Dynamic Data Structures - Linked lists

Dynamic Data Structures

- Basic types (eg. int, char, bool, ...), and arrays of these, store a fixed amount of data.

- We want implementations of ADTs like stacks + queues to grow & shrink (their memory use) as needed.

  - Eg. like Vector, ArrayList, String classes

- Basic Idea:
  - store data in a collection of (simple) objects
  - add/delete these as needed
  - (Link them all together to make the main object.)

Linked Lists

- A sequence of simple objects (nodes), each storing one datum, (plus a link...) linked together in a chain

---

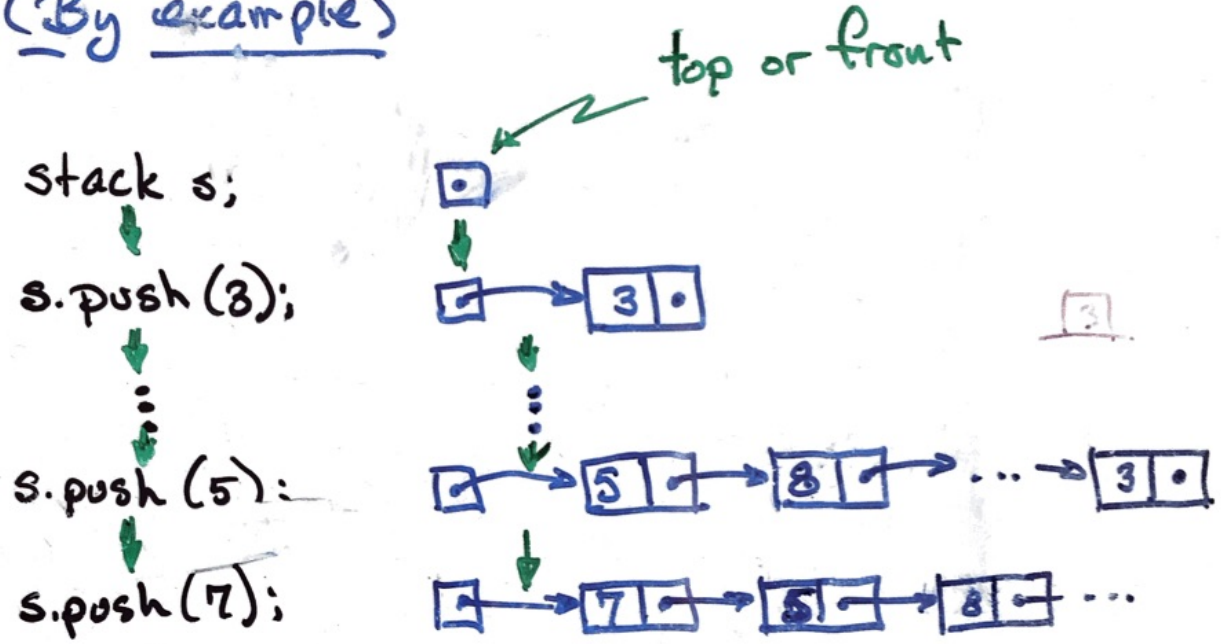
- Eg. to store the list ⟨3,5,7⟩

- These objects have no names, (in contrast to declared variables)
  - we access them by following links
    - in Java, references ← implemented as pointers
    - in C++, pointers
- Need one named place to start:

↑ a normal variable of type "pointer to a node"

Implementing a Stack with a Linked list

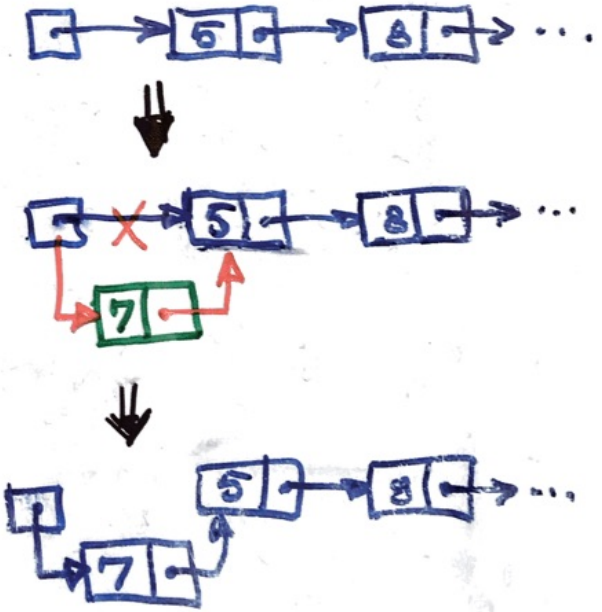
(By example)

