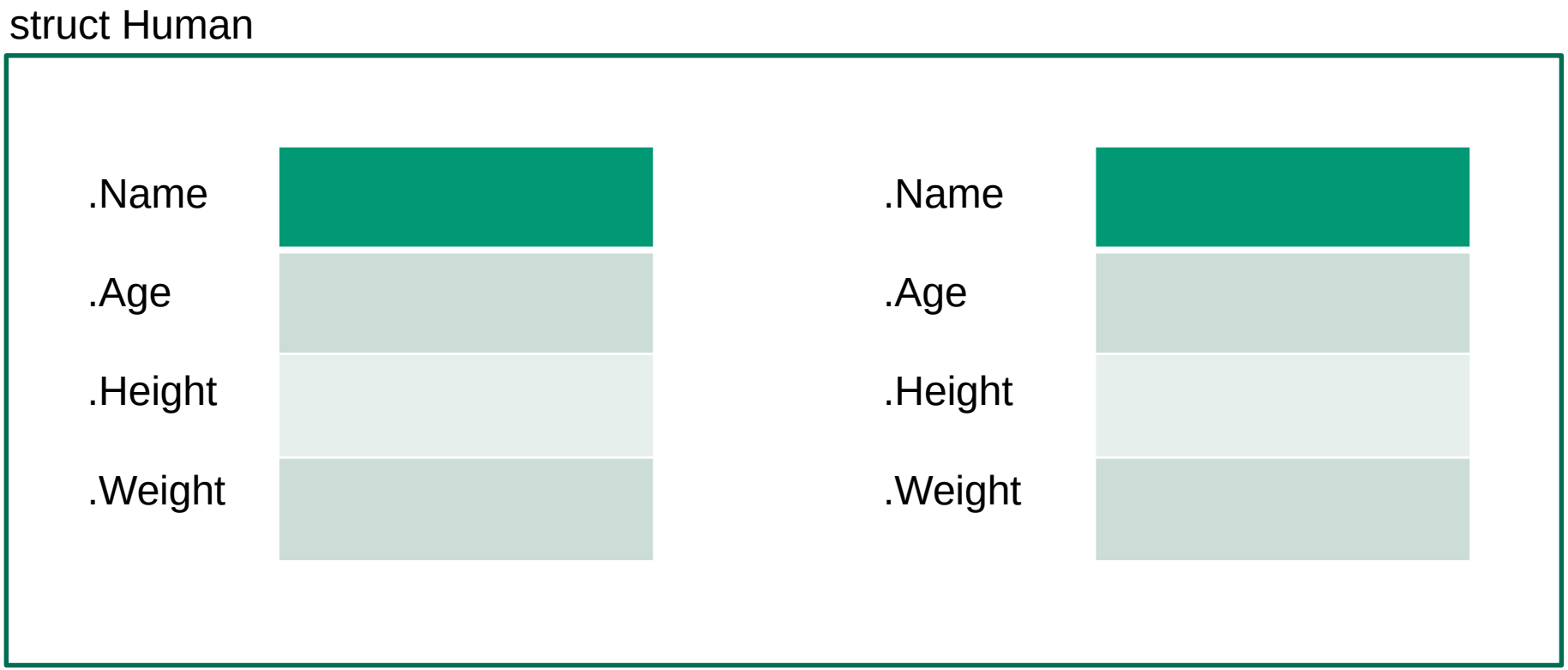
POINTERS & STRUCTURES

Fundamentals of System Programming Exercise's lecture note 9

Structures

- The structure mechanism provide a means to aggregate variables of different types.



Pointers and Structures

- Suppose the following structure:

C code excerpt:

```
struct person {
    char name[20];
    int age;  floa
    t weight;
};

struct person a;

...

printf("%s", a.name);
```

How to **print** the **name** field?

C code excerpt:

```
struct person *b;

...

printf("%s", (*b).name);

printf("%s", b->name);
```

How to **print** the **name** field?

