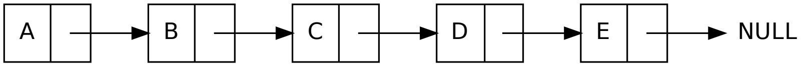
# CS 241
# Data Organization
# Linked Lists

Brooke Chenoweth

University of New Mexico

Fall 2014

## Linked List



- A *linked list* is a *data structure* that consists of a sequence of data records such that in each record there is a field that contains a reference (a link) to the next record in the sequence.
- In the C programming language, the “link” is usually implemented as a pointer. The link could, however, be implemented in other ways (i.e. as an array index).

## Why Linked Lists?

Why not just stick with arrays?

- The order of the linked items may be different from the order that the data items are stored in memory or on disk.
- Size is not predetermined, so can make better use of memory.
- We can insert and remove elements without having to reorganize the entire structure.

## Linked Lists in File Systems

- Most file systems store data as linked lists of data blocks.
- “Defragmenting” a hard disk moves the blocks to maximize the number of blocks that are physically adjacent.

## Linked List Access and Insertion

- Access to the $i^{th}$ element requires walking the list from the beginning and counting links to i. Such a process is said to have a time complexity of O(n).
- Insertion or Deletion at a known access point has a constant time complexity time O(1).

## Basic Structure

Nodes are self-referential structures.

```
struct ListNode
{
  int data;
  struct ListNode* next;
};
```

## Initializing

```
struct ListNode* createNode(int data)
{
  struct ListNode* node =
              malloc(sizeof(struct ListNode));
  node->data = data;
  node->next = NULL;
  return node;
}
```

## Looking through a list

```c
void printlist(struct ListNode* head)
{
  struct ListNode* current = head;
  while (current != NULL)
  {
    printf("%d ", current->data);
    current = current->next;
  }
  printf("\n");
}
```

## List Length

```c
int listlength(struct ListNode* head)
{
  struct ListNode* current = head;
  int count = 0;

  while (current != NULL)
  {
    count++;
    current = current->next;
  }
  return count;
}
```

## Inserting an element

- Beginning of the list
- Middle of the list
- End of the list

## Insert at beginning

```
struct ListNode* newNode = createNode(data);
newNode->next = head;
head = newNode;
```

## Insert in middle

```c
struct ListNode* newNode = createNode(data);
newNode->next = currentNode->next;
currentNode->next = newNode;
```

## Remove from beginning

```
struct ListNode* node = head;
head = node->next;
free(node);
```