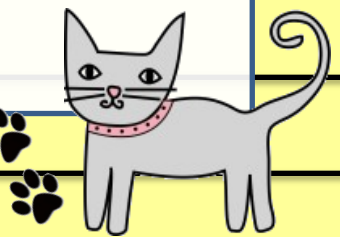
CMSC 21
Fundamentals of Programming

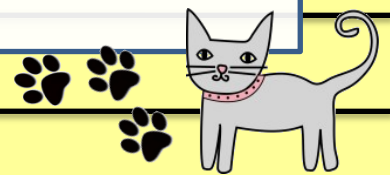
2nd Semester 2011-2012

Linked Lists

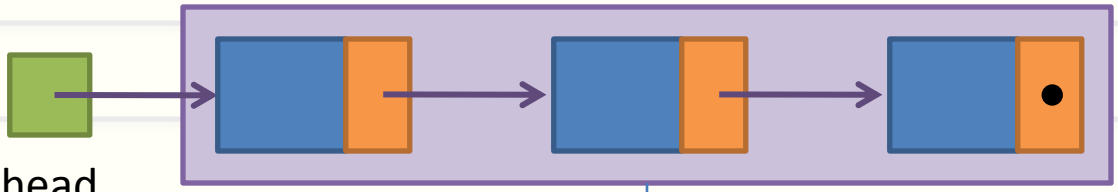


Linked Lists

- Used as alternative to arrays
- The size can grow or shrink during the execution of the program
- Used when the size of data varies during the execution of the program
- Saves memory, unlike arrays
  - Allocated memory will never exceed what is needed by the program
  - Dynamic arrays can handle the change in the maximum size of data, but it is possible that the allocated memory will not be used.

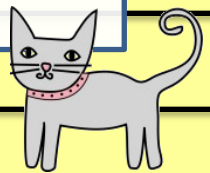


Linked Lists



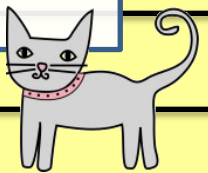
A pointer to a structure that holds the address of the first element of the linked list; analogous to the constant pointer employed by arrays

Elements of the linked list



Creating a linked list

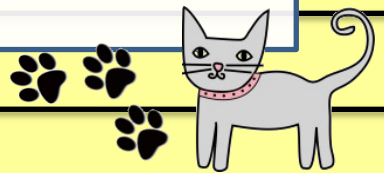
- A linked list consists of dynamic variables linked together to form a chain-like structure
- Each linked list element is called a node
- A node is a self-referential structure, that is, a structure that has a pointer to an instance of itself as a field
- The `malloc` and free `functions` are used to dynamically grow and shrink the linked list



Self-referential Structures

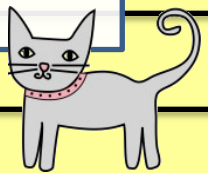
- Structure that contains as a field a pointer to a structure similar to itself.
- Example:

```
struct node {
  int x;
  //pointer to an instance of
  //struct node
  struct node *ptr;
}
```



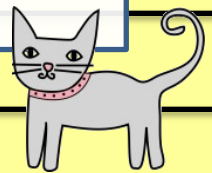
Linked Lists

- In a linked list, the pointer in each element will point to the next node in the list
- The pointer to the first element of the list is called head
- Furthermore, linked lists may have a dummy node (OPTIONAL!)