The **same** as '**(*b).**'

Type Definition

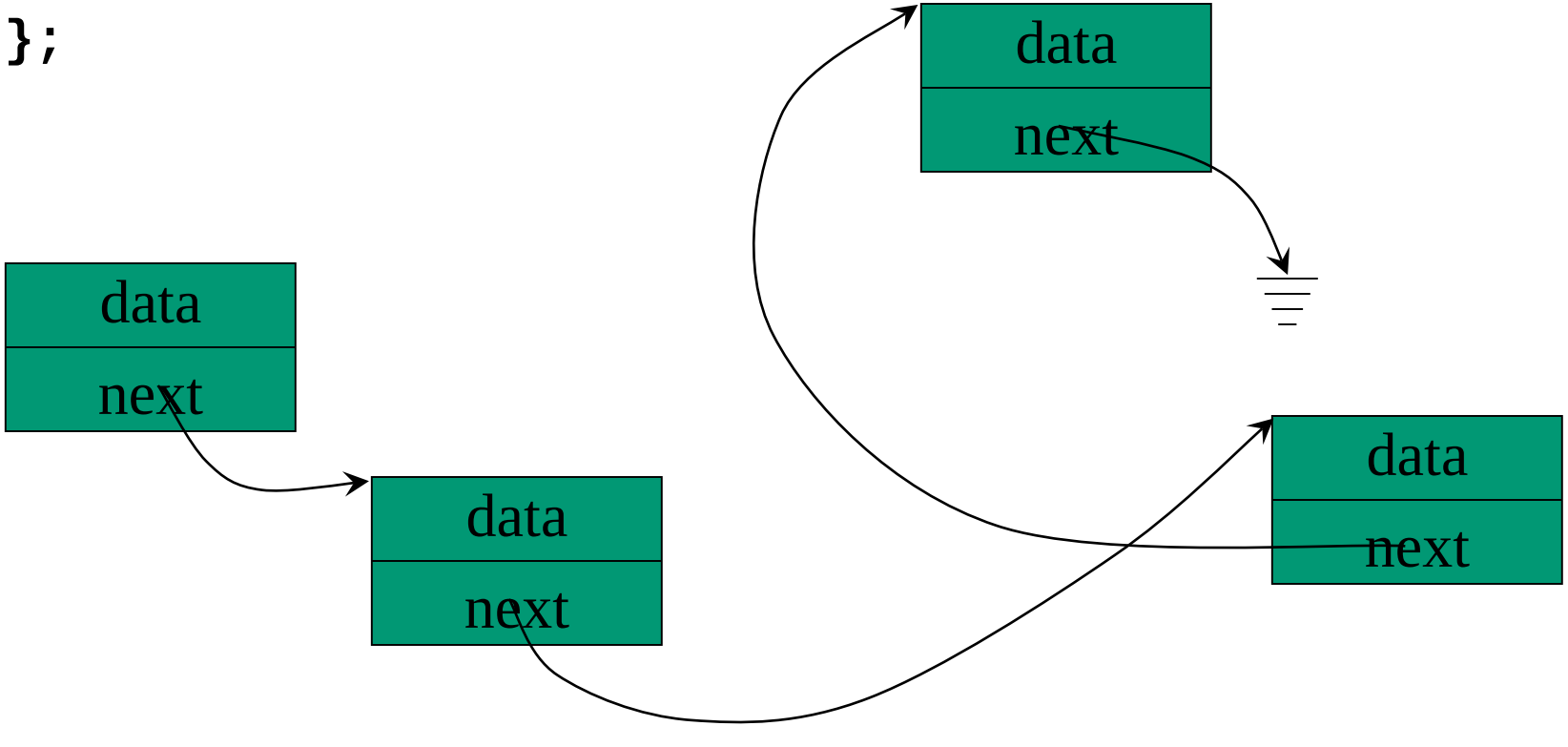
- C provides the ***typedef*** facility so that an identifier can be associated with a specific type.

