

Window Programming

Visual C++ MFC Programming

Lecture 9

김예진

Dept. of Game Software

Announcement

- 03/16: HW #1 on ClassNet (Due: 03/24) → Avg: 8.1
- 04/06: HW #2 on ClassNet (Due: 04/15) → Avg: 6.3
- 04/27: Midterm 실기 시험 (5 문제, ~90 min.) → Avg: 4.3
 - Sample 문제 on ClassNet

Plan

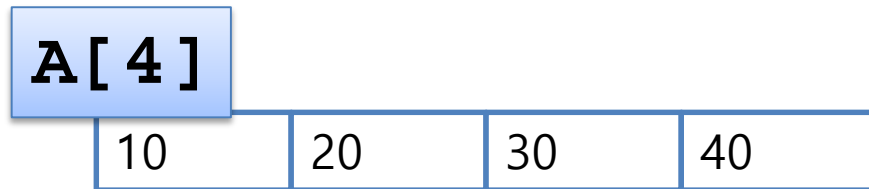
- Linked List
 - 연습: Linked List
- MFC Template Library CList
 - 연습: 점찍고 지우기

잠깐..! 번외편:
Linked List 와
MFC Template Library
CList

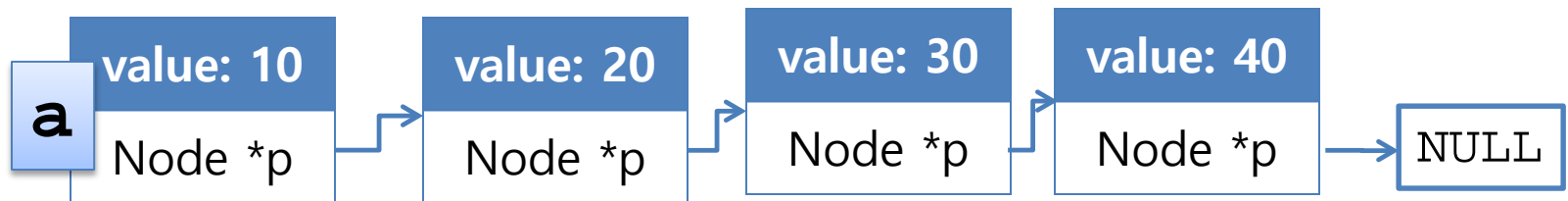


많은 데이터를 저장하는 방법

- Array 이용



- Linked List 이용



구조체와 포인터를 이용한 Linked List

- 구조체 내에 포인터를 멤버로 가질 수 있음

```
struct IHavePointer  
{  
    int v;  
    int *p;  
};
```

구조체와 포인터를 이용한 Linked List

- 구조체를 가리키는 포인터를 멤버로 가질 수 있음

```
struct test
{
    int v;
    test *p;
};
```

구조체를 가리키는 멤버 포인터

```
#include <iostream>
using namespace std;

struct List
{
    int id;
    List *p;
};

void main ()
{
    List item;
    item.id = 20;
    item.p = &item;

    cout << "item.id: " << item.id << "\n";
    cout << "item.p->id: " << item.p->id << "\n";
    cout << "item.p->p->id: " << item.p->p->id << "\n";
}
```


