Linked lists

Comp Sci 1575 Data Structures

Linked lists

```
prev->next = toDelete->next;
delete toDelete;

// if only forgetting were
// this easy for me.
```

```
assert "It's going to be okay.";
```

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

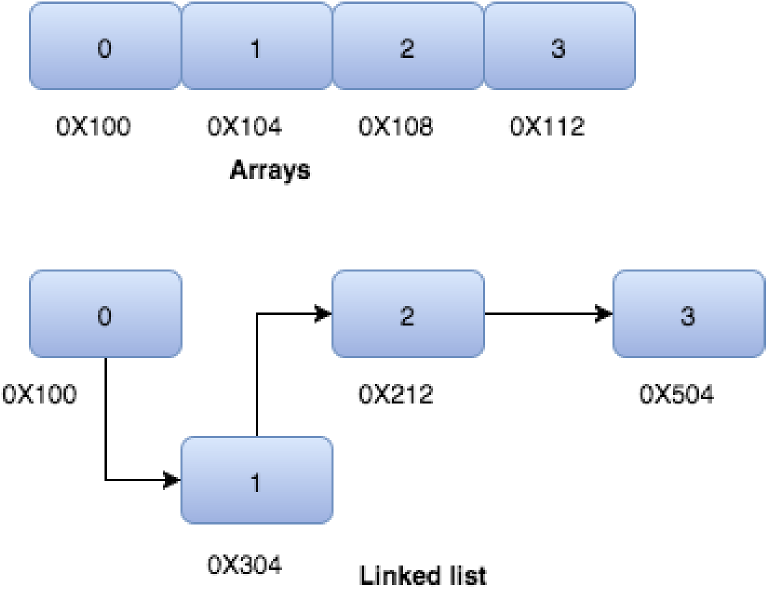
Linked list

- **Linked lists** are of a linear collection of data elements, called **nodes**, each pointing to the next node
- Each **node** is composed of:
  - **data**, and
  - **a pointer** (link) to the next node in the sequence.
- Enable efficient insertion or removal of elements from any position in the sequence
- Uses dynamic memory allocation for new list elements as needed

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Storage in memory



0 | 1 | 2 | 3

0X100 | 0X104 | 0X108 | 0X112

Arrays

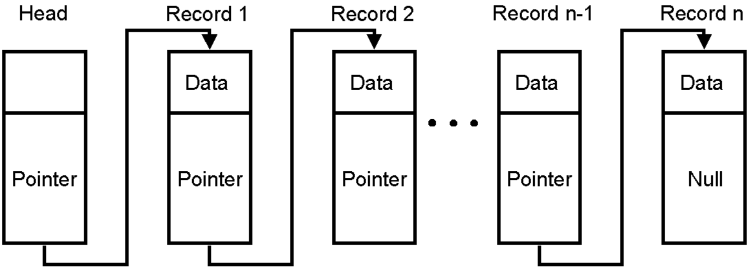
0 → 1 → 2 → 3

0X100, 0X304, 0X212, 0X504

Linked list

Data and pointers

Note the header node: and the tail node:



Empty data: and **NULL pointer:**

- Which node-external features of this list do we want to keep track of for our list?
- What is the minimum number of node-external features we must track?

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Outline

1. Definitions
2. Memory structure
3. Implementation
    - Modular classes
    - Header node
    - Current position?
    - Code
4. Sketchpad slide
5. Operations
    - Insertion
    - Removal
    - setValue?
    - Recursive traversals
6. Comparison
    - Table of step counts
    - Pros and cons
    - Space requirements

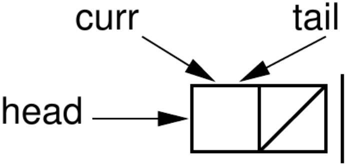
Node and list classes

- A list node is a separate object (not just an array element)
- Thus, it is common to create separate
  list class and
  node class
- **Node** class can also be reused by other linked data structures, like our tree, etc.

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Initial state of a linked list when using header node



```
// Three pointers to nodes of type E
Node<E> *head;
Node<E> *curr;
Node<E> *tail;
int cnt;
// Grab new memory for a node
head = curr = tail = new Node<E>;
cnt = 0;
```

Which of these is variables is necessary?
Which are merely convenient?

Outline

1 Definitions

2 Memory structure

3 Implementation
- Modular classes
- Header node
- Current position?
- Code

4 Sketchpad slide

5 Operations
- Insertion
- Removal
- setValue?
- Recursive traversals

6 Comparison
- Table of step counts
- Pros and cons
- Space requirements

Q1

Do Q1

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Code

Check out the uploaded LinkedList implementation, which shares some files with the previous array list, and with new files including:

- *node.h* is the node class
- *list_L.h* is the list class
- *main_LList.cpp* calls these with the same test as before

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Sketchpad slide

Q2

Do Q2

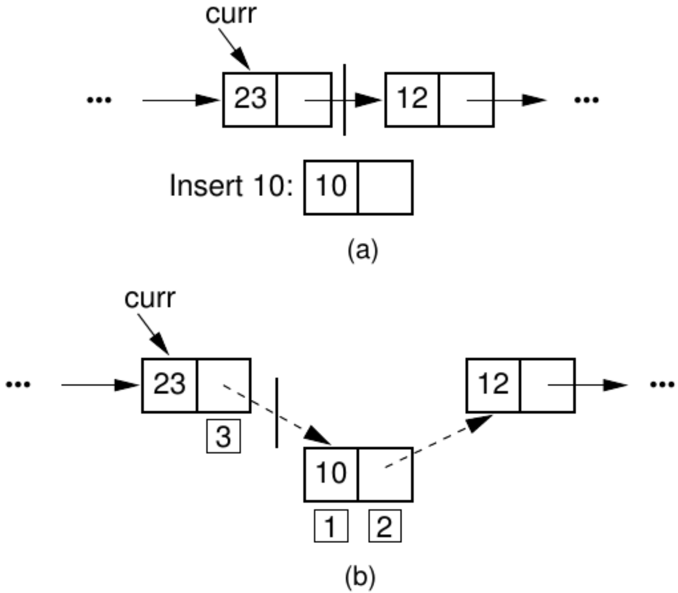
Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. **Operations**
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