Example)

```
typedef     char  uppercase;
uppercase   c, u[100];
```


Linked List

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

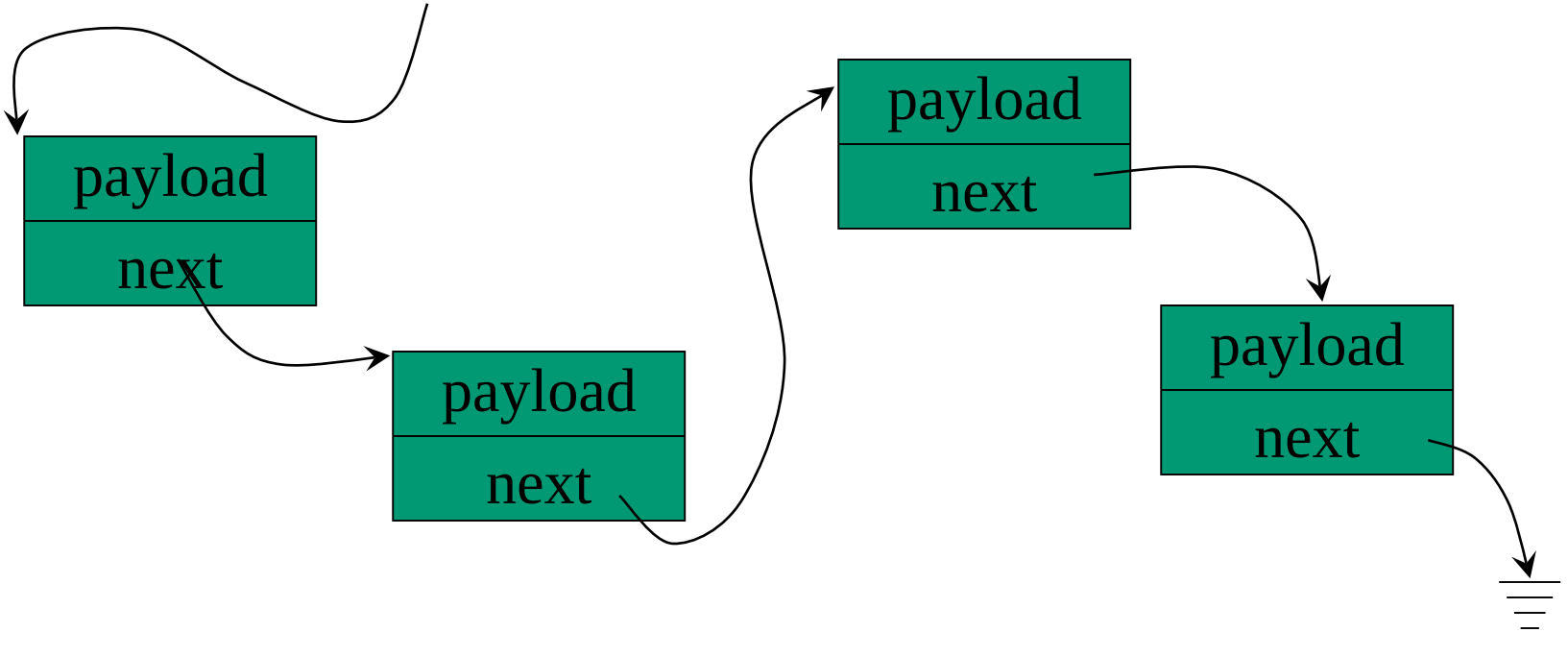


Linked List (Cont'd)

- node는 일반적으로 같은 타입을 갖게 된다.
  - 바로 앞 슬라이드를 보면 모든 노드들은 struct node라는 타입을 갖는걸 볼 수 있음
- 각 node는 항상 다른 node를 가리키고 있다.
- 마지막 node는 NULL을 가리킨다.
  - Linkded list의 끝은 NULL이다.
- 처음 우리가 linked list에 접근하기 위한 head라는 포인터가 필요하다.
- Data는 어떤 종류의 타입이 와도 상관없다.
  - Struct, int, float 등 어느 타입이 와도 상관없다.

Linked List (Cont'd)

