Sorted Lists and Their Implementations

Chapter 12

Contents

- Specifying the ADT Sorted List
- Link-Based Implementation
- Implementations That Use the ADT List

Specifying the ADT Sorted List

- The ADT sorted list maintains its entries in sorted order.
- It is a container of items that determines and maintains order of entries by their values.

Specifying the ADT Sorted List

- Test whether sorted list is empty.
- Get number of entries in sorted list.
- Insert entry into a sorted list.
- Remove given entry sorted list.
- Remove entry at given position.
- Remove all entries.
- Look at (get) th entry at a given position
- Get position of a given entry.

Specifying the ADT Sorted List

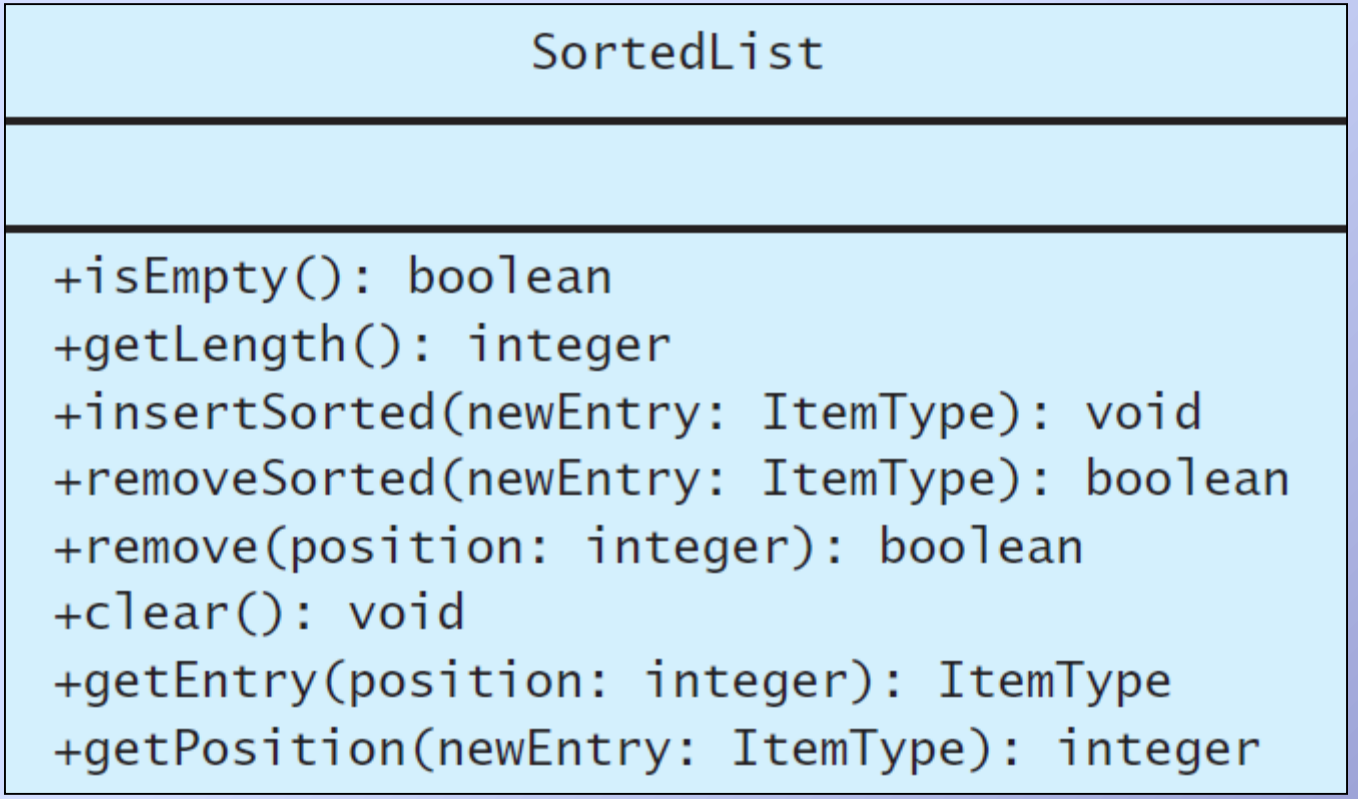


FIGURE 12-1 UML diagram for the ADT sorted list

Specifying the ADT Sorted List

- View C++ interface in Listing 12-1
  - Formalizes initial specifications of ADT sorted list
- ADT sorted list can ac ate entry, given entry as a
- Sorted list will not allo ment of entry by position