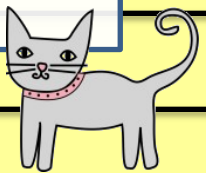
Linked Lists

- There are several kinds of linked lists
  - Singly Linked List (most basic)
  - Doubly Linked List
  - Circular Singly Linked List
  - Circular Doubly Linked List



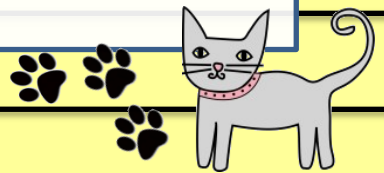
Dummy Nodes

- A dummy node is a node in the linked list that does not contain data and is used to simplify some linked list operations
- Can be advisable for non-circular linked lists
- Optional



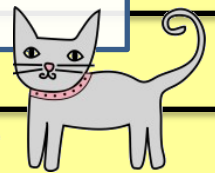
Operations on Linked Lists

- In general, linked lists have four basic operations:
  - Insert
  - Delete
  - Search
  - View

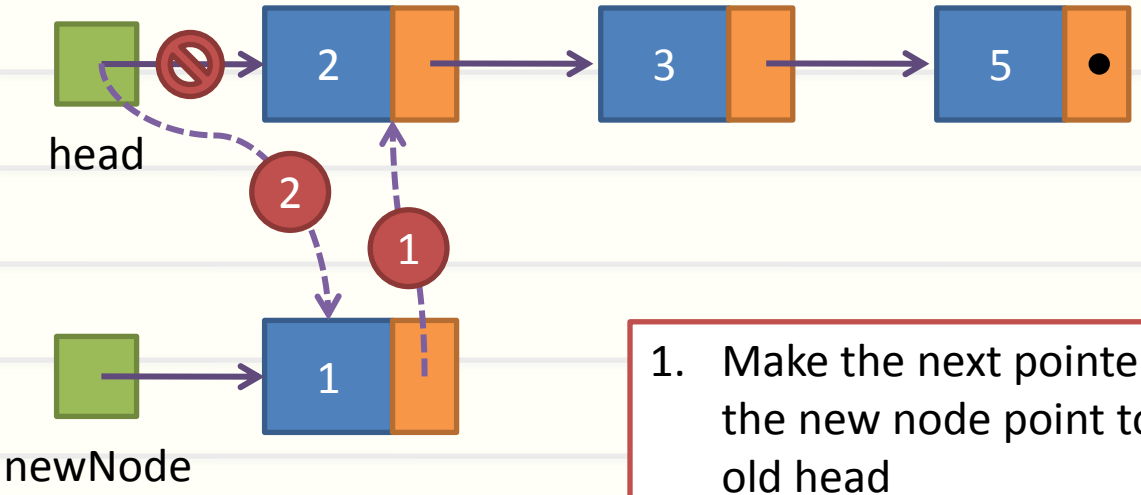


Insert

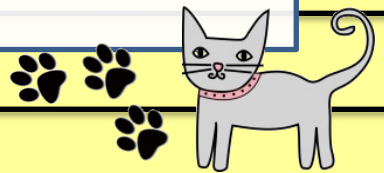
- Used to insert values to the linked list
- Can be:
  - Insert at head
  - Insert at the middle
  - Insert at tail
- In the case of sorted lists, insertion in the middle of the list is used in order to insert new values in their rightful positions in the list