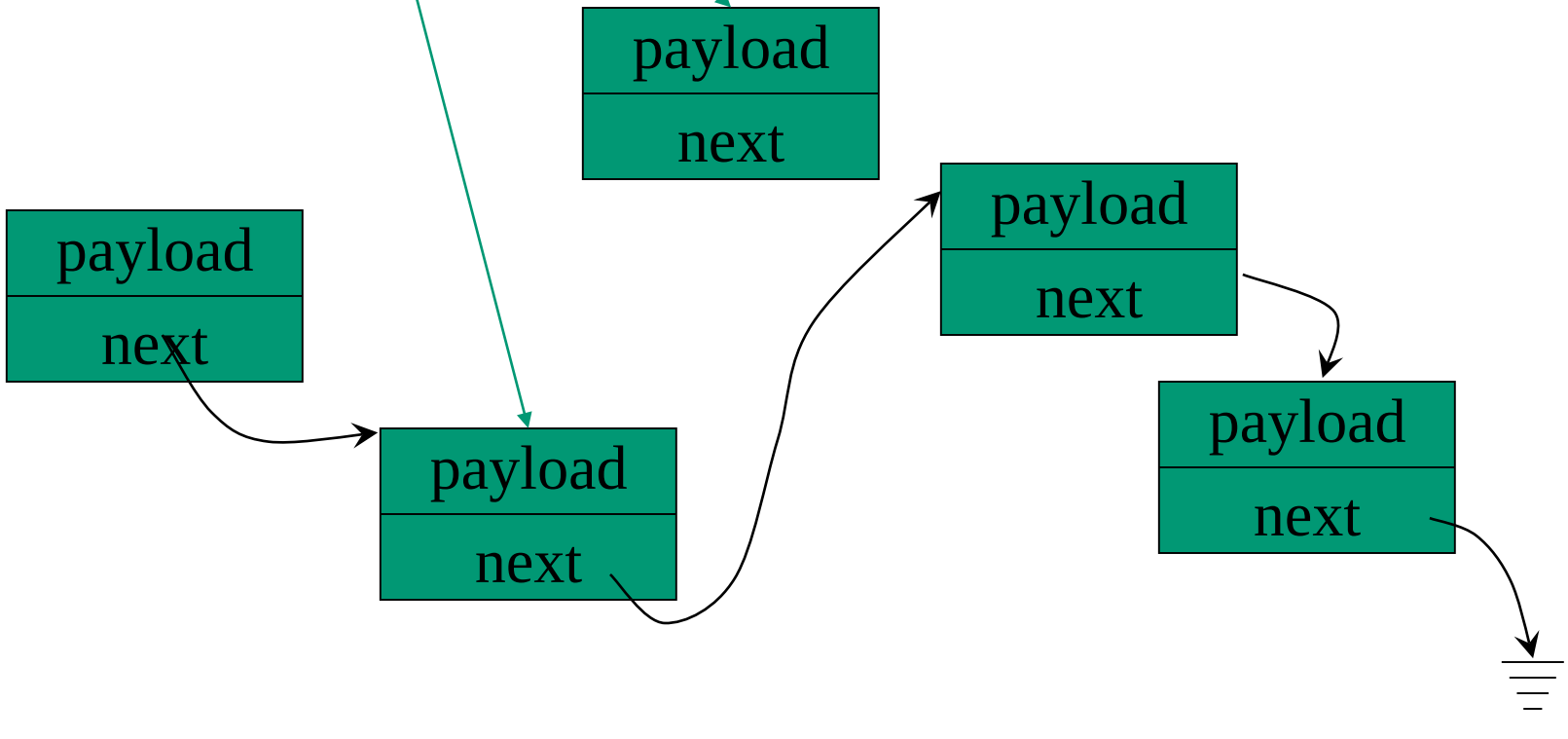
```
struct node {
  int data;
  struct node *next;
};
struct node *head;
```



