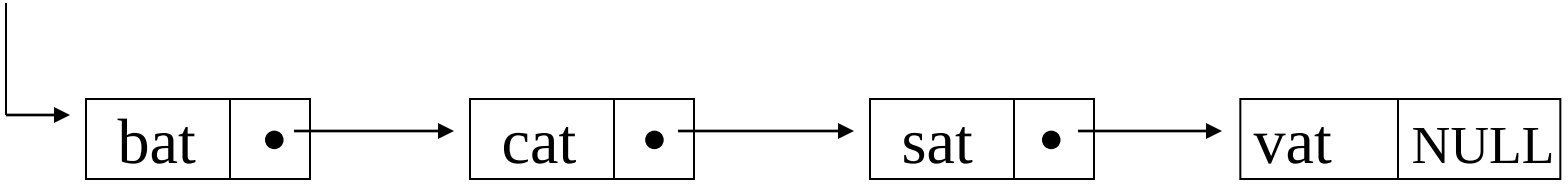
LISTS

Bruce Nan

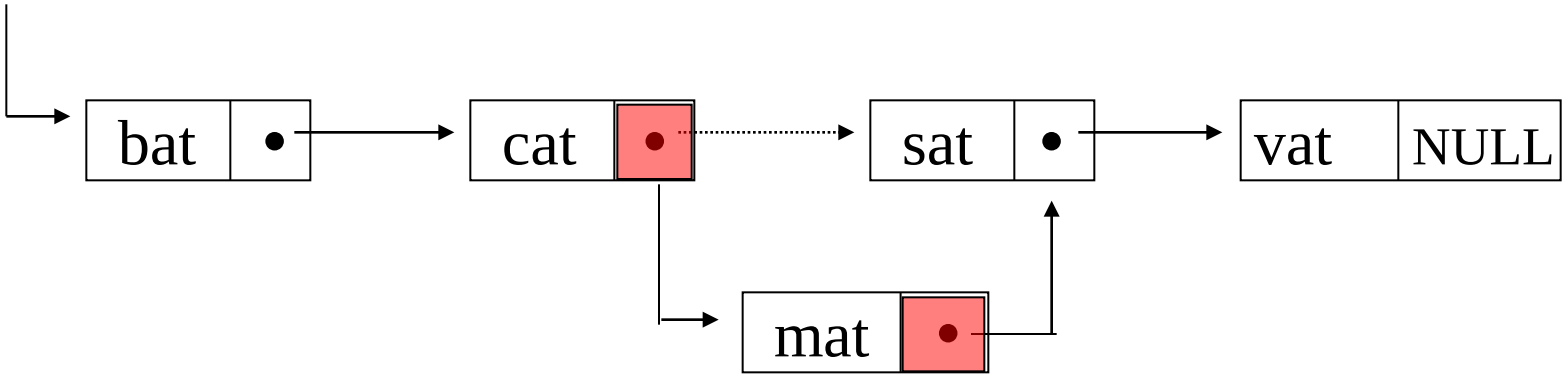
Linked Lists

- Avoid the drawbacks of fixed size arrays
  - Linked lists

Linked Lists

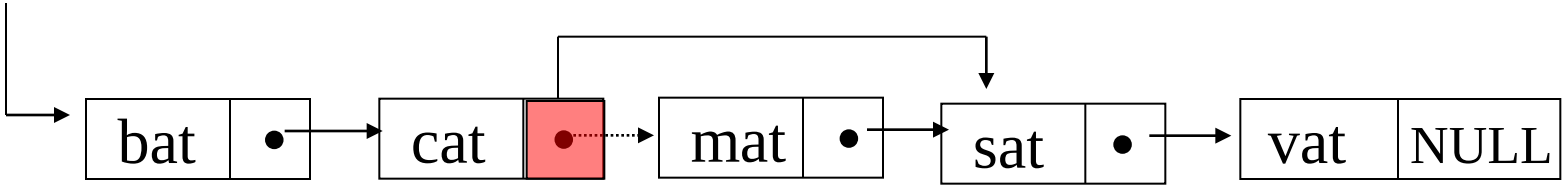


Insertion



Compare this with the insertion in arrays!

