

자료구조의 기초

## Lab 3. Linked List

Taewhan Kim

# Lab Introduction

## ■ Visual Studio

- 모랩에 설치된 **Visual Studio 2017** 사용
- 개인 노트북에 설치된 Visual Studio 사용 가능

## ■ 출석

- **출석부에 서명 + eTL에 실습 코드 업로드**로 출석 체크
- 둘 중 하나라도 누락 시 **결석** 처리

# Linked List

## ■ Linked List

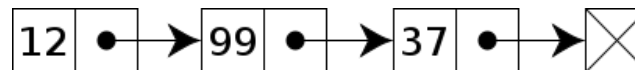
- 각 원소(Element)가 **데이터**와 다른 원소의 **포인터**를 포함하는 자료구조

## ■ Array와의 차이점

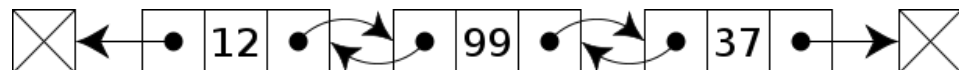
- 각 원소가 연속하여 배치되어 있지 않아도 됨
  - 포인터를 이용하여 앞/뒤 원소에 순차적으로 접근
  - Array와 달리 Random Access 불가

## ■ Linked List 종류

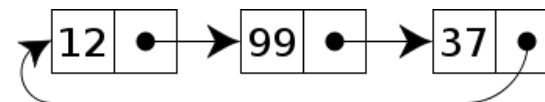
- Singly Linked List



- Doubly Linked List

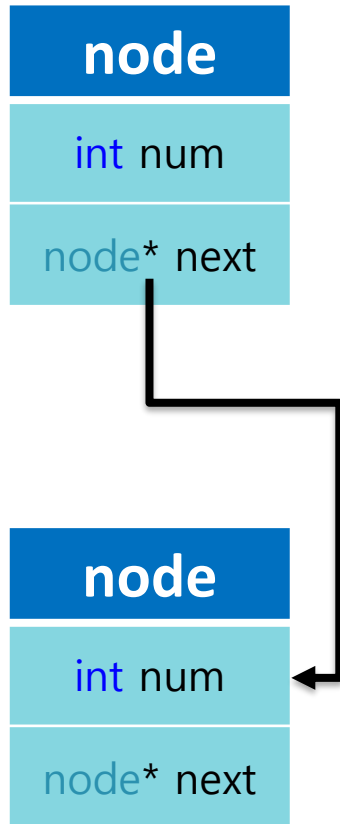


- Circular Linked List



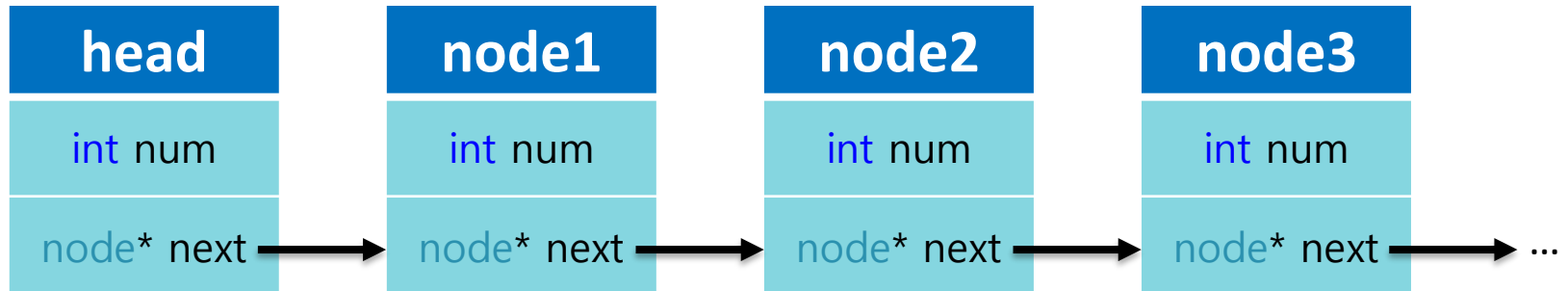
# Singly Linked List Node

- List class 안에 nested class 형태로 Node class 구현



sllist.h	sllist.cpp
<pre>#ifndef SLLIST_H #define SLLIST_H  class list { private:     class node { private:         int num;         node* next; public:         node(int);         node(int, node*);         void setNext(node*);         node* getNext();         int getNum();     };     ... };  #endif</pre>	<pre>#include "sllist.h"  list::node::node(int n) {     num = n;     next = nullptr; }  list::node::node(int n, node* link) {     num = n;     next = link; }  void list::node::setNext(node* link) {     next = link; }  list::node* list::node::getNext() {     return next; }  int list::node::getNum() {     return num; }</pre>

