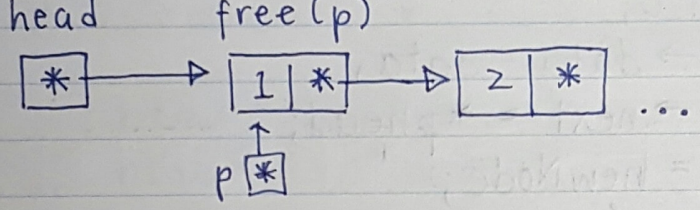
2510 : Lecture 28 || April 5/17

Destroying a List

- each node is dynamically-allocated; we need to free each node

Standard idiom to tranverse list:

```
for (p = head; p != ∅; p = p → next) {
    /* process p → data */   ← call free(p)?
```



- we can't access p → next after we call free(p)!
  SOLUTION: REMEMBER p → next before we call free(p)

Standard list to destroy a list:

```
void destroy(node * head) {
    node *p, *q;
    for (p = head; p != ∅; p = q) {
        q = p → next;
        free(p);
    }
}
```

note: this does not reset head.

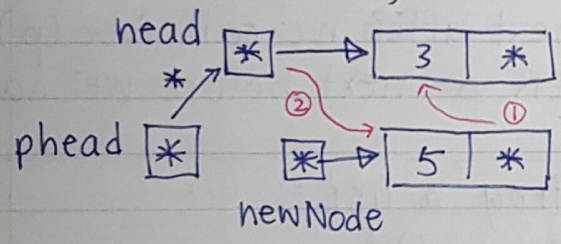
Hilroy

▷ Inserting data into a list

1) insert at the beginning
2) insert in sorted order

① inserting at the beginning

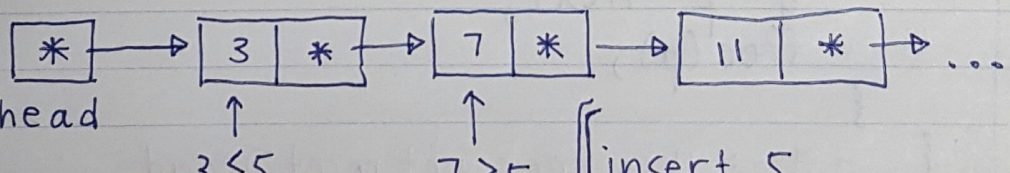
```
int insert_front (node ** phead, int data) {
    node * newNode = malloc (sizeof(node));
    if (newNode == 0)
        return 0;
    newNode → data = data;
    newNode → next = *phead;
    *phead = newNode;
    return 1; }
```



insert 5 at the beginning

② insert in sorted order

ex. insert in ascending order

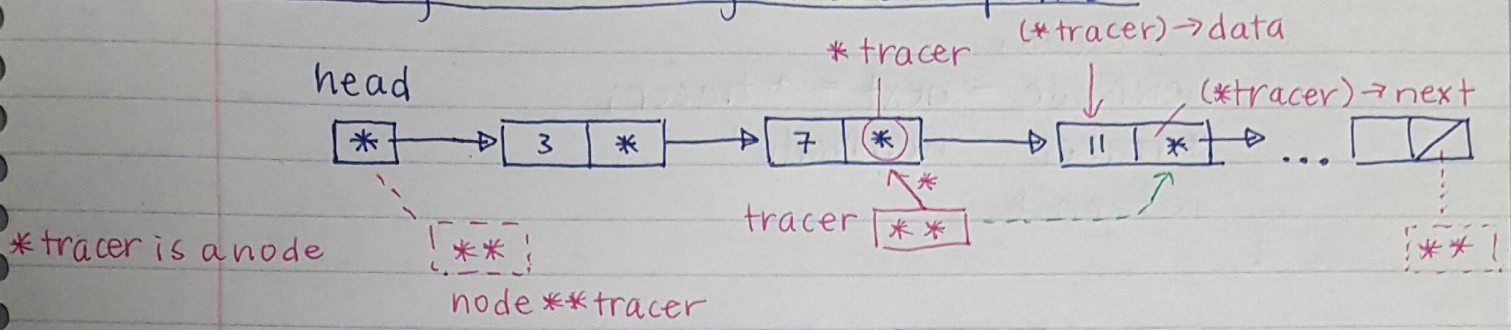


insert 5
- need to find correct location
- need to tranverse list + compare data.

2 SOLUTIONS:

① use a pair of pointers to tranverse the list
② use a double pointer

traversing a list using a double pointer




```
for (p tracer = &head; *tracer != ∅; tracer = &(*tracer)→next)
```

standard idiom to traverse a linked list (** version)

```
head - head pointer
node **tracer;
for (tracer = &head; *tracer != ∅; tracer = &(*tracer)→next){
  /* process (*tracer) → data */
```

Note: needed when we insert or delete data

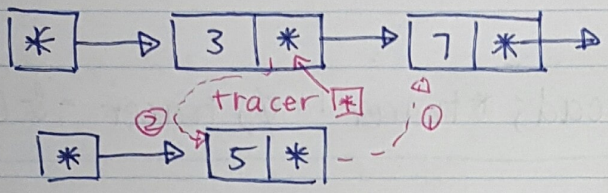
eg. insert in ascending order

```
int insert_asc (node **phead, int data) {
  node *newNode;
  node **tracer;
  newNode = malloc (sizeof (node));
  if (newNode == 0)
     return 0;
  newNode →data = data;
  for (tracer = phead; *tracer != ∅; tracer = &(*tracer)→next)
    if ((*tracer) →data >= data)
       break;
```

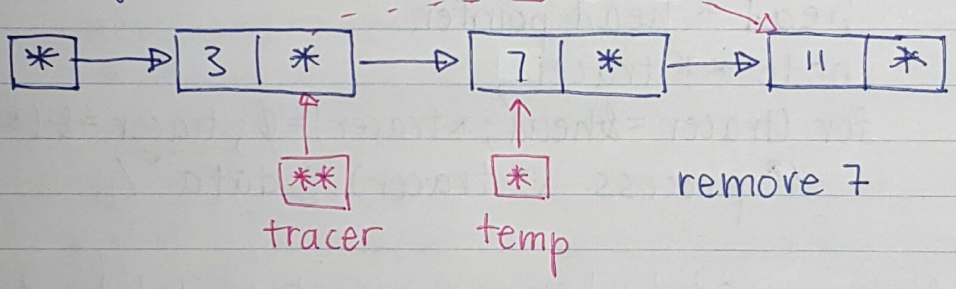
Hilroy

→ continued...

```
    if ((*tracer) -> data >= data)
        break;
    newNode -> next = *tracer;
    *tracer = newNode;
    return 1;
}
```



▷ Removing data from a list




```
tmp = *tracer;
*tracer = temp -> next;
free(tmp);
```