# اللوائح المترابطة Linked Lists

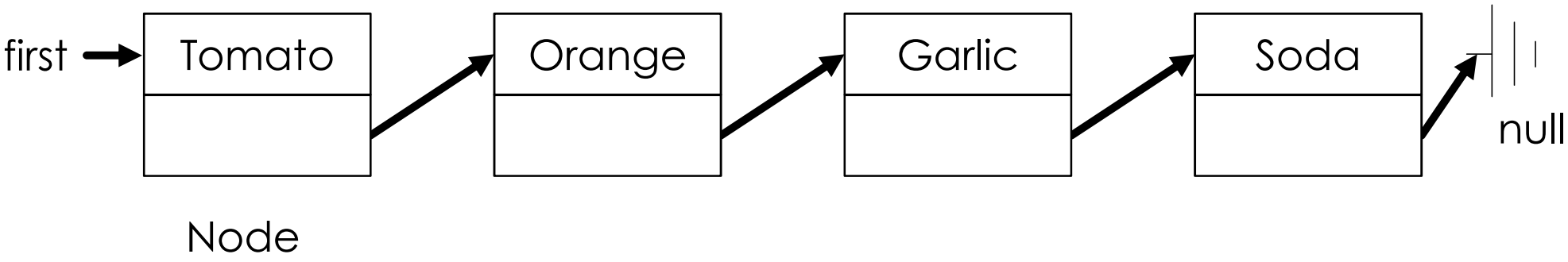
إعداد: م. يوسف دعبول

# اللوائح المترابطة Linked Lists

- اللائحة عبارة عن تتال منته من عناصر البيانات
  - مثال: قائمة المشتريات, قائمة الطعام في مطعم, قائمة الموظفين
  - المكدسات Stacks والأرتال Queues هي أنواع خاصة من اللوائح

# اللوائح المترابطة Linked Lists

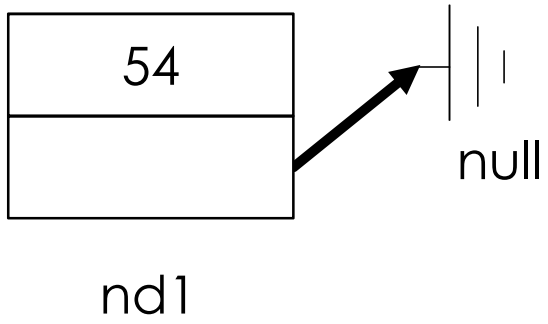
- اللائحة عبارة عن تتال منته من عناصر البيانات
  - مثال: قائمة المشتريات, قائمة الطعام في مطعم, قائمة الموظفين
  - المكدسات Stacks والأرتال Queues هي أنواع خاصة من اللوائح



# تخزين العقدة Node

```
class Node {
    public:
        int data;
        Node * next;
}
Node nd1;
nd1.data = 54;
nd1.next = 0;

Node * nd2 = new Node();
Nd2->data = 54;
```



# تحقيق اللوائح المترابطة باستخدام المؤشرات

- أولا: نحن بحاجة للصنف Node والذي سيتم اشتقاق العقد منه

```
class Node {
    public:
        int data;
        Node * next;
        Node(int data, Node* next) {
            this.data = data;
            this.next = next;
        }
}
```

```
Node node1(53, null);

Node* nd1 = &node1;
Node* nd2 = new Node(10, null);

Nd1->next = nd2;
nd2->next = &node1;
```

# تحقيق اللوائح المترابطة باستخدام المؤشرات

- ثانيا: نحن بحاجة لبناء الصنف LinkedList وهو صنف اللائحة المترابطة

```
class LinkedList {
    private:
        Node * first;
        int size;
    public:
        //هنا نعرف التوابع الأعضاء
}
```

- تعريف الصنفين Node و LinkedList
  - يمكن تعريف كلا الصنفين في نفس الملف تلو بعضهم
  - يمكن تعريف كل صنف في ملف
  - يمكن تعريف الصنف Node في الجزء الخاص من الصنف LinkedList

