Linked List

Juan Zhai

juan.zhai@rutgers.edu

Linked List Operations

- Traverse the whole list
- Search a specified item
- Insert an item
- Delete an item

Traverse a linked list

- Basic operations of a linked list require traversing the list
  - Search the list for an item
  - Insert an item in the list
  - Delete an item from the list
- We cannot use head to traverse the list
  - We would lose the nodes of the list
  - Use another reference variable of the same type as head to do the traversal: ptr
- Refer to Sakai code

Insertion

- Insert to the beginning of a linked list
- Insert to the end of a linked list
- Insert between two nodes of a linked list

Deletion

- Delete the first node of a linked list
- Delete the last node of a linked list
- Delete some in-between node of of a linked list.

Insert to the beginning

1. Create a new node
2. Make the new node point to the first node of the original list
3. Make front (the variable which points to the first node of the list) point to the new node, otherwise the newly added node cannot be accessed

Refer to Sakai code

Remove the head

1. Make the first node head point to head.link

```
public static IntNode deleteFront(IntNode front){
    return front.next; //head has been changed
}
```

Work when the original list has only one node

What if the list is empty?

Remove the head

1. Make the first node head point to head.link

```
public static IntNode deleteFront(IntNode front){
    if (front == null)   //watch out for empty list
        return null;
    return front.next; //head has been changed
}
```

Array v.s. Linked List

Operation	Array	Linked List
Traverse	O(n)	O(n)
Insert at beginning	O(n)	O(1)
Delete at beginning	O(n)	O(1)

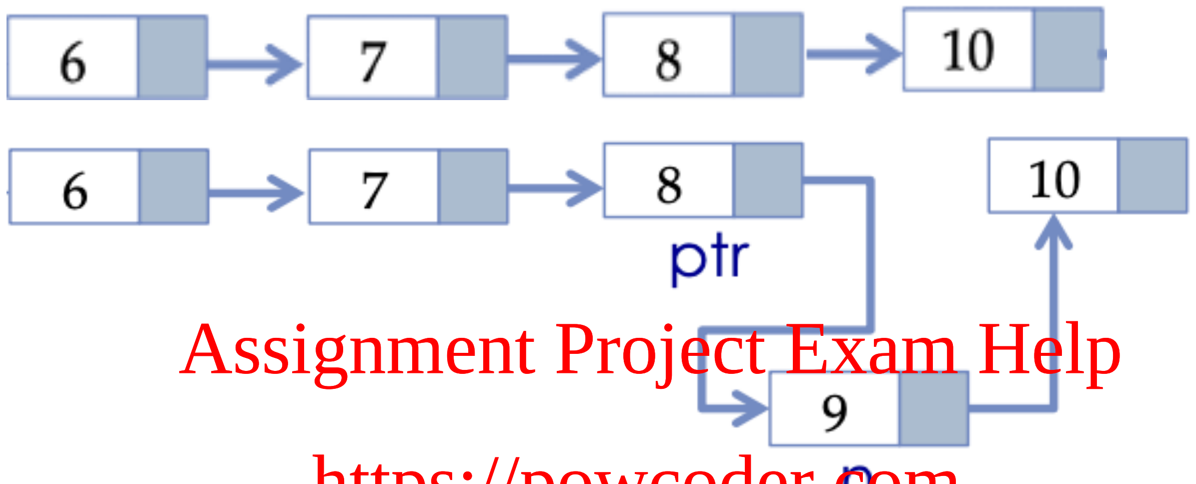
- Constant order: O(1), the execution time does not depend on the size of the input
- Linear order: O(n), the execution time increases at most linearly with the size of the input

Search a specified entry

1. Use a variable to iterate over items
2. Stop searching if
   a. The specified entry is found
   b. Reach the end of the list