Deletion



dangling
reference

Abstract Data Type

- Defined by:
  - What information it stores
  - How the information is organized
  - How the information can be accessed
- Doesn't specify **implementation**

Vs. Implementation

- What information it stores
  - `What classes/types of variables`
- How the information is organized
  - `How it is stored in memory`
- How the information can be accessed
  - `What methods (and algorithms)`

ADT: Lists

- An ordered series of objects
- Each object has a previous and next
  - Except *first* has no previous, *last* has no next
- We can insert an object to a list (at location *k*)
- We can remove an object from a list
- We can read an object from a list (location *k*)

Application of Lists

- To Do: insert tasks, remove when done
- Word Processor:
  - typing text inserts to list,
  - deleting text removes (simple array won't work)
- Shopping: insert needed items, remove when bought

Array List

- 1st Hurdle: arrays have sizes
  - Create bigger array when we run out of space, copy old array to big array
- 2nd Hurdle: Inserting object anywhere but the end
  - Shift all entries forward one. O(N)
- Get kth and insertion to end constant time O(1)

Linked List

- Store list objects anywhere in memory
- Each object has a reference to its next object
- Insert at *k* requires T(get kth) + constant
  - Insert to front is constant time
- T(get kth) = O(N), if naïve

Linked List vs. Array List

- Linked Lists
  - No additional penalty on size
  - Insert/remove O(1)*
  - get kth costs O(N)*
  - Need some extra memory for links
- Array Lists
  - Need to estimate size/grow array
  - Insert/remove O(N)*
  - get kth costs O(1)
  - Arrays are compact in memory

Build-in List

- C++, Java, Python have the build-in list structure.
- Just use it and call functions.

C++ List

```
#include <list>

list <int> L;          //define a list variable
L.push_back(1);        //add element at the end
L.push_back(2);
L.push_front(0);       //add element at beginning
```

List Functions

empty ()	Test whether list is empty
size()	Return size
front()	Access first element
back()	Access last element
push_front()	Insert element at beginning
pop_front()	Delete first element
push_back()	Add element at the end
pop_back()	Delete last element

List Functions -- cond

insert()	Insert elements
erase()	Erase elements
swap()	Swap list
clear()	Clear list
unique()	Remove duplicate values
merge()	Merge sorted lists
sort()	Sort elements in list

More functions can be found from google c++ list

Java List

- Java has a public interface list
- All java list methods can be found from http://docs.oracle.com/javase/1.4.2/docs/api/java/util/List.html

List in Java

- **Collection** Interface extends **Iterable**
- A Collection stores a group of objects
- We can add and remove from a Collection
- **Iterator** objects let us iterate over objects in a Collection (also enhanced **for** loop)
- Built in **LinkedList** and **ArrayList** implementations of Collection

Java List Methods

- void clear()
- int size()
- boolean isEmpty()
- boolean add(AnyType x)
- void add(int idx, AnyType x)
- AnyType get(int idx)
- AnyType set(int idx, AnyType newVal)
- AnyType remove(int idx)
- String toString()
- java.util.Iterator<AnyType> iterator()

Operations of linked lists

- Create two lists L1 and L2, and store the following data into L1 and L2
  L1 Data: 2, 8, 10, 3, 8, 1 L2 Data: 3, 2, 5, 9, 1, 5
- Print all elements of each list
- Print the size of each list
- Sort each list in increasing order
- Print all elements of each list
- Remove duplicate values in each list
- Print all elements of each list
- Find all share elements of two lists
- Merge two sorted lists L1 and L2 into one sorted list L1
- Print all elements of list L1
- Clear list L1