Lists

(ordered set of data objects)

Sequential representation

- successive nodes of the data object are stored a fixed distance apart → array 구조
- (physical) order of elements is the same as in ordered list
- adequate for functions such as accessing an arbitrary node in a table
- operations such as insertion and deletion of arbitrary elements from ordered lists become expensive

Linked representation

- successive items of a list may be placed anywhere in memory (효율성)
- (physical) order of elements need not be the same as order in list
- each data item is associated with a pointer (link) to the next item
- We will study singly linked lists (SLL)

List ADT (description of functionality of list data structure)

```
template < class T>
class List {
public:
    List();
    ~List();
    void insert(int loc, T d);
    void remove(int loc);
    int getSize();
    T & getData(int loc);
private:
    // not yet !!
};
```

```
int main() {
    List<int> mylist;
    mylist.insert(1, 10);
    mylist.insert(2, 20);
    mylist.insert(3, 15);
    mylist.remove(2);
    mylist.insert(1, 35);
    mylist.insert(3, mylist.getData(2));

    cout << mylist.getSize() << endl;
    return 0;
}
```

Implementation by Array

```
template < class T>
class List {
public:
    List() { size = 0; }
    //~List();
    void insert(int loc, T d);
    void remove(int loc);
    int getSize();
    T & getData(int loc);
private:
    T data[5];
    int size;
};
```

insert a new data object?

remove an existing data object?

Implementation by Links

```
template < class T>
class List {
private:
    class listNode {
    public:
       T data;
       listNode *next;
       listNode(T newItem);
    };
public:
    List() : size(0), head(NULL) { }
    ~List();
    listNode * Find(listNode *nptr, int k);
    void insert(int loc, T d);
    void remove(int loc);
    int getSize() const;
    T & getData(int loc);
private:
    listNode *head;
    int size;
};
```

- head
  - Point to an object of listNode in List.

- ~List();

✓ Need to delete dynamically allocated objects of listNode.

- Find(nptr, k);

✓ Take k steps forward in the list from the object pointed by ‘nptr’.

✓ Array implementation 의 index 역할.

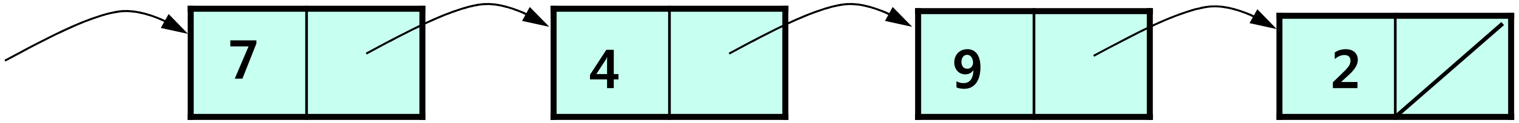
Implementation by Links (계속)

```
template < class T>

class List {
private:
    class listNode {
    public:
        T data;
        listNode *next;
        listNode(T newItem);
    };
public:
    List() { size = 0; head = NULL; }
    ~List();
    listNode * Find(listNode *nptr, int k);
    void insert(int loc, T d);
    void remove(int loc);
    int getSize() const;
    T & getData(int loc);
private:
    listNode *head;
    int size;
};
```

- class listNode { … }
  - Nested class of List class.
  - Only List functions can create objects of the private class listNode.
  - Only List functions can access listNode objects.
  - Implementation of constructor:

Insert a new node at the front

```
List<int> mylist;
.....
listNode *h = mylist.getHead();
mylist.insertAtFront(h, 10);
```

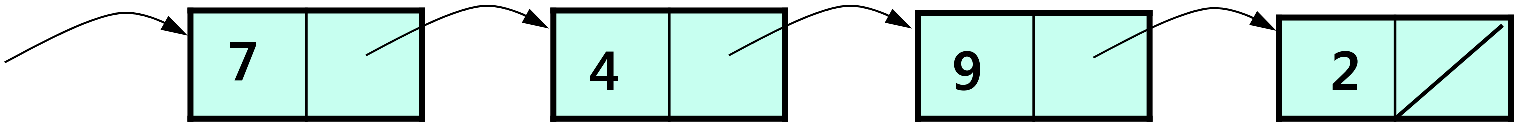


```
// Insert a new node with data d at the front of the list node
// pointed by curr. In addition, curr should be updated to point
// the new node.
void List<T>::insertAtFront(listNode * curr, T d) {









}
```

Time complexity:

For array case ?

Print data in list in reverse order

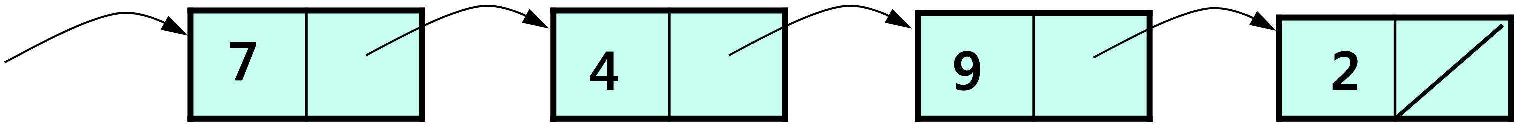


```
// 2 9 4 7 should be printed for the input of above list.
void List<T>::printReverse(listNode *curr) {








}
```

Time complexity:

For array case ?