구조체를 가리키는 멤버 포인터

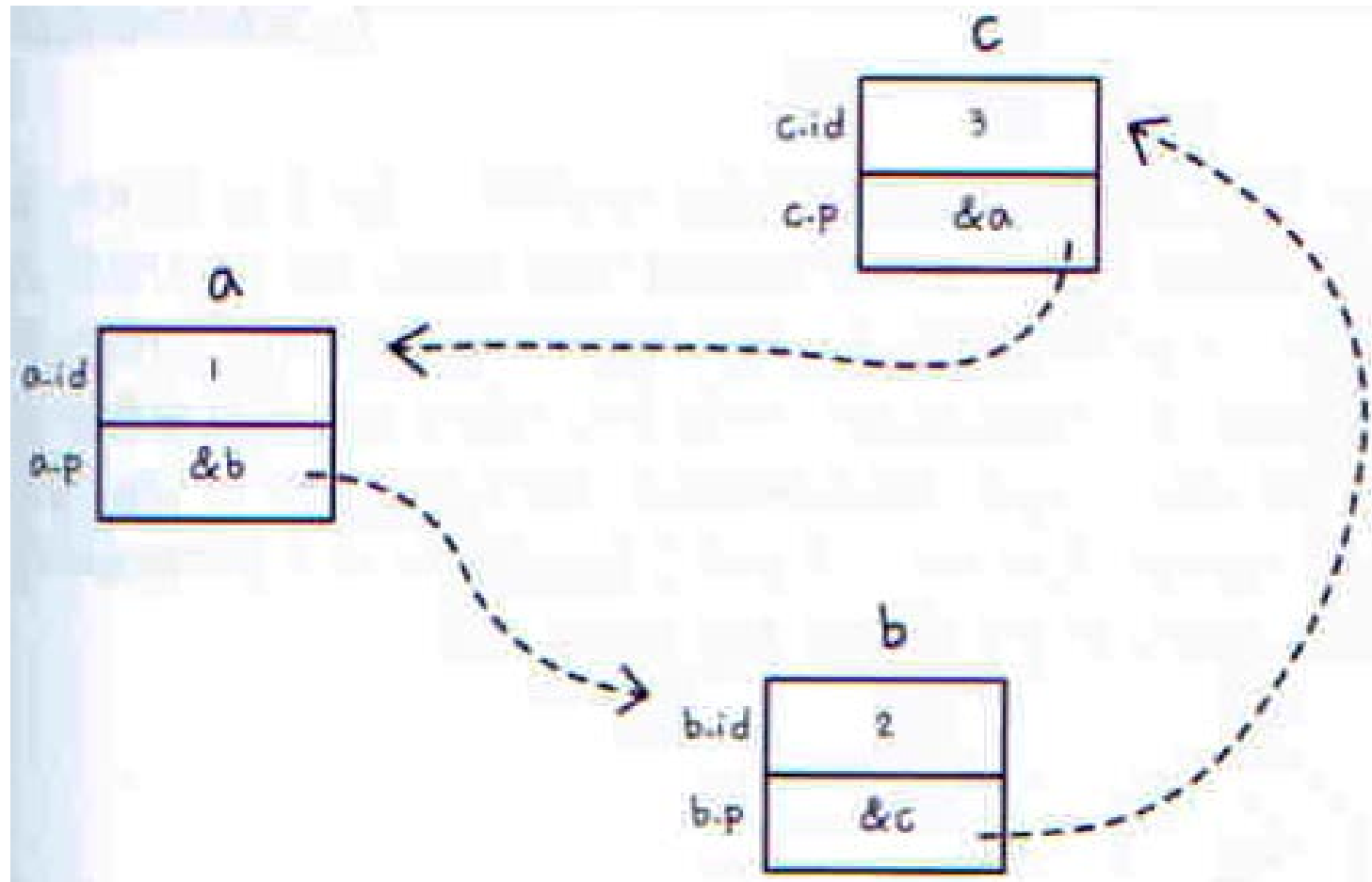
```
#include <iostream>
using namespace std;

struct List
{
    int id;
    List *p;
};

void main ()
{
    List a, b, c;
    a.id = 1;  a.p = &b;
    b.id = 2;  b.p = &c;
    c.id = 3;  c.p = &a;

    cout << "a.id: " << a.id << "\n";
    cout << "b.id: " << a.p->id << "\n";
    cout << "c.id: " << a.p->p->id << "\n";
}
```

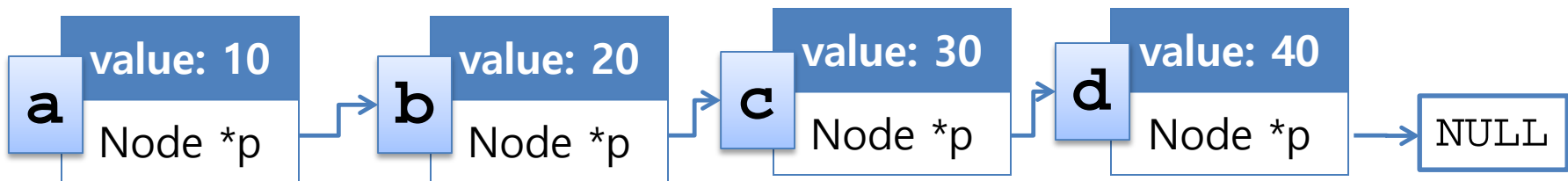
구조체를 가리키는 멤버 포인터



연습: Linked List

- 다음과 같은 구조체를 만들고 Node 변수 a, b, c, d 4개를 만든다.
- 각 변수를 그림과 같이 연결한다.
- a부터 시작해 **반복문** (while) 을 통해 모든 변수의 value값을 출력 해 보자

```
struct Node
{
    int value;
    Node *p;
};
```



템플릿과 STL

템플릿 (Template) ?

- te
1
2/
3
4/
5/
당



氏)

키에 할

일반성이 필요한 예제:

- 두 정수 중 큰 수를 알려주는 max 함수

```
int max(int a, int b)
{
    if(a>b) return a;
    else return b;
}
```

두 실수 간의 비교가 필요하다면???

오버로딩! OVERLOADING!

일반성이 필요한 예제:

- 두 실수 중 큰 수를 알려주는 max 함수

두 double 간의 비교가 필요하다면???

```
float max(float a, float b)
```

```
{
```

```
    if(a > b)
```

```
}
```

오버로딩! OVERLOADING!

두 char 간의 비교가 필요하다면???

오버로딩! OVERLOADING!

두 complex 간의 비교가 필요하다면???

오버로딩! OVERLOADING!



한 번에 해결할 수 없을까?

템플릿 (Template) !

다시 붕어빵...



Copyright Doosan Encyber.



Copyright Doosan Encyber.



Copyright Doosan Encyber.



템플릿 (Template) !

템플릿의 활용방법

- 템플릿 클래스
- 템플릿 함수

일반적인 스마트 포인터

```
class AutoArray
{
public:
    AutoArray(int *ptr)
    {
        _ptr = ptr; }
    ~AutoArray()
    {
        delete [] _ptr; }
    int& operator[] (int index)
    {
        return _ptr[index]; }
private:
    int *_ptr;
};
```

스마트 포인터 활용

```
int main()
{
    AutoArray arr( new int[100] );
    arr[20] = 30;

    return 0;
}
```

AutoArray는 정수형에 대해서만 동작!

