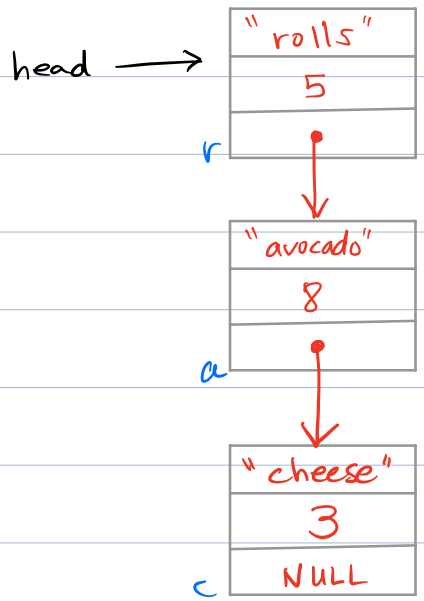
Linked Lists

We often want to organize data items into a structure.

An array is a kind of linear structure that is somewhat rigid
- you need to know its size right from start
- if you want to insert or remove an item you may need to shift the entire array
- hard to concatenate two arrays

For these reasons among others, we may want to use linked lists

Grocery (linked) List




```
struct ListNode {
    string item;
    int count;
    ListNode *next;
};
```

ListNode

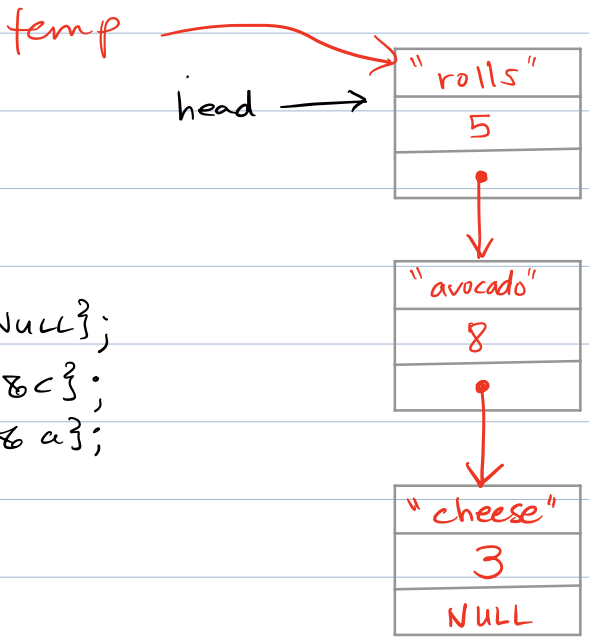
string item
int count
ListNode *next

```
ListNode c = {"cheese", 3, NULL};
ListNode a = {"avocado", 8, &c};
ListNode r = {"rolls", 5, &a};

ListNode *head = &r;
```

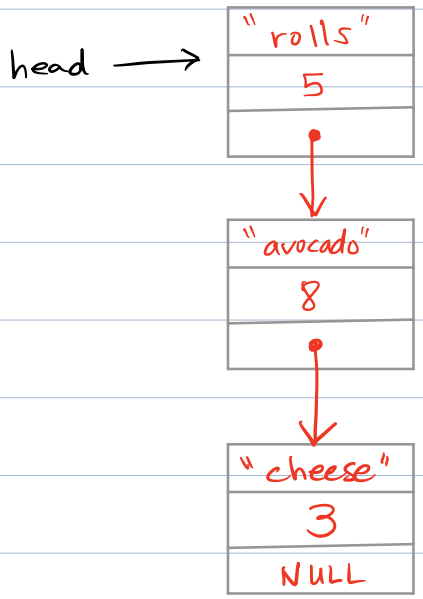
```
struct ListNode {
        string item;
        int count;
        ListNode *next;
};
ListNode c = {"cheese", 3, NULL};
ListNode a = {"avocado", 8, &c};
ListNode r = {"rolls", 5, &a};

ListNode *head = &r;

ListNode *temp = head;

while (temp != NULL) {
        cout << temp->count << " " << temp->item << endl;

        temp = temp->next;
} //end while
```

count pointed to by temp

item pointed to by temp



```
struct ListNode {
        string  item ;
        int     count ;
        ListNode *next ;
};
ListNode  c = {"cheese", 3, NULL};
ListNode  a = {"avocado", 8, &c};
ListNode  r = {"rolls", 5, &a};

ListNode *head = &r;
```




```
for (ListNode *temp = head; temp != NULL; ____________){
        cout << ____________ << " " << ____________ << endl;
} //end for
```