Adding an Item to a List

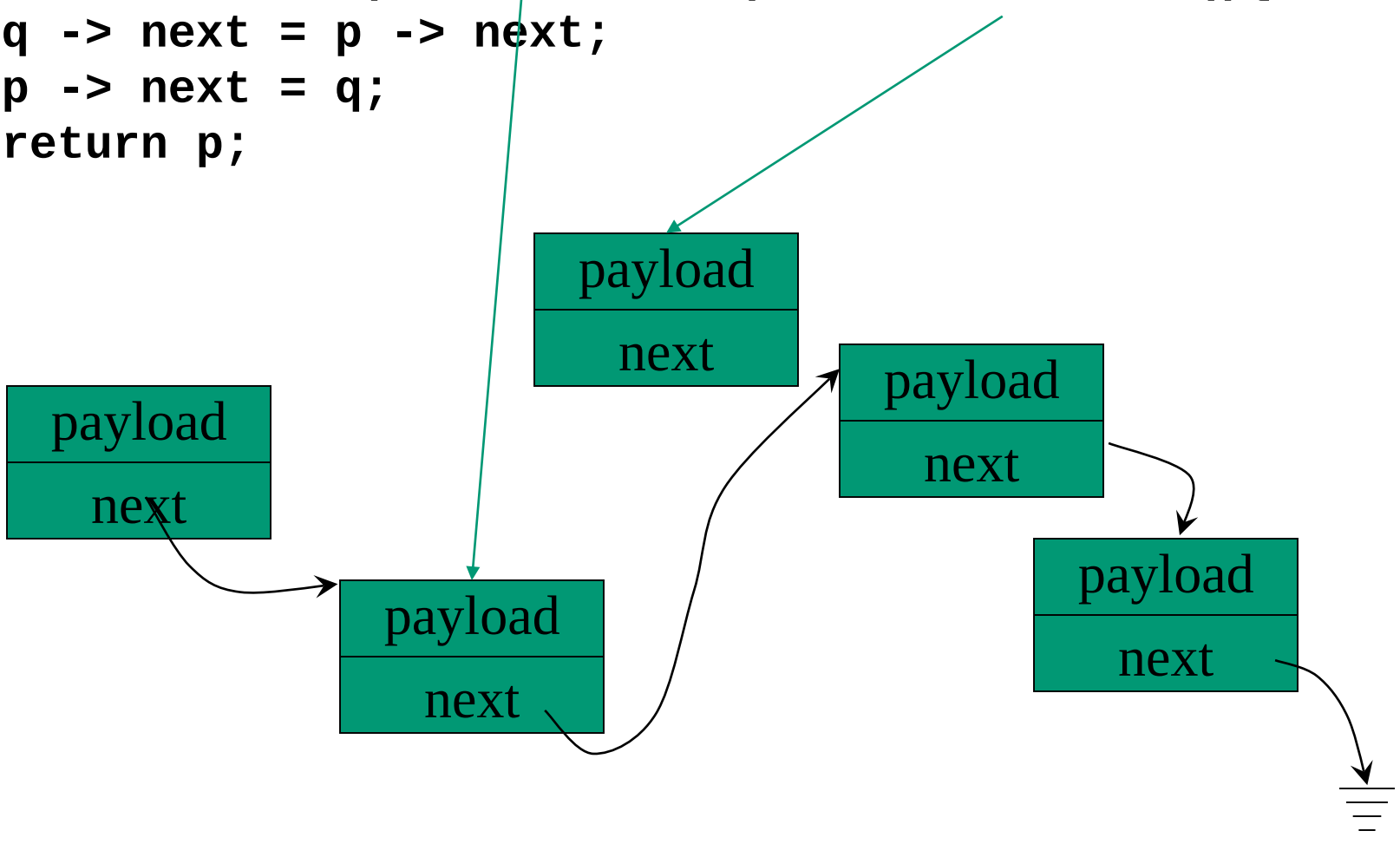
`struct node *p, *q;`

- 추가할 node를 포인터 q로 가리키고 있고 추가할 위치 바로전 node를 p로 가리키고 있는다.
- P와 q는 NULL이 되면 안된다.



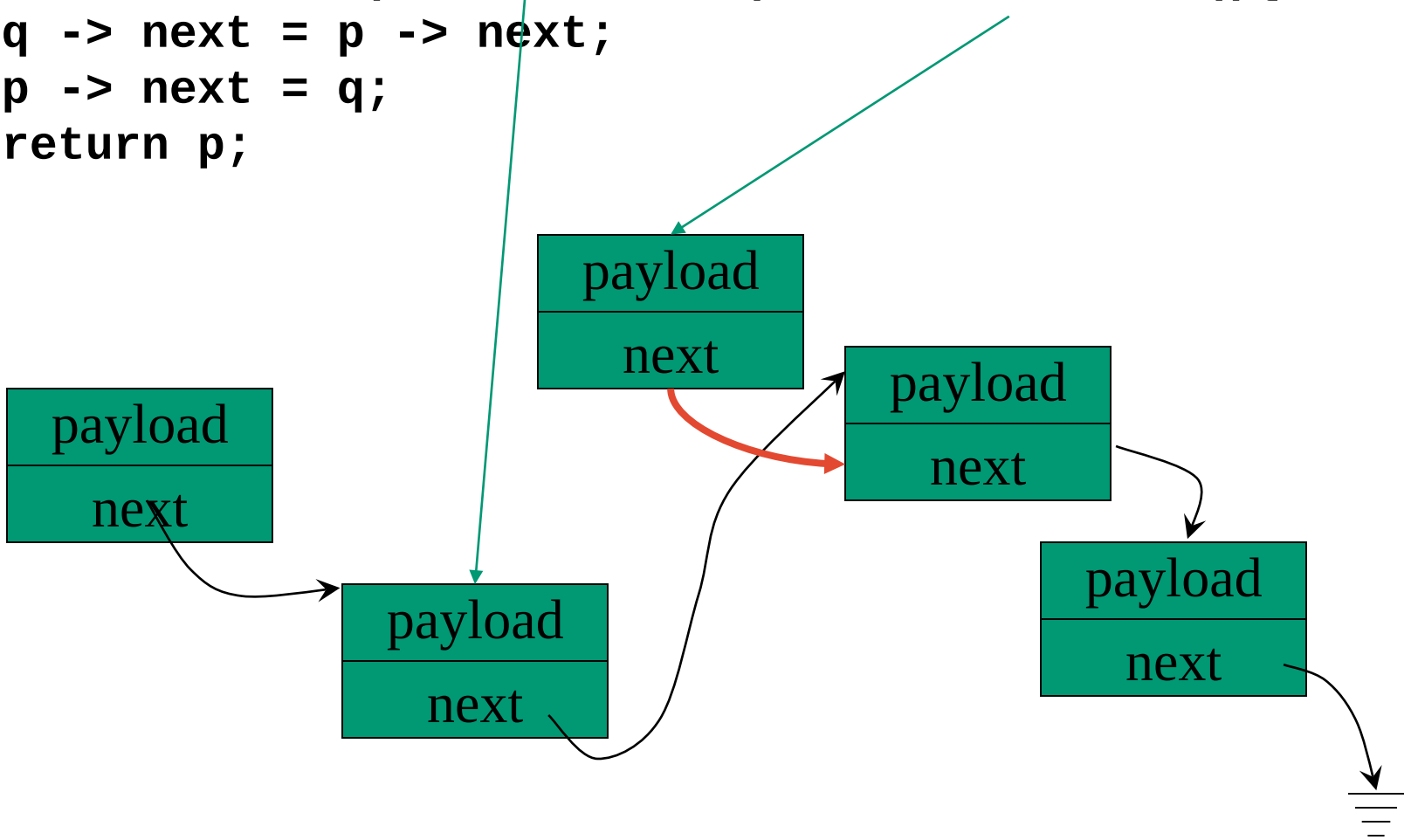
Adding an Item to a List

```
node *addAfter(struct node *p, struct node *q){
  q -> next = p -> next;
  p -> next = q;
  return p;
}
```



