Final LAB Exam duration: **2hrs 45mins.**

On Saturday, 7th Nov @ 2 PM

B1-6 (Mon/Tue Lab Batch).

Report at **New Core Labs** before 2pm**.**

On Sunday, 8th Nov @ 10 AM

B7-12 (Wed/Thu Lab Batch).

PH Category (all sections).

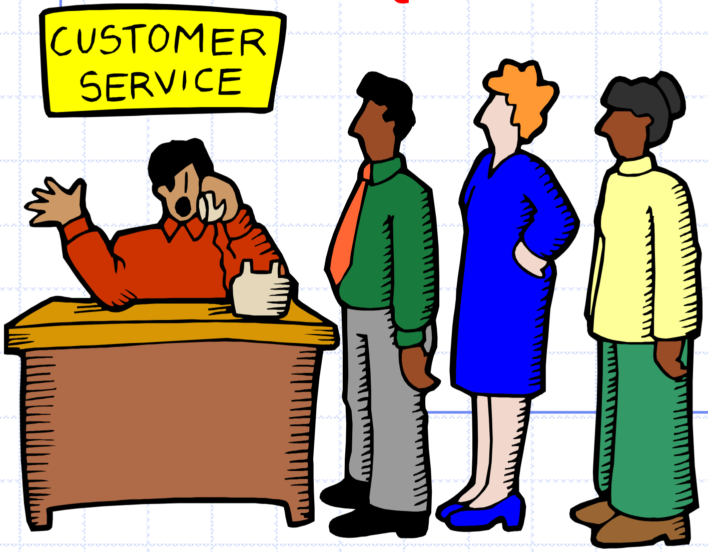
Report at **New Core Labs** before 10am.

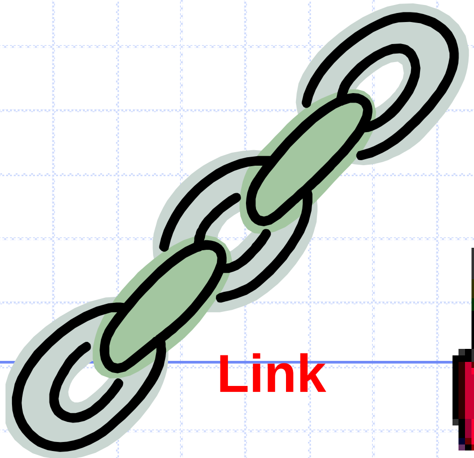
Syllabus: Everything covered till Friday, 6th Nov.