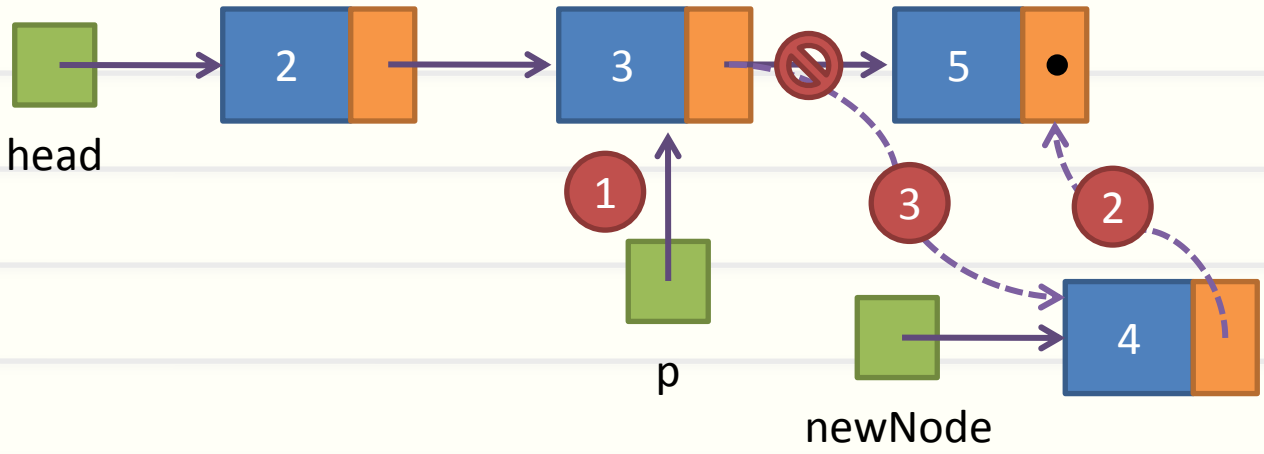
Insert at head



1. Make the next pointer of the new node point to the old head
2. Make the head pointer point to the new node



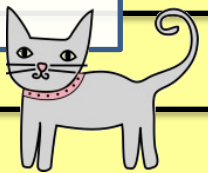
Insert at the middle



1. Find the position where the node is to be inserted. Find the last node whose value is less than the new node.
2. Make the next pointer of the new node point to the node next to the one selected in step 1.
3. Make the next pointer of the node selected in step 1 refer to the new node

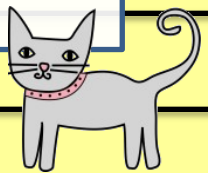
Insert at the middle

- Usually used in conjunction with insertion at head
  - If the value to be inserted cannot be inserted at head (due to sorting purposes), insert at middle is used to find the node's correct position in the rest of the list (not including the head)
- Can be modified slightly to accommodate insertion at tail

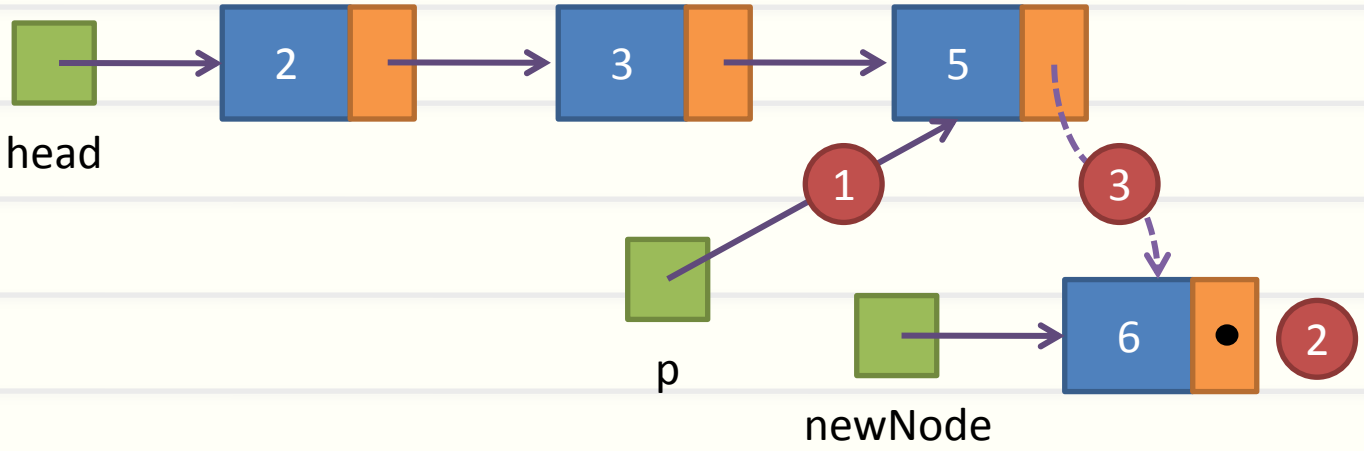


Insert at tail

- Is used when the position of the new value is not in the middle (and therefore not at the head either)
- Can be treated as a special case of insert in the middle