.htm code listing files must be in the same folder as the .ppt files for these links to work

A Link-Based Implementation

- View header file for linkSortedList, Listing 12-2
- Begin Implementation: Copy Constructor

```
template<class ItemType>
LinkedSortedList<ItemType>::
LinkedSortedList(const LinkedSortedList<ItemType>& aList)
{
   headPtr = copyChain(aList.headPtr);
}  // end copy constructor
```

A Link-Based Implementation

Method **`copyChain`**

```
template<class ItemType>
Node<ItemType>* LinkedSortedList<ItemType>::
                copyChain(const Node<ItemType>* origChainPtr)
{
   Node<ItemType>* copiedChainPtr;
   if (origChainPtr == nullptr)
   {
      copiedChainPtr = nullptr;
      itemCount = 0;
   }
   else
   {
      // Build new chain from given one
      Node<ItemType>* copiedChainPtr =
                      new Node<ItemType>(origChainPtr->getItem());
      copiedChainPtr->setNext(copyChain(origChainPtr->getNext()));
      itemCount++;
   }  // end if

   return copiedChainPtr;
}  // end copyChain
```

A Link-Based Implementation

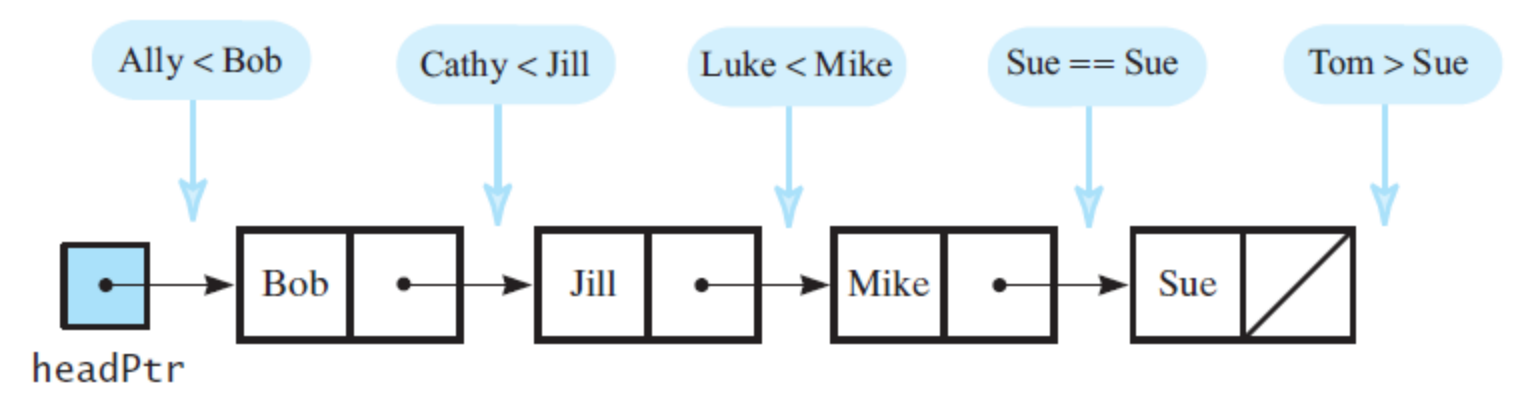


FIGURE 12-2 Places to insert strings into a sorted chain of linked nodes

A Link-Based Implementation

Method **`insertSorted`**

```
template<class ItemType>
void LinkedSortedList<ItemType>::insertSorted(const ItemType& newEntry)
{
   Node<ItemType>* newNodePtr = new Node<ItemType>(newEntry);
   Node<ItemType>* prevPtr = getNodeBefore(newEntry);

   if (isEmpty() || (prevPtr == nullptr)) // Add at beginning
   {
      newNodePtr->setNext(headPtr);
      headPtr = newNodePtr;
   }
   else                                   // Add after node before
   {
      Node<ItemType>* aftPtr = prevPtr->getNext();
      newNodePtr->setNext(aftPtr);
      prevPtr->setNext(newNodePtr);

   }  // end if

   itemCount++;
}  // end insertSorted
```