# العمليات الأساسية

1. البناء Construction
2. اختبار كون اللائحة فارغة isEmpty
3. التجوال عبر اللائحة traverse
4. جلب عنصر من اللائحة getNode
5. إضافة عنصر insert
   - إضافة عنصر إلى مكان محدد في اللائحة
6. حذف عنصر delete
   - حذف عنصر من مكان محدد في اللائحة

# التابع الباني Constructor

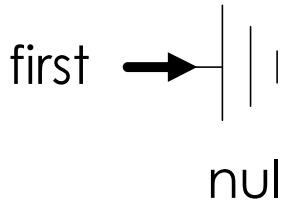
- يقوم بحجز المؤشر first والمتغير size في الذاكرة
- يقوم بتهيئتهما بالقيمة 0

```
class LinkedList {
    private:
        Node * first;
        int size;
    public:
        LinkedList() {
            first = 0;
            size = 0;
        }
}
```

# اختبار كون اللائحة فارغة

- اللائحة الفارغة لا تؤشر فيها البداية إلى أي شيء null

first == 0



# تابع اختبار كون اللائحة فارغة isEmpty

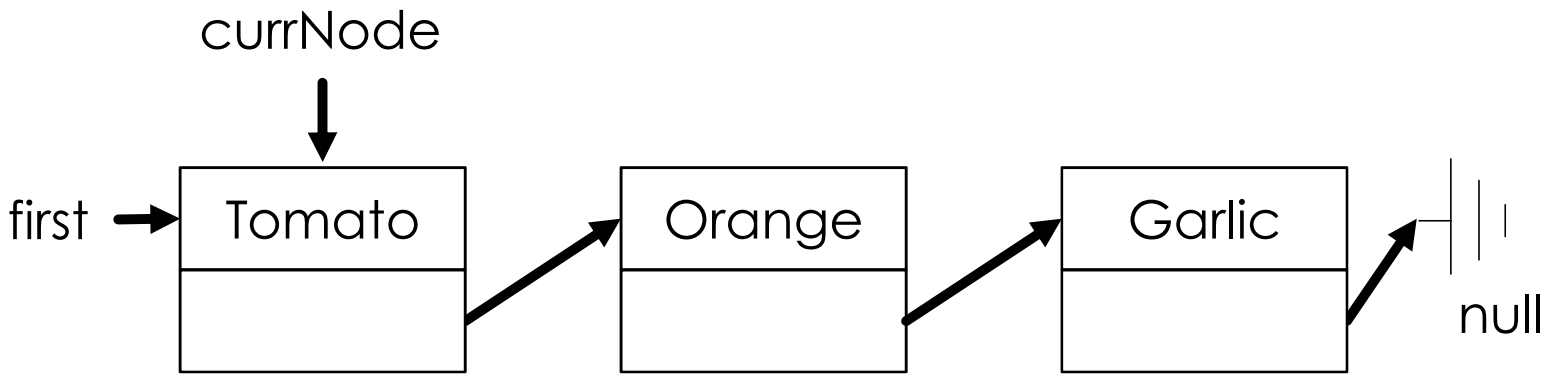
- يتأكّد من كون المؤشر first لا يشير إلى شيء

```
class LinkedList {
    private:
        Node * first;
        int size;
    public:
        bool isEmpty() {
            return (first == 0);
        }
}
```