ESC101: Introduction to Computing

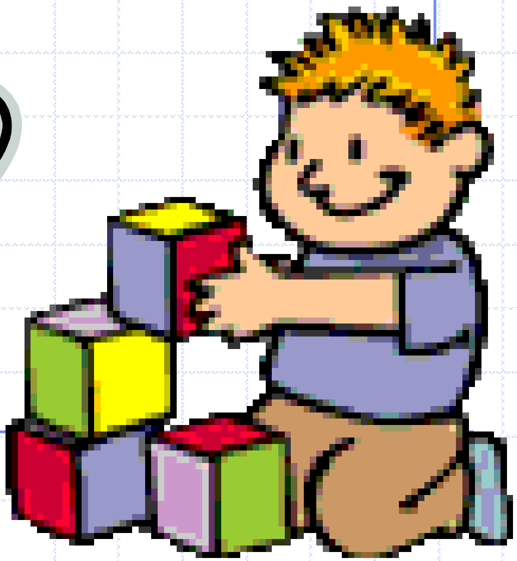
Data Structures

Queue





Link



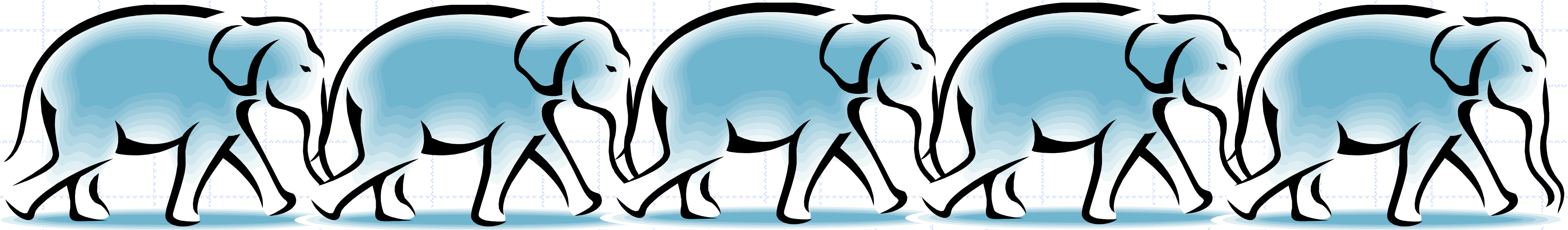
Stack

Data Structure

- What is a data structure?
- According to Wikipedia:
  - … a particular way of storing and organizing data in a computer so that it can be used efficiently…
  - … highly specialized to specific tasks.
- Examples: array, a dictionary, a set, etc.

Linked List

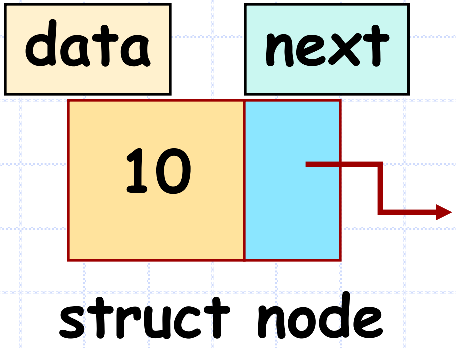
- A linear, dynamic data structure, consisting of nodes. Each node consists of two parts:
  - a “data" component, and
  - a “next" component, which is a pointer to the next node (the last node points to nothing).



Linked List : A Self-referential structure

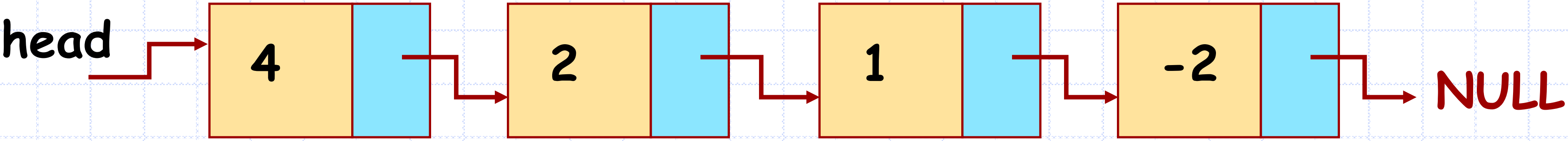
Example:

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



1. Defines the structure struct node, which will be used as a node in a "linked list" of nodes.
2. Note that the field next is of type struct node *
3. If next was of type struct node, it could not be permitted (recursive definition, of unknown or infinite size).

An example of a (singly) linked list structure is:

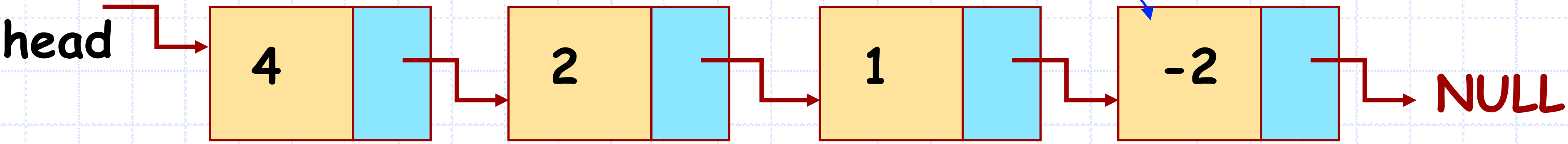


There is only one link (pointer) from each node, hence, it is also called "singly linked list".