정수형 스마트 포인터

```
class AutoArray
{
public:
    AutoArray(int * ptr)
    {
        _ptr = ptr;
    }
    ~AutoArray()
    {
        delete [] _ptr;
    }
    int& operator[] (int index)
    {
        return _ptr[index];
    }
private:
    int * _ptr;
};
```

템플릿 클래스의 사용(1)

- 모든 타입의 배열을 위한 스마트 포인터 클래스

```
template <typename T>
class AutoArray
{
public:
    AutoArray(T* ptr)
    { _ptr = ptr; }
    ~AutoArray()
    { delete[] _ptr; }
    T& operator[] (int index)
    { return _ptr[index]; }
private:
    T* _ptr;
};
```

템플릿 클래스의 사용(2)

- 모든 타입의 배열을 위한 스마트 포인터 클래스

```
int main()
{
    AutoArray<float> arr( new float [100] );

    arr[0] = 99.99f;

    return 0;
}
```

템플릿 클래스의 사용(3)

- 템플릿 매개 변수의 사용

typename 대신에
class라고 적을 수 있다

템플릿 클래스의
정의

```
template <typename A, typename B, int MAX>  
class TwoArray  
{  
    // 중간 생략  
    A arr1[MAX];  
    B arr2[MAX];  
};
```

템플릿 클래스의
사용

```
TwoArray<char, double, 20> arr;
```


템플릿 클래스의 이해

- 템플릿 클래스 객체를 생성하는 순간
➔ 컴파일러 내부적으로 알맞은 클래스 생성
– 개발자가 만든 코드

```
template< typename A, typename B, int MAX >
class TwoArray
{
    // 중간 생략
    A arr1[ MAX ];
    B arr2[ MAX ];
};

TwoArray< char, double, 20 > arr;
```

템플릿 클래스의 이해

- 템플릿 클래스 객체를 생성하는 순간
➔ 컴파일러 내부적으로 알맞은 클래스 생성
– 컴파일러가 새로 만든 클래스

```
class TwoArray_char_double_20 // 이 이름은 임의로 만든 것
{
    // 중간 생략
    char arr1[ 20 ];

    컴파일러에 의해 생성된 클래스

    double arr2[ 20 ];
};
```

Standard Template Library

- C++ = 언어
 - 클래스, 멤버함수, 상속, 템플릿, 오버로딩 등의 기술
- STL = 템플릿 기반의 편한 사용 툴:
 - 클래스, 함수 등의 집합체
 - array 나 pointer를 대체하는 데이터 관리에 용이

Standard Template Library

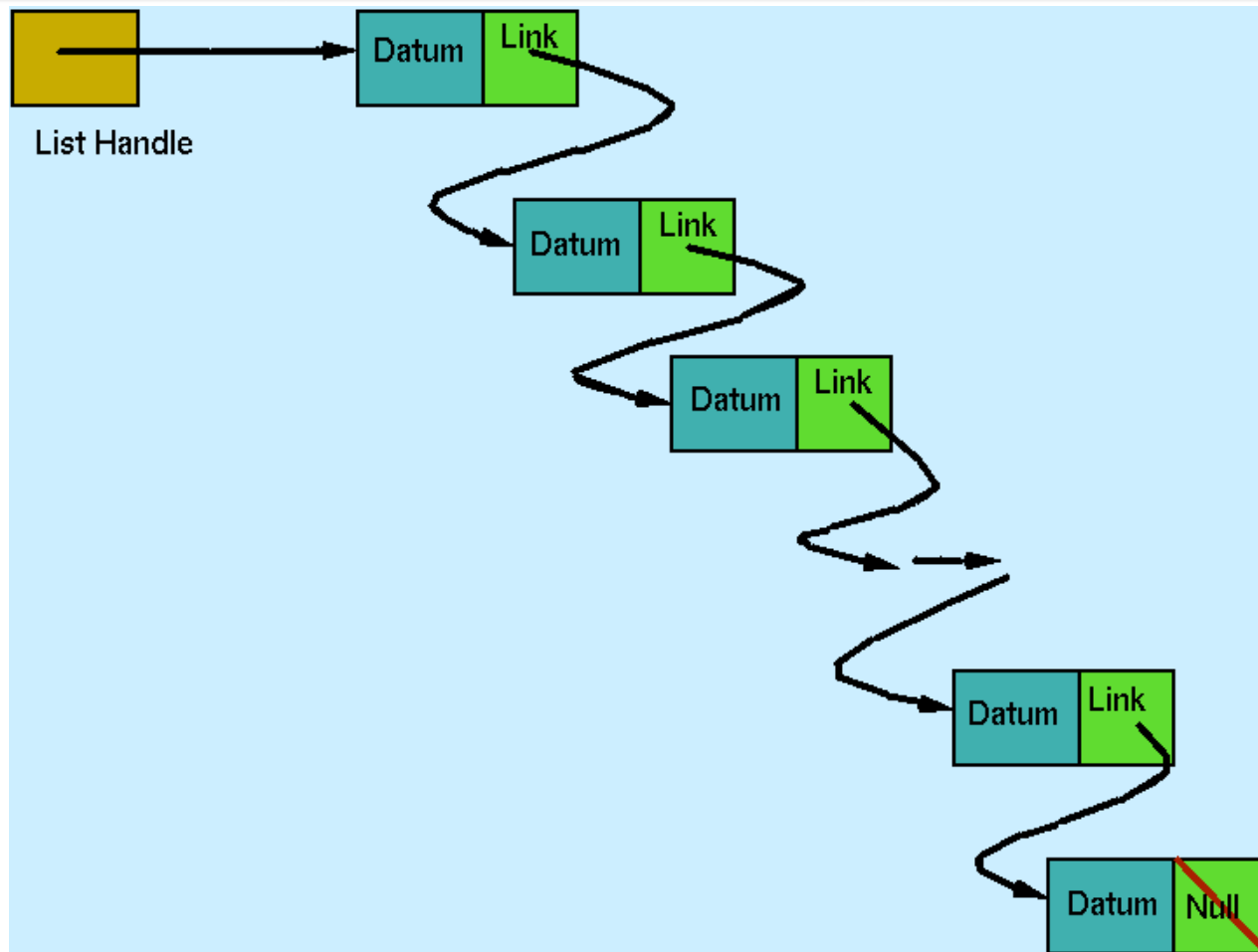
- 구성:
 - Container
 - 같은 타입의 원소를 관리하는 데이터 구조 지원 클래스
 - list, vector, deque, ...
 - Algorithm
 - Container에 대해 복사, 전환, 병합, 정렬
 - random_shuffle, replace, fill, remove, sort, ...
 - Iterator
 - 원소의 관리 방법 (= 포인터)
 - 함수 개체
 - 함수 연산자 () 를 오버로딩

