UCSC Silicon Valley Extension

Advanced C Programming

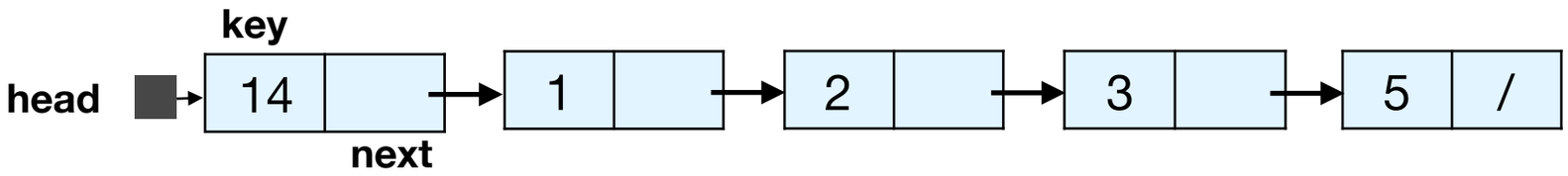
Linked Lists

Instructor: Radhika Grover

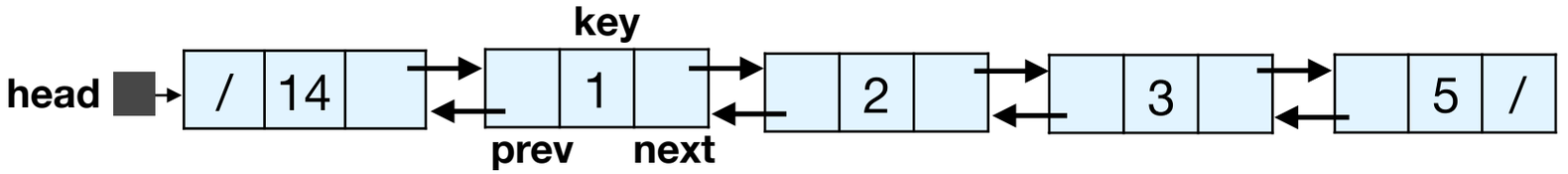
Overview

- Linked List
  - insert, search and delete operations
  - Using a sentinel
  - Implementation and applications

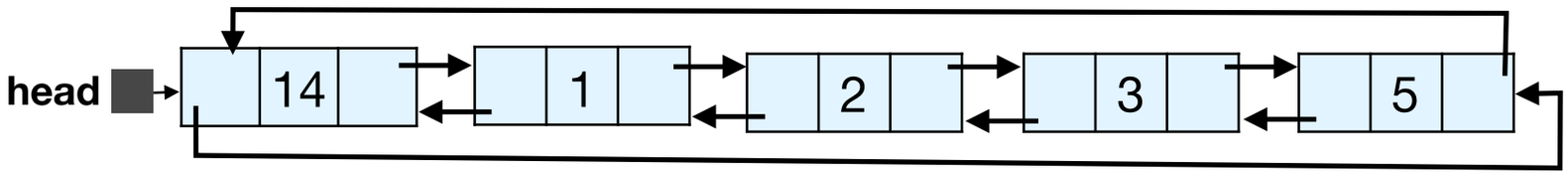
Linked list example



Singly linked list L



Doubly linked list L



Circular doubly linked list L

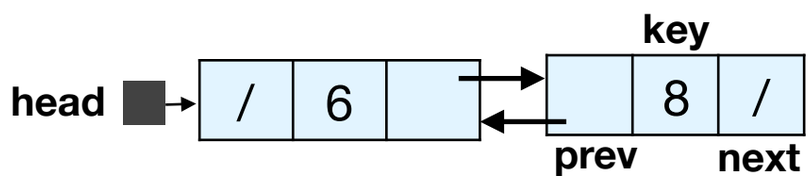
Linked list search

```c
// DoublyLinkedList/linkedlist.c
struct Node {
    struct Data data;
    struct Node *prev;
    struct Node *next;
};

// search for Node with given key and return a pointer to it
struct Node* listSearch(struct Node *head, int key) {
    struct Node *ptr = head;

    while (ptr != NULL && ptr->data.key != key) {
        ptr = ptr->next;
    }
    return ptr;
}
```