Linked Lists

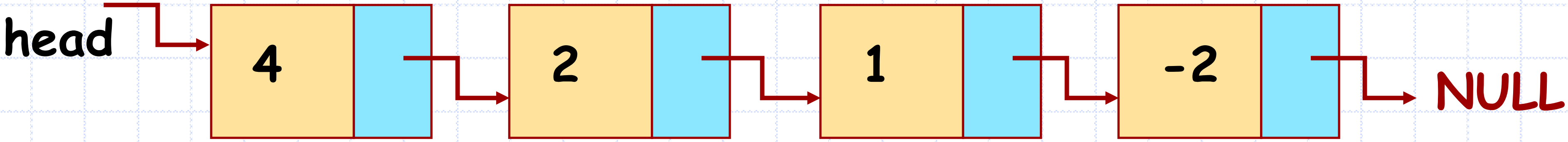
List starts at node pointed to by head

next field == NULL pointer indicates the last node of the list



1. The list is modeled by a variable called head that points to the first node of the list.
2. head == NULL implies empty list.
3. The next field of the last node is NULL.
4. Note that the name head is just a convention – it is possible to give any name to the pointer to first node, but head is used most often.

Displaying a Linked List

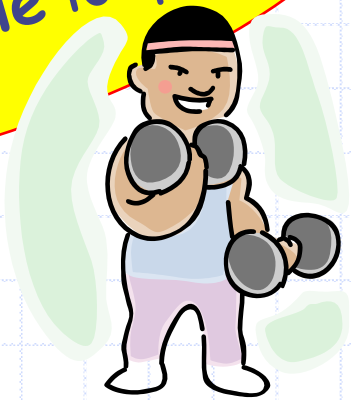


```
void display_list(struct node *head)
{
  struct node *cur = head;
  while (cur != NULL) {
    printf("%d ", cur->data);
    cur = cur->next;
  }
  printf("\n");
}
```

OUTPUT

```
4 2 1 -2
```

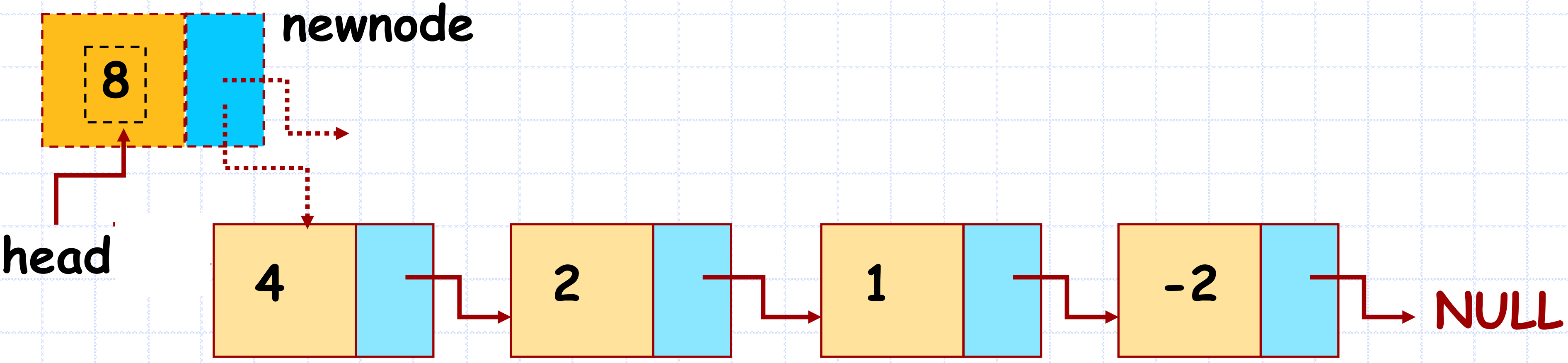
Exercise: Rewrite the code using **for** loop instead of **while** loop.



Insert at Front

Inserting at the front of the list.

1. Create a new node of type struct node. Set its data field to the value given.
2. "Add" it to the front of the list:
   Make its next pointer point to target of head.
3. Adjust head correctly to point to newnode.