STL 컨테이너

- 자주 사용하는 STL의 컨테이너 클래스

클래스	요약
vector	동적인 배열. 동적으로 원소의 개수를 조절할 수 있는 배열이다.
list	링크드 리스트.
deque	배열과 링크드 리스트의 장점을 모아놓은 컨테이너. 배열만큼 원소에 접근하는 시간이 빠른 동시에, 맨 앞과 끝에 원소를 추가하고 제거하는 시간에 링크드 리스트 만큼 빠르다.
map	<p>맵은 원소를 가리키는 인덱스까지도 다양한 타입을 사용할 수 있다. 예를 들어서 다음과 같이 문자열 타입의 인덱스를 사용할 수도 있다.</p> <pre>map<string, string> m; m["add"] = "더하다.;"</pre>

Linked List



장점: 데이터의 추가 및 삭제가 용이한 데이터 구조
단점: 데이터의 접근이 순차적으로만 가능

CList

- MFC에서 제공하는 **Linked List Template Class**

정의 방법:

```
CList < datatype > a;
```

데이터 추가:

```
CList::AddTail(..)
```

```
CList::AddHead(..)
```

데이터 삭제:

```
CList::RemoveTail();
```

```
CList::RemoveHead();
```

```
CList::RemoveAt(..)
```

사용예:

```
CList < int > a;
```

데이터 추가:

```
CList::AddTail(3)
```

```
CList::AddHead(4)
```

데이터 삭제:

```
CList::RemoveTail();
```

```
CList::RemoveHead();
```

```
CList::RemoveAt(..)
```

iterator:
POSITION

Iterator

- Container에서 Pointer와 같은 역할을 하는 것
- MFC의 iterator 변수형 : **POSITION**

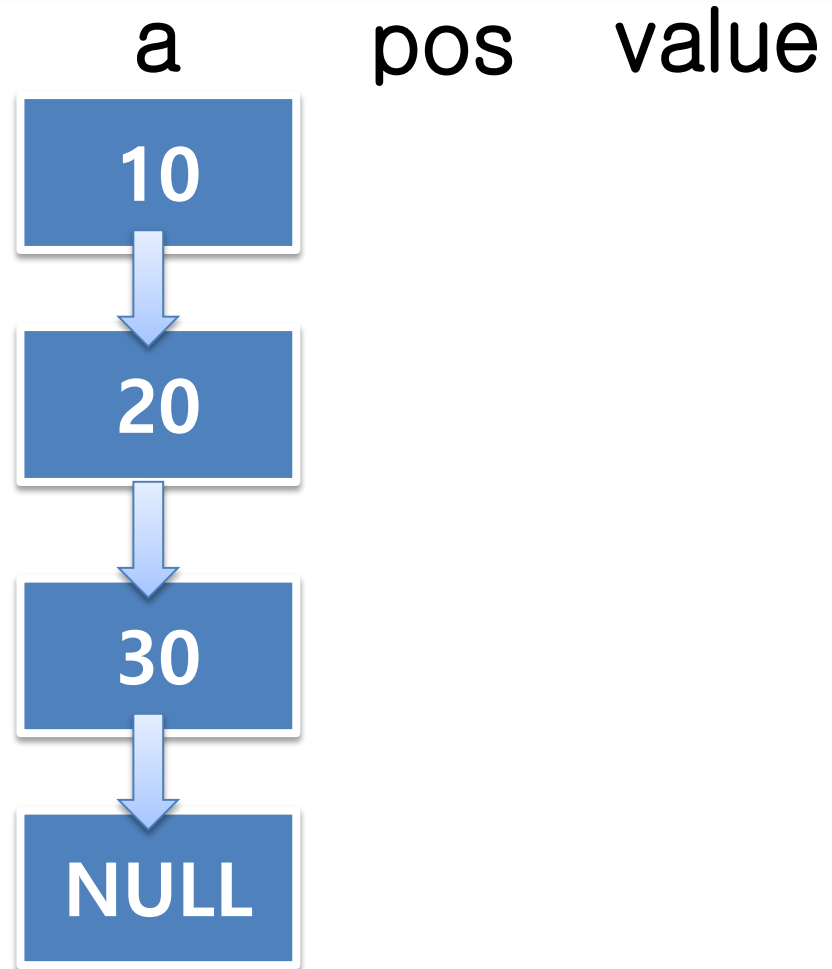
```
CList <int> a;
a.AddTail(10);
a.AddTail(20);
a.AddTail(30);

POSITION pos = a.GetHeadPosition(); // 첫 데이터 위치 반환
while(pos != NULL)
{
    int value = a.GetNext(pos); // pos위치의 데이터 반환
                                // pos는 다음 위치 가르킴
}
```