# التجوال عبر اللائحة traverse

- نستخدم مؤشر للتنقل بين العناصر بدءا من العنصر الأول

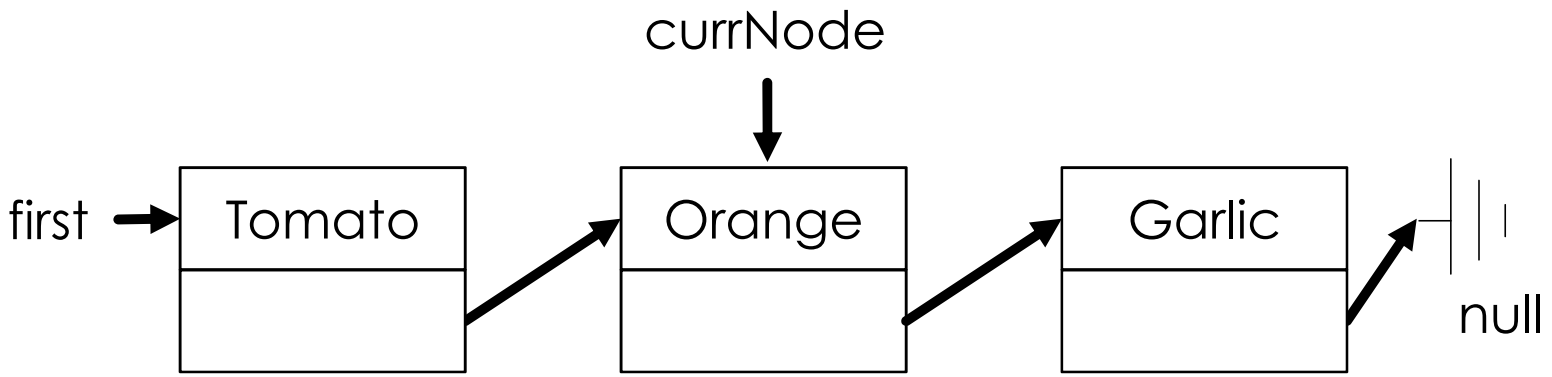
```
Node * currNode = first;
While ( currNode != null ) {
        currNode = currNode->next;
}
```



# التجوال عبر اللائحة traverse

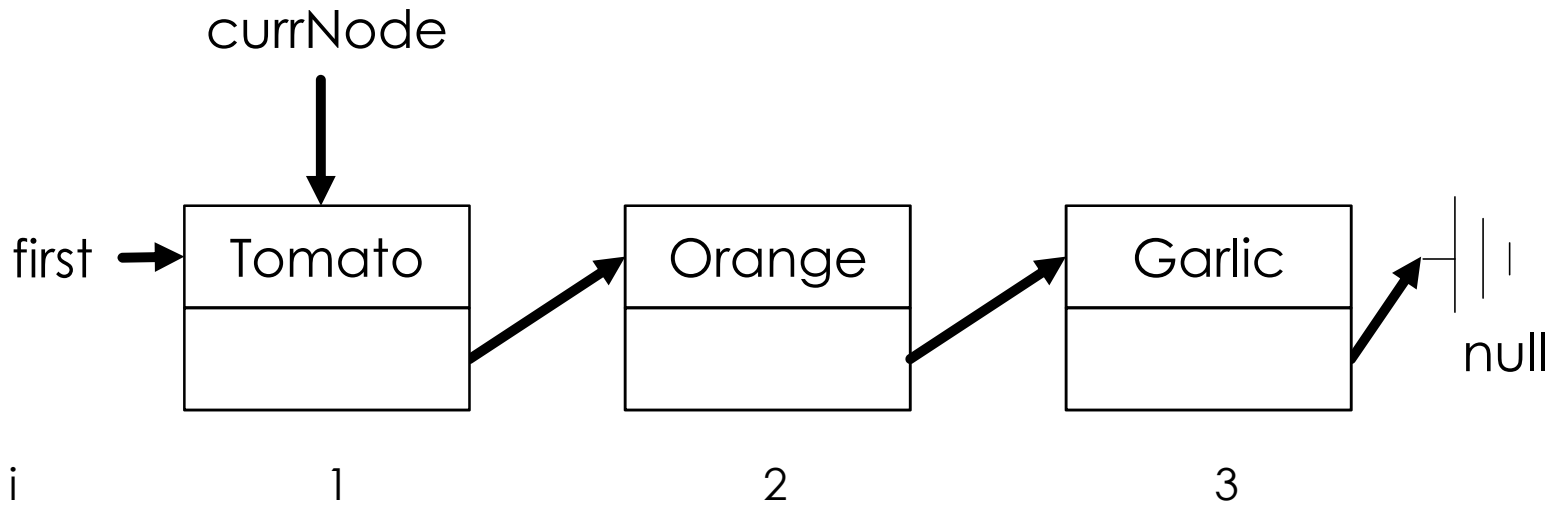
- نستخدم مؤشر للتنقل بين العناصر بدءا من العنصر الأول

```
Node * currNode = first;
While ( currNode != null ) {
        currNode = currNode->next;
}
```