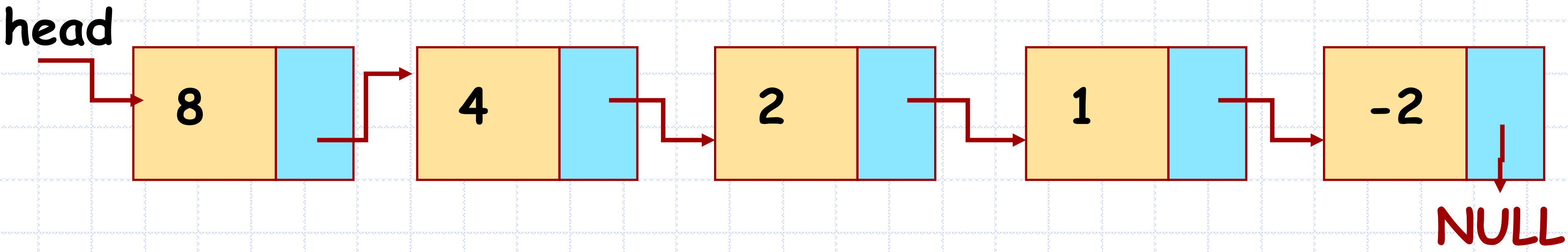


```
struct node * make_node(int val) {
    struct node *nd;
    nd = (struct node *)
        calloc(1, sizeof(struct node));
    nd->data = val;
    return nd;
}
```

```
/* Allocates new node
pointer and sets the
data field to val,
next field initialized
to NULL */
```

```
struct node *insert_front(int val, struct node *head) {
   struct node *newnode= make_node(val);
   newnode->next = head;
   head = newnode;
   return head;
}
```

```
/* Inserts a node with data field val at the head
    of the list currently pointed to by head.
   Returns pointer to the head of new list.
   Works even when the original list is empty,
     i.e. head == NULL */
```



Suppose we want to start with an empty list and insert in sequence -2, 1,2, 4 and 8, as provided by user. The following code gives an example. Final list should be as above.

```
struct node *head = NULL;
int val; scanf ("%d", &val);
while (val != -1) {
   insert_front (val, head);
   scanf ("%d", &val);
}
```

