CSCI 2270 – CS 2: Data Structures

University of Colorado
Boulder

Reminders

Topics

- Abstract Data Types using Lists
- Linked List Data Structure Implementation
  - Traverse
  - Search
  - Insert
  - Delete

Abstract Data Type

- ADT
  - Is like an interface/contract
  - Defines operations that will support it
  - Doesn't define the algorithm(s)
  - No implementation details!
- Data structure
  - Specific implementation of an ADT
  - Singly, Doubly, Circular, etc.
- A linked list is an abstract data type

Singly Linked List

- In a singly linked list, each element, which is also called a node, contains the data stored in the node and a pointer to the next node in the list

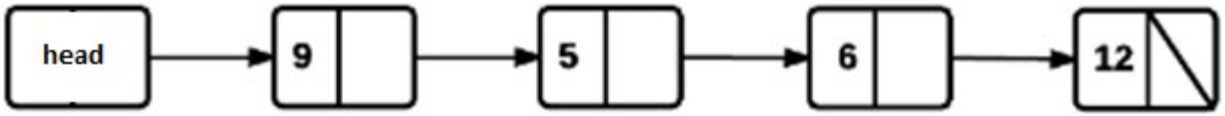


Figure 1. Singly linked list with four elements, called nodes.
In this example, each node has an integer key value and a pointer to the next node in the list.

Doubly Linked List

- Each node has three properties: an integer key, a pointer to the next node in the list, and a pointer to the previous node in the list.

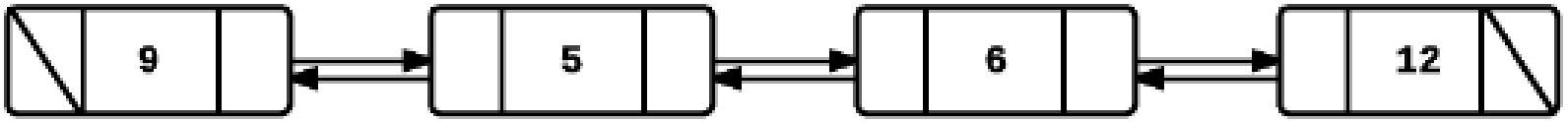
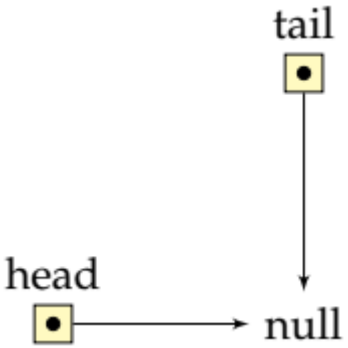
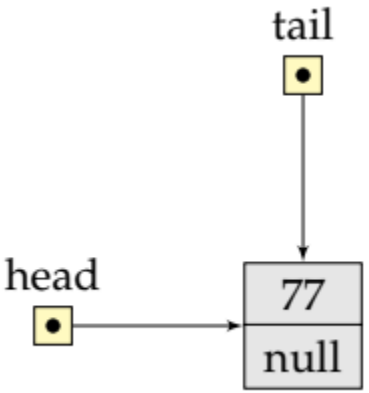


Figure 2. Doubly linked list with four nodes. Each node has an integer key value, a pointer to the previous node, and a pointer to the next node in the list.

More LL Examples





Linked List as ADT

- Basic operations on LL
  - Initialize the list
  - Determine whether the list is empty
  - Print the list
  - Find the length of the list
  - Destroy the list
  - Retrieve the info contained in the first node
  - Retrieve the info contained in the last node
  - Search the list for a given item
  - Insert an item in the list
  - Delete an item from the list
  - Make a copy of the LL

Linked List as ADT

- Let's define the class **linkedListType**

- **# Main operations**
- prepend(value) -> Add a node in the beginning
- append(value) -> Add a node in the end
- pop() -> Remove a node from the end
- popFirst() -> Remove a node from the beginning
- head() -> Return the first node
- tail() -> Return the last node
- deleteNode(Node) -> Remove Node from the list

Linked List as ADT

linkedListType:

1. private:
2. head
3. tail
4. public:
5. Init()
6. insertNode(previousValue, value)
7. search(value)
8. traverse()
9. deleteNode(value)
10. deleteList()

