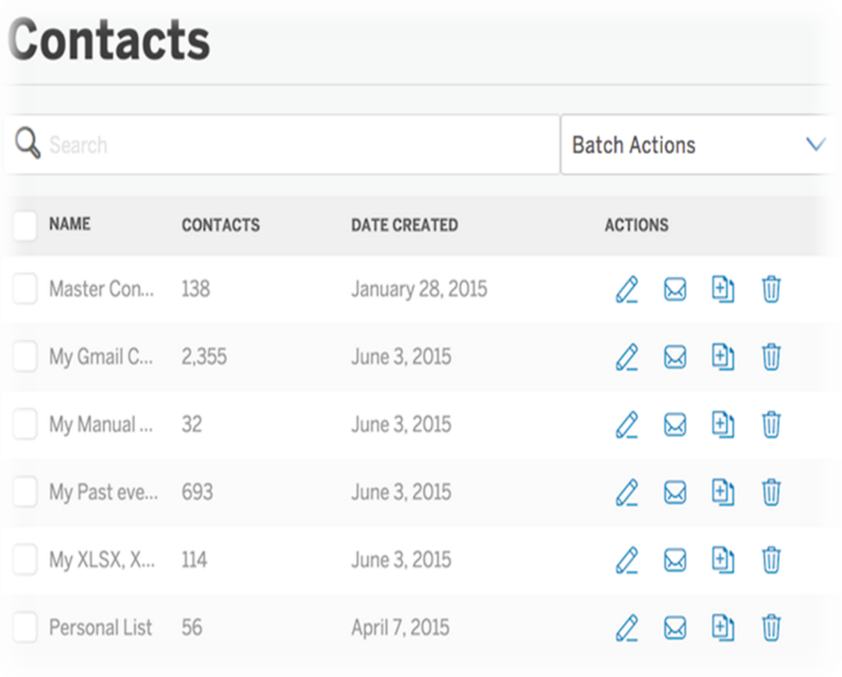
CS214-Data Structure

Lecturer: Dr. Salwa Osama

Linked Lists

List- What?