Find a pointer to node k steps forward from *current



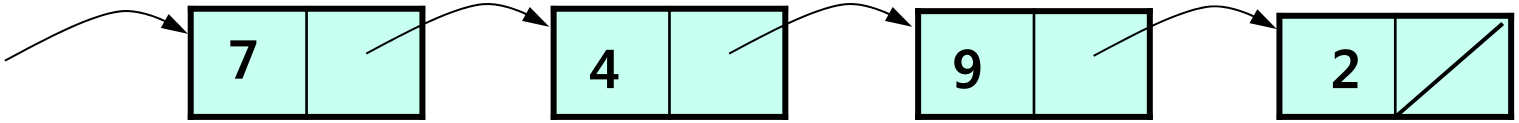
```
// returns pointer to node k step forward.
listNode * List<T>::Find(listNode *cur, int k) {









}
```

Time complexity:

For array case ?

Insert a new node in k-th position

```
List<int> mylist;
….
mylist.insert(3, 100);
```

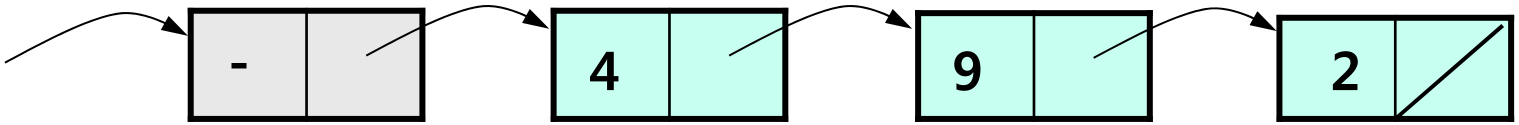


```
void List<T>::insert(int k, T d) {

   // (1) Create a new node
   listNode *temp = new listNode(d);


   // (2) Fix up pointers
   listNode *p = Find(head, k-2);// what if k is 1?
   temp->next = p->next;
   p->next = temp;
}
```

Time complexity:

For array case ?

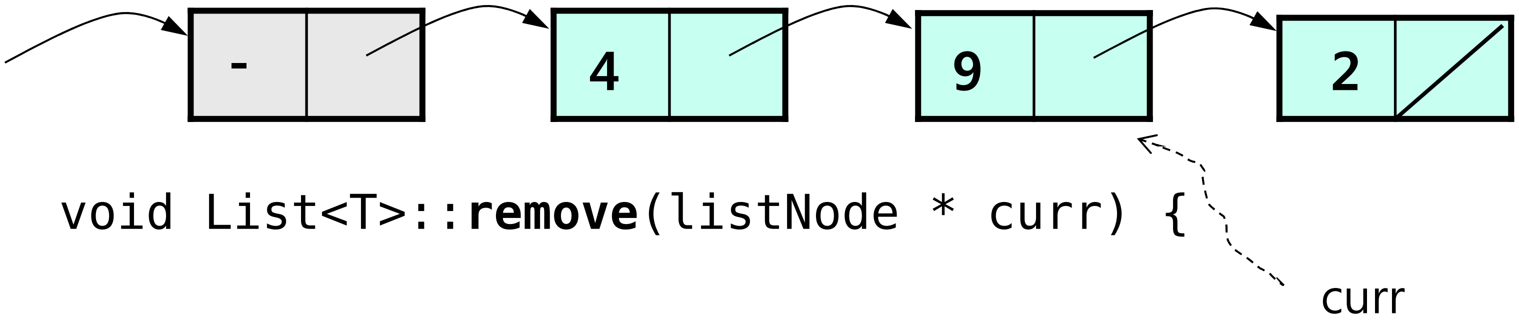
Insert a new node in a list with a sentinel node



```
void List<T>::insert(int k, T d) {
   // (1) Create a new node
   listNode *temp = new listNode(d);

   // (2) Fix up pointers
   listNode *p = Find(head, k-1);// is it OK?
   temp->next = p->next;
   p->next = temp;
}
```

Time complexity:

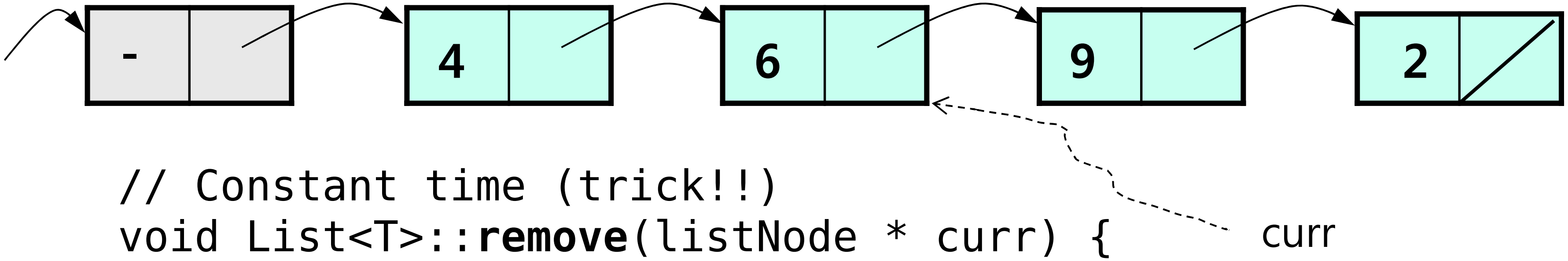
Remove a node in fixed position (given by a node pointer)



```
void List<T>::remove(listNode * curr) {







}
```

Time complexity:

Remove a node in fixed position (given by a node pointer)



```
// Constant time (trick!!)
void List<T>::remove(listNode * curr) {







}
```

Run times for List functions

	Singly linked List	Array
Insert/Remove at front	O(1)	O(1)
Insert at given location	O(1) : inserting a node after the given location	O(n) shift : inserting a node at the given index
Remove at given location	O(1) trick	O(n) shift
Insert at arbitrary location	O(n) find	O(n) shift
Remove at arbitrary location	O(n) find	O(n) shift