●Linked List 주제

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. General Linked List

1. Singly Linked List(SLL)

- Ordered List 의 문제점: 삽입, 삭제시 많은 양의 자료이동 필요

예) (A, C, D) 에서 "insert B between A and C
or "remove "C" from the list

⇨ sequential representation 에서 임의의 삽입과 삭제는 time-consuming.
⇨ Another difficulty is "waste of storage"
⇨ Solution: Linked List Representation

- Linked List
  - 순차적 표현 : 기억장소에서도 인접한 위치
  - 연결표현(LL) : 기억장소의 어느곳에 위치해도 무관함. 단, List 의 원소들은 다음원소 찾는 정보 필요 (pointer) (주소, 위치번호)

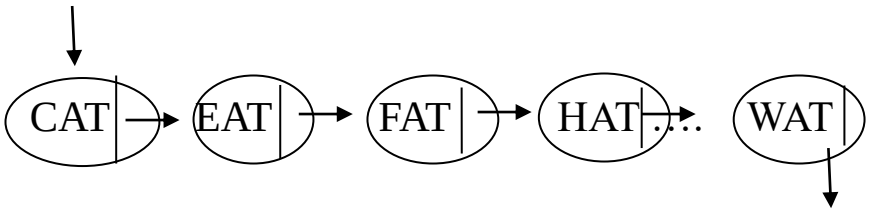
  - Node(원소): consists of two fields
    - Data , Pointer to next node
    - The pointers are called LINK

  - Singly Linked Lists:
    . ordered sequence of nodes
    . nodes do not reside in sequential locations

- 배열 이용한 Linked List 시뮬레이션

주소 Data Link

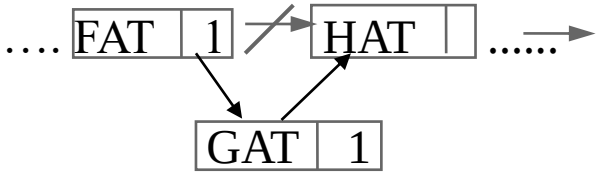
1	HAT	15
2		
3	CAT	4
4	EAT	8
5		
6	WAT	0
7		
8	FAT	1



- head = 3 //시작지점
- Data[3] = CAT//data, Link[3] = 4 //주소
- Data[link[3]] = EAT

<Non Sequential List Representation>

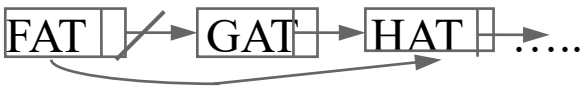
Ex) **Insert "GAT"**



1) Get unused Node, ex)주소 5
2) Set data field 5 GAT
3) Set link field 5 GAT 1
4) FAT 의 link 1 → 5 교체

1	HAT	15
2		
3	CAT	4
4	EAT	8
5	*GAT*	1
6	WAT	0
7		
8	FAT	~~1~~ 5

■ **Delete "GAT"**



1	HAT	15
2		
3	CAT	4
4	EAT	8
5	*GAT*	1
6	WAT	0
7		
8	FAT	~~5~~ 1

■ **Node 정의**

```
struct Node {
   int data;
   Node *next;
};

 class List {
  private:
       Node *head;

  public:
       List () { head = 0;}
       void insert(int);
       void append(int);

       bool isEmpty();
       void display();
  };
```

```
class Node {
      private:
            int data;
            Node *next;
       friend class List;
};
```

● 노드생성: "**new**"

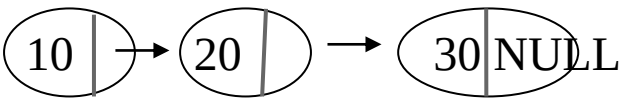
Ex) Node* f; f = new Node;

● 노트 삭제

delete f;

1) Insert (front, middle, last)
2) Delete (front, middle, last)
3) Isempty
4) Display