# Singly Linked List



- Singly Linked List
  - List의 처음을 가리키는 **head**를 멤버변수로 가짐
  - **head**는 Dummy node로, 무의미한 데이터를 가짐
  - 전체 List의 크기인 **size**를 멤버변수로 가짐
  - **head**로부터 각 Element가 포인터로 순차적으로 연결

## sllist.h

```
#ifndef SLLIST_H
#define SLLIST_H

class list {
...
private:
    node* head;
    int size;
...
};

#endif
```

# Singly Linked List – Ctor/Dtor

## sllist.cpp

```
list::list() {  
    head = new node(0, nullptr);  
    size = 0;  
}  
  
list::~~list() {  
    node* curr = head;  
    while (curr != nullptr)  
    {  
        node* temp = curr->getNext();  
        delete curr;  
        curr = temp;  
    }  
}
```

### ■ Constructor

- Dummy node **head** 생성
- 멤버변수 초기화

### ■ Destructor

- node의 **next** 포인터를 이용하여 순차적으로 node 삭제

# Singly Linked List – Find

## sllist.cpp

```
list::node* list::find(int num) {  
    node* temp = head;  
    while (temp->getNext() != nullptr) {  
        temp = temp->getNext();  
        if (temp->getNum() == num) break;  
    }  
    if (temp->getNum() == num)  
        return temp;  
    else  
        return nullptr;  
}
```

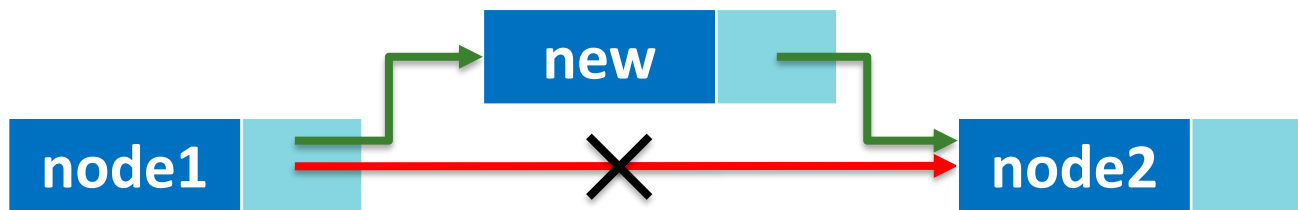
} node 위치 탐색 및 반환

# Singly Linked List – Insert

## sllist.cpp

```
void list::insert(int k, int num) {  
    node* temp = head;  
    if (size < k || k < 0) return;  
    while (k > 0 && temp->getNext() != nullptr) {  
        temp = temp->getNext();  
        k--;  
    }  
    node *newNode = new node(num, temp->getNext());  
    temp->setNext(newNode);  
    size++;  
}
```

} 새 node 삽입 위치 탐색  
} 새 node 생성 및 연결





# Singly Linked List – Remove

## sllist.cpp

```
void list::remove(int k) {  
    node* curr = head;  
    if (size < k || k < 0) return;  
    while (k > 0 && curr->getNext() != nullptr) {  
        curr = curr->getNext();  
        k--;  
    }  
    node* temp = curr->getNext();  
    curr->setNext(temp->getNext());  
    delete temp;  
}
```

