# CS 32: Discussion 1D

TA: Shichang Zhang
LA: Stephanie Doan, Rish Jain

# About Us

**Discussion:**

- Discussion 1D: 2:00 - 3:50 PM PST, Friday
- Discussion will be recorded and uploaded on CCLE

**Office Hours:**

- Shichang: Wednesday 3:30-5:30PM PST, Thursday 10:30-12:30PM PST
- Rish: Tuesdays 2:30-3:30PM PST
- Stephanie: Thursdays 10:30AM-12:30PM PST

**Email:**

- Shichang: shichang@cs.ucla.edu
- Rish: rishabjain@g.ucla.edu
- Stephanie: stephaniekdoan@gmail.com

**Course Website:**

- http://web.cs.ucla.edu/classes/spring21/cs32/

# Announcements

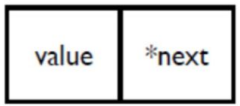
- Project 2 due Tuesday (April 20th)
- Workshop on linked lists happened on Wednesday April 14 -- watch recording here

# Overview

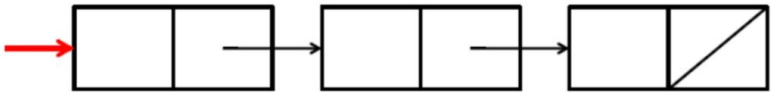
- Linked list
  - Insertion
  - Search
  - Removal
- Sorted linked list
- Circular linked list
- Doubly linked list

# Linked List

- Minimum Requirement
  - Key component as unit: Node (with `value` and `pointer` to next node)
  - Head pointer → points to the first term
  - Loop-free (except in some special case: circular listed list)
- Regular operations
  - Insertion
  - Search
  - Removal
- Pros and cons
  - Efficient insertion, flexible memory allocation, simple implementation
  - High complexity of search

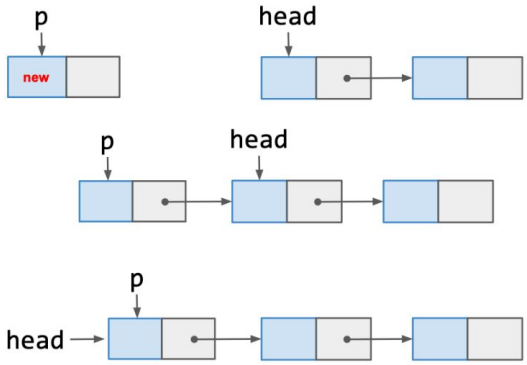



```
typedef int ItemType;
Struct Node
{
      ItemType value;
      Node *next;
};
```



# Linked List: Insertion

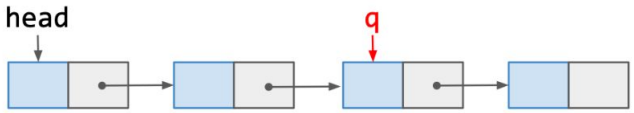
- Example: Insert as head in a list
- Steps
  a) Create a new node and call the pointer p
  b) Make its `next` pointer point to the first item
  c) Make the head pointer to the new node




```
//Skeleton: Linked list insertion
//====================================

//insert as head
p->next = head;
Head = p;

//insert after end: End node: q
q->next = p;
p->next = nullptr;

//insert in the middle: node q
p->next = q->next;
q->next = p;
```

# Linked List: Search

- Steps
  a) Find matched node and return
  b) If no match, return NULL



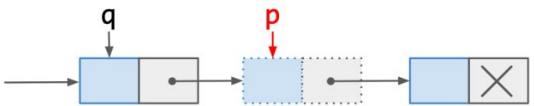

```
// Skeleton Code: Linked list search
// ==================================


Node* Search(int key, Node* head){
  Node *q = head;
  while(q != NULL)
  {
    if(q -> value != key) q = q -> next;
    else return q;
  }
  return NULL;
}
```

# Linked List: Removal

- Remember to set the previous node q's `next` pointer to point the next node of p

```
q->next = p->next;
delete p
```

- What if `p == head`? What if `p` prints to the last node in the linked list?



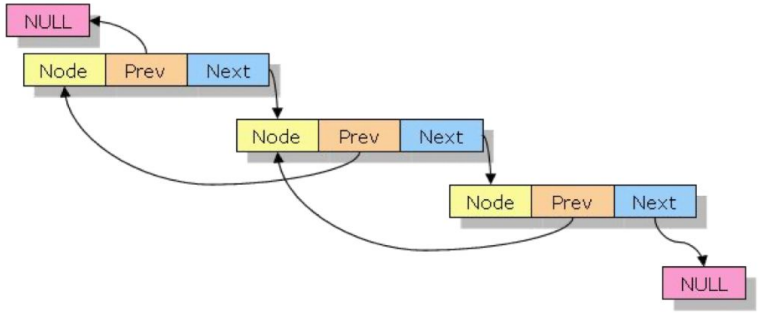

```
// Skeleton Code: Linked list removal
// ====================================

void remove(int valToRemove, Node* head) {
      Node *p = head, *q = NULL;
      while (p != NULL) {
            if (p->value == valToRemove)
                  break;
            q = p;
            p = p->next;}
      if (p == NULL) return;
      if (p == head) //special case
            head = p->next;
      else
            q->next = p->next;
      delete p;
}
```