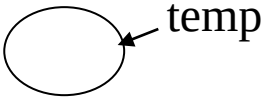
● 일반적인 순차 연결리스트의 모양

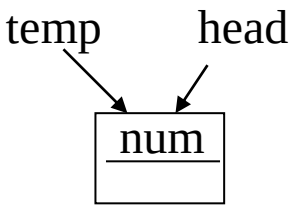


■ **node insert**

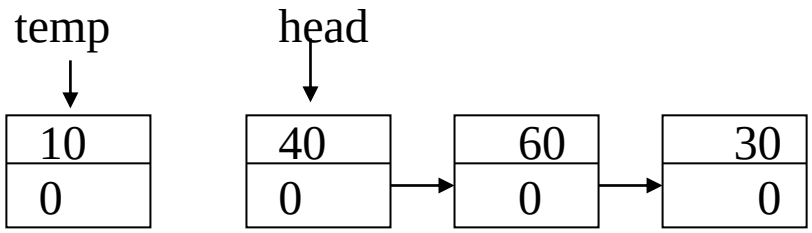
1) **맨앞에 삽입하기** (list 가 empty 일 경우)

```
Node *temp = new Node;
temp->data = num;
temp->next = 0;
head = temp;        // linked list 가 empty 인 경우에는
                    // head 가 temp node를 가리키게 한다.
```

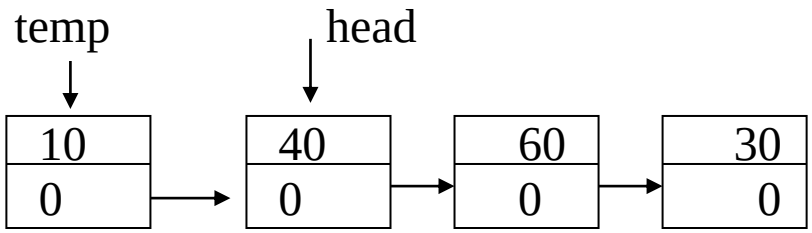




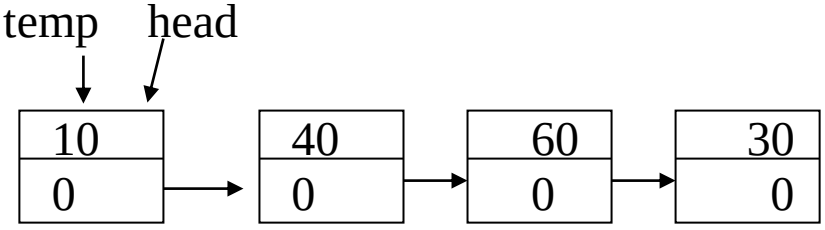
2) **head node 가 0이 아닌 경우** (즉, List에 여러 개의 노드가 있을 경우, 맨 앞에 삽입하기)



- 우선 temp 와 head 연결 `temp->next = head;`



- head 가 list 의 맨 앞을 가리키게 한다 `head = temp;`



3) head node 뒤에 node 삽입할 경우 (insert middle)

```
if (head->next = 0)
     head->next = temp;    // head node 하나밖에 없을 경우
else
   {    temp->next = head->next;
        head->next = temp;
    }
```

4) 맨 뒤에 node를 만들 경우 (insert last)

- 우선 삽입할 노드를 만들고 temp가 가리키게 한다.

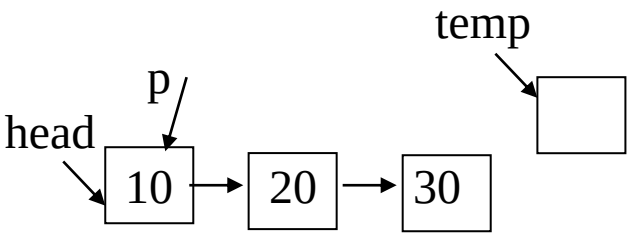
```
Node *temp = new Node;
temp->data = num;    temp->next = 0;
```

- head 가 0인 경우 (list가 empty일 경우, head 가 temp 를 가리키게 한다)

```
head = temp;
```

- head가 0이 아닌경우 (list에 node 가 여러개 있을 경우)

```
p= head;
while (p->next != 0)
        p = p->next;
p->next = temp;
```