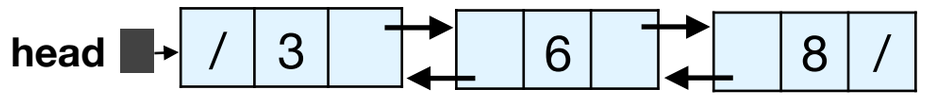
Time complexity = O(n)

Linked list insert

```
// insert a new node with given key at front of list
void listInsert(struct Node **head, int key) {
    struct Node *newNode = (struct Node *)
calloc(sizeof(struct Node), 1);
    if (newNode == NULL) {
        printf("Error: memory could not be allocated");
        exit (-1);
    } else {
        newNode->next = (*head);
        newNode->prev = NULL;
        newNode->data.key = key;

        if ((*head) != NULL)
            (*head)->prev = newNode;

        (*head) = newNode;

    }
}


Time complexity = O(1)
```



Doubly linked list L

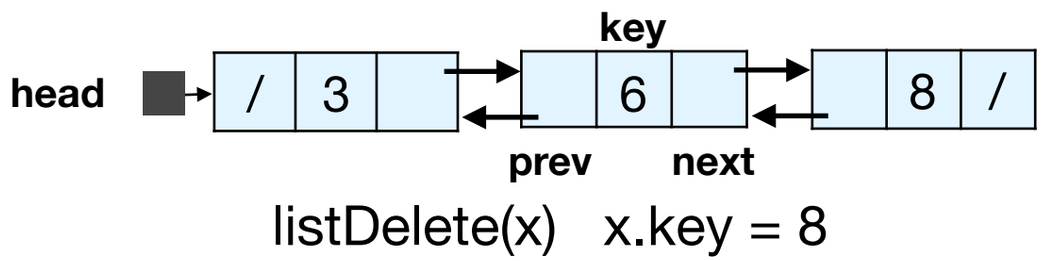


listInsert(x) x.key = 3

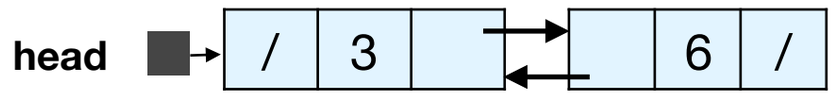
Linked list delete

```
// delete the given node x
void listDelete(struct Node **head, struct Node *x)
{
    // x is the first node
    if (x->prev == NULL)
        (*head) = x->next;
    // x is the last node
    else if (x->next == NULL)
        x->prev->next = NULL;
    // x is in the middle
    else {
        x->prev->next = x->next;
        x->next->prev = x->prev;
    }

    free(x);
    x = NULL;
}
```

Time complexity = O(1)



listDelete(x) x.key = 8



Doubly linked list L

Sentinel

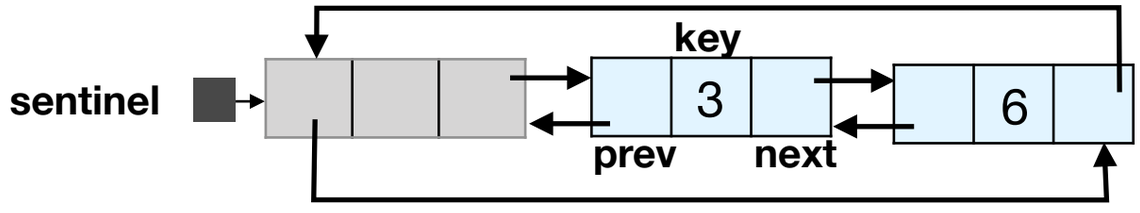
- Node with no data for simplifying the boundary condition checks in doubly linked lists.

- Add sentinel to start and end of list - removes the need to check position of x (first and last positions) before deletion in listDelete.