Contacts

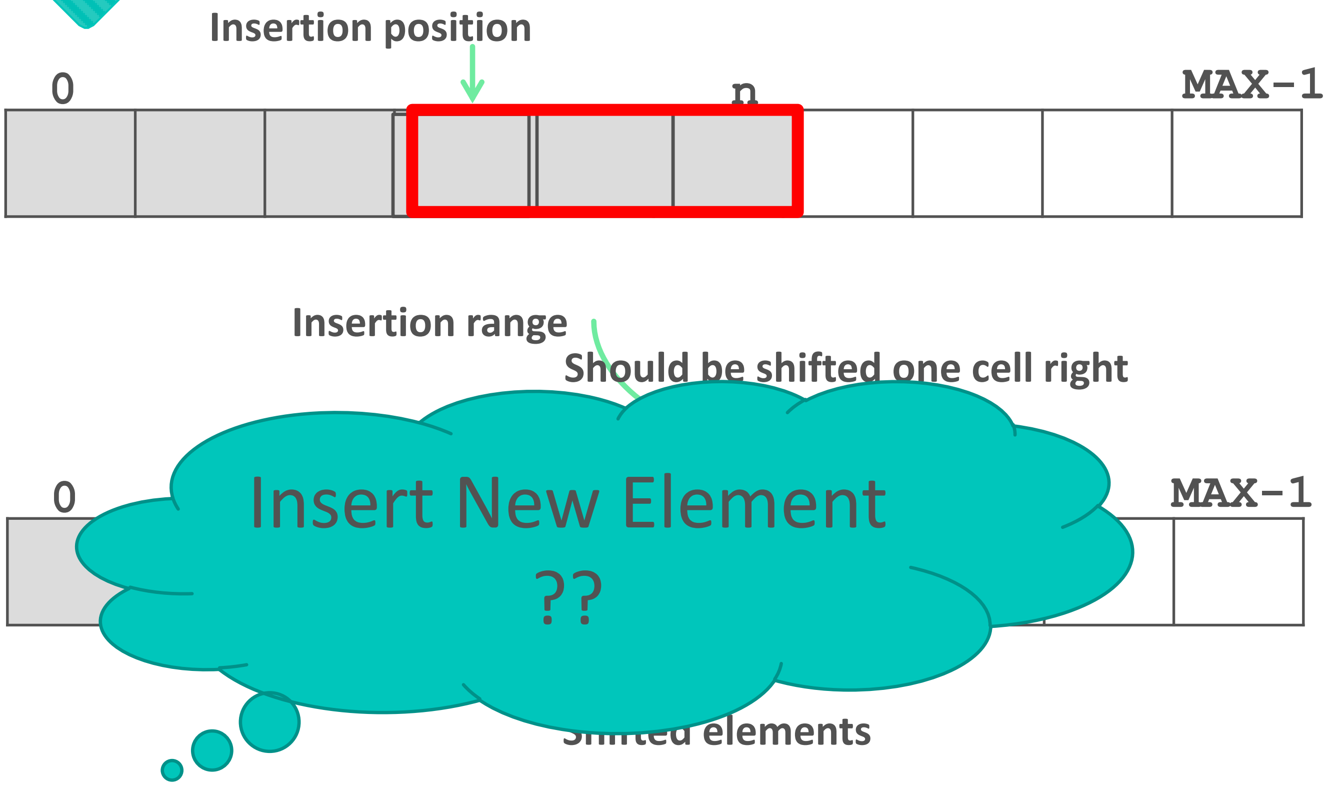
Search | Batch Actions

□	NAME	CONTACTS	DATE CREATED	ACTIONS
□	Master Con...	138	January 28, 2015	
□	My Gmail C...	2,355	June 3, 2015	
□	My Manual ...	32	June 3, 2015	
□	My Past eve...	693	June 3, 2015	
□	My XLSX, X...	114	June 3, 2015	
□	Personal List	56	April 7, 2015	

Lists

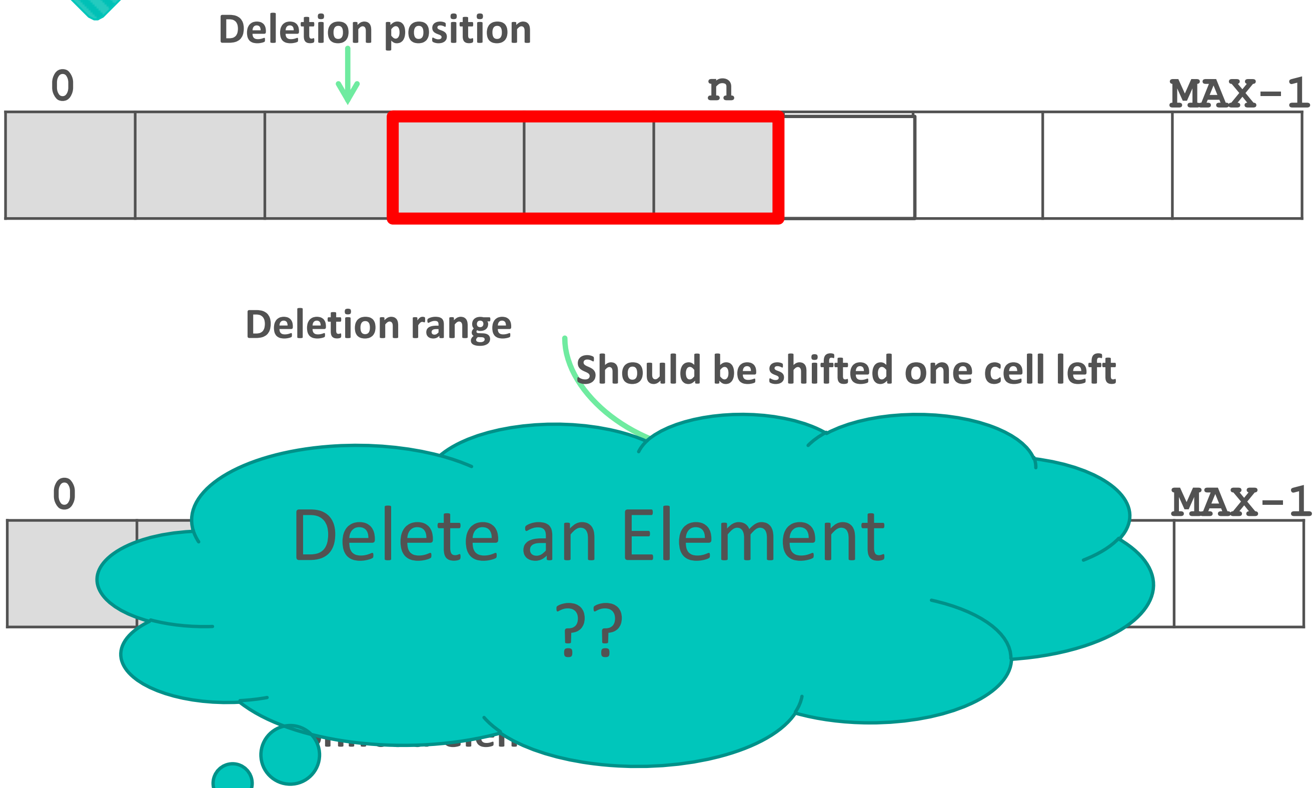
- A list is an sequential set of data items (values).
- **In a general list:**
  - New values are added in position determined by the user.
  - Element is removed from a position determined by the user.