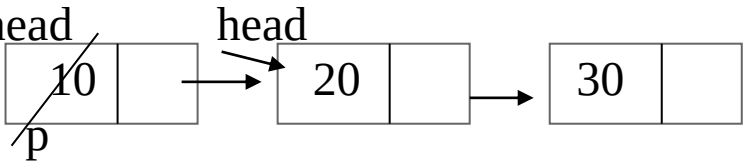
● 출력 하기

```
p = head;
While (p != 0) {
      cout <<   p->data;
      p = p-> next;
}
cout << endl;
```

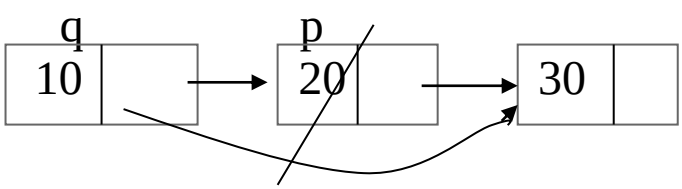
- **node 삭제**

1) Delete from Front (ex. delete 10)

```
If (head->data ==num){
 p = head;
 head = head->next;
 delete p;
}
```

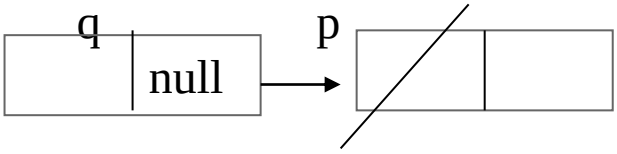


2) Delete from Middle




```
p = head; q= head;
while (p != NULL && p->data != num) {    // delete 20
     q=p;     p= p->next;
}
if (p != NULL) {
        q->next = p->next;
        delete p;
}
else
       cout << num << " is not in the list\n";
```

3) Delete from End




```
p = head;
q=head;

while ( !p->next= null) {
      q=p;       p = p->next;
 }
q->next = null;
delete p;
```

- 우측으로 이동: p = p->next;
- 현재 pointer를 head 가르키게: p = head;

- Traverse

```
p = head;
while (p != NULL) {
        /* cout << p->data; */
        p = p->next;
}
```

Singly Linked List 알고리즘

(1) Singly Linked List ADT

```
class Node {
   private:
      int data;
      Node *next;
      Node (int value) {data = value; next=0}
   friend class List;
};

class List {
   private:
         Node *head;
   public:
         List () { head = 0;}
         void insertNode(int);
         void deleteNode(int);

         bool isEmpty();
         void display();
};
```

(2) List ADT 의 operations (member functions)

변수 및 함수선언부	설명
bool isEmpty()	연결리스트가 비었는지의 여부를 검사하는 함수
void insertNode(int num)	연결 리스트에 노드를 삽입하는 함수
void deleteNode(int num)	연결 리스트에 노드를 삭제하는 함수
void traverseList()	연결 리스트의 노드들의 내용을 출력하는 함수
void searchList()	연결 리스트에서 데이터를 찾는 함수
~List()	연결 리스트의 각노드들을 시스템에 반환 함수

(3) insert 함수 (데이터값의 크기에 따라 입력될 경우)

```
void List::insertNode(int data)
{
   Node *temp = new Node(data);
   Node *p, *q;

   if (head == 0)    head = temp;          // 처음 노드
   else if (temp->data < head->data) { // head 앞에 삽입
         temp->next = head;
         head = temp;
   }
   else {                                  // insert middle
      p = head;
      while ((p != 0) && (p->data < temp->data)) {
            q = p;
            p = p->next;
      }
      if (p != 0) {
            temp->next = p;        q->next = temp;
      }
      else
         q->next = temp;    // 맨뒤에 insert
   }
}
```

- 첫 노드(head)가 만들어지는 경우 (head == NULL)
- head 노드 앞에 노드가 삽입될 경우 (temp->data < head->data)
- 연결리스트의 가운데에 노드가 삽입되는 경우