Iterator

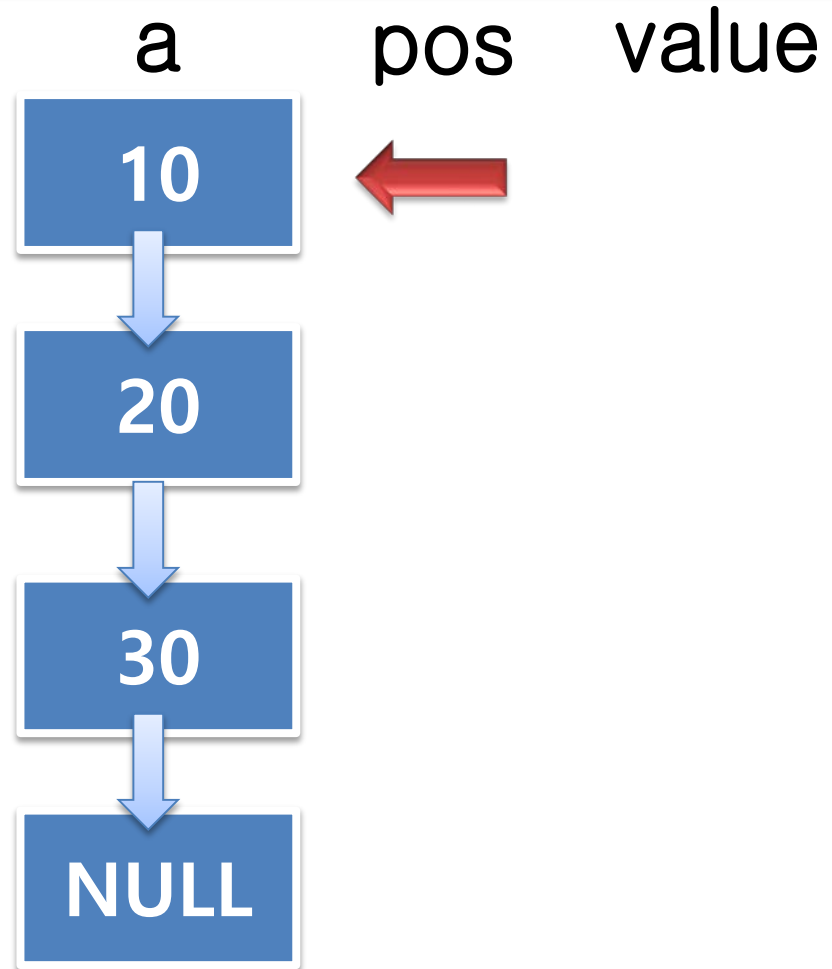
```
CList <int> a;  
a.AddTail(10);  
a.AddTail(20);  
a.AddTail(30);
```

```
POSITION pos = a.GetHeadPosition();  
while(pos != NULL)  
{  
    int value = a.GetNext(pos);  
}
```