Linked List as ADT - Traverse

- **Traverse Operation**

```
list = LinkedList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

currentNode = LinkedList.head()    # Get the first Node

while the currentNode.next
    print currentNode.value
    currentNode = currentNode.next     # Assign next element
```

Linked List as ADT - Search

- **Search Operation**

```
list = LinkedList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

targetValue = 7

currentNode = LinkedList.head()              # Get the first Node

while the currentNode.next
    if currentNode.value is equal to targetValue
        print Node found
    currentNode = currentNode.next           # Assign the next element
```

Implementation of a Node

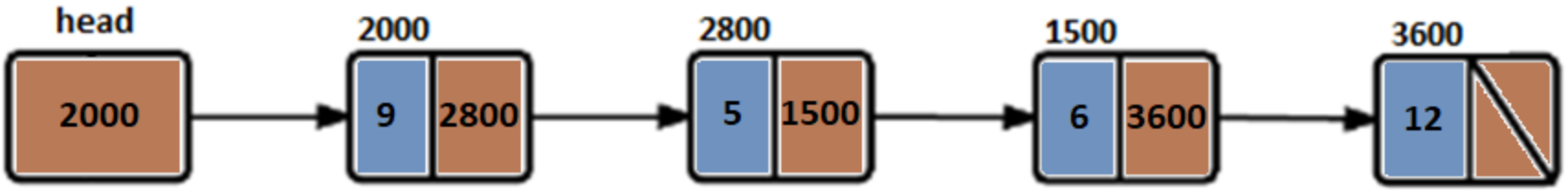
- Each node of a LL has two components
  - Each node is declared as a **class** (C++) or **struct** (C/C++)

```
struct Node {
    int data;           /* Data field */
    Node *next;         /* Next pointer */
};


Node *head;           /* Variable declaration */
```

Implementation of a Node

```
struct Node {
    int data;          /* Data field */
    Node *next;        /* Next pointer */
};

Node *head;                    /* Variable declaration */
```

- head Value = 2000
- head->data Value = 9
- head->next Value = 2800
- head->next->data Value = 5

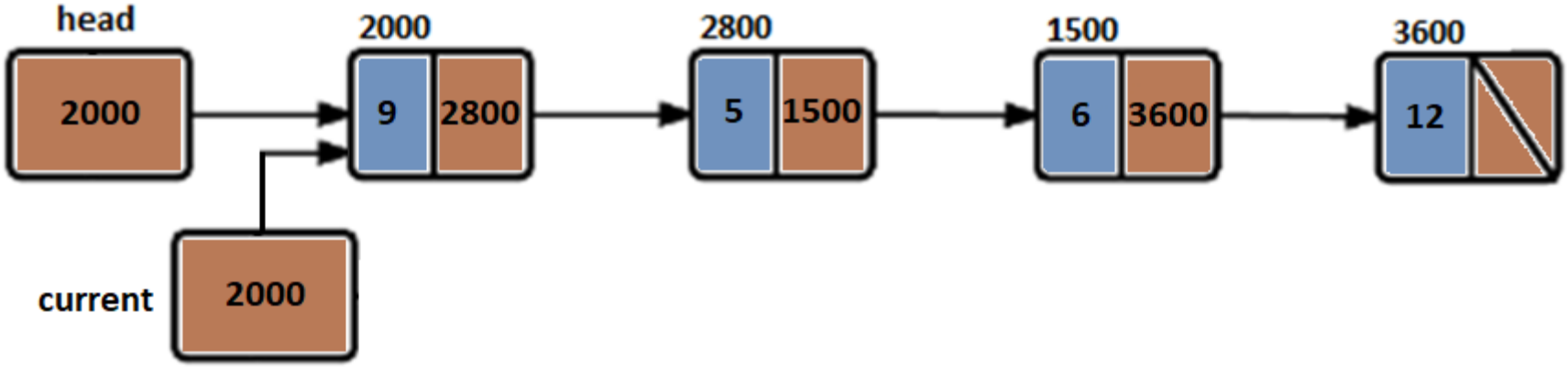


Implementation of a Node

Suppose that current is a pointer of the same type as the pointer head.

```
current = head;   // copies value of head into current
```