# تابع جلب عنصر من اللائحة getNode

- يقوم بإعادة مؤشر إلى العقدة المطلوبة



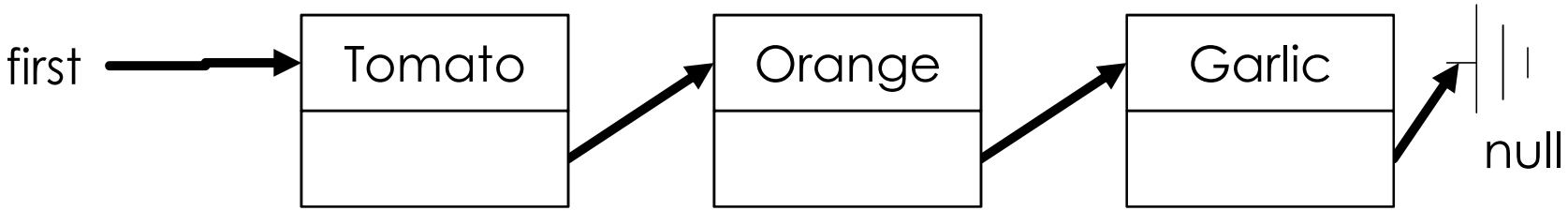
# تابع جلب عنصر من اللائحة getNode

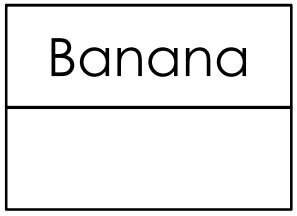
- يقوم بإعادة مؤشر إلى العقدة المطلوبة

```
class LinkedList {
    public:
        Node* getNode(int pos) {
            int i = 1;
            Node * currNode = first;
            while (i != pos and currNode->next != null) {
                currNode = currNode->next;
                i++;
            }
            return currNode;
        }
}
```

# إضافة عنصر insert

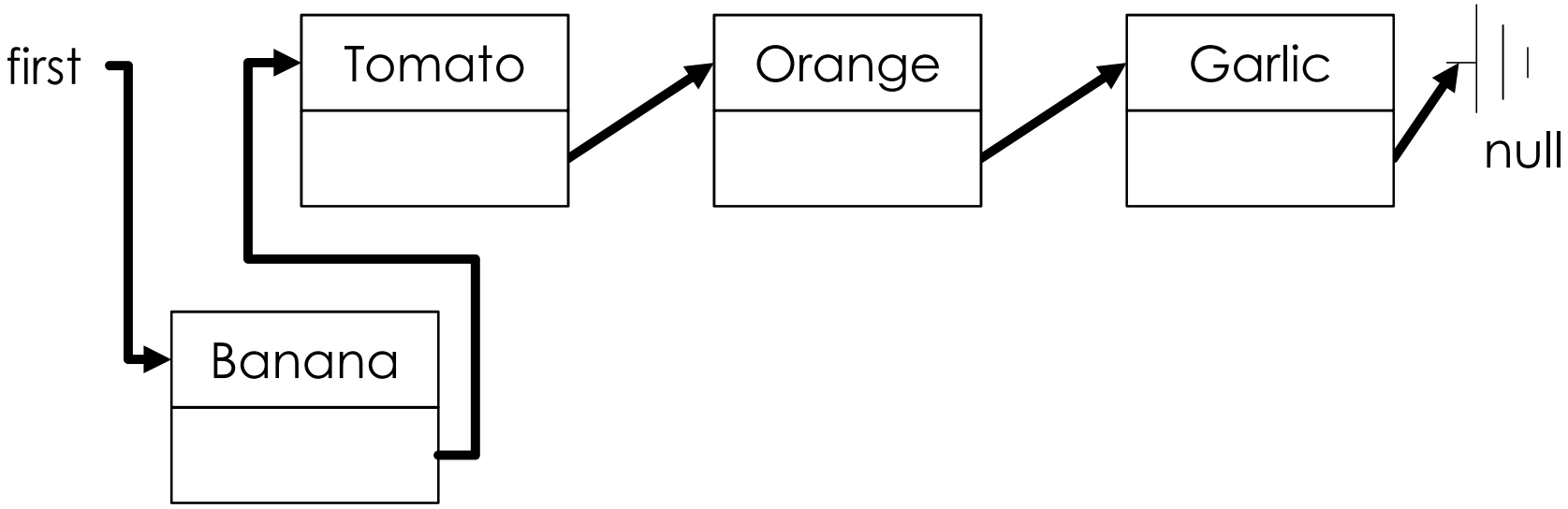
- الحالة الأولى: الحشر في بداية اللائحة
- نجعل مؤشر العنصر الجديد يشير إلى العنصر الأول
- نجعل مؤشر البداية first يشير إلى العنصر الجديد