Iterator

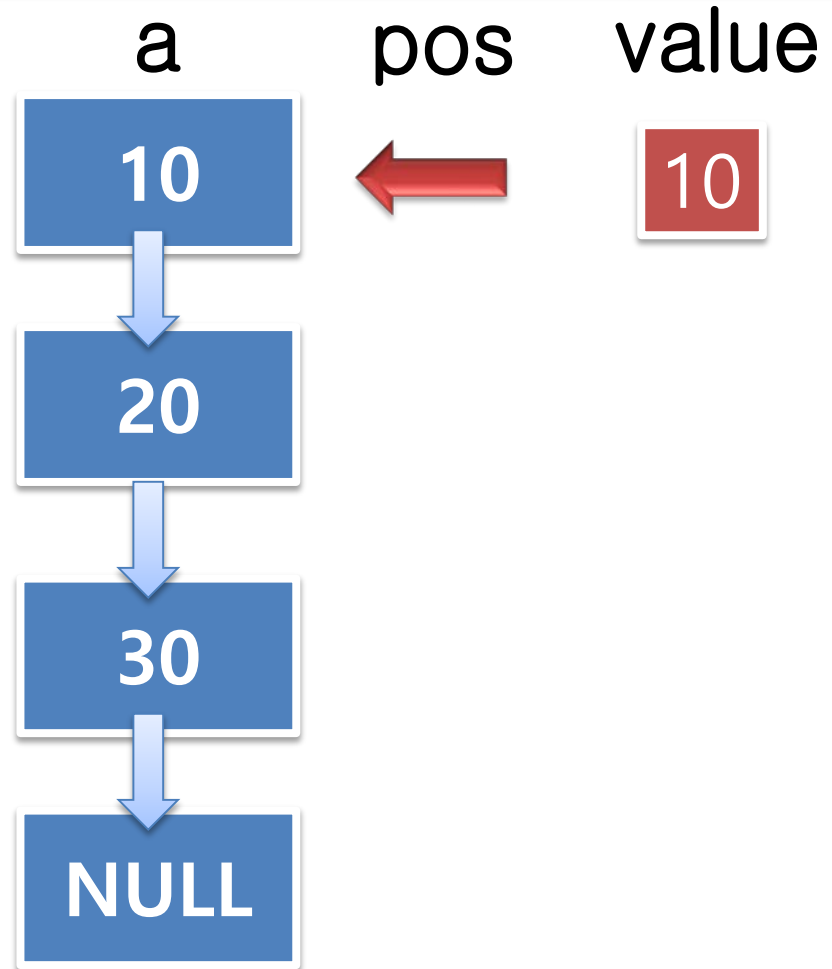
```
CList <int> a;  
a.AddTail(10);  
a.AddTail(20);  
a.AddTail(30);  
  
POSITION pos = a.GetHeadPosition();  
  
while(pos != NULL)  
{  
    int value = a.GetNext(pos);  
}
```



Iterator

```
CList <int> a;  
a.AddTail(10);  
a.AddTail(20);  
a.AddTail(30);  
  
POSITION pos = a.GetHeadPosition();  
  
while(pos != NULL)  
{  
    int value = a.GetNext(pos);  
}
```

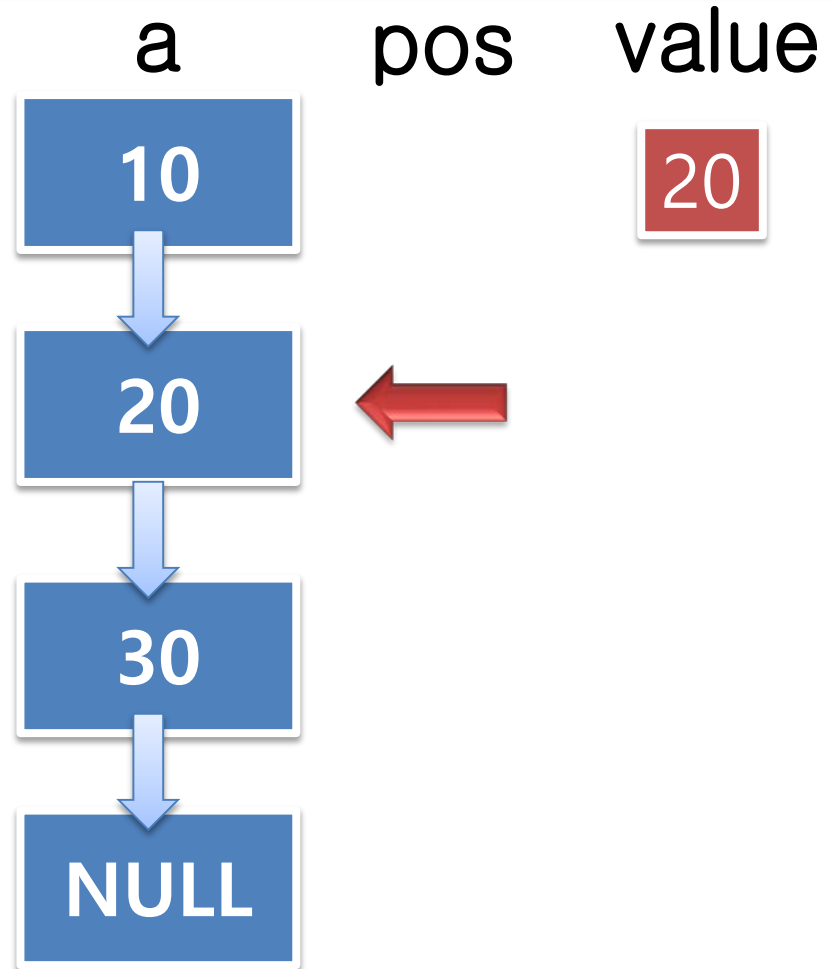
1회 호출



Iterator

```
CList <int> a;  
a.AddTail(10);  
a.AddTail(20);  
a.AddTail(30);  
  
POSITION pos = a.GetHeadPosition();  
while(pos != NULL)  
{  
    int value = a.GetNext(pos);  
}
```

