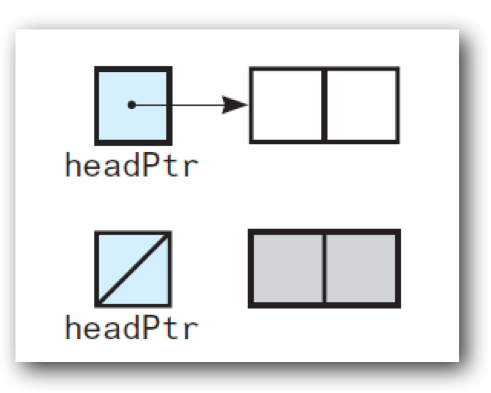
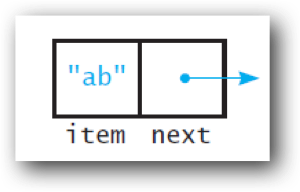
The Empty The Empty Chain



Nod

The Class Node

#ifndef NODE\_H\_   
#define NODE\_H\_

template<class ItemType>

class Node

{

public:

Node(const ItemType& an\_item);   
Node(const ItemType& an\_item, Node<ItemType>\*next\_node\_ptr);  void setItem(const ItemType& an\_item);   
void setNext(Node<ItemType>\* next\_node\_ptr);   
ItemType getItem() const;   
Node<ItemType>\* getNext() const;

private:

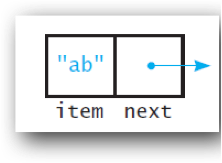
ItemType item\_; // A data item

Node<ItemType>\* next\_; // Pointer to next node

}; // end Node

#include "Node.cpp"

#endif // NODE\_H\_

Node Implementation

#include “Node.hpp”.

template<typename ItemType> Node<ItemType>::Node() : next\_(nullptr)

{  
} // end default constructor

The constructor

template<class ItemType>

Node<ItemType>::Node(const ItemType& an\_item) : item\_(an\_item), next\_(nullptr)   
{  
} // end constructor

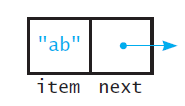
template<class ItemType>

Node<ItemType>::Node(const ItemType& an\_item,

Node<ItemType>\* next\_node\_ptr) :  item\_(an\_item), next\_(next\_node\_ptr)

{

} // end constructor

Node Implementation

The “*setData*” members

#include “Node.hpp”

template<typename ItemType>   
void Node<ItemType>::setItem(const ItemType& an\_item)

{

item\_ = an\_item;

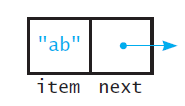
} // end setItem

template<class ItemType>   
void Node<ItemType>::setNext(Node<ItemType>\* next\_node\_ptr)

{

next\_ = next\_node\_ptr;

} // end setNext

Node Implementation

The “*getData*” members

#include “Node.hpp”

template<typename ItemType>   
ItemType Node<ItemType>::getItem() const

{

return item\_;

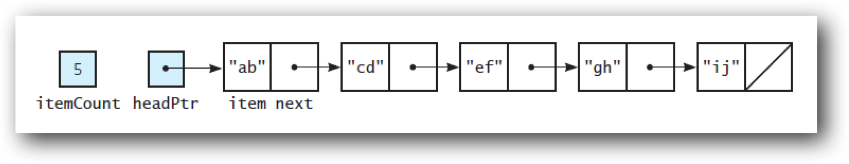
} // end getItem

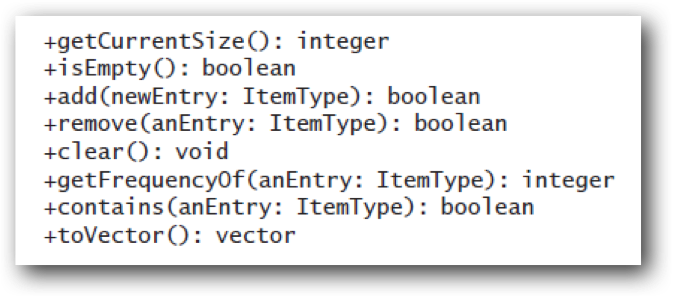
template<class ItemType>   
Node<ItemType>\* Node<ItemType>::getNext() const

{

return next\_;