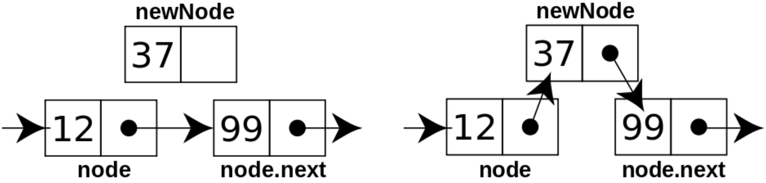
Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Insertion: numbered steps

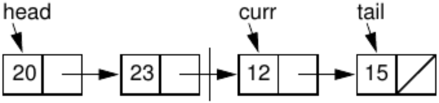


Insertion

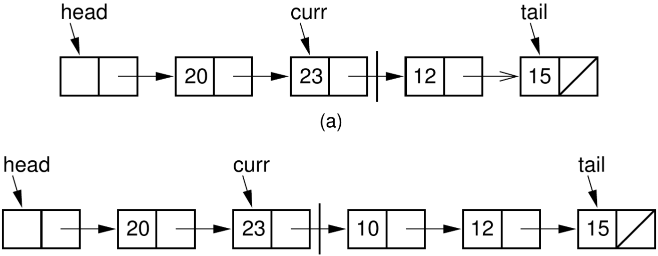

newNode
37
12
99
node
node.next
newNode
37
12
99
node
node.next

Can we insert a node before a pointed-to-node?

Insert 10 before node being pointed to?



Insert 10 after node being pointed to?



(a)

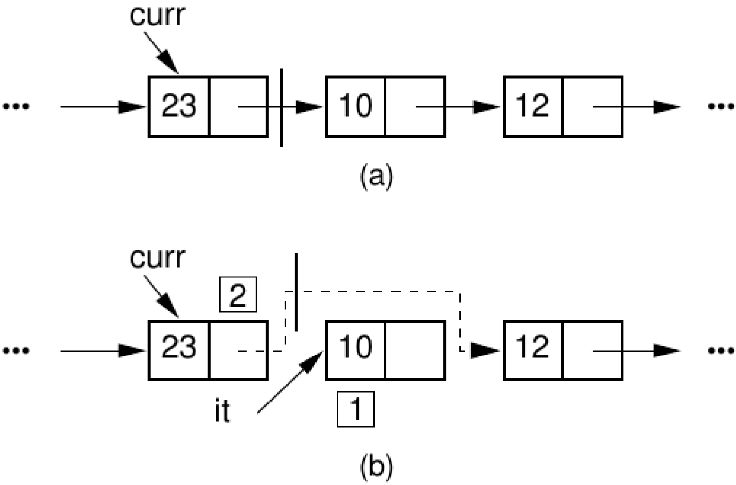
(b)

Empty header node means we don't have to use special cases with pointer operations performed on element after curr

Outline

1. Definitions
2. Memory structure
3. Implementation
    - Modular classes
    - Header node
    - Current position?
    - Code
4. Sketchpad slide
5. Operations
    - Insertion
    - Removal
    - setValue?
    - Recursive traversals
6. Comparison
    - Table of step counts
    - Pros and cons
    - Space requirements

Removal



Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

setValue?

Remember, curr points to the object before the one being modified, though we're pretending it's the current object for ease.

```
void setValue(const E &it)
{
    curr->next->element = it;
}
```

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Recursive traversal?

- What is base case?
- What is the 1-smaller version of the same problem?

Check out recursive length function.

Q3

Do Q3

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Operation steps related list size increase?

	ArrayList	LinkedList
Insert()	n	1
append()	1	1
remove()	n	1
moveToStart/End()	1	1
prev()	1	n
next()	1	1
length() cnt or re-calc	1	1 or n
currPos()	1	n or 1
moveToPos()	1	n
getValue()	1	1
clear()	=	=

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Pros and cons of linked lists

Advantages

- Dynamic data structure; can grow or shrink, allocating and deallocating memory during runtime
- Insertion and deletion node operations easy
- No need to define an initial size for a linked list.
- Items can be added or removed from the middle of list.
- Dynamic data structures such as stacks and queues can be implemented using a linked list.

Disadvantages

- Can use more memory than arrays because of the storage used by pointers.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
- Nodes are stored incontiguously, increasing the time required to access individual elements within the list
- Singly linked lists can't directly navigate backwards

Outline

1. Definitions
2. Memory structure
3. Implementation
   - Modular classes
   - Header node
   - Current position?
   - Code
4. Sketchpad slide
5. Operations
   - Insertion
   - Removal
   - setValue?
   - Recursive traversals
6. Comparison
   - Table of step counts
   - Pros and cons
   - Space requirements

Space requirements

ArrayList

- Array must be allocated in advance.
- No overhead if all array positions are full.

LinkedList

- Space grows with number of elements.
- Every element requires overhead

If your data type is the same size as the pointer, then when is space equivalent with an ArrayList and LinkdeList?

Space heuristics

- Generally, linked lists are more space efficient when implementing lists whose number of elements varies widely or is unknown.
- Array-based lists are generally more space efficient when the user knows in advance approximately how large the list will become.

Operation times for insertion/deletion/etc will often be more important than space, but not always.