A Link-Based Implementation

Private method **`getNodeBefore`**

```
template<class ItemType>
Node<ItemType>* LinkedSortedList<ItemType>::
                getNodeBefore(const ItemType& anEntry) const
{
   Node<ItemType>* curPtr = headPtr;
   Node<ItemType>* prevPtr = nullptr;
   while ( (curPtr != nullptr) && (anEntry > curPtr->getItem()) )
   {
      prevPtr = curPtr;
      curPtr = curPtr->getNext();
   }  // end while

   return prevPtr;
}  // end getNodeBefore
```

Implementations That Use the ADT List

- Containment
- Public inheritance
- Private inheritance

Containment

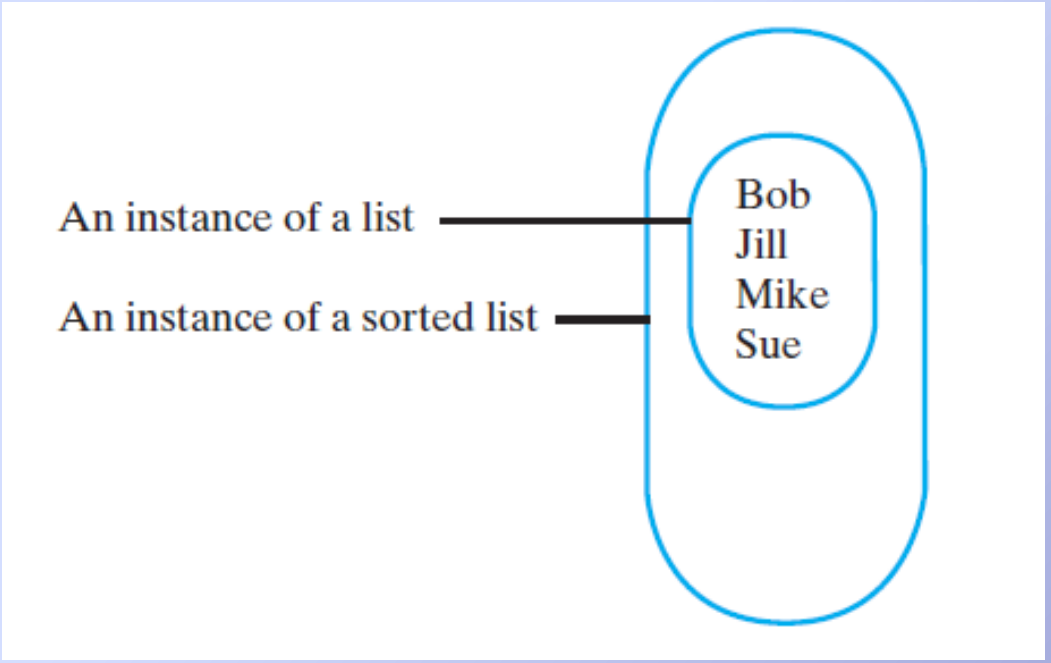
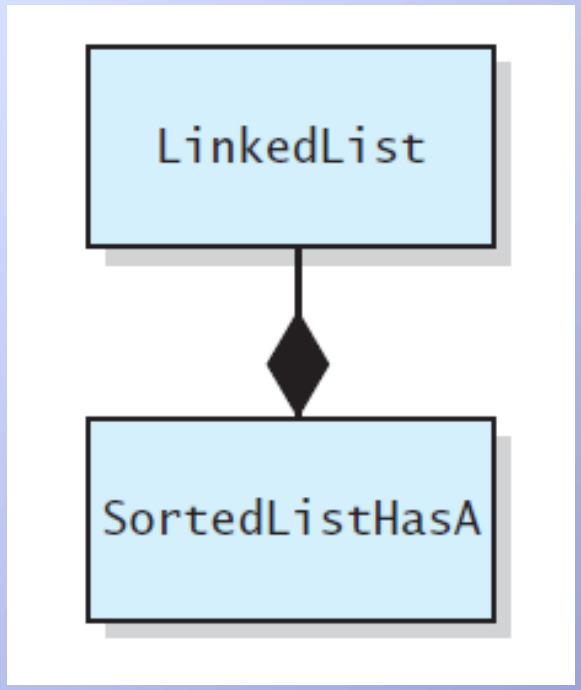