} 삭제할 node 위치 탐색  
} node 삭제 및 연결



# Singly Linked List – Etc.

## sllist.cpp

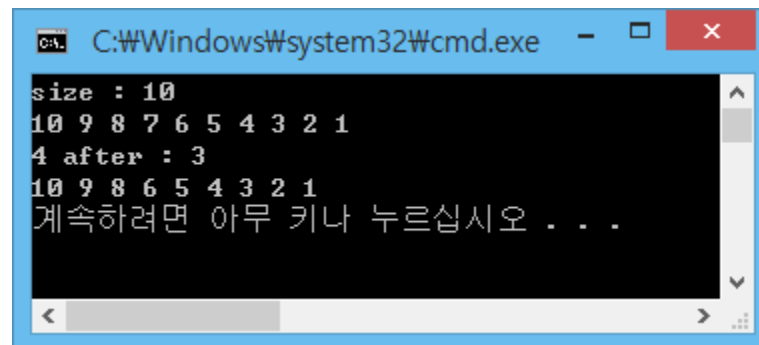
```
int list::getSize() const {  
    return size;  
}  
  
void list::printAll() {  
    node* temp = head;  
    while (temp->getNext() != nullptr) {  
        temp = temp->getNext();  
        cout << temp->getNum() << " ";  
    }  
    cout << endl;  
}
```

# Singly Linked List – Test

## sllist\_test.cpp

```
#include "sllist.h"

int main() {
    list numList;
    for (int i = 0; i < 10; i++) {
        numList.insert(i, 10 - i);
    }
    cout << "size : " << numList.getSize() << endl;
    numList.printAll();
    cout << "4 after : " << numList.find(4)->getNext()->getNum() << endl;
    numList.remove(3);
    numList.printAll();
}
```



The screenshot shows a Windows command prompt window with the title "C:\Windows\system32\cmd.exe". The output of the program is as follows:

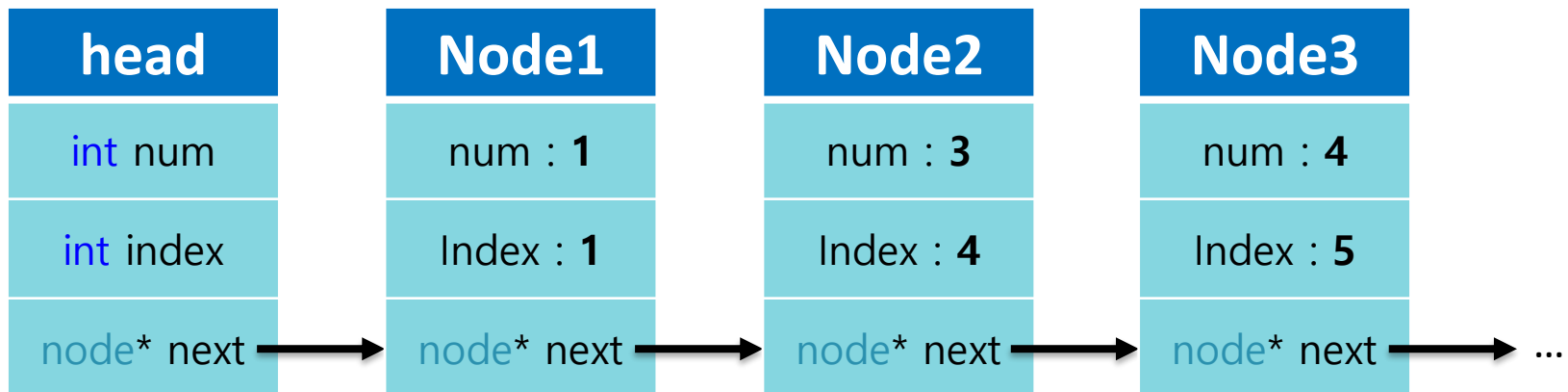
```
size : 10
10 9 8 7 6 5 4 3 2 1
4 after : 3
10 9 8 6 5 4 3 2 1
계속하려면 아무 키나 누르십시오 . . .
```

# TODO : 1D Sparse Matrix

## ■ Sparse Matrix

- 행렬 값 대부분이 0인 경우, 행렬 값과 그 인덱스만 저장
- 행렬 데이터를 압축하여 저장 공간을 효율적으로 사용

[0 1 0 0 3 4 0 0 2 0]



# TODO : 1D Sparse Matrix

- node class
  - 멤버변수로 인덱스를 나타내는 **index**를 추가
  - Constructor 수정 및 Getter 추가

## node class in sllist.h

```
class node {  
private:  
    int num;  
    node* next;  
public:  
    node(int);  
    node(int, node*);  
    void setNext(node*);  
    node* getNext();  
    int getNum();  
};
```



## node class in smlist.h

```
class node {  
private:  
    int num, index;  
    node* next;  
public:  
    node(int idx, int n);  
    node(int idx, int n, node*);  
    void setNext(node*);  
    node* getNext();  
    int getNum();  
    int getIndex();  
};
```

