Linked List - I

Abstract Data Type (ADT)

- ADT is a data type whose properties (domain and operations) are specified independently of any particular implementation.
  - Logical (or ADT) level: abstract view of the domain and operations.
  - Implementation level: specific representation of the structure to hold the data items, and the coding for operations.

4 Basic Kinds of ADT Operations

- Constructor/Destructor - creates/deletes a new instance (object) of an ADT.
- Transformer -- changes the state of one or more of the data values of an instance.
- Observer -- allows us to observe the state of one or more of the data values of an instance without changing them.
- Iterator -- allows us to process all the components in a data structure sequentially.
List

• **Linear relationship** Each element except the first has a unique predecessor, and each element except the last has a unique successor.
• **Length** The number of items in a list; the length can vary over time.

Class ShoppingList

class ShoppingList
{
public:
  int LengthIs() const;
  void RetrieveItem(int pos);
  void InsertItem(const Item& item, int pos);
  void RemoveItem(int pos);
  void DeleteAll();
private:
  int size;
  Node* front;
};

Singly Linked List Diagram

[Diagram of a singly linked list showing pointers to nodes and data]
Class Node

```cpp
class Node {
public:
    Node(const Item & anItem, int pos) :
        item(anItem),
        size(pos)
    { /* Constructor */ }

private:
    Item item;
    Node *link;
};
```

Front Insert Example (1 of 6)

```cpp
void ShoppingList::insert(const Item & anItem, int pos) {
    prev = cur = NULL;
    Node *newNode = new Node(anItem, pos);
    cur = newNode;
    if (pos == 0) {
        cur->link = front;
        front = cur;
    } else {
        pos--;
        while (pos > 0) {
            prev = cur;
            cur = cur->link;
            pos--;
        }
        prev->link = newNode;
        newNode->link = cur;
    }
}
```

Front Insert Example (2 of 6)

```cpp
assert(pos >= 0 && pos <= size);
size++;
```
if (pos == 0) { // Inserting at the front
    cur = front;
}

front = new Node(anItem);

front->link = cur;
Front Insert Example (6 of 6)

```cpp
return;
}
```

Empty Insert Example (1 of 6)

```cpp
void ShoppingList::insert(const Item & anItem, int pos) {
    Node *prev, *cur;
    Inserting A at position 0 in empty list
    cur
    pos
    anItem
    size
    front
    0
}```

Empty Insert Example (2 of 6)

```cpp
assert(pos >= 0 && pos <= size);
size++;
```
Empty Insert Example (3 of 6)

```java
if (pos == 0) { // Inserting at the front
    cur = front;
}
```

Empty Insert Example (4 of 6)

```java
front = new Node(anItem);
```

Empty Insert Example (5 of 6)

```java
front->link = cur;
```
Empty Insert Example (6 of 6)

```cpp
return;
}
```

Middle Insert Example (1 of 8)

```cpp
void ShoppingList::insert(const Item & anItem, int pos) {
    Node *prev, *cur;
    assert(pos >= 0 && pos <= size);
    size++;
    // Inserting D at position 2
    // state: prev, cur, item, pos
    // state: 3
    // state: A, B, C
    // state: D
    // state: 2
    // state: A, B, C, D
    // state: A, B, D, C
    // state: A, D, B, C
    // state: D, A, B, C
}
```
Middle Insert Example (3 of 8)

```c
prev = NULL;
cur = front;
```

![Diagram of linked list with pointers]

```c
while (pos > 0) {
    prev = cur;
cur = cur->link;
pos--;
}
```

Middle Insert Example (4 of 8)

```c
First Iteration
```

![Diagram of linked list with pointers after first iteration]

Middle Insert Example (5 of 8)

```c
Second Iteration
```

![Diagram of linked list with pointers after second iteration]

```c
while (pos > 0) {
    prev = cur;
cur = cur->link;
pos--;
}
```
Middle Insert Example (6 of 8)

```java
prev->link = new Node(anItem);
```

Middle Insert Example (7 of 8)

```java
prev->link->link = cur;
```

Middle Insert Example (8 of 8)
void ShoppingList::insert(const Item & anItem, int pos) {
  Node *prev, *cur;
  
  Inserting D at position 3

  assert(pos >= 0 && pos <= size);
  size++;

  prev = 0;
  cur = front;

  // More code here...
}
while (pos > 0) {
  prev = cur;
  cur = cur->link;
  pos--;
}

---

End Insert Example (4 of 8)

while (pos > 0) {
  prev = cur;
  cur = cur->link;
  pos--;
}

---

End Insert Example (5 of 8)

while (pos > 0) {
  prev = cur;
  cur = cur->link;
  pos--;
}

---

End Insert Example (6 of 8)

while (pos > 0) {
  prev = cur;
  cur = cur->link;
  pos--;
}
```c
void ShoppingList::insert(const Item & anItem, int pos) {
    Node *prev, *cur;
    assert(pos >= 0 && pos <= size);
    size++; 
    if (pos == 0) { // Inserting at the front
        cur = front;
        front = new Node(anItem);
        front->link = cur;
        return;
    }
    prev = NULL;
    cur = front;
    while (pos > 0) {
        prev = cur;
        cur = cur->link;
        pos--;
    }
    prev->link = new Node(anItem);
    prev->link->link = cur;
}
```
Remove Example (1 of 7)

```c
void ShoppingList::remove(int pos) {
    Node *prev, *cur;
    Removing at position 1
    front
    ABC
    item link item link item link
    prev cur pos
```

```c
size = 3
```

Remove Example (2 of 7)

```c
assert(pos >= 0 && pos < size);
size--;
```

```c
prev = NULL;
cur = front;
```

Remove Example (3 of 7)

```c
front
ABC
item link item link item link
prev cur pos
```

```c
size = 2
```
while (pos > 0) {
    prev = cur;
    cur = cur->link;
    pos--;
}

First iteration
Delete the node pointed by cur

Remove Example (5 of 7)

prev->link = cur->link;

Remove Example (6 of 7)

delete cur;
void ShoppingList::deleteAll() {
    Node *kil;
    while (front != NULL) {
        kil = front;
        front = front->link;
        // First iteration
    }
}
Delete All Example (3 of 8)

```c
kil->link = NULL;
delete kil;
```

First iteration

```
A --- B --- C
```

Delete All Example (4 of 8)

```c
while (front != NULL) {
    kil = front;
    front = front->link;
}
```

Second iteration

```
B --- C
```

Delete All Example (5 of 8)

```c
kil->link = NULL;
delete kil;
```

Second iteration

```
B --- C
```
while (front != NULL) {
    kil = front;
    front = front->link;
    kil->link = 0;
    delete kil;
}

size = 0;