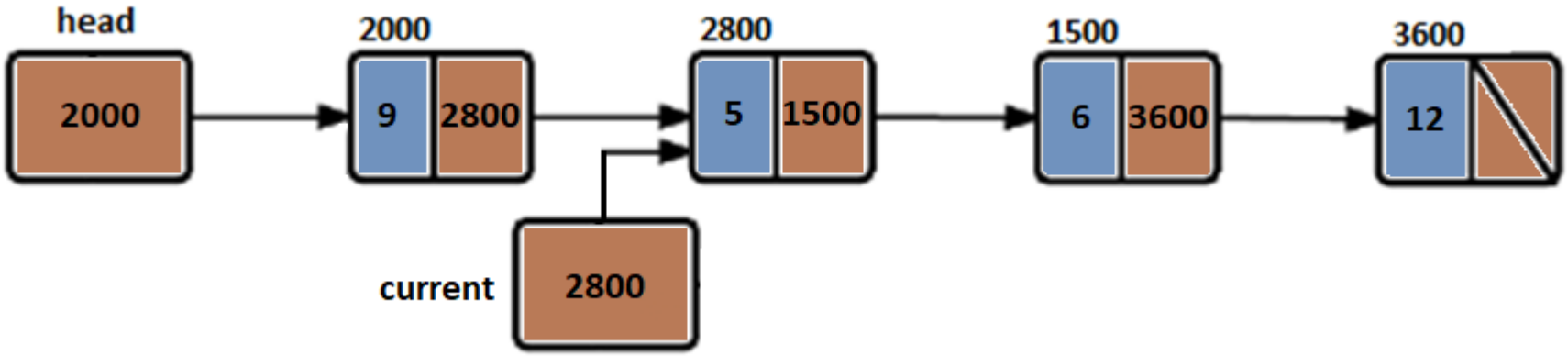
- `current` Value = 2000
- `current->data` Value = 9
- `current->next` Value = 2800
- `current->next->data` Value = 5



Implementation of a Node

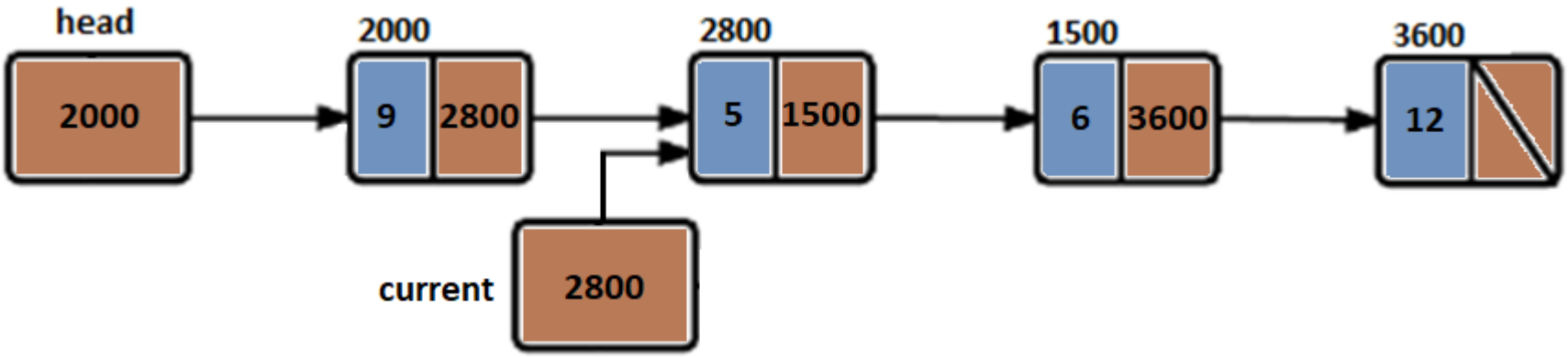
```
current = current->link; // copies over value of current->link
                         // to current, which is 2800
```

- `current` Value = 2800
- `current->data` Value = 5
- `current->next` Value = 1500
- `current->next->data` Value = 6



Implementation of a Node

- `current->next->next->next` `Value = nullptr`
- `current->next->next->next->data` `Value = Does not exist`



Implementation of a Node