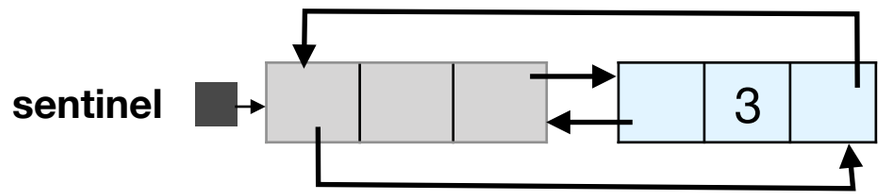
- Does not typically improve the asymptotic running time.

Linked list with sentinel - pseudocode for delete

```
listDelete(Node *x){
   x->prev->next = x->next;
   x->next->prev = x->prev;
}

Time complexity = O(1)
```

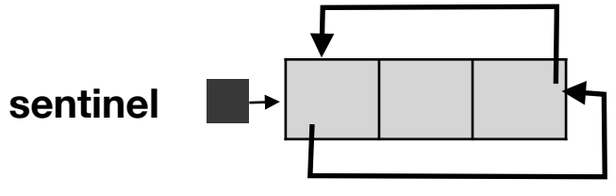


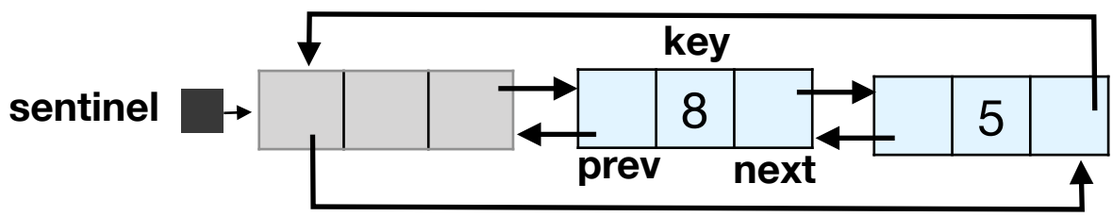
listDelete'(x) x.key = 6



Linked list with sentinel - pseudocode for search

```
listSearch(int k){
    x =  sentinel->next;
    while(x!=sentinel && x->key!=k)
        x = x->next;

    return x;
}

Time complexity = O(n)
```

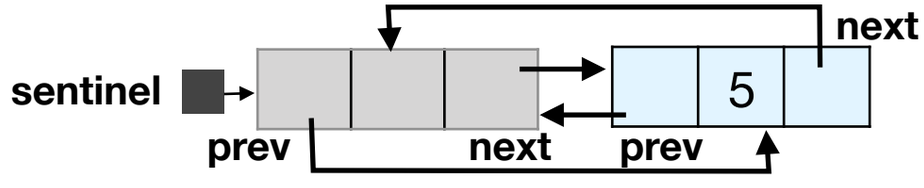


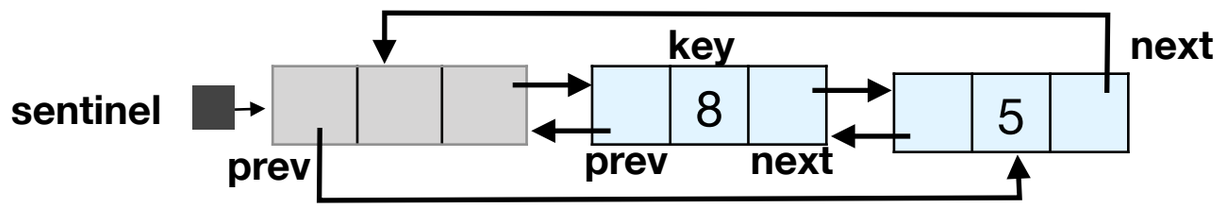


Circular doubly linked list with a sentinel

Linked list with sentinel - pseudocode for insert

```
listInsert(Node *x){
    x->next = sentinel->next;
    sentinel->next->prev = x;
    sentinel->next = x;
    x->prev = sentinel;
}

Time complexity = O(1)
```





listInsert'(x) x->key = 8

Applications

- Implementation of other types of data structures, such as stacks, queues, and trees
- Adjacency list in graph algorithms
- Dynamic memory allocation (malloc algorithm)
- Representing polynomials