} // end getNext

A Linked Bag ADT



The Class LinkedBag

#ifndef LINKED\_BAG\_H\_

#define LINKED\_BAG\_H\_

#include "BagInterface.hpp"

#include “Node.hpp" template

<typename ItemType>

class LinkedBag

{  public:

LinkedBag();   
LinkedBag(const LinkedBag<ItemType>& a\_bag); // Copy constructor

~LinkedBag(); // Destructor   
int getCurrentSize() const;   
bool isEmpty() const;   
bool add(const ItemType& new\_entry);   
bool remove(const ItemType& an\_entry);   
void clear();   
bool contains(const ItemType& an\_entry) const;   
int getFrequencyOf(const ItemType& an\_entry) const;  std::vector<ItemType> toVector() const;

private:

Node<ItemType>\* head\_ptr\_; // Pointer to first node   
int item\_count\_; // Current count of bag items

// Returns either a pointer to the node containing a given entry  // or the null pointer if the entry is not in the bag.  Node<ItemType>\* getPointerTo(const ItemType& target) const;

}; // end LinkedBag

//More than one public method will need to know if there is a pointer to a target so we separate it out into a private helper function (similar to ArrayBag but here we get pointers rather than indices)

LinkedBag Implementation

#include “LinkedBag.hpp”  the default constructor

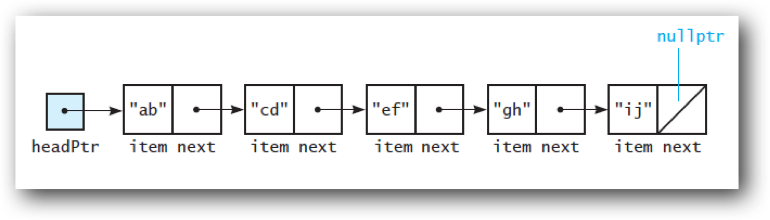
template<typename ItemType>  privet data mem intializ

LinkedBag<ItemType>::LinkedBag() : head\_ptr\_(nullptr), item\_count\_(0)   
{

} // end default constructor

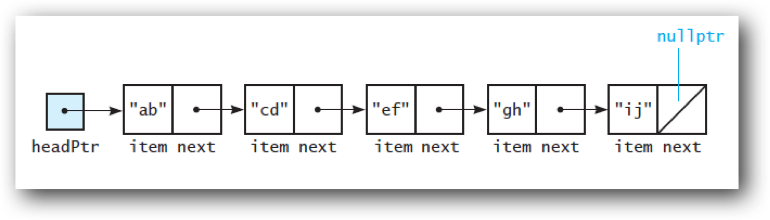
add(const ItemType& new\_entry)

Where should we add?



Lecture Activity

Write **pseudocode** for a sequence of steps to add to the **front** of the chain



    Pseudocode (English-like)

• Instantiate a new node and let temp\_ptr point to it

• Set temp\_ptr->next to point to the same node head\_ptr\_ points to

• Set head\_ptr to point to the same node temp\_ptr points to

• Set temp\_ptr to nullptr

Pseudocode (Code-like)

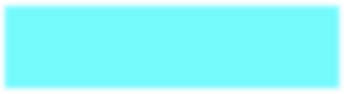
temp\_ptr = new node

temp\_ptr->next = head\_ptr\_

head\_ptr = temp\_ptr

temp\_ptr = nullptr

LinkedBag Implementation



The add method ADD at beginning of chain is easy because we have head\_ptr\_

#include “LinkedBag.hpp”

template<typename ItemType>  
bool LinkedBag<ItemType>::add(const ItemType& new\_entry)

{

// Add to beginning of chain: new node references rest of chain; // (head\_ptr\_ is null if chain is empty)   
Node<ItemType>\* new\_node\_ptr = newNode<ItemType>;



Dynamic memory allocation adding notes to heap\_

new\_node\_ptr->setItem(new\_entry);

new\_node\_ptr->setNext(head\_ptr\_); // New node points to chain

item\_count\_++;

return true; } // end add



DYNAMIC MEMORY ALLOCATION ADDING NODES TO THE HEAP

head\_ptr\_ = new\_node\_ptr;// New node is now first node  Dynamic memory

!23

The add method  
Add at beginning of chain is easy because we have head\_ptr\_

allocation  
Adding nodes to the heap!