# إضافة عنصر insert

- الحالة الأولى: الحشر في بداية اللائحة
- نجعل مؤشر العنصر الجديد يشير إلى العنصر الأول
- نجعل مؤشر البداية first يشير إلى العنصر الجديد

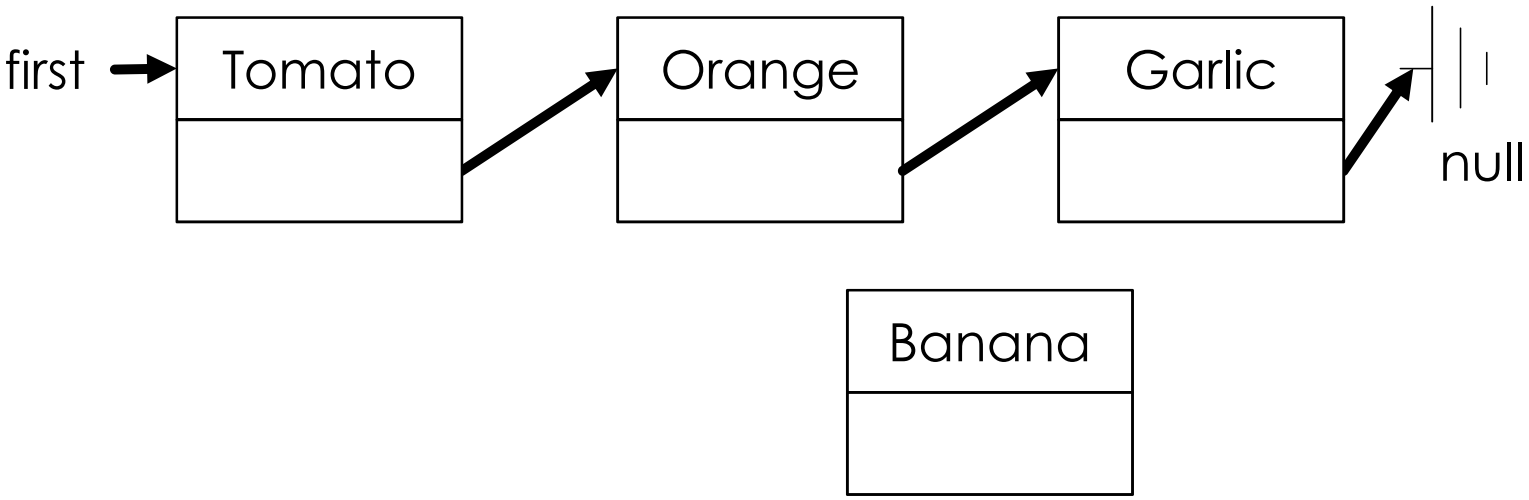


# تابع إضافة عنصر إلى اللائحة insert

```
class LinkedList {
    public:
        void insert(int pos, int value) {
            Node* newNode = new Node(value, 0);
            if (pos == 1) {
                newNode->next = first;
                first = newNode;
            } else {
                ...
            }
        }
}
```

