Linked Lists - Introduction
Dynamic Arrays Revisited

• Dynamic array can sometimes be slow
  – When?
  – Why?
Linked Lists to the Rescue

Dynamic Array in memory

Data
Size = 3
Capacity = 5

Linked List in memory

Linked List

Head
Size = 3

What can we now do...and not do ...quickly?
• Data elements held in structures called “links”
• Like a chain: each link is tied to the next

• Links are 1–1 with elements, allocated and released as necessary
struct Link {
    TYPE val; /* Data contained by this link. */
    struct Link *next; /* Pointer to next link. */
};
Linked List Variations

List

8 → 5 → 13

List

8 → 5 → 13

List

next
Implementing a stack interface with a linked list:

- Header with head reference only: null if empty
- Null terminated
- Singly linked

**Where should the ‘top’ of the stack be????**
struct linkedListStack {
    struct Link *firstLink; /* Initialize routine sets to zero/NULL. */
};

void linkedListStackInit (struct linkedListStack s) {
    s->firstLink = 0;
}
void pushListStack(struct ListStack *s, TYPE d) {
    /* You are going to write this:
       1. Allocate (malloc) a new link (check that it works!).
       2. Set data fields in the new link.
       3. Change head to point to new link. */
}

• Draw the diagram!
• Go through the steps visually, labeling each step
• Convert each step to C code
• Try the boundary cases:
  – Empty list?
  – List with several items?
• How do you tell if stack is empty?
• How do you return first element (i.e., firstLink)?
• How do you remove an element?
Your Turn:

Worksheet 17

- pushLinkedListStack
- popLinkedListStack