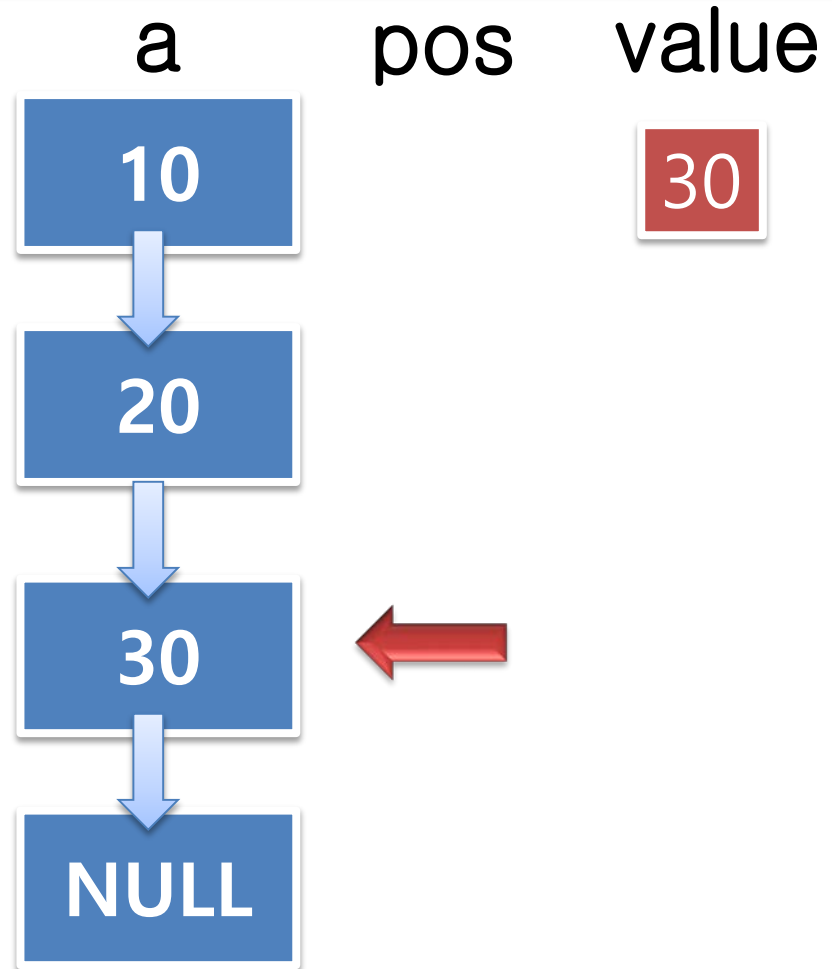
2회호출



Iterator

```
CList <int> a;  
a.AddTail(10);  
a.AddTail(20);  
a.AddTail(30);  
  
POSITION pos = a.GetHeadPosition();  
  
while(pos != NULL)  
{  
    int value = a.GetNext(pos);  
}
```

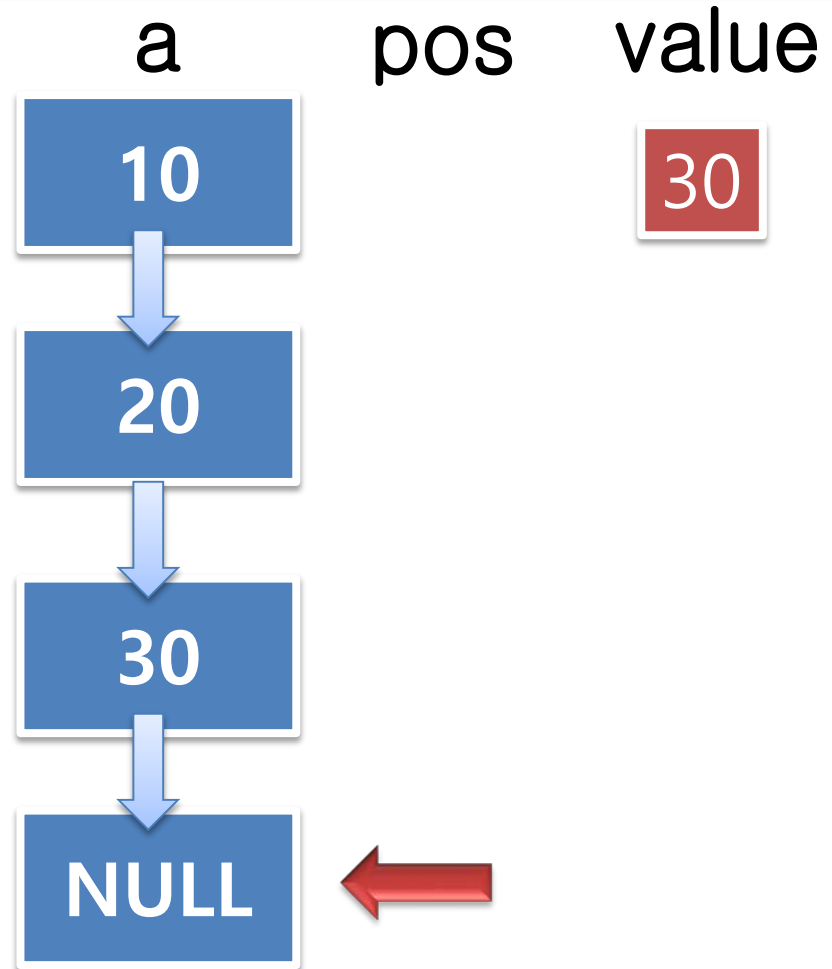
3회 호출



Iterator

```
CList <int> a;  
a.AddTail(10);  
a.AddTail(20);  
a.AddTail(30);  
  
POSITION pos = a.GetHeadPosition();  
  
while(pos != NULL)  
{  
    int value = a.GetNext(pos);  
}
```