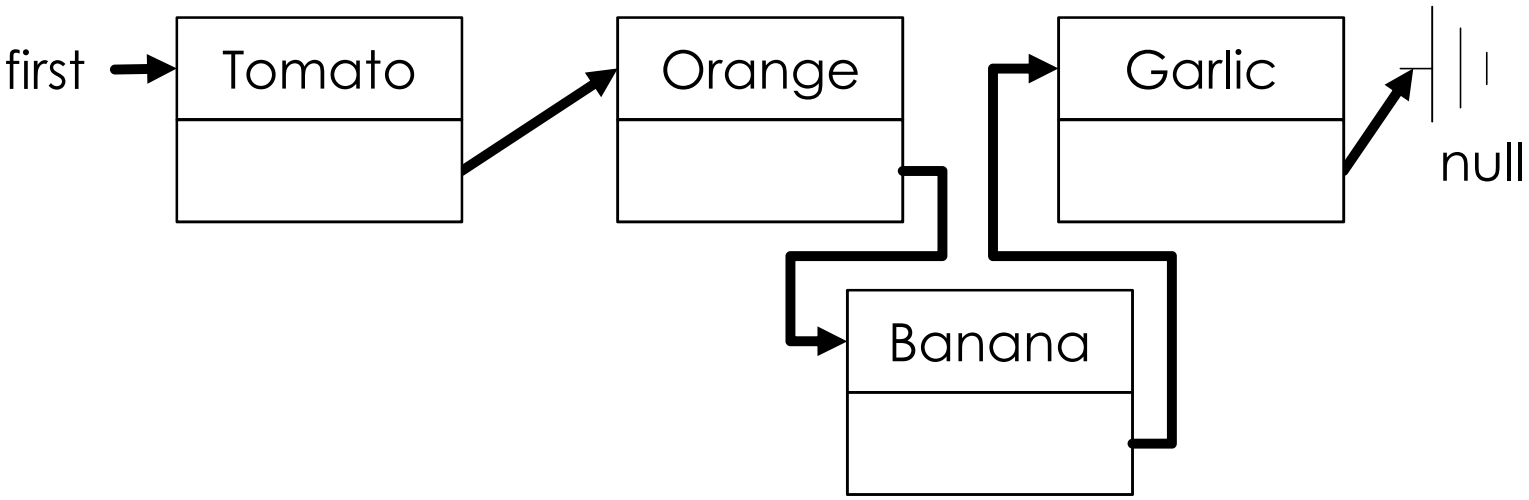
# إضافة عنصر insert

- الحالة الثانية: الحشر بعد عنصر ما في اللائحة
- نجعل مؤشر العنصر الجديد يشير إلى العنصر التالي للعنصر السابق
- نجعل مؤشر العنصر السابق يشير إلى العنصر الجديد