INPUT: -2 1 2 4 8 -1

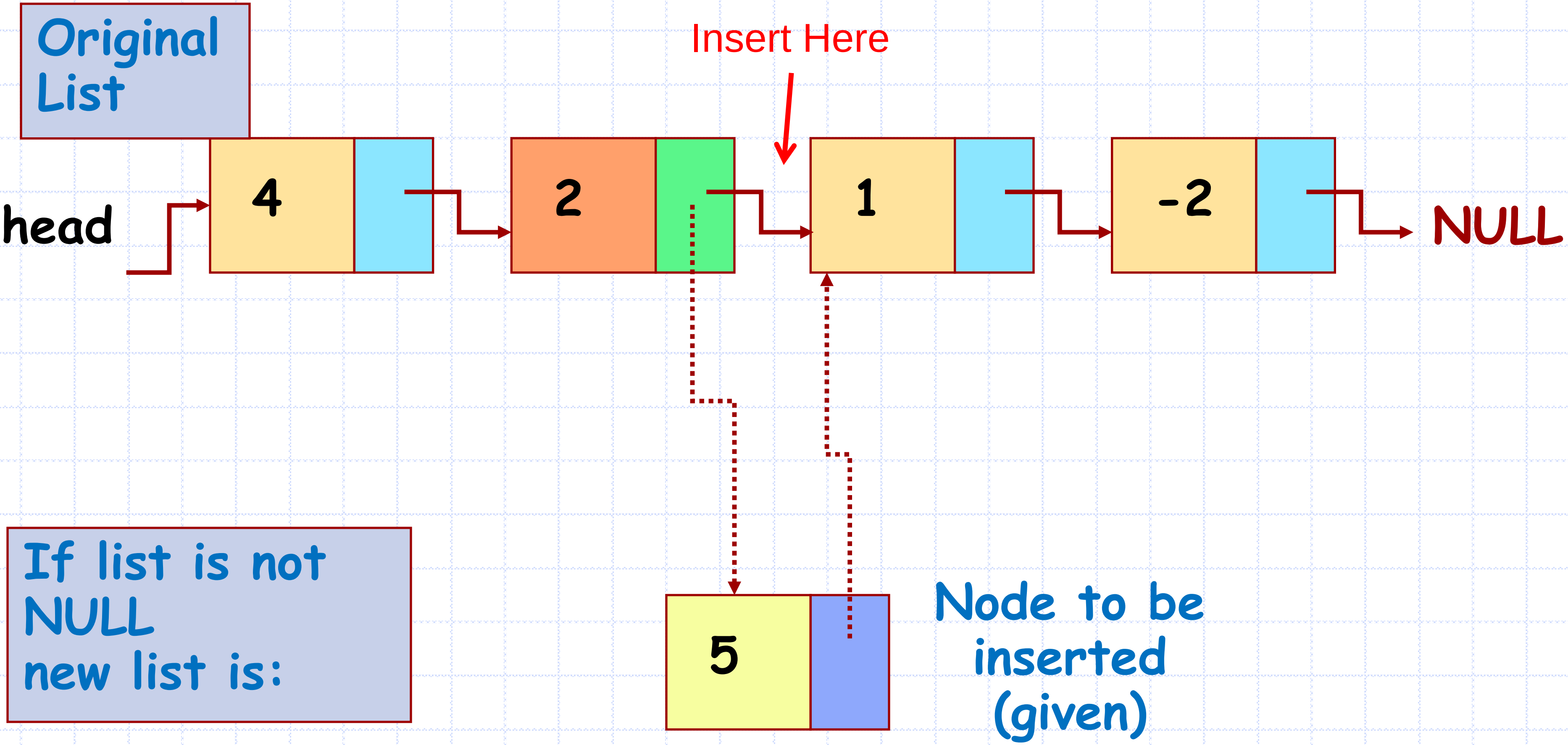
This creates the list in the reverse order of input: head points to the last element inserted.
How to create list in the same order as input?

Generic Insertion in linked list

List Insertion

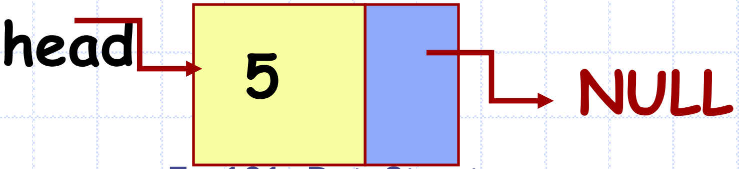
Given a node, insert it after a specified node in the linked list.

Original List

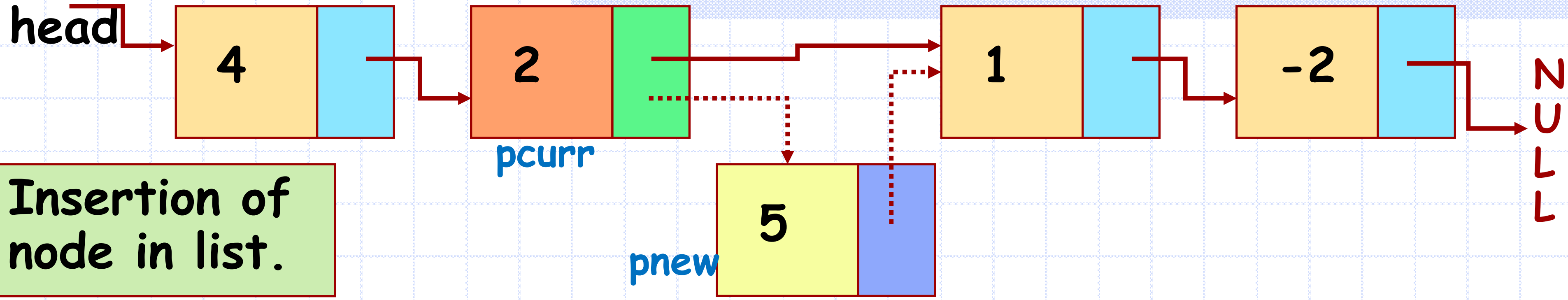


If list is not NULL new list is:

If list is NULL new list is:



Insertion of node in list.