(4) delete 함수

- (delete head, middle, and last node)

```
void List::deleteNode(int num)
{
   Node *p, *q;


   if (head->data == num) {      // delete head
      p = head;
      head = head->next;
      delete p;
   }
   else {                              //delete middle node
      p = head;
      while (p != 0 && p->data != num) {
         q = p;
         p = p->next;
      }
      if (p != 0) {
         q->next = p->next;      // p->next 는 NULL
         delete p;
      }
      else
         cout << num << " is not in the list\n";
   }
}
```

(5) isEmpty 함수 설명

기능: 현재 연결 리스트가 비어있는지의 여부를 검사
반환값: head 가 NULL 이면 1 을 그렇지 않으면 0 을 반환

```
bool List::isEmpty()
{
   if (head == 0)      return TRUE;
   else      return FALSE;
}
```

(6) traverse 함수

```
void List::traverseList()
{
 Node *p;

 if (!isEmpty()) {
    p = head;
    while (p) {
       cout << setw(8) << p->data;
       p = p->next;
    }
    cout << endl;
 }
 else
    cout << "List is empty!\n";
}
```

(7) search 함수

```
void List::searchList(int num)
{
  Node *p;

  if (head != 0) {
      p = head;
      while (p != 0 && p->data != num)
            p = p->next;

      if (p)
         cout << p->data << " is found." << endl;
      else
         cout << num << " is not in the list." << endl;
    }
    else
      cout << "List is empty\n";
}
```

(8) ~List() 함수 : ~List()는 소멸자

```
List::~List()
{
    Node *p;

    while (head != 0) {
        p = head;
        head = head->next;
        delete p;
    }
}
```

* Single List Sample Code#1

//singly1.cpp (Structure Node 사용)

```
struct Node {
   int data;
   Node *next;
};

class List {
   private:
      Node *head;

   public:
      List () { head = 0;}
      void insert(int);
      void append(int);

      bool isEmpty();
      void display();
};

// insert() places a new item at the front of the list.

void List::insert(int data) {
   Node *temp = new Node;

   temp->data = data;
   temp->next = 0;

   if (head != 0) {
      temp->next = head;
      head = temp;
   }
   else     head = temp;
}

// append() places a new item at the end of
      the list.

void List::append(int data) {
   Node *temp = new Node;
   temp->data = data;
   temp->next = 0;

   if (head == 0)
         head = temp;
   else {
      Node *ptr = head;
      while (ptr->next != 0)
            ptr = ptr->next;
      ptr->next = temp;
   }
}

bool List::isEmpty()
{
   if (head == 0)    return true;
   else              return false;
}

void List::display() {
   Node *ptr;

   ptr = head;
   while (ptr) {
      cout << ptr->data;
      ptr = ptr->next;
   }
   cout << endl;
}


void main()    {
   List l1;

   l1.insert(30);
   l1.insert(40);
   l1.append(50);
   l1.append(80);

   l1.display();
}
/* output: 30 40 50 80
```

// Sample Code #2

// singly2.cpp Class Node와
// friend function 사용.

```
class Node {
private:
    int data;
    Node *next;
    friend class List;
};
class List {
private:
    Node *head;
public:
    List () { head = 0;}
    void insert(int);
    void append(int);
    bool isEmpty();
    void display();
};

void List::insert(int data) {
  Node *temp = new Node;
  temp->data = data;
  temp->next = 0;

    if (head != 0) {
       temp->next = head;
       head = temp;
      }
      else head = temp;
  }

void List::append(int data) {}
  ……
bool List::isEmpty() {}
……
void List::display() { }
  ……
void main() {
  ……
}
```

// Sample Code #3

// singly3.cpp
// (1) Class Node와
// friend function 사용.
// (2) Node Constructor 사용

```
class Node {
  private:
       int data;
       Node *next;
       Node(int value)
          {data = value; next = 0;}
       friend class List;
};

class List {
private:
       Node *head;
public:
       List () { head = 0;}
       void insert(int);
       void append(int);
       bool isEmpty();
       void display();
  };

void List::insert(int data) {
Node *temp = new Node(data);

  // no need to specify temp

    if (head != 0) {
       temp->next = head;
       head = temp;
    }
      else
         head = temp;
  }
```