Refer to Sakai code

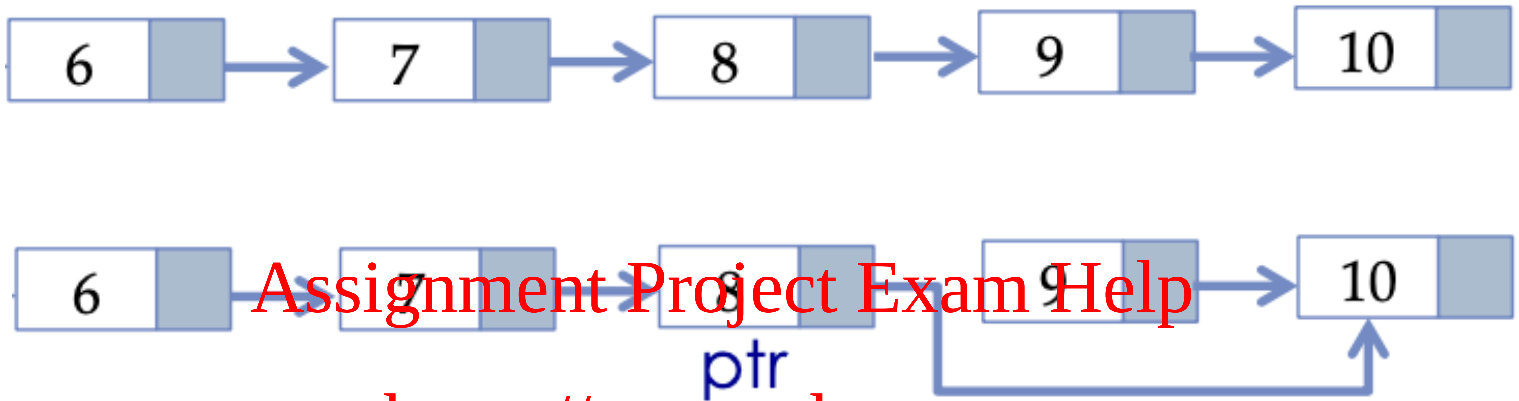
Insert after the integer *target*



1. Locate the node containing the specified integer target, denoted as ptr
2. Create a new node, denoted as n
3. Make the new node point to the node that follows *target* → n.next = ptr.next
4. Make the node containing *target* point to the new node → ptr.next = n

Refer to Sakai code

Remove the specified integer



1. Locate the node that precedes the node containing the integer-to-be-deleted, denoted as ptr
2. Delink the node from the linked list

   → ptr.next = ptr.next.next

Array v.s. Linked List

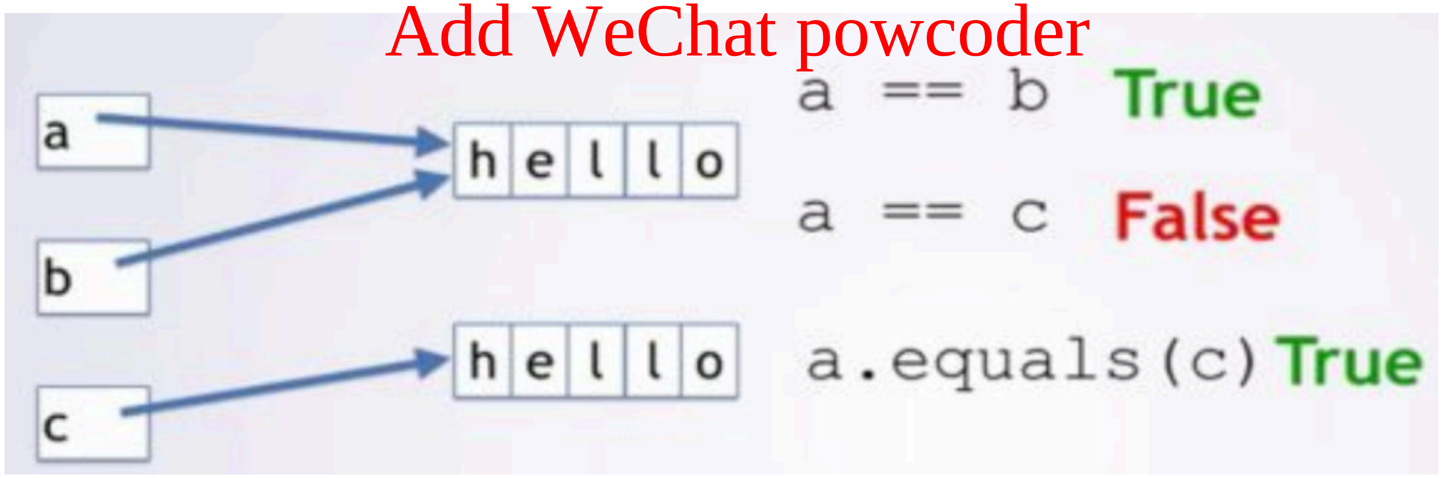
Operation	Array	Linked List
Search specified entry	O(n)	O(n)
Insert/delete at beginning	O(n)	O(1)
Insert/delete in middle*	O(n)	O(1)