Given **pcurr**: Pointer to node after which insertion is to be made
pnew: Pointer to new node to be inserted.

```
struct node *insert_after_node (struct node *pcurr,
                                    struct node *pnew) {
    if (pcurr != NULL) {
        // Order of next two stmts is important
        pnew->next = pcurr->next;
        pcurr->next = pnew;
       return pcurr; // return the prev node
    }
    else return pnew; // return the new node itself
}
```

Use of typedef

- Repetitive to keep writing the type struct node for parameters, variables etc.
- C allows naming types— the typedef statement.

Defines a new type Listnode as struct node *

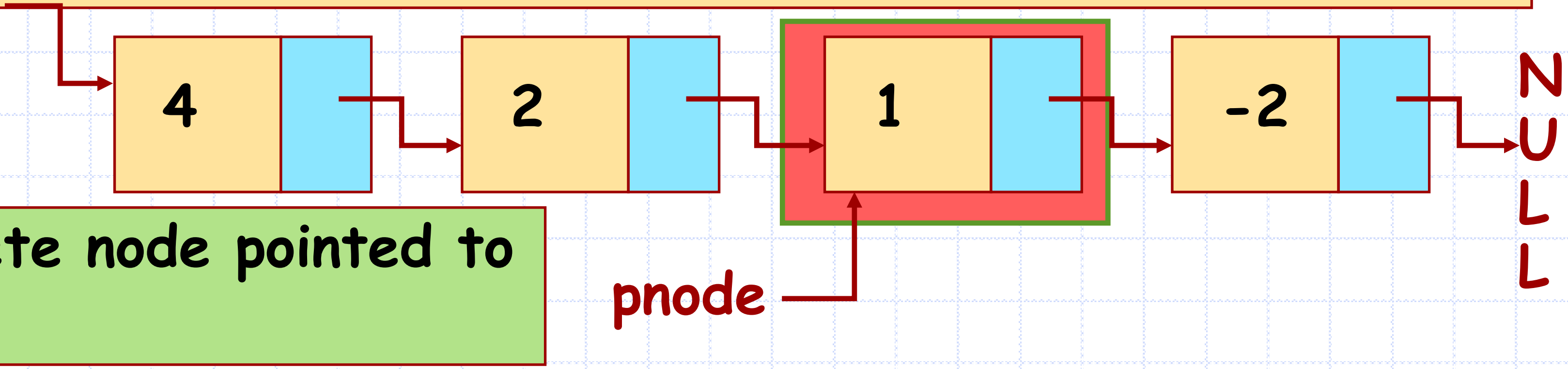
```
typedef struct node * Listnode;
```

Listnode is a type. It can now be used in place of struct node * for variables, parameters, etc..

```
Listnode head, curr;
 /* search in list for key */
Listnode search(Listnode list, int key);
/* insert the listnode n in front of listnode list */
Listnode insert_front(Listnode list, Listnode n);
 /* insert the listnode n after the listnode curr */
Listnode insert_after(Listnode curr, Listnode n);
```