How Lists Work?



We can **insert** a new element in position 0≤p≤n+1, where n is the number of elements within the list.

How Lists Work?

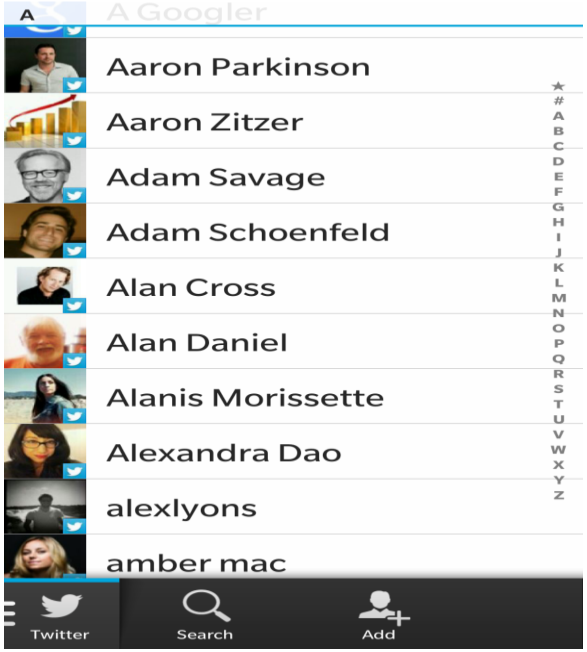


Delete an Element
??

We can **delete** an element from position $0 \le p \le n$, where n is the number of elements within the list .

Lists

Implementation



A Googler

Aaron Parkinson

Aaron Zitzer

Adam Savage

Adam Schoenfeld

Alan Cross

Alan Daniel

Alanis Morissette

Alexandra Dao

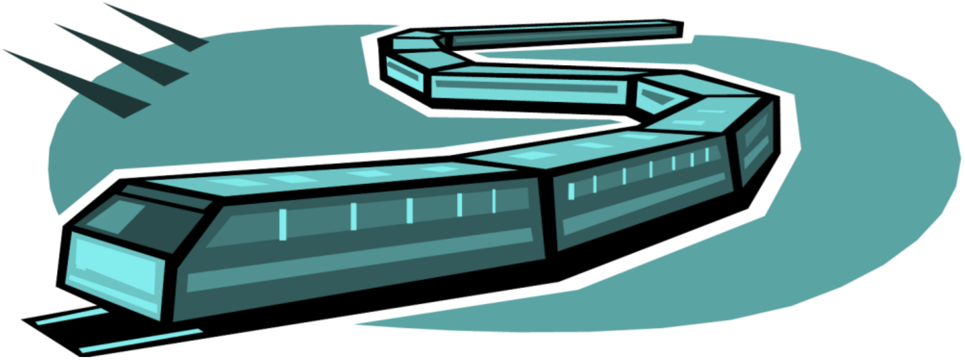
alexlyons

amber mac

Lists

- Many applications require resizing, and the required size not always immediately available.