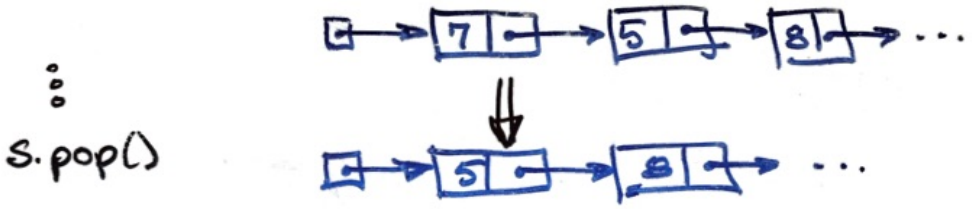


To perform the push (7):

1) Make a new node to store the 7.

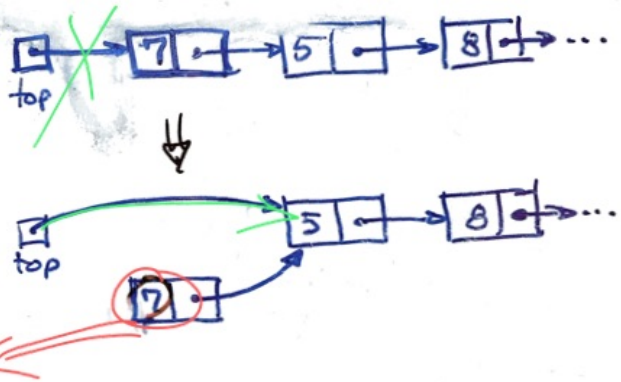
2) modify links to insert it correctly.





To perform the pop():

1. Change the "top" link
2. return the old top value.



Caveat 1: don't lose the old top value

Caveat 2: don't ignore the old top node!
(It still consumes space!)

"Improved" pop():

1) store the old top value in 'temp'
2) make top link to the new top node
3) free the space for the old top node
4) return 'temp'

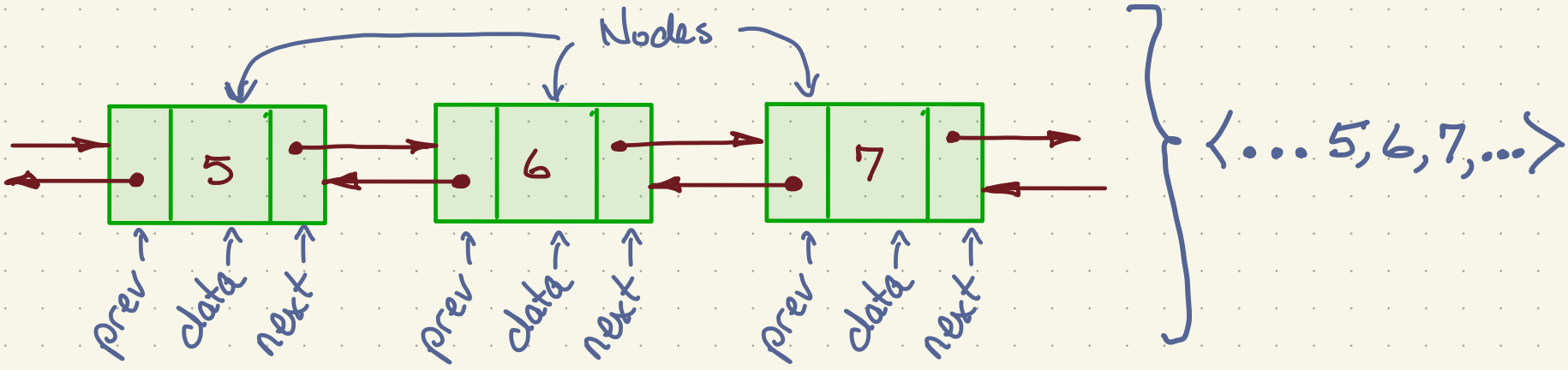
The List Class (A doubly-linked list implementation of a List ADT)

```
template <typename Object>
class List
{
  private:
    // The basic doubly linked list node.
    // Nested inside of List, can be public
    // because the Node is itself private
    struct Node
    {
        Object  data;
        Node   *prev;
        Node   *next;

        Node( const Object & d = Object{ }, Node * p = nullptr, Node * n = nullptr )
          : data{ d }, prev{ p }, next{ n } { }

        Node( Object && d, Node * p = nullptr, Node * n = nullptr )
          : data{ std::move( d ) }, prev{ p }, next{ n } { }
    };
```

list element

pointer to next node

pointer to previous node



End