FIGURE 12-3 An instance of a sorted list that contains a list of its entries

Containment

FIGURE 12-4
`SortedListHasA` is composed of an instance of the class **`LinkedList`**



View header file source code, Listing 12-3

Containment

Constructors

```
template<class ItemType>
SortedListHasA<ItemType>::SortedListHasA()
{
   listPtr = new LinkedList<ItemType>();
}  // end default constructor
```

```
template<class ItemType>
SortedListHasA<ItemType>::
SortedListHasA(const SortedListHasA<ItemType>& sList)
{
   listPtr = new LinkedList<ItemType>();
   for (int position = 1; position <= sList.getLength(); position++)
     listPtr->insert(position, sList.getEntry(position));
}  // end copy constructor
```

Containment

Destructor

```
template<class ItemType>
SortedListHasA<ItemType>::~SortedListHasA()
{
   clear();
}  // end destructor
```

Method **`insertSorted`**

```
template<class ItemType>
void SortedListHasA<ItemType>::insertSorted(const ItemType& newEntry)
  {
     int newPosition = fabs(getPosition(newEntry));
     listPtr->insert(newPosition, newEntry);
  }  // end insertSorted
```

Containment

- Methods **`isEmpty`** , **`getLength`** , **`remove`, `clear`** , and **`getEntry`** of ADT sorted list has same specifications as in ADT list
- Example, method **`remove`**

```
template<class ItemType>
bool SortedListHasA<ItemType>::remove(int position)
{
   return listPtr->remove(position);
}  // end remove
```

Efficiency Issues

ADT List Operation	Array-based	Link-based
`insert(newPosition, newEntry)`	$O(n)$	$O(n)$
`remove(position)`	$O(n)$	$O(n)$
`getEntry(position)`	$O(1)$	$O(n)$
`setEntry(position, newEntry)`	$O(1)$	$O(n)$
`clear()`	$O(1)$	$O(n)$
`getLength(), isEmpty()`	$O(1)$	$O(1)$

FIGURE 12-5 The worst-case efficiencies of ADT list operations for array-based and linkbased implementations

Efficiency Issues

ADT Sorted List Operation	List Implementation	
	Array-based	Link-based
`insertSorted(newEntry)`	$O(n)$	$O(n^2)$
`removeSorted(anEntry)`	$O(n)$	$O(n^2)$
`getPosition(anEntry)`	$O(n)$	$O(n^2)$
`getEntry(position)`	$O(1)$	$O(n)$
`remove(givenPosition)`	$O(n)$	$O(n)$
`clear()`	$O(1)$	$O(n)$
`getLength(), isEmpty()`	$O(1)$	$O(1)$

FIGURE 12-6 The worst-case efficiencies of the ADT sorted list operations when implemented using an instance of the ADT list