# TODO : 1D Sparse Matrix

## ■ list class

- insert의 매개변수가 행렬의 값과 인덱스로 변경되며, list 마지막에 삽입
- printAll의 출력값이 인덱스와 값의 쌍으로 출력되도록 수정

### sllist.h

```
class list {  
private:  
    (node class declaration)  
public:  
    list();  
    ~list();  
    void insert(int, int);  
    void remove(int);  
    node* find(int);  
    int getSize() const;  
    void printAll();  
private:  
    node* head;  
    int size;  
};
```



### smlist.h

```
class list {  
private:  
    (node class declaration)  
public:  
    list();  
    ~list();  
    void insert(int idx, int n);  
    void remove(int);  
    node* find(int);  
    int getSize() const;  
    void printAll();  
private:  
    node* head;  
    int size;  
};
```

# TODO : 1D Sparse Matrix

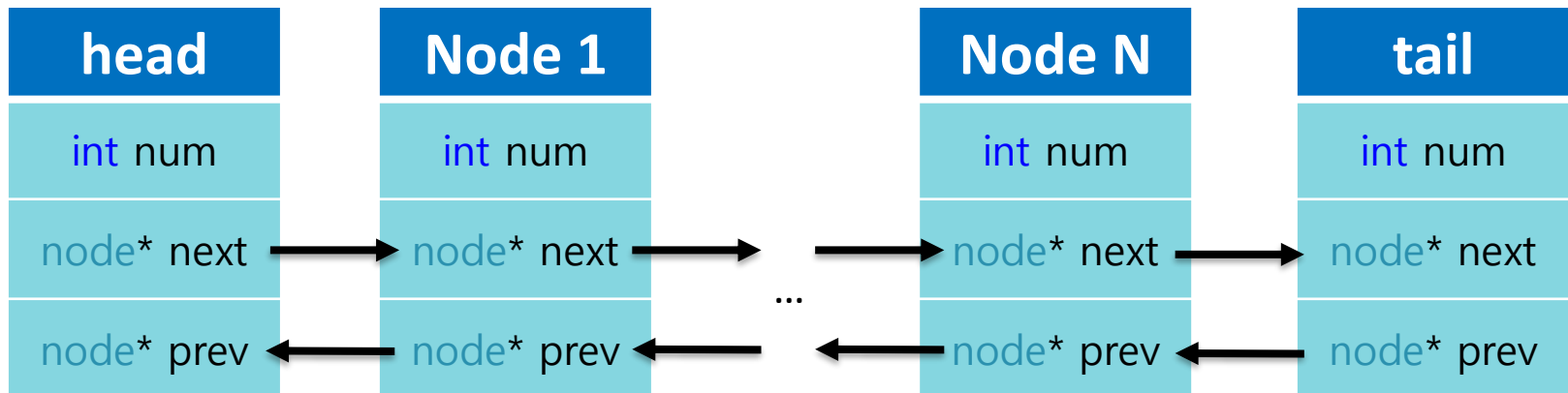
- smlist\_test.cpp 실행 결과
  - 입력된 Dense Matrix가 아래와 같이 Sparse Matrix로 변환되는지 확인

[0 1 0 0 3 4 0 0 2 0]



# TODO : Doubly Linked List

- TODO : 오늘 실습 내용을 응용하여 Doubly Linked List 구현
  - 이전 node를 가리키는 prev가 추가로 존재
  - List의 처음과 마지막을 가리키는 head/tail 존재





# TODO : Doubly Linked List

## ■ Node class

- 멤버변수로 이전 node를 가리키는 pointer **prev**를 추가
- Prev를 추가로 저장할 수 있도록 constructor 수정
- Prev의 정보를 저장하고 불러오는 **setPrev(node\*)**, **getPrev()** 함수 추가

### node class in slist.h

```
class node {  
private:  
    int num;  
    node* next;  
public:  
    node(int);  
    node(int, node*);  
    void setNext(node*);  
    node* getNext();  
    int getNum();  
};
```