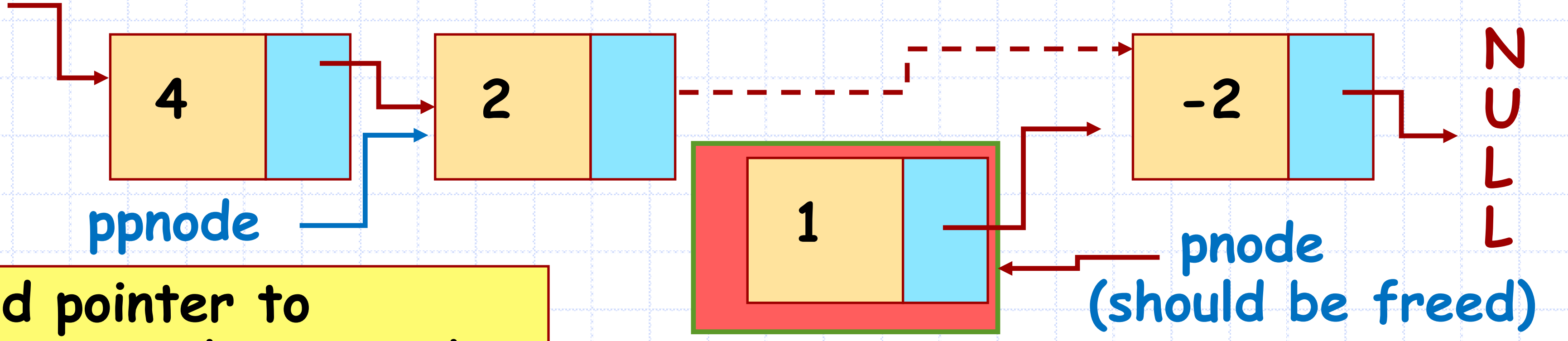
Deletion in linked list

Given a pointer to a node pnode that has to be deleted. Can we delete the node?



E.g, delete node pointed to by pnode

After deletion, we want the following state



Need pointer to previous node to pnode to adjust pointers.

call free() to release storage for deleted node.

prototype

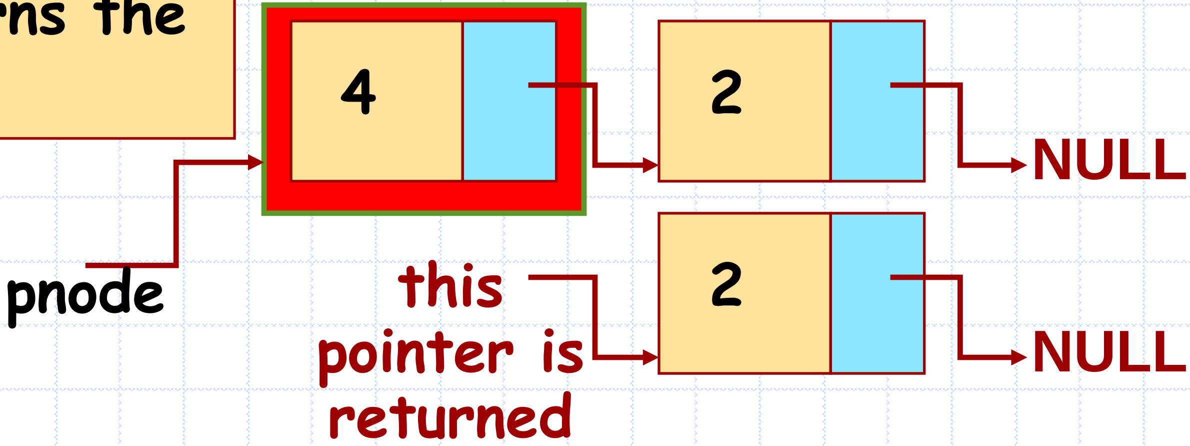
delete(Listnode pnode, Listnode ppnode)

```
Listnode delete(Listnode pnode, Listnode ppnode)
{
    Listnode t;
    if (ppnode)
        ppnode->next = pnode->next;
    t = ppnode ? ppnode : pnode->next;
    free (pnode);
    return t;
}
```

Delete the node pointed to by pnode. ppnode is pointer to the node previous to pnode in the list, if such a node exists, otherwise it is NULL.

Function returns ppnode if it is non-null, else returns the successor of pnode.

The case when pnode is the head of a list. Then ppnode == NULL.


4
2
NULL
pnode
this pointer is returned
2
NULL