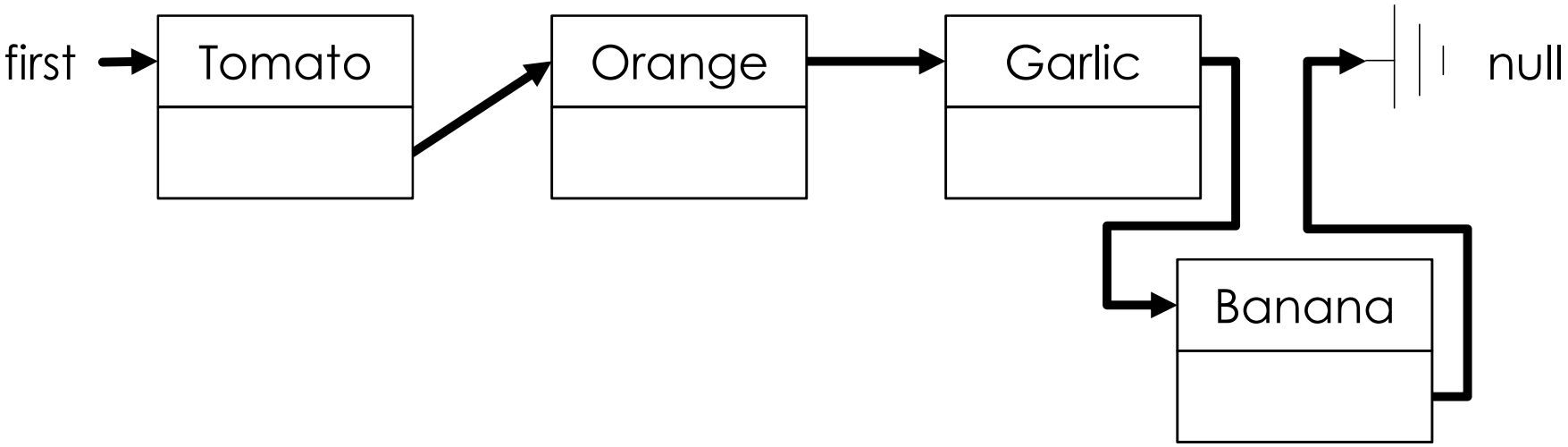
# إضافة عنصر insert

- الحالة الثانية: الحشر بعد عنصر ما في اللائحة
- نجعل مؤشر العنصر الجديد يشير إلى العنصر التالي للعنصر السابق
- نجعل مؤشر العنصر السابق يشير إلى العنصر الجديد



# إضافة عنصر insert

- الحالة الثانية: الحشر بعد عنصر ما في اللائحة
- نجعل مؤشر العنصر الجديد يشير إلى العنصر التالي للعنصر السابق
- نجعل مؤشر العنصر السابق يشير إلى العنصر الجديد