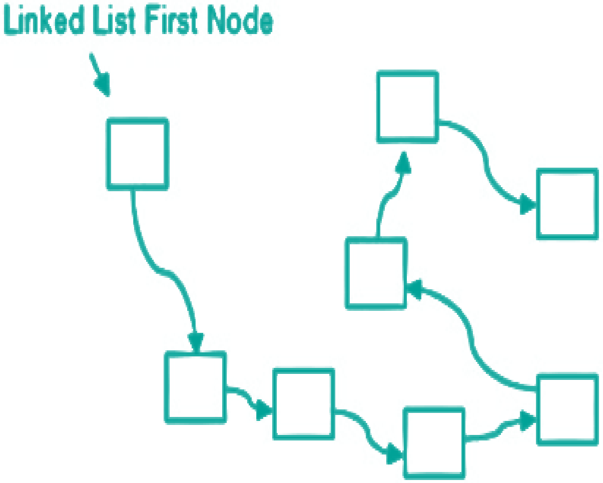
- For those applications, the linked list is preferred.



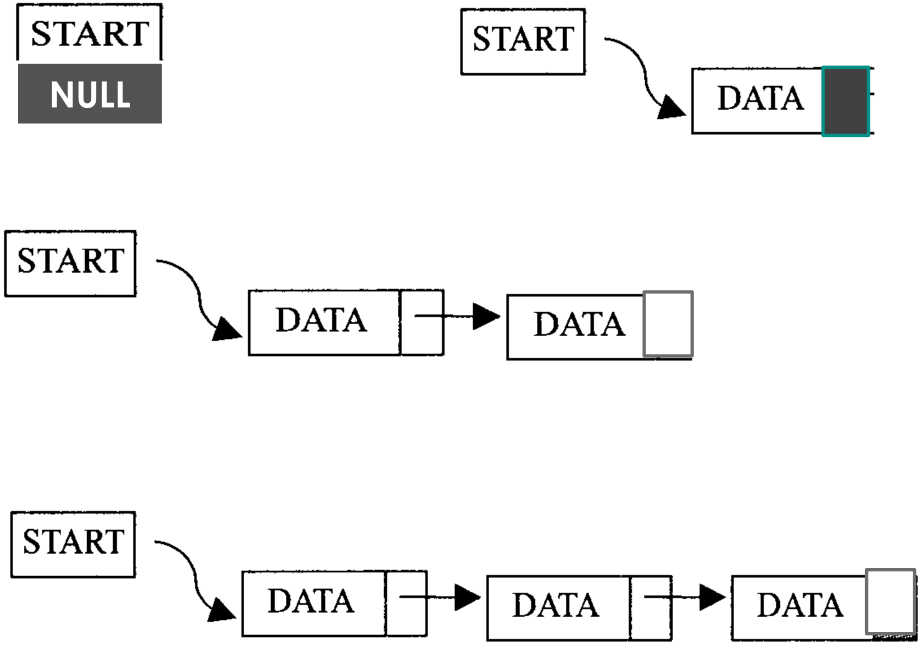
Linked Lists

- Linked list is a **linear** data structure consisting of a sequence of nodes **connected** to each other