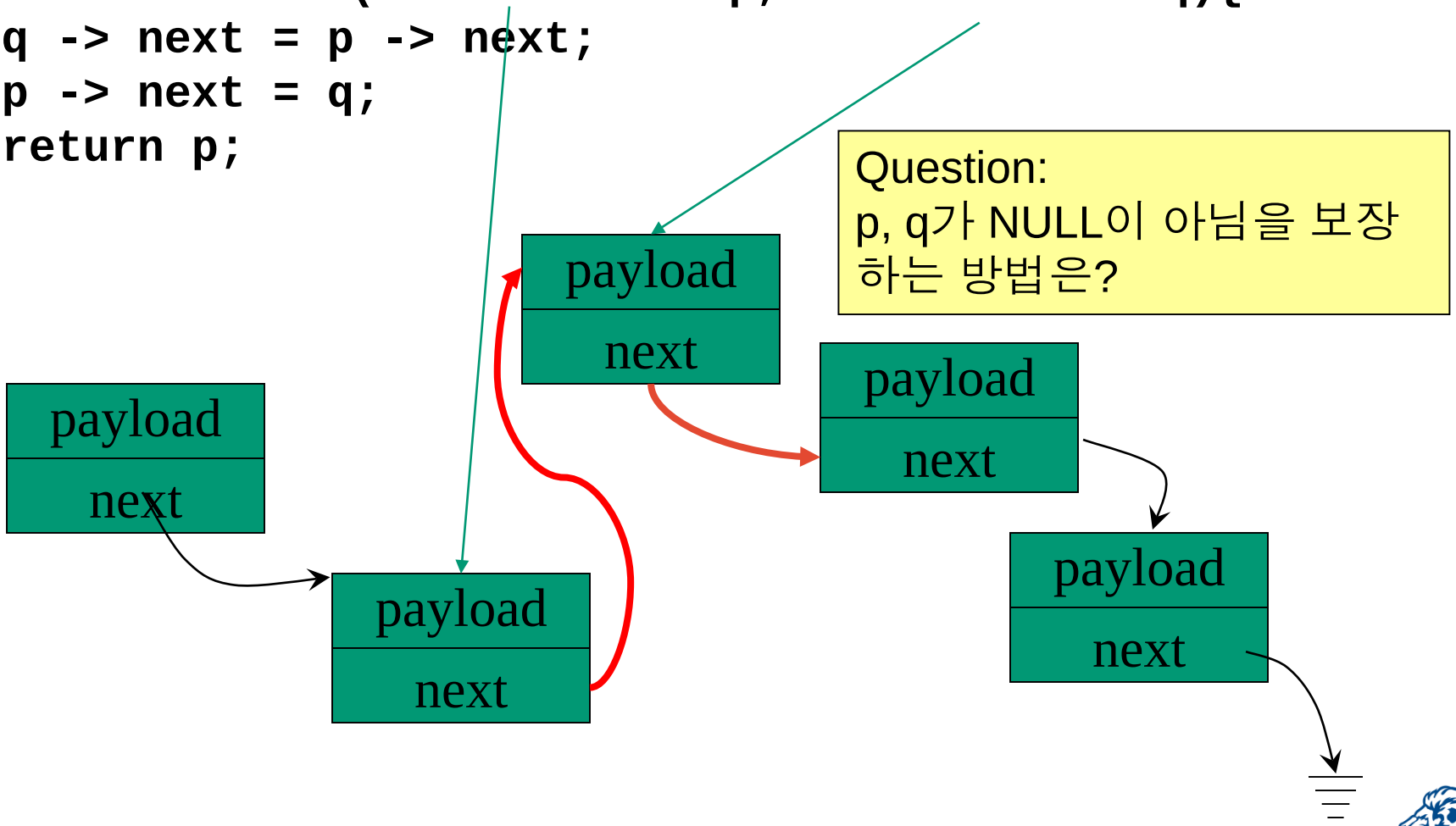
Adding an Item to a List

```
node *addAfter(struct node *p, struct node *q){
  q -> next = p -> next;
  p -> next = q;
  return p;
}
```