* the search time is not included

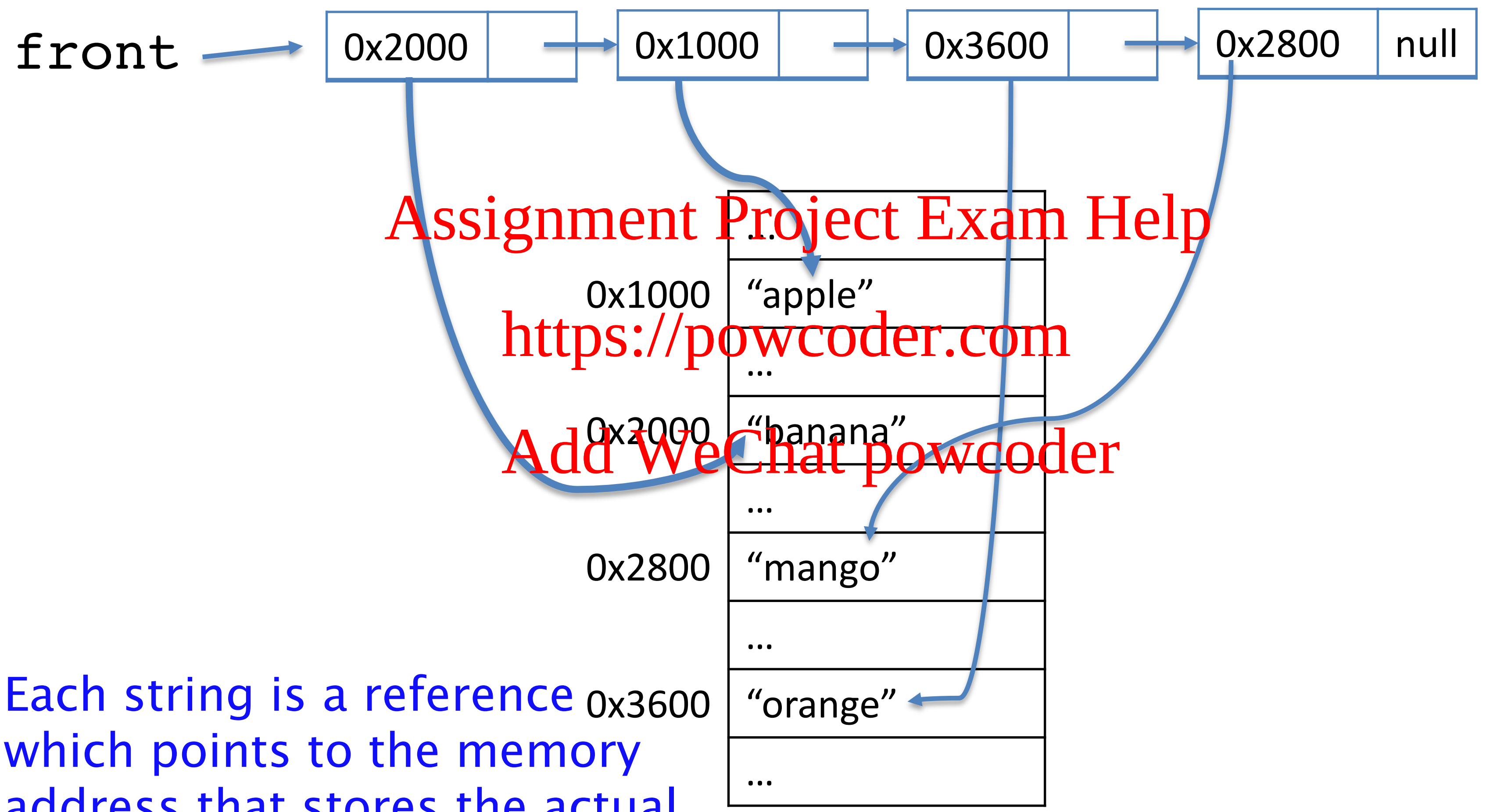
String Compare

- ==: compare object/reference, if refer to the same memory, result is true

- equals(): compare content, if the contents in the memories are the same, result is true



Linked List Containing String



Each string is a reference which points to the memory address that stores the actual string value like "apple"

Linked List Containing Strings

- How does code change if we need to have a linked list of Strings?
  - Define a node class for data of String
  - Substitute == with .equals()

Refer to Sakai code