// Sample Code #4, singly4.cpp

// (1) Class Node 에서
// public 사용
// (2) Node Constructor 사용
// (3) template 사용

```
#include <iostream>
#include <iomanip>

template <class Type>
class Node {
   public:
      Type data;
      Node *next;
      Node(Type value)
         {data = value; next = 0;}
};

template <class Type>
 class List {
      private:
            Node<Type> *head;
      public:
            List () { head = 0;}
            void insert(Type);
            void append(Type);
            void display();
 };

template <class Type>
void List<Type>::insert(Type data) {
Node<Type> *temp = new Node <Type>
(data);
 if (head != 0) {
       temp->next = head;
       head = temp;
 }
  Else    head = temp;
  }
template <class Type>
void List<Type>::append(Type data){
Node<Type> *temp = new Node <Type>
(data);
 if (head == 0)
       head = temp;
 else {
       Node<Type> *ptr = head;
       while (ptr->next != 0)
             ptr = ptr->next;
       ptr->next = temp;
       }
 }

template <class Type>
void List<Type>::display() {
Node<Type> *ptr;
   ptr = head;
   while (ptr) {
        cout << setw(8) << ptr->data;
        ptr = ptr->next;
    }
    cout << endl;
   }

void main() {
 List<int> l1; // instantiation
 l1.insert(30);    l1.insert(40);
 l1.append(50); l1.append(80);
 l1.display();
}
```

- Unordered SLL

멤버 함수	설명

int is_empty();
void insertAfter(Type);
void insertBefore(Type);
void insertLast(Type);
void insertFirst(Type);
 	
void deleteCurrent();
Type retrieveCurrent()
void locateCurrent(int);
void updateCurrent();
void dispalyList();
int listLength()
	리스트의 empty 여부를 검사하는 함수.
	current의 뒤에 노드를 삽입하는 함수.
	current의 앞에 노드를 삽입하는 함수.
	리스트의 마지막에 노드를 삽입하는 함수.
	리스트의 제일 앞에 노드를 삽입하는 함수.
	current가 가리키는 곳의 노드를 삭제하는 함수.
	current가 가리키는 곳의 자료를 반환하는 함수.
	current를 옮겨주는 함수.
	current가 가리키는 곳의 값을 변경시키는 함수.
	List의 data를 출력해주는 함수.
	List의 길이를 반환하는 함수.

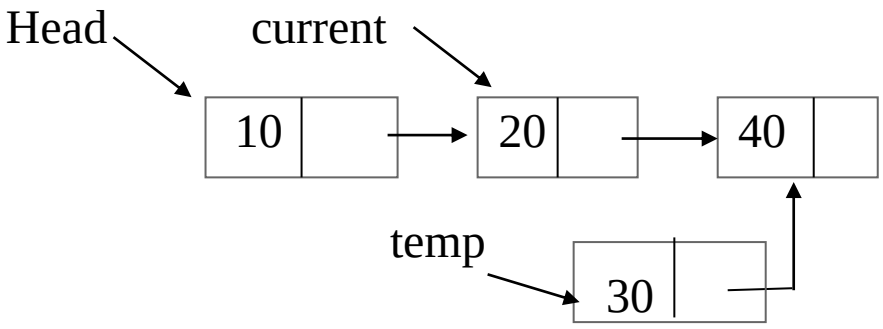
```
class Node {
private:
        Type val;
        Node *next;
        Node(Type data) {val = data; next = 0;}
        friend class List;
};

class List {
 private:
        Node *head;
        Node *current;
public:
        List();
        ~List();
        void insertafter(Type);    void insertbefore(Type);
        void insertfirst(Type);    void insertlast(Type);
        void deleteCurrent();      void locateCurrent(int);
        void updateCurrent(Type);    Type retrieveCurrent();
        void displayList();    int listLength();    int is_empty();
};
```