- Each node stores:
  - **Data**
  - **Link to the next node**



Linked List



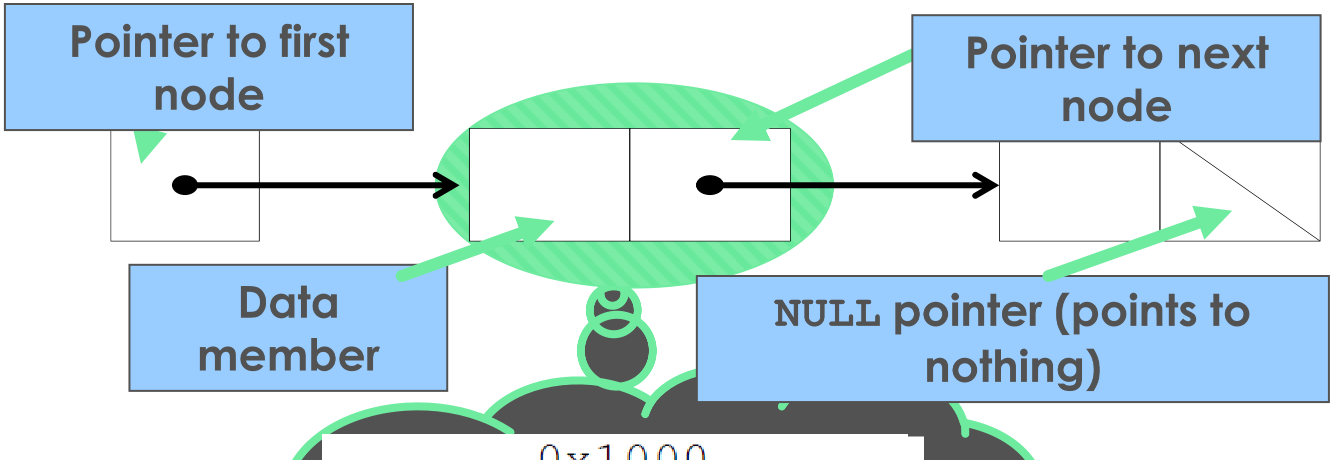
START
NULL

START
DATA

START
DATA
DATA

START
DATA
DATA
DATA

Linked List



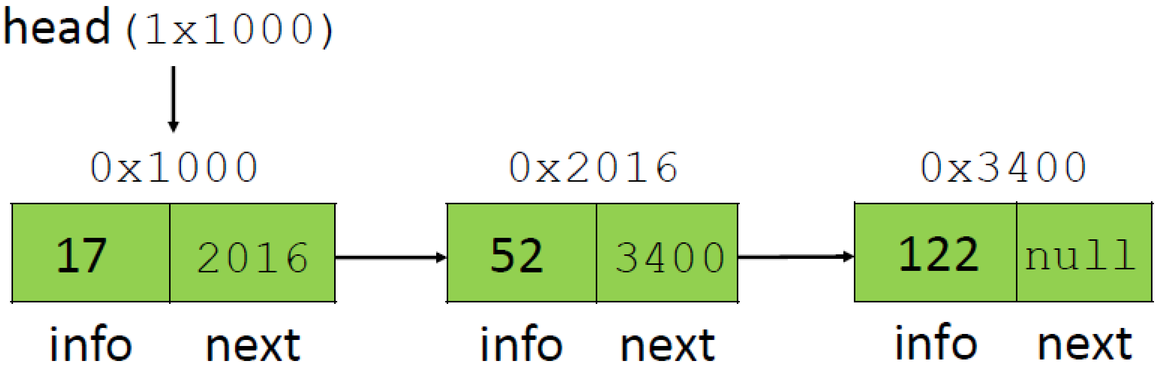
Pointer to first node

Pointer to next node

Data member

NULL pointer (points to nothing)

0x1000



head (1x1000)

0x1000

0x2016

0x3400

17	2016	52	3400	122	null
info	next	info	next	info	next

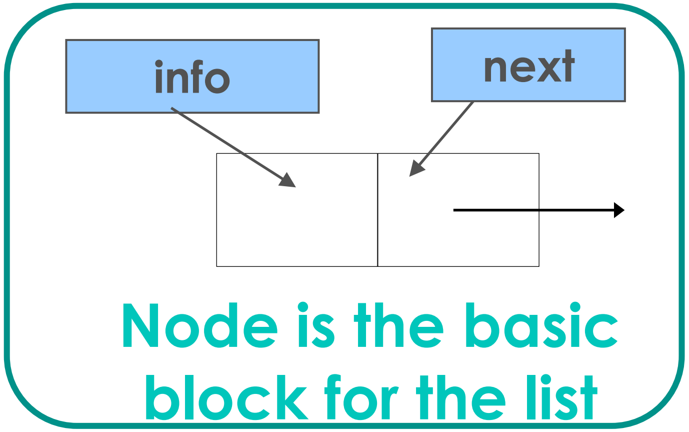
Linked List

- ***They are dynamic data structure:*** That **grow** or **shrink** during the **execution** of a program.