Public Inheritance

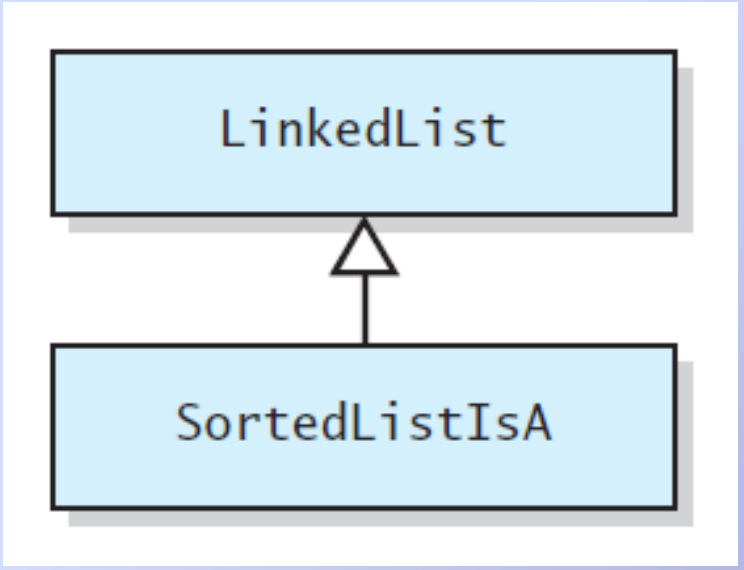


FIGURE 12-7 **`SortedListIsA`** as a descendant of **`LinkedList`**

Public Inheritance

- View header file, Listing 12-4
- Implementation – constructors, destructor

```
template<class ItemType>
SortedListIsA<ItemType>::SortedListIsA()
{
} // end default constructor

template<class ItemType>
SortedListIsA<ItemType>::
SortedListIsA(const SortedListIsA<ItemType>& sList):
                                        LinkedList<ItemType>(sList)

{
} // end copy constructor

template<class ItemType>

SortedListIsA<ItemType>::~SortedListIsA()
{
} // end destructor
```

Public Inheritance

- Method **`insertedSorted`**

```
template<class ItemType>
void SortedListIsA<ItemType>::insertSorted(const ItemType& newEntry)
{
   int newPosition = fabs(getPosition(newEntry));
  // We need to call the LinkedList version of insert, since the
  // SortedListIsA version does nothing but return false
  LinkedList<ItemType>::insert(newPosition, newEntry);
}  // end insertSorted
```

Public Inheritance

- Method **`removeSorted`**

```
template<class ItemType>
bool SortedListIsA<ItemType>::removeSorted(const ItemType& anEntry)
{
   bool ableToRemove = false;
   if (!LinkedList<ItemType>::isEmpty())
   {
      int position = getPosition(anEntry);
      ableToRemove = position > 0;
      if (ableToRemove)
         ableToRemove = LinkedList<ItemType>::remove(position);
   }  // end if

   return ableToRemove;
}  // end removeSorted
```

Public Inheritance

- Method **`getPosition`**

```
template<class ItemType>
int SortedListIsA<ItemType>::getPosition(const ItemType& anEntry) const
{
   int position = 1;
   int length = LinkedList<ItemType>::getLength();

   while ( (position <= length) &&
           (anEntry > LinkedList<ItemType>::getEntry(position)) )
   {
      position++;
   }  // end while

   if ( (position > length) ||
        (anEntry != LinkedList<ItemType>::getEntry(position)) )
   {
      position = -position;
   }  // end if

   return position;
}  // end getPosition
```

Public Inheritance

- Overridden method **`insert`**

```
template<class ItemType>
bool SortedListIsA<ItemType>::
     insert(int newPosition, const ItemType& newEntry)
{
   return false;
}  // end insert
```

Private Inheritance

- View header file for **`SortedListAsA`**, Listing 12-5
- Implementation
  - Method **`getEntry`**

```
template<class ItemType>
ItemType SortedListAsA<ItemType>::getEntry(int position) const
                                    throw(PrecondViolatedExcep)
  {
    return LinkedList<ItemType>::getEntry(position);
  } // end getEntry
```