Adding an Item to a List

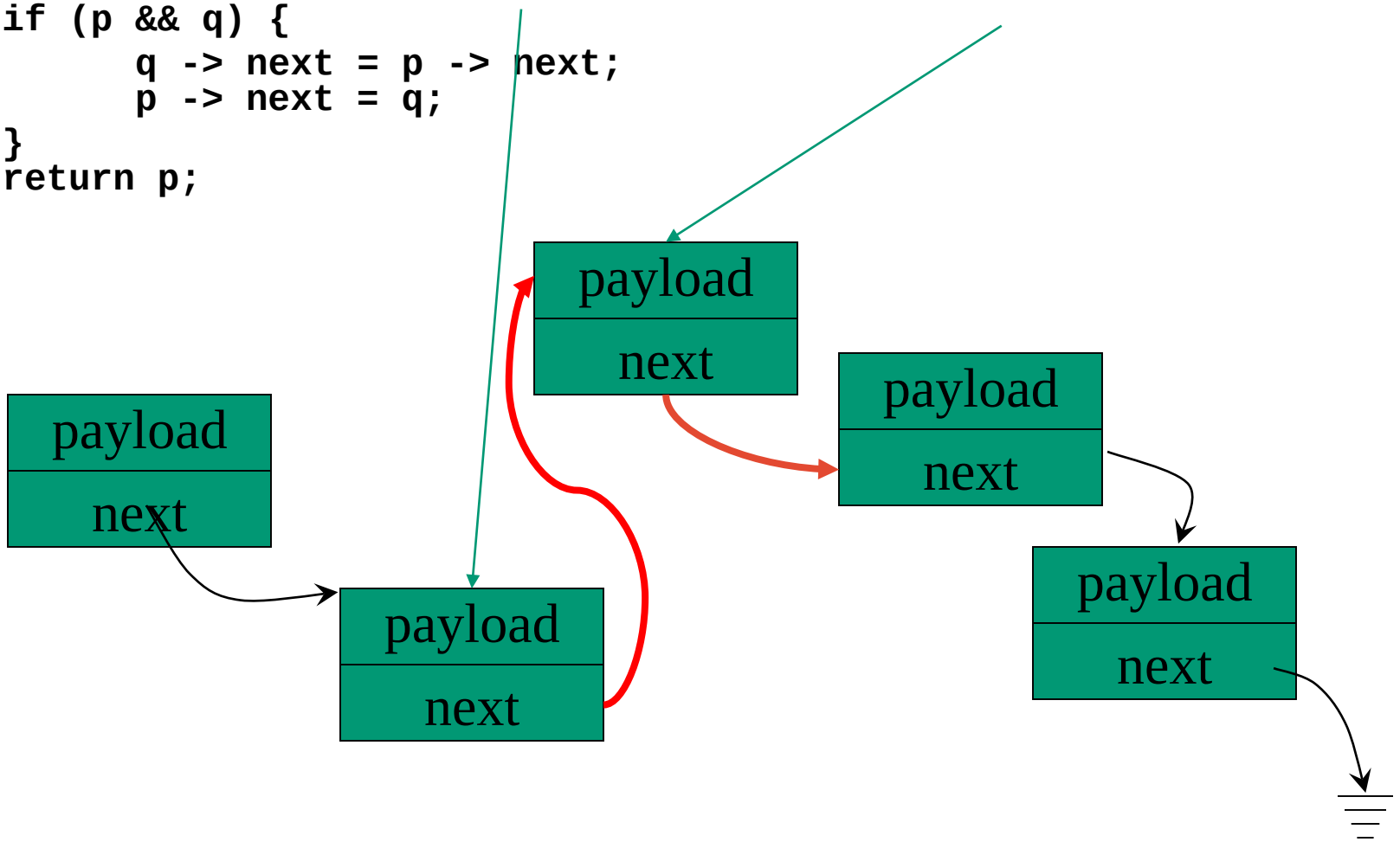
```
node *addAfter(struct node *p, struct node *q){
  q -> next = p -> next;
  p -> next = q;
  return p;
}
```