# Linked List: Summary

- Pros:
  - Efficient insertion (add new data items)
  - Flexible memory allocation
- Cons:
  - Slow search (search is more important than insertion and removal in real situations)

- Many variations
  - Doubly  linked lists
  - Sorted linked lists
  - Circularly linked lists

# Doubly Linked List

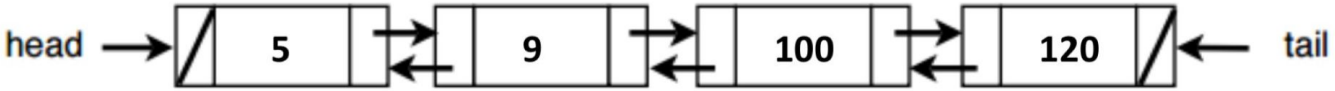



- A linked list where each node has two pointers:
  - Next – pointing to the next node
  - Prev – pointing to the previous node
- Features
  - head, tail pointers
  - `head->prev = NULL; tail->next = NLL;`
  - `head == tail == NULL` when doubly linked list is empty

```
typedef int ItemType;
Struct Node
{
      ItemType value;
      Node *next;
      Node *prev;
};
```

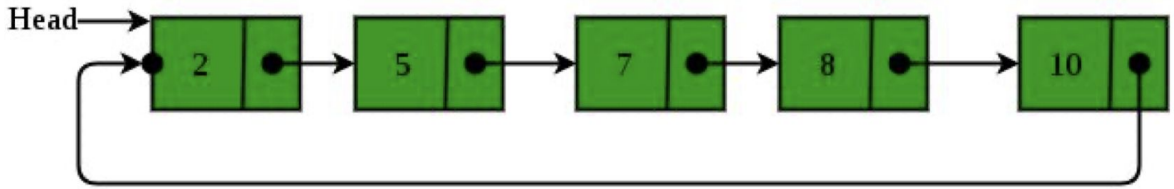
# Sorted Linked List

- Do we need to search the entire linked list?
- What if we store all values in an ascending sorted (or descending order)?




- How do you change insertion function? → Worksheet Q6
  - Find the node **q** whose value is the greatest lower bound to the new node **p**.
- How do you change search function?
  - Early stop when we see a node which stores a value that is larger than key for ascending sorted linked list.
- How do you update removal functions?

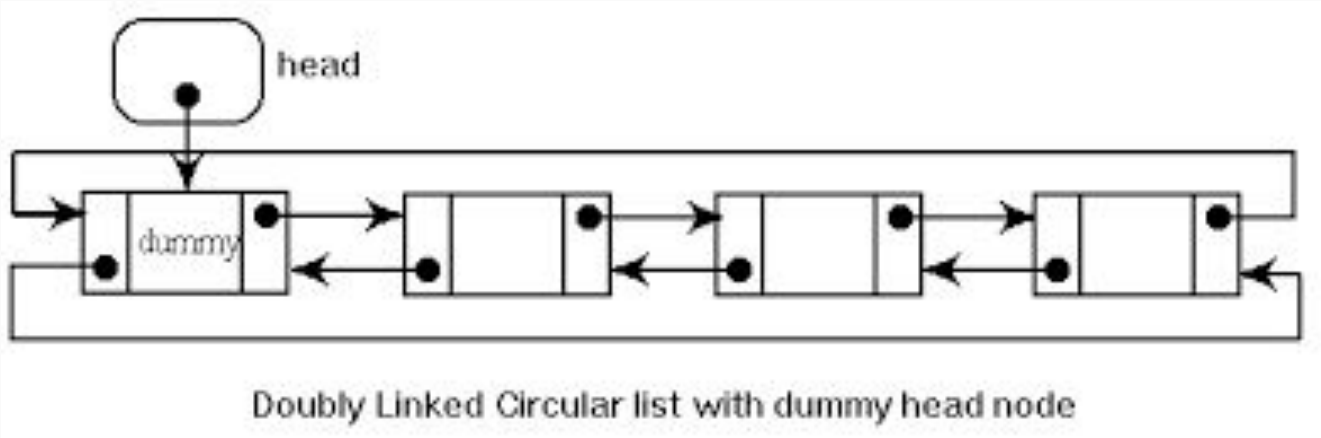
# Circular Linked List




- Linked list where all nodes are connected to form a circle.
  - There is no NULL at the end.
  - Can be a singly circular linked list or doubly circular linked list.
- Pros:
  - Any points can be head (starting point).
  - Implementation for queue
  - Fit to repeatedly go around the list.
- It is also very tricky though.

# Project 2 Hint

- Make a dummy node to make sure your linked list is always non-empty.




Doubly Linked Circular list with dummy head node

# Break: 5 mins

# Worksheet

# Codeshare

Room 1

Room 2

Room 3

Room 4

Room 5

Room 6

# Worksheet Solution