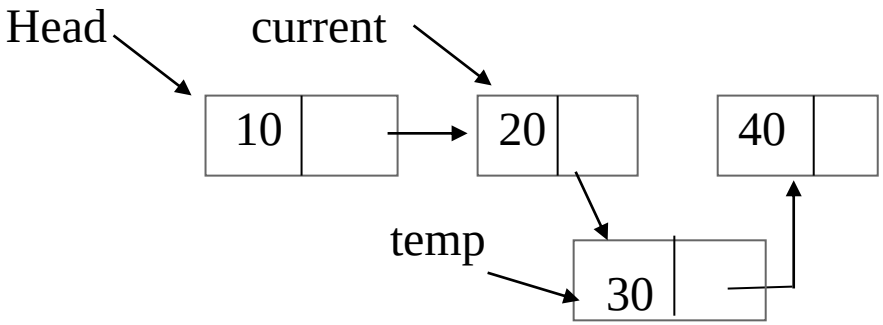
1) InsertAfter 함수

current 포인터의 위치 다음에 노드를 삽입하는 함수. 메모리 할당을 받은 후 data를 넣고, next부분에는 NULL을 넣는다. 그 다음 head가 NULL인지를 검사한 후, current의 다음 노드를 temp의 next가 가리키도록 한다.

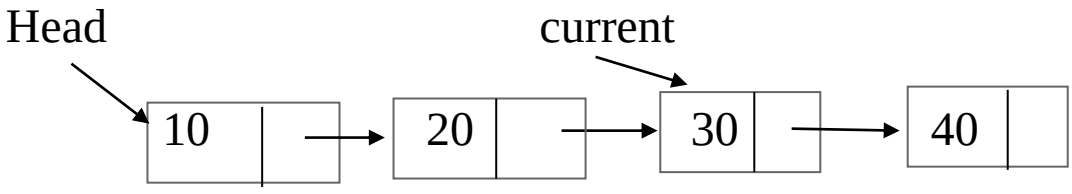
- temp->next = current->next;