Private Inheritance

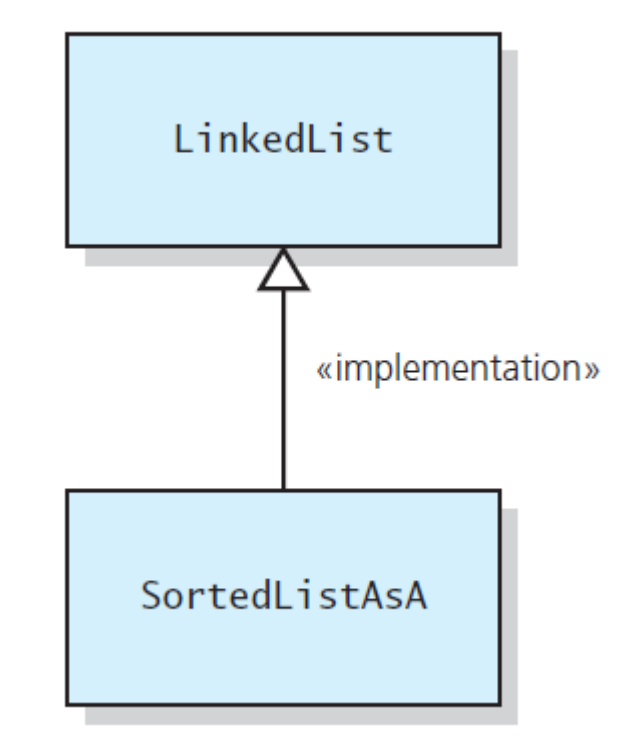
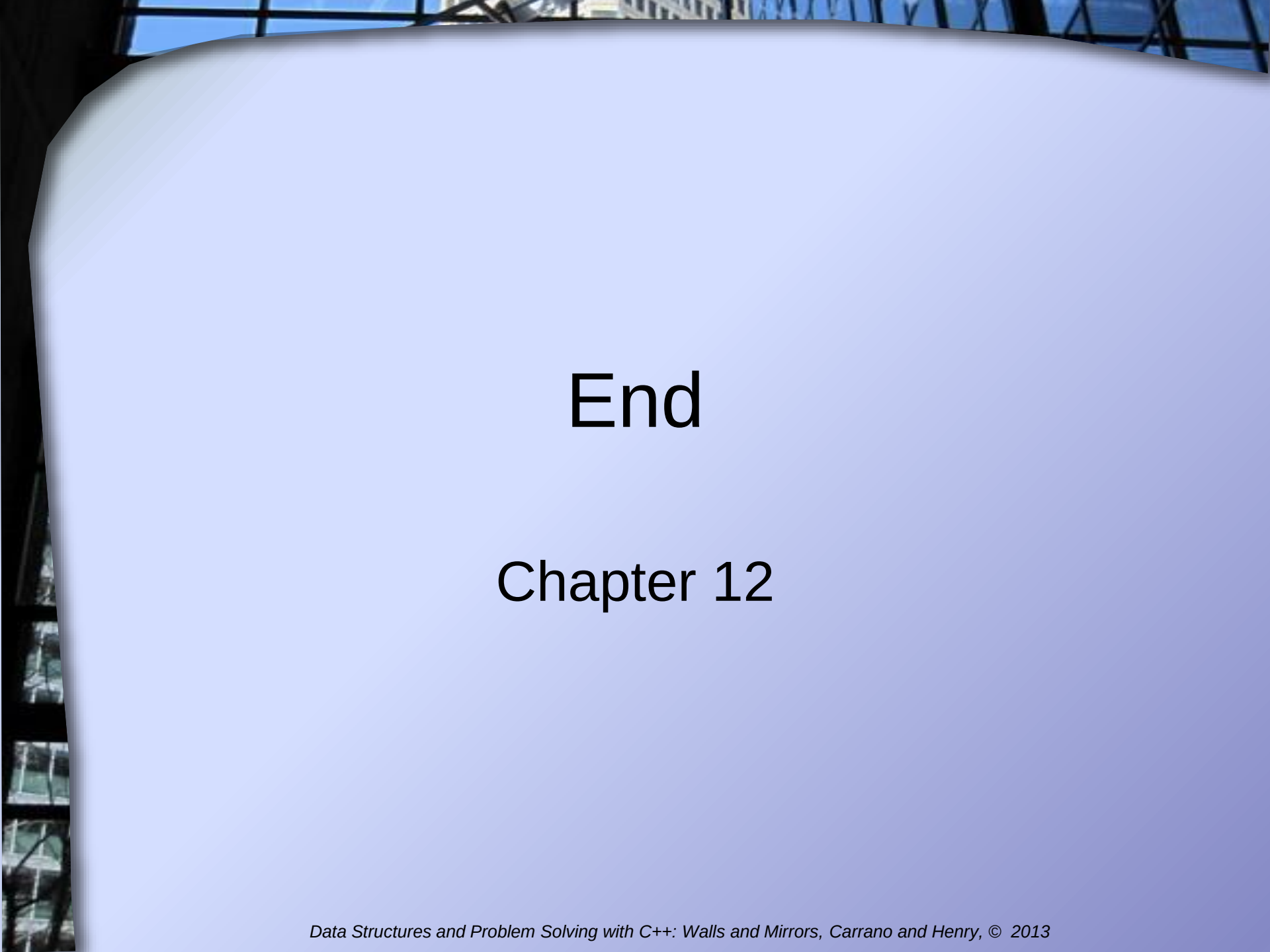


FIGURE 12-8 The **`SortedListAsA`** class implemented in terms of the **`LinkedList`** class



End

Chapter 12