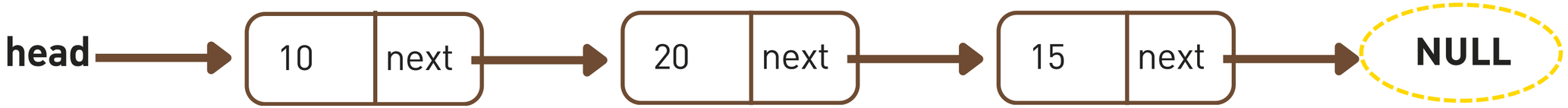
```
int main() {
    List head = NULL;
}
```



insert

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

typedef struct __Node Node;
typedef Node* List;

void insert(List *list, int value);
```

```
int main() {
    List head = NULL;
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 15);
}
```



Operation on Linked List ▶ Insertion

insert

Step 1

- allocate memory เพื่อสร้าง Node*
- ถ้า list ว่าง ให้ list ชี้ที่ node

Step 2

- Loop หา Node สุดท้าย (prev)
- prev->next ชี้ไปที่ Node ที่สร้างใหม่

```
void insert(List *list, int value) {
    // Step 1
    // Node* node = (Node*)malloc(sizeof(Node));
    List node = (List) malloc(sizeof(Node));
    node->data = value;
    node->next = NULL;

    if (*list == NULL) {
        *list = node;
        return;
    }
    // Step 2 >> NEXT...
}
```

Operation on Linked List ▶ Insertion

insert

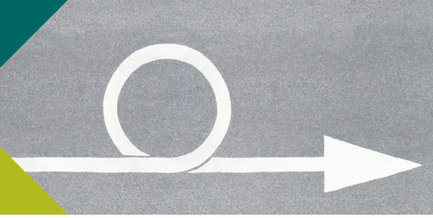
Step 1

- allocate memory เพื่อสร้าง Node*
- ถ้า list ว่าง ให้ list ชี้ที่ node

Step 2

- Loop หา Node สุดท้าย (prev)
- prev->next ชี้ไปที่ Node ที่สร้างใหม่

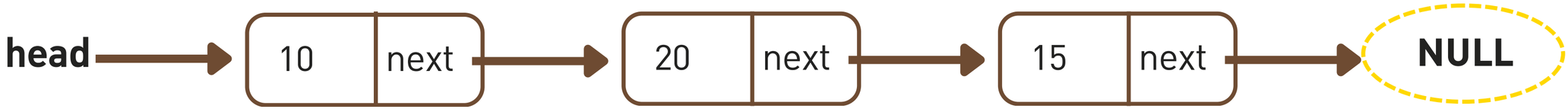
```
void insert(List *list, int value) {
    // Step 1  >> ...PREVIOUS
    // Step 2
    List current = *list;
    List prev = NULL;

    while (current != NULL) {
        prev = current;
        current = current->next;
    }
    prev->next = node;
}
```



print

```
struct __Node {
    int data;
    struct __Node *next;
};
typedef struct __Node Node;
typedef Node* List;

void insert(List *list, int value);
void print(List list);
```

```
int main() {
    List head = NULL;
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 15);
    print(head);
    printf("\n");
}
```



Operation on Linked List ▶ Iteration

print

Version While Loop

```
void print(List list) {
    List current = list;

    while (current != NULL) {
        printf("[%d]->", current->data);
        current = current->next;
    }
}
```

Operation on Linked List ▶ Iteration

print

Version Recursive

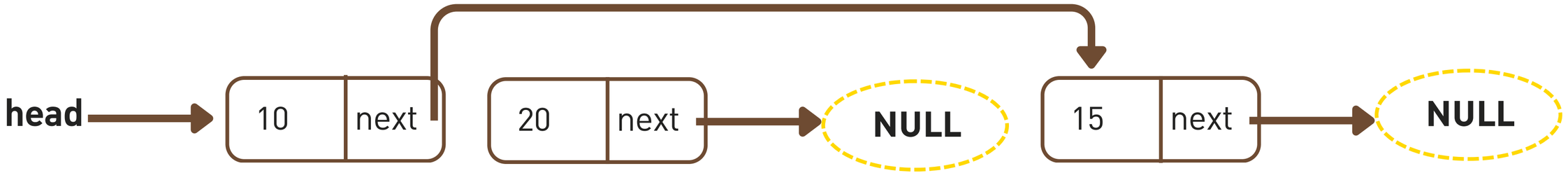
```
void print(List list) {
    if (list == NULL) return;

    printf("[%d]->", list->data);
    print(list->next);
}
```

Operation on Linked List ▶ Deletion

delete

```
struct __Node {
    int data;
    struct __Node *next;
};
typedef struct __Node Node;
typedef Node* List;
void insert(List *list, int value);
void print(List list);
void delete(List* list, int value);
```

```
int main() {
    List head = NULL;
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 15);
    delete(&head, 20);
    print(head);
    printf("\n");
}
```



Operation on Linked List ▶ Deletion

delete

Step 1

- หาว่า value อยู่ Node ใด
- prev ชี้ Node ก่อนหน้า
- current ชี้ Node ที่หา

Step 2 (ถ้าเจอ)

- ให้ prev->next ชี้ไปที่ current->next
- ให้ current->next ชี้ไปที่ NULL

ระวัง

- ลบ root node

```
void delete(List *list, int value) {
    // Step 1
    List current = *list;
    List prev = NULL;

    while (current != NULL) {
        if (current->data == value) break;
        prev = current;
        current = current->next;
    }
    // Step 2 >> NEXT...
}
```

Operation on Linked List ▶ Deletion

delete

Step 1

- หาว่า value อยู่ Node ใด
- prev ชี้ Node ก่อนหน้า
- current ชี้ Node ที่หา

Step 2 (ถ้าเจอ)

- ให้ prev->next ชี้ไปที่ current->next
- ให้ current->next ชี้ไปที่ NULL

ระวัง

- ลบ root node

```
void delete(List *list, int value) {
    // Step 1 >> ...PREVIOUS
    // Step 2
    if (current == NULL) return; // not found

    if (prev == NULL) { // delete root node
        *list = current->next;
    } else {
        prev->next = current->next;
    }
    current->next = NULL;
    free(current);
}
```