종료



Insertion

- CList의 구성 요소(element) 추가

```
// 맨 앞에 추가  
CList::AddHead( value )  
  
// 맨 뒤에 추가  
CList::AddTail( value )  
  
// 임의의 위치에 추가  
CList::InsertAfter (POSITION, value)
```

Retrieval/Modification

- CList의 구성 요소(element) 회수 및 수정

```
// 값 얻어오기  
value = CList::GetAt( POSITION )  
  
// 값의 reference 얻어오기  
value & = CList::GetAt( POSITION )
```

```
CList <int> a;  
a.AddHead(10);  
a.AddHead(20);  
a.AddHead(30);
```

```
POSITION pos;  
pos = a.GetHeadPosition();  
  
int b = a.GetAt(pos); // value  
int &c = a.GetAt(pos); // reference
```


Removal

- CList의 구성 요소(element) 삭제

```
// 맨 앞 삭제
```

```
CList::RemoveHead( )
```

```
// 맨 뒤 삭제
```

```
CList::RemoveTail( )
```

```
// 임의의 위치 삭제
```

```
CList::RemoveAt(POSITION)
```

```
// 모두 삭제
```

```
CList::RemoveAll( )
```

MFC Container Class

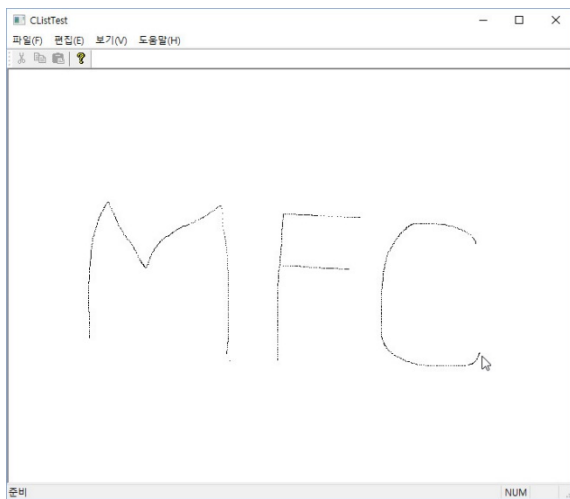
- 그 밖의 dynamic data 관리를 위한 MFC collection classes

CList
CVector
CMap

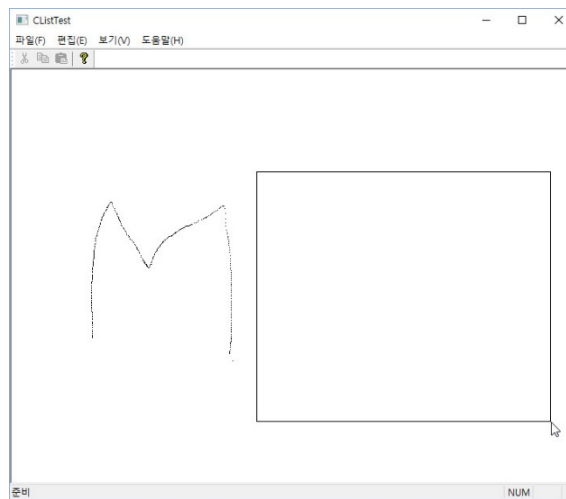
```
#include <afxtempl.h> // MFC suport for Collections
```

연습: 점찍고 지우기

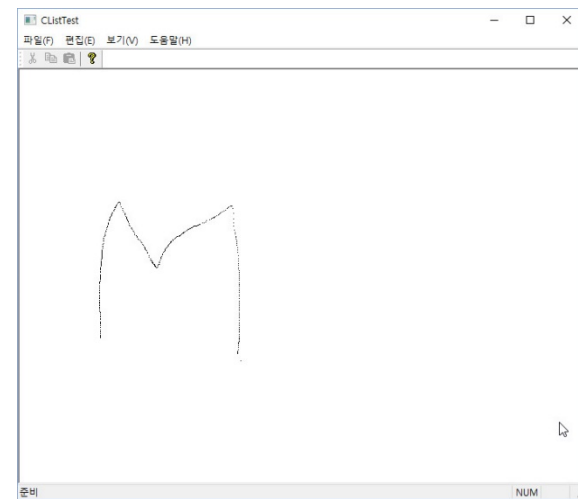
- 마우스 왼쪽 버튼 드래깅을 하면 그 위치에 점을 그리되 점들의 위치를 CList를 이용하여 저장
- 마우스 오른쪽 버튼 드래깅으로 사각형 안에 든 점들을 삭제



마우스 왼쪽
드래깅으로 점찍기



마우스 오른쪽
드래깅으로 사각형 지정



포함된 점들 지우기

Q & A