- currnet->next = temp

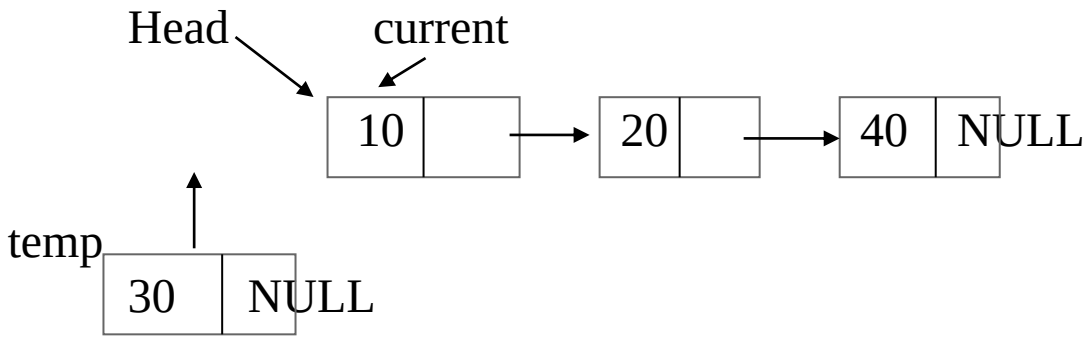


- current = temp

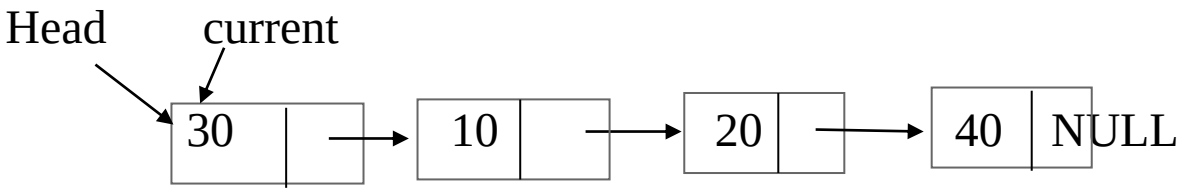


2) InsertBefore

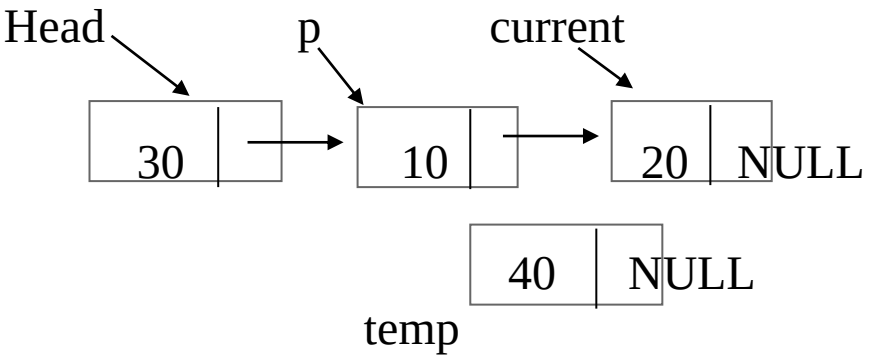
- Current와 head가 같은경우 => head 앞에 삽입



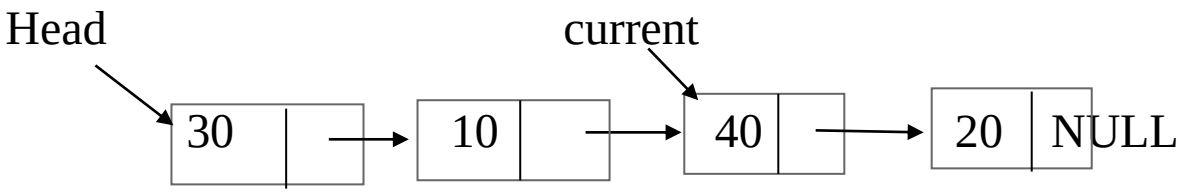
temp->next = head; head = temp; current = temp



- Head 와 current가 다른 경우 (p위치는 current 이전)



p->next = temp; temp->next = current current = temp;



3) InsertLast

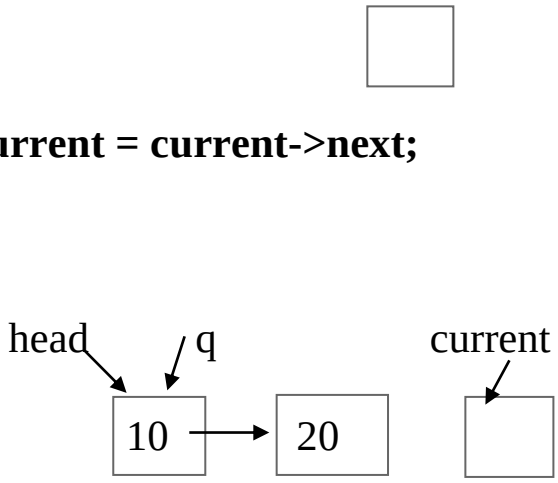
```
void List::insertLast(Type data) {
   Node *temp = new Node(data);      Node *p;

   if (head == 0)    head = temp;
   else {
        p = head;
        while (p->next != 0)
              p = p->next;
        p->next = temp;
     }
   current = temp;
}
```

3) DeleteCurrent

```
void List::deleteCurrent() {
Node *p, *q;

 if (head == 0)    cout << "List is empty!" << endl;
  else {
         p = current;
         if (head == current) {
            head = head->next;      current = current->next;
          }
   else {
        q = head
         while (q->next != current)
                   q = q->next;
           q->next = current->next;
         if (current->next != 0)
               current = current->next;
         else    current = head;
         }
         delete p;      // p = current
 }//end of else
}
```



2. Dynamically Linked Stacks and Queues

1) Linked List implementation of a **STACK**