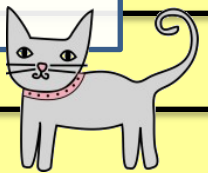
Insert at tail



1. Find the position where the node is to be inserted. Find the last node whose value is less than the new node.
2. Make the next pointer of the new node point to the node next to the one selected in step 1. However, since there is no more next node, the pointer is set to NULL
3. Make the next pointer of the node selected in step 1 refer to the new node

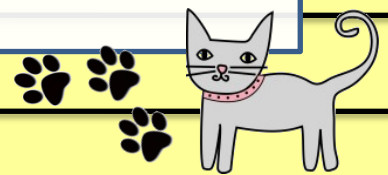
Notes

- To mark the end of the list, the next pointer of the last node should have a value of NULL
  - NULL is a constant value that is defined in stdlib.h
  - NULL is usually symbolized by a pointer that is not pointing anywhere
- In our linked list visualizations, we assume that a pointer field that does not have an outgoing arrow has a NULL value
- To prevent pointers that have garbage values, ALWAYS initialize your pointers to NULL

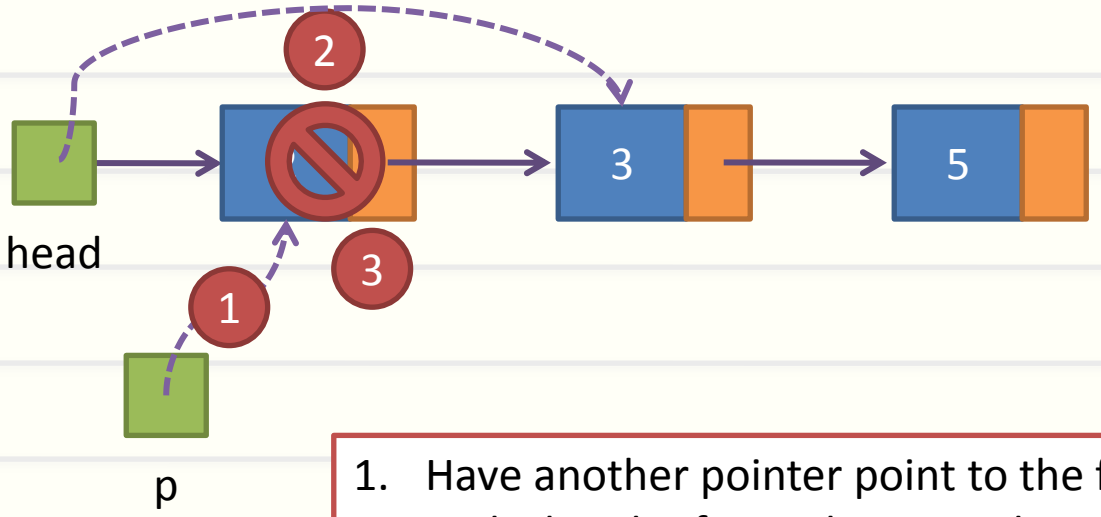


Delete

- Used to delete elements from the list
- Similar to insertion, it has two cases:
  - Delete at head
  - Delete at the middle
  - Delete at tail