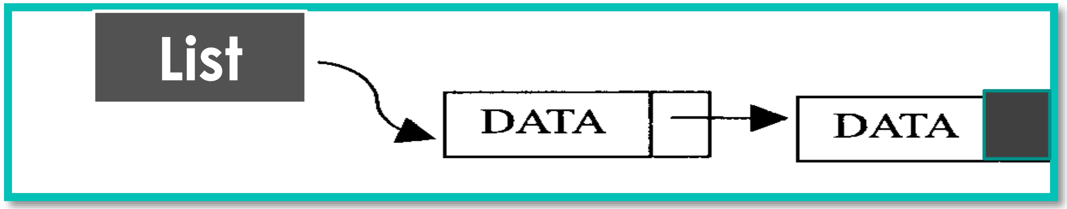
- ***Efficient memory utilization:*** Memory is **allocated** whenever it is required. And it is **de-allocated** when it is not needed.

Linked List Implementation

```
typedef  char Entrytype;


typedef struct nodeT {
     Entrytype       info;
     struct nodeT     *next;
     }Node;
typedef        Node *  ListType;
```

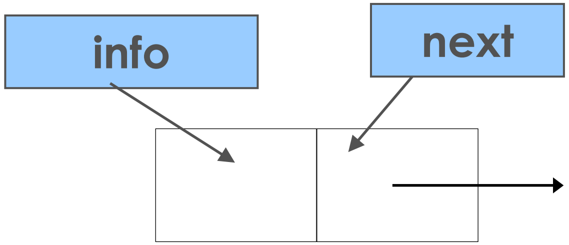


Node is the basic block for the list



Linked List Implementation

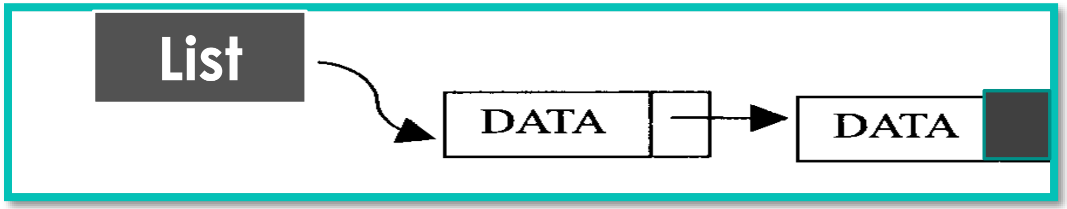
```
typedef  char Entrytype;


typedef struct nodeT {
     Entrytype        info;
     struct nodeT     *next;
     }Node;
typedef        Node *  ListType;
```



Node is the basic block for the list

Better declaration:

```
typedef struct{
        Node *     head;
} ListType;
```



OPERATIONS PERFORMED ON LINKED LIST

- **Create** the list, leaving it empty.
- Determine whether the list **is empty** or not.
- Determine whether the list **is full** or not.
- **Insert** a new entry in a specific position.
- **Retrieve** an entry from a specific position.
- **Clear** the list to make it empty

Linked List Implementation

- Create operation:

  Pre: None.

  Post: The list is initialized to be empty.

```
void CreateList(ListType *L){
  *L= NULL;
}
```

Linked List Implementation

- **Create operation:**

  Pre: None.

  Post: The list is initialized to be empty.

```
void CreateList(ListType *L){
 *L= NULL;
}
```

If the list is declared as:

```
typedef struct{
      Node *     head;
} ListType;
```

This statement should be replaced by:

```
L->head = NULL;
```

Linked List Implementation

Empty operation:

Pre: The list is initialized.

Post: If the list is empty (1) is returned. Otherwise (0) is returned.

```
int  EmptyList(ListType   L){
        return  (L==NULL);}
```

Full operation:

Pre: The list is initialized.