### node class in dllist.h

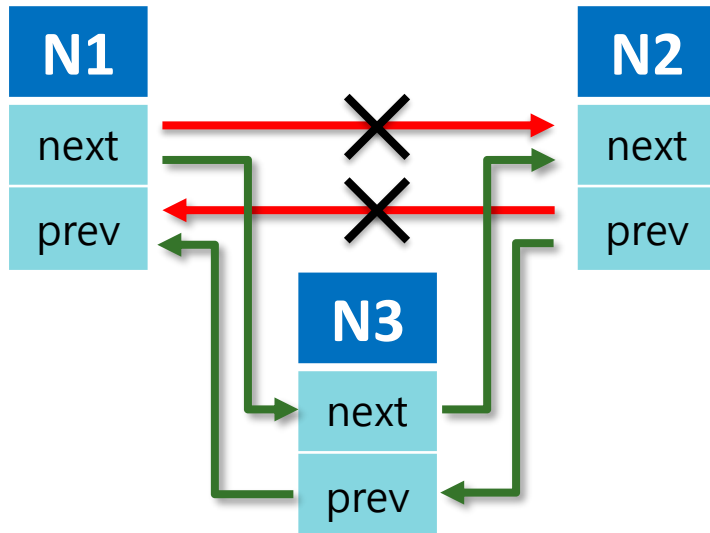
```
class node {  
private:  
    int num;  
    node *next, *prev;  
public:  
    node(int);  
    node(int, node*, node*);  
    void setNext(node*);  
    node* getNext();  
    void setPrev(node*);  
    node* getPrev();  
    int getNum();  
};
```

# TODO : Doubly Linked List

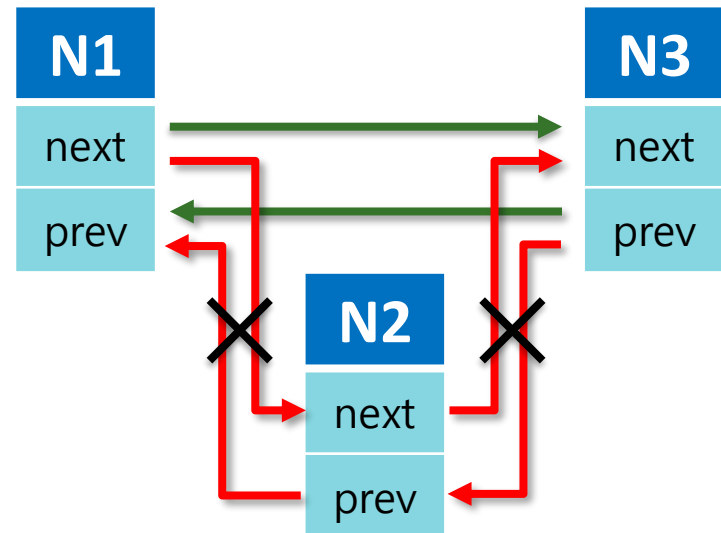
## ■ List Class

- 마지막 node가 가리키는 **tail**을 멤버변수로 추가
- Constructor에서 head, tail 초기 생성 후 head와 tail이 서로를 가리키도록 지정
- Node 추가, 삭제 과정에서 prev가 올바르게 update되도록 **insert()**, **remove()** 함수 수정

### □ Node Insertion



### □ Node Deletion



# TODO : Doubly Linked List

- dllist\_test.cpp 실행 결과
  - Node 삽입, 삭제 후에도 Doubly Linked List가 유지되는지 확인

```
C:\WINDOWS\system32\cmd.exe
size : 10
10 9 8 7 6 5 4 3 2 1 0
4 after : 3
7 before : 8

10 9 8 6 5 4 3 2 1 0
8 after : 6
6 before : 8

10 9 8 6 5 4 3 17 2 1 0
3 after : 17
17 before : 3
17 after : 2
2 before : 17
계속하려면 아무 키나 누르십시오 . . .
```

# Code Submission

## ■ 코드 제출

- 구현한 코드를 다음과 같이 압축
  - 제출할 코드 : sllist.h/.cpp, smlist.h/.cpp, dllist.h/.cpp
  - 파일명 : lab3\_홍길동\_2017-10000.zip
- 오늘 (2018년 4월 4일) **오후 11시**까지 eTL에 제출
- 제출된 코드는 따로 채점하지 않음

## ■ 출석

- **출석부에 서명 + eTL에 실습 코드 업로드**로 출석 체크
- 둘 중 하나라도 누락 시 **결석** 처리