CS106B Supplement to Section 4 (Week 5) - Solutions

---

0. Pre-section

part 1:

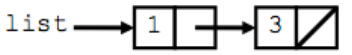
```
v1 : 70
v2 : 25
p1 : points to v1
p2 : points to v1
```

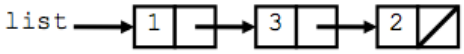
part 2:

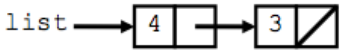
```
list->next->next->next->next = list->next->next; // 40->30
list->next->next = list->next->next->next;       // 20->40
list->next->next->next->next = list;             // 30->10
list = list->next;                               // list->20
list->next->next->next->next = null;             // make 10 the end
```

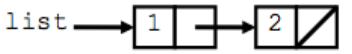
---

1. LinkedNodes (1)

a) 

b) 

c) 

d) 

---

2. LinkedNodes (2)

a)
```
list->next->next = new ListNode(3, NULL);    // 2 -> 3
```

b)
```
list = new ListNode(3, list);            // 3 -> 1  and  list -> 3
```

c)
```
temp->next->next = list->next;               // 4 -> 2
list->next = temp;                           // 1 -> 3
```

d)
```
list->next->next = temp->next;               // 2 -> 4
temp->next = list->next;                     // 3 -> 2
list->next = temp;                           // 1 -> 3
```

e)
```
ListNode* list2 = list;                      // list2 -> 1
list = list->next;                           // list -> 2
list2->next = list2->next->next;             // 1 -> 3
list->next = NULL;                           // 2 /
```

f)
```
ListNode* temp = list->next->next;           // temp -> 3
temp->next = list->next;                     // 3 -> 4
list->next->next = list;                     // 4 -> 5
list->next->next->next = NULL;               // 5 /
list = temp;                                 // list -> 3
```

g)
```
list->next->next->next = list;               // 3 -> 5
list = list->next->next;                     // list -> 3
ListNode* list2 = list->next->next;          // list2 -> 4
list->next->next = NULL;                     // 5 /
```

CS106B Supplement to Section 4 (Week 5) - Solutions

---

3. min

```
int LinkedList::min() const {
    if (m_front == NULL) {
        throw "list is empty";
    } else {
        int min = m_front->data;
        ListNode* current = m_front->next;
        while (current != NULL) {
            if (current->data < min) {
                min = current->data;
            }
            current = current->next;
        }
        return min;
    }
}
```

---

4. isSorted

```
bool LinkedList::isSorted() const {
    if (m_front != NULL) {
        ListNode* current = m_front;
        while (current->next != NULL) {
            if (current->data > current->next->data) {
                return false;
            }
            current = current->next;
        }
    }
    return true;
}
```

---

5. countDuplicates

```
int LinkedList::countDuplicates() const {
    int count = 0;
    if (m_front != NULL) {
        ListNode* current = m_front;
        while (current->next != NULL) {
            if (current->data == current->next->data) {
                count++;
            }
            current = current->next;
        }
    }
    return count;
}
```

---

6. stutter

```
void LinkedList::stutter() {
    ListNode* current = m_front;
    while (current != NULL) {
        current->next = new ListNode(current->data, current->next);
        current = current->next->next;
    }
}
```

CS106B Supplement to Section 4 (Week 5) - Solutions

---

7. deleteBack

```
int LinkedList::deleteBack() {
    if (m_front == NULL) {
        throw "empty list";
    }
    int result = 0;
    if (m_front->next == NULL) {
        result = m_front->data;
        delete m_front;
        m_front = NULL;
    } else {
        ListNode* current = m_front;
        while (current->next->next != NULL) {
            current = current->next;
        }
        result = current->next->data;
        delete current->next;
        current->next = NULL;
    }
    return result;
}
```

---

8. split

```
void LinkedList::split() {
    if (m_front != NULL) {
        ListNode* current = m_front;
        while (current->next != NULL) {
            if (current->next->data < 0) {
                ListNode* temp = current->next;
                current->next = current->next->next;
                temp->next = m_front;
                m_front = temp;
            } else {
                current = current->next;
            }
        }
    }
}
```

---

9. removeAll

```
void LinkedList::removeAll(int value) {
    while (m_front != NULL && m_front->data == value) {
        ListNode* trash = m_front;
        m_front = m_front->next;
        delete trash;
    }
    if (m_front != NULL) {
        ListNode* current = m_front;
        while (current->next != NULL) {
            if (current->next->data == value) {
                ListNode* trash = current->next;
                current->next = current->next->next;
                delete trash;
            } else {
                current = current->next;
            }
        }
    }
}
```

CS106B Supplement to Section 4 (Week 5) - Solutions

---

10. doubleList

```
void LinkedList::doubleList() {
    if (m_front != NULL) {
        ListNode* half2 = new ListNode(m_front->data);
        ListNode* back = half2;
        ListNode* current = m_front;
        while (current->next != NULL) {
            current = current->next;
            back->next = new ListNode(current->data);
            back = back->next;
        }
        current->next = half2;
    }
}
```

---

11. rotate

```
void LinkedList::rotate() {
    if (m_front != NULL && m_front->next != NULL) {
        ListNode* temp = m_front;
        m_front = m_front->next;
        ListNode* current = m_front;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = temp;
        temp->next = NULL;
    }
}
```

---

12. reverse

```
void LinkedList::reverse() {
    ListNode* current = m_front;
    ListNode* previous = NULL;
    while (current != NULL) {
        ListNode* nextNode = current->next;
        current->next = previous;
        previous = current;
        current = nextNode;
    }
    m_front = previous;
}
```