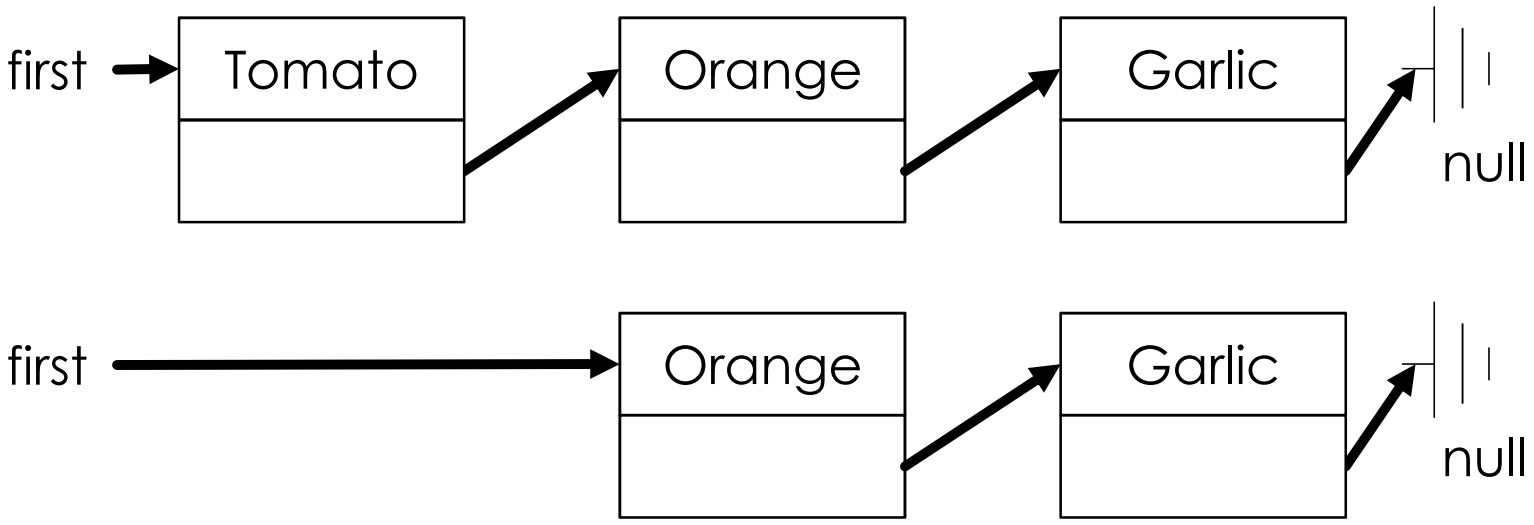


# تابع إضافة عنصر إلى اللائحة insert

```
class LinkedList {
    public:
        void insert(int pos, int value) {
            Node* newNode = new Node(value, 0);
            if (pos == 1) {
                newNode->next = first;
                first = newNode;
            } else {
                Node* preNode = this.getNode(pos - 1);
                newNode->next = preNode->next;
                preNode->next = newNode;
            } size++;
        }
}
```

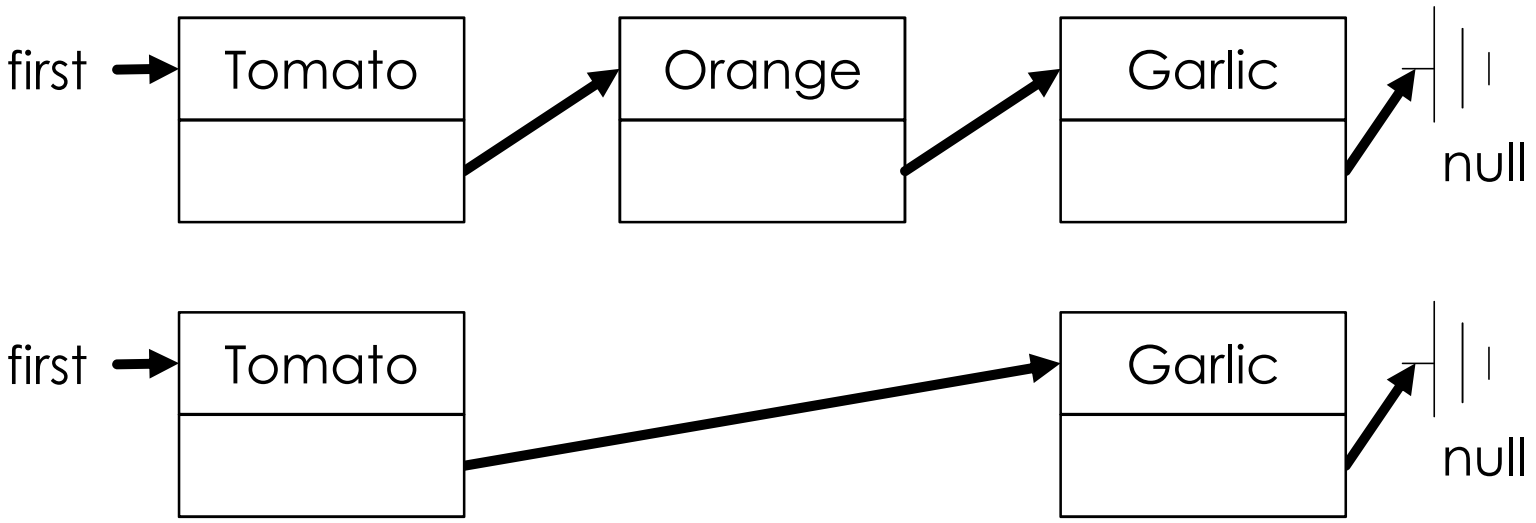
# حذف عنصر delete

- الحالة الأولى: حذف العنصر الأول في اللائحة
- نجل مؤشر البداية يشير إلى العنصر التالي للعنصر الأول



# حذف عنصر delete

- الحالة الثانية: حذف عنصر غير العنصر الأول
- نجل مؤشر العنصر السابق يشير للعنصر التالي للعنصر المحذوف



# تابع حذف عنصر من اللائحة delete

```
class LinkedList {
    public:
        void delete(int pos) {
            Node* currNode = this.getNode(pos);
            if (pos == 1) {
                first = currNode->next;
            } else {
                Node* preNode = this.getNode(pos - 1);
                preNode->next = currNode->next;
            }
            delete currNode; size--;
        }
}
```