count pointed to by temp

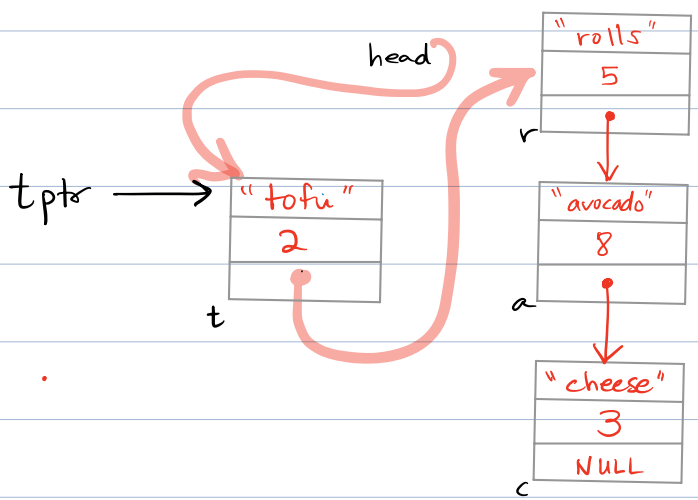
item pointed to by temp

When you have a struct and need to access a field...

use .

When you have a pointer to a struct and need to access a field.... use →

```
ListNode t, *tptr = &a;
  = {"tofu", 2, NULL};

cout << t.item << endl;
cout << tptr->item << endl;

//Now insert ListNode t into head of list

tptr->next = head;
head = tptr;
```



When you have a struct and need to access a field...

use •

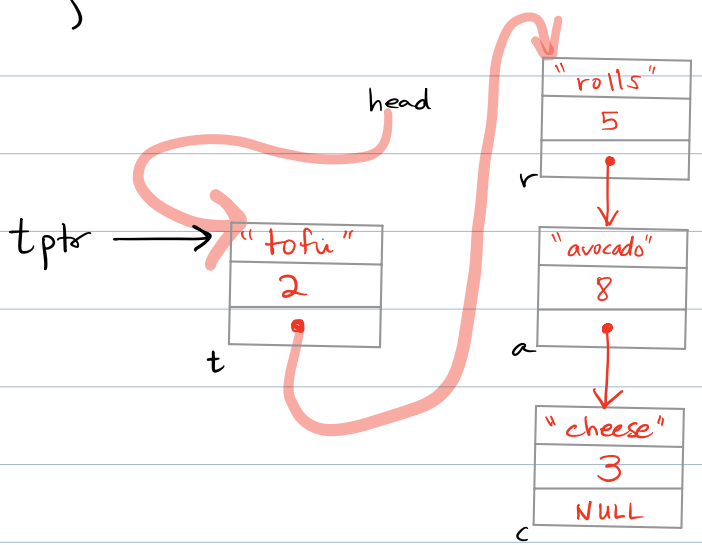
When you have a pointer to a struct and need to access a field.... use →

```
ListNode t, *tptr = &a;
  = {"tofu", 2, NULL};
```

```
cout << t.item << endl;
cout << tptr->item << endl;
```

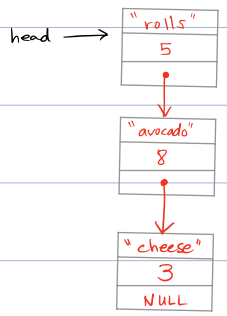
```
//Now insert ListNode t into head of list
```

```
tptr -> next = head;
head = tptr;
```



```
cout << (*tptr).item << endl;
cout << tptr -> item << endl;
```

You might notice that we are rarely using the "name" of a node (r , a , c) but are now managing access entirely through pointers. This is common, and C++ language designers have introduced **new** as a way to directly produce the memory location of a new struct and return a pointer to it.




```
struct ListNode {
    string item;
    int    count;
    ListNode *next;
};
```

```
// code fragment. Assumes head points to first element of a list, or is NULL if List is empty
ListNode *temp = NULL;

temp = new ListNode; // returns a pointer to the
                     // newly reserved space for a
                     // ListNode

temp->item = "sugar";
temp->count = 2;
temp->next = head;

head = temp;
```

Functions that can be useful in an app to maintain a grocery list.

```
struct ListNode {
    string item;
    int count;
    ListNode *next;
};
```

```
// read in a grocery list of item, count
// pairs and put them into an unsorte
// linked list
```

```
ListNode* getItem()
{
    ListNode *t = new ListNode;
    cout << "Enter item name: ";
    cin >> t -> item;
    cout << "Enter number of " << t->item << ": ";
    cin >> t -> count;
    t -> next = NULL;
    return t;
}
```