Question:
p, q가 NULL이 아님을 보장하는 방법은?

Adding an Item to a List

```
node *addAfter(struct node *p, struct node *q){
  if (p && q) {
        q -> next = p -> next;
        p -> next = q;
  }
  return p;
}
```



Exercise #1 : Linked List 구현

주어진 source code(linkedlist.c)를 채워 넣으시오.

- 위에서 살펴본 예제와는 달리 add_tail과 delete를 작성해야함
- add_tail 함수는 linked list의 끝에 node를 추가하는 함수임
- delete 함수는 linked list에서 인자로 받은 id의 노드를 삭제함

Exercise #1 : 출력 예제

```
[CPSLAB@ce jeong]$ gcc linkedlist.c -g
[CPSLAB@ce jeong]$ ./a.out
------------ Add id 1 -------------
HEAD --->   ID :   1 --->  TAIL(NULL)
          DATA :  30

------------ Add id 2 -------------
HEAD --->   ID :   1 --->   ID :   2 --->  TAIL(NULL)
          DATA :  30       DATA :  35

------------ Add id 4 -------------
HEAD --->   ID :   1 --->   ID :   2 --->   ID :   4 --->  TAIL(NULL)
          DATA :  30       DATA :  35       DATA :  45

------------ Add id 3 -------------
HEAD --->   ID :   1 --->   ID :   2 --->   ID :   4 --->   ID :   3 --->  TAIL(NULL)
          DATA :  30       DATA :  35       DATA :  45       DATA :  40

------------ Delete id 3 ----------
HEAD --->   ID :   1 --->   ID :   2 --->   ID :   4 --->  TAIL(NULL)
          DATA :  30       DATA :  35       DATA :  45

------------ Delete id 2 ----------
HEAD --->   ID :   1 --->   ID :   4 --->  TAIL(NULL)
          DATA :  30       DATA :  45

------------ Add Node after id 1 ----------
HEAD --->   ID :   1 --->   ID :   4 --->  TAIL(NULL)
          DATA :  30       DATA :  45
```

Exercise #2 : add_after 함수 구현

소스코드에 비워져 있는 `add_after` 함수를 구현하시오.

- `Id`를 찾지 못할경우 `NULL`을 반환

Exercise #2 : 출력 결과

```
[CPSLAB@ce jeong]$ gcc linkedlist.c -g
[CPSLAB@ce jeong]$ ./a.out
------------ Add id 1 -------------
HEAD --->   ID :   1 --->  TAIL(NULL)
          DATA :  30

------------ Add id 2 -------------
HEAD --->   ID :   1 --->   ID :   2 --->  TAIL(NULL)
          DATA :  30       DATA :  35

------------ Add id 4 -------------
HEAD --->   ID :   1 --->   ID :   2 --->   ID :   4 --->  TAIL(NULL)
          DATA :  30       DATA :  35       DATA :  45

------------ Add id 3 -------------
HEAD --->   ID :   1 --->   ID :   2 --->   ID :   4 --->   ID :   3 --->  TAIL(NULL)
          DATA :  30       DATA :  35       DATA :  45       DATA :  40

------------ Delete id 3 ----------
HEAD --->   ID :   1 --->   ID :   2 --->   ID :   4 --->  TAIL(NULL)
          DATA :  30       DATA :  35       DATA :  45

------------ Delete id 2 ----------
HEAD --->   ID :   1 --->   ID :   4 --->  TAIL(NULL)
          DATA :  30       DATA :  45

------------ Add Node after id 1 ----------
HEAD --->   ID :   1 --->   ID :   5 --->   ID :   4 --->  TAIL(NULL)
          DATA :  30       DATA :  55       DATA :  45
```