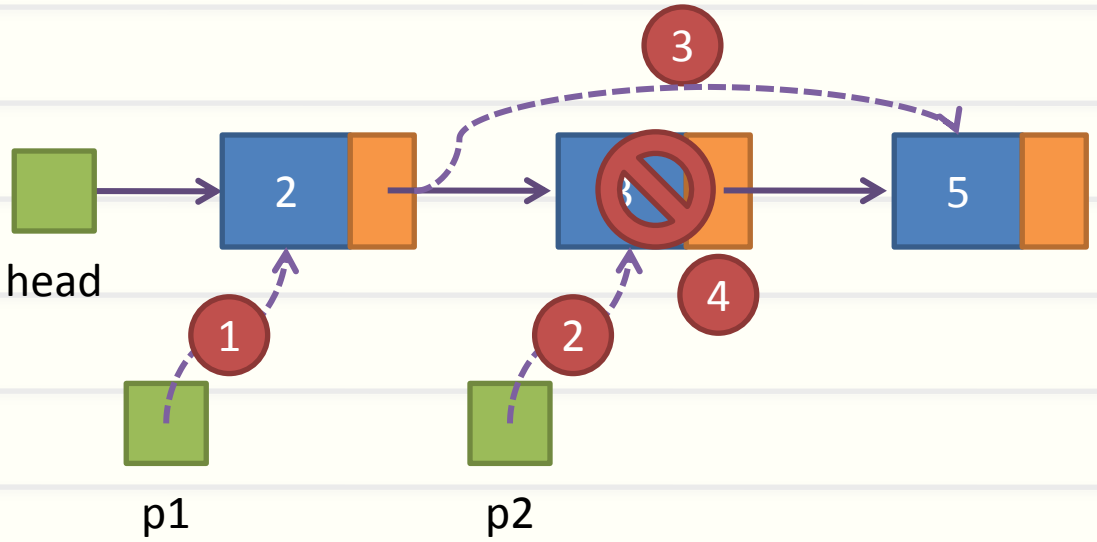


Delete at head



1. Have another pointer point to the first node
2. Make head refer to the second node
3. Free the first node

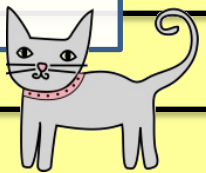
Delete at the middle



1. Find the position before the node to be deleted
2. Have another pointer refer to the node to be deleted
3. Make the next pointer of the node found in step 1 refer to the next node pointed to by the node in step 2
4. Delete the node in step 2

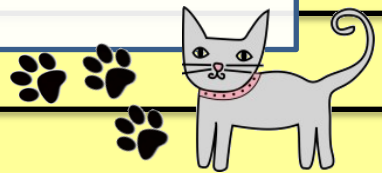
Delete at the middle

- Like insertion in the middle, it is used in conjunction with deletion at middle
- Can also be modified to accommodate deletion at tail

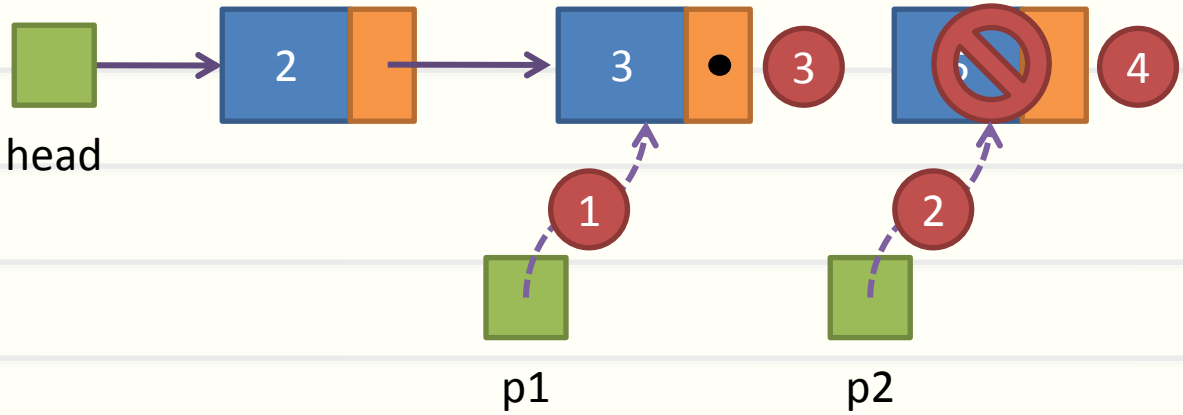


Delete at tail

- Like insertion at tail, can be treated as special case of deletion at tail



Delete at tail



1. Find the node before the tail using an extra pointer
2. Have another pointer refer to the last node
3. Make the next pointer of the node found in step 1 point to NULL
4. Delete the node in step 2