Post: If the list is full (1) is returned. Otherwise (0) is returned.

```
int FullList(ListType  L){
        return 0;}
```

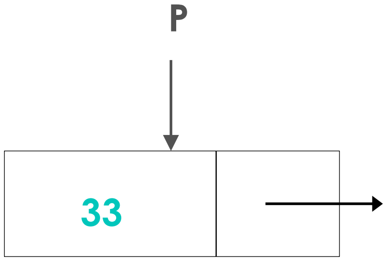
Linked List Implementation

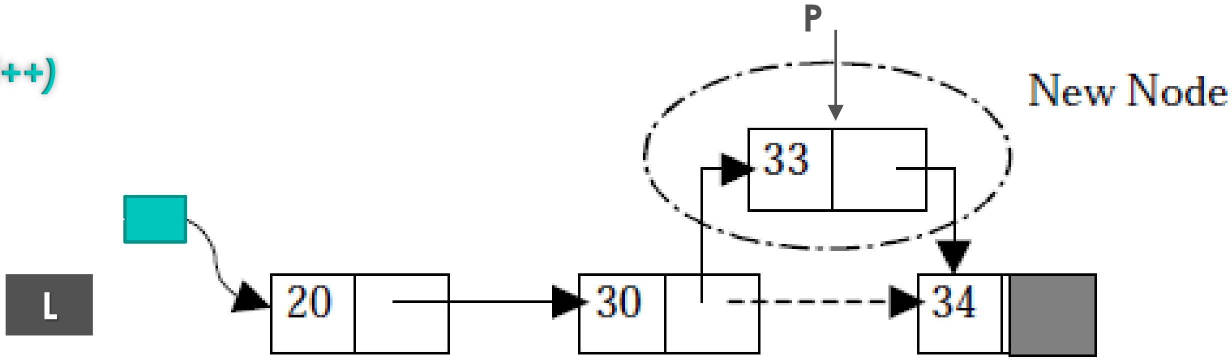
- Insert operation:

Pre: The list is initialized, not full and 0<=pos<=size of the list.

Post: Item is added to specific position of the list.

```
void  Insert(ListType *L, Entrytype  item, int  pos){
        Node *p = (Node *)malloc(sizeof(Node));
        p->info = item;
        if (pos==0){        //will work also for empty list
        p->next=*L;              *L = p;       }
else{   Node  *q;        int i;
          for(q=*L, i=0; i<pos-1; i++)
                q=q->next;
         p->next=q->next;
        q->next=p; } }
```





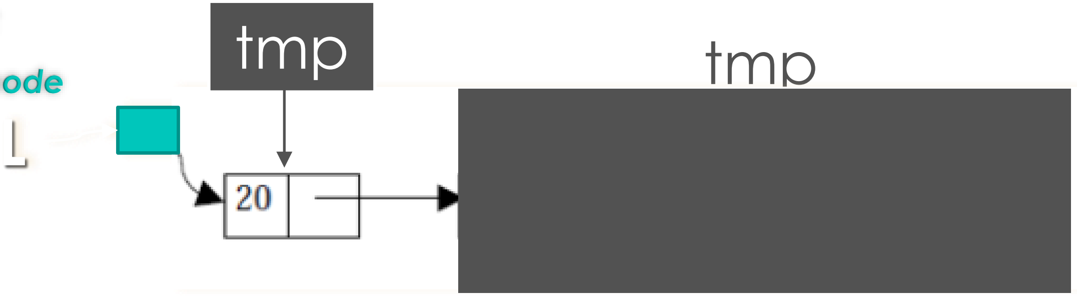
Linked List Implementation

- Retrieve operation:

Pre: The list is initialized, not empty and 0<=pos<=size of the list.

Post: An element has been retrieved from position pos.

```
void  Retrieve(ListType *L, Entrytype *item, int pos){
 int i;   Node *q, *tmp;
if (pos==0){
            *item=(*L)->info;       tmp=*L;       *L=(*L)->next;
            free(tmp);}       // it works also for one node
else{
           for(q=*L,  i=0; i<pos-1; i++)    q=q->next;
       *item=q->next->info;
       tmp=q->next;         q->next=tmp->next;
       free(tmp); }// check for retrieving last node
}
```



Linked List Implementation

- Clear operation:

  Pre: The list is initialized.

  Post: the list is cleared to be empty.

```
  void  ClearList(ListType  *L){
  Node *q;
  while(*L){
        q = *L;
        *L=(*L)->next;
        free(q);
  }
}
```

Think

- Could you keep track with the list size in the List ADT?!! How?!! is that useful?!!
- Discuss, which is better array based or linked implementation for lists.
- How to use the Linked List as a Linked Stack?!!
- How to use the Linked List as a Linked Queue?!!

Any
Questions