문제 해결을 위한
C++ 야매 정복 for Beginners

Sejong Algorithm Lab

강연자 소개

한정환 pizzaroot

Diamond III 2410

강재구 ash9river

Platinum V 1728

강의 목표

- C++ 편하게 이용하기
- 자료구조와 알고리즘 맛보기
  - string, vector, pair, tuple, stack, queue, deque, list, map, unordered_map, set, unordered_set,

기본 용어

- 자료구조(Data Structure)
- 알고리즘(Algorithm)

- PS (Problem Solving)
  - 문제를 적절한 수학적 관찰, 알고리즘, 자료구조 등을 활용해 해결

- CP (Competitive Programming)
  - 여러 문제들을 주어진 시간 안에 해결

알고리즘 대회와 코딩 테스트

알고리즘 대회

- 논리적인 사고력을 요구하는 창의적인 문제
- 아이디어성 문제
- 깊은 수학적 지식의 이해를 요구하는 문제

코딩 테스트

- 전형적인 문제
- 구현이 많은 문제
- 주로 실제 상황과 유사한 기능을 구현하는 문제

왜 알고리즘을 푸나요?

- 단순히 코드를 복붙(?) 하는 것보다, 문제를 해결하기까지의 모든 과정을 중요하게 생각
- 문제해결에 필요한 논리력, 사고력, 수학적 직관 등
- 구현 및 디버깅 능력 향상

바람직한 코딩 습관

- 필요한 라이브러리만 사용
- 변수 이름은 알아보기 쉽게
- 주석은 필수
- 전역 변수 사용 지양
- Etc…

```cpp
#include <iostream>
#include <memory>
int main() {
    // code that prints maximum value of a given array
    int length; // length of an array
    std::cin >> length;
    int* array = (int*)malloc(sizeof(int) * length);
    for (int i = 0; i < length; i++) {
        std::cin >> array[i];
    }
    int answer = 0;
    for (int i = 0; i < length; i++) {
        // renew max element using max function
        answer = std::max(answer, array[i]);
    }
    std::cout << answer << std::endl;
    free(array); // free allocated memory
    return 0;
}
```

PS에서는…

- 모든 라이브러리 포함
- 무의미한, 편한 변수명
- 나만 알면 돼(?) 주석 없이 작성
- 자유로운 전역 변수 사용
- Etc…

```
#include <bits/stdc++.h>
using namespace std;
int arr[10001];
int main() {
    int l, ans = 0; cin >> l;
    for (int i = 0; i < l; i++) cin >> arr[i];
    for (int i = 0; i < l; i++) ans = max(ans, arr[i]);
    cout << ans << '₩n';
    return 0;
}
```

C++를 사용해야 하는 이유

- C언어만 사용하는 것에 비해 C++ 스타일을 사용하면 코딩 실수를 할 확률이 훨씬 적음
- 이미 잘 구현된 여러 함수 및 기능들을 사용할 수 있음
- STL을 사용해 더 쉽고 빠르게 원하는 기능을 구현할 수 있음

- 다만, 학교 수업은 C로만 진행됨

C 스타일 vs C++ 스타일

C 스타일

- 포인터 사용
- malloc을 통한 동적 할당 및 해제

```c
#include <stdio.h>
#include <stdlib.h>
void add(int* a){
    *a += 3;
}
int main() {
    int* arr = (int*)malloc(sizeof(int) *
3);
    int b = 3;
    add(&b);
    printf("%d",b);
    free(arr);
    return 0;
}
```

C++ 스타일

- 참조자 사용
- STL을 사용한 동적 할당 해제는 STL이 해줌

```cpp
#include <iostream>
#include <vector>
void add(int& a) {
    a += 3;
}
int main() {
    std::vector<int> arr(3);
    int b = 3;
    add(b);
    std::cout << b;
    return 0;
}
```

printf와 scanf를 대신하는 입출력

```
#include <iostream>
int main(){
    int num;
    std::cin>>num;
    std::cout<<"Hello World!"<<std::endl;
    std::cout<<num<<' '<<'A';
    std::cout<<' '<<3.14<<std::endl;
    return 0;
}
```

함수 오버로딩

```
#include <iostream>
int add(int a,int b){
    return a+b;
}
int add(int a,int b,int c){
    return a+b+c;
}
int main(){
    std::cout<<add(1,2)<<std::endl;
    std::cout<<add(1,2,3);
    return 0;
}
```

이름공간(namespace)

```
#include <iostream>
namespace MySpace{
    void SimpleF(void){
        std::cout<<"My Space"<<std::endl;
    }
}
namespace YourSpace{
    void SimpleF(void){
        std::cout<<"Your Space"<<std::endl;
    }
}
```

```
int main(){
    MySpace::SimpleF();
    YourSpace::SimpleF();
    return 0;
}
```

using을 통한 이름 공간 생략

```
#include <iostream>
int main() {
    std::cin >> 변수;
    std::cout << "Hello, C++!" << std::endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {
    cin >> 변수;
    cout << "Hello, C++!" << endl;
    return 0;
}
```

using을 통한 이름 공간 명시

```cpp
#include <iostream>
using namespace std;
int main(){
    int num;
    cin>>num;
    cout<<"Hello World!"<<endl;
    cout<<num<<' '<<'A';
    cout<<' '<<3.14<<endl;
    return 0;
}
```

```cpp
#include <iostream>

int main(){
    int num;
    std::cin>>num;
    std::cout<<"Hello World!"<<std::endl;
    std::cout<<num<<' '<<'A';
    std::cout<<' '<<3.14<<std::endl;
    return 0;
}
```

전역 변수 접근자 ::

```
#include <iostream>
using namespace std;
int val=100;
int main(){
    int val=200;
    cout<<val<<endl;
    cout<<::val<<endl;
}
```

새로운 자료형 bool

- true와 false의 등장

```
#include <iostream>
using namespace std;
int main() {
    bool state1 = true;
    bool state2 = false;
    cout << state1 << endl;
    cout << state2 << endl;
    state1 = 0;
    cout << state1 << endl;
}
```

참조자 &

```
#include <iostream>
using namespace std;
int main() {
    int num1 = 10;
    int& num2 = num1;
    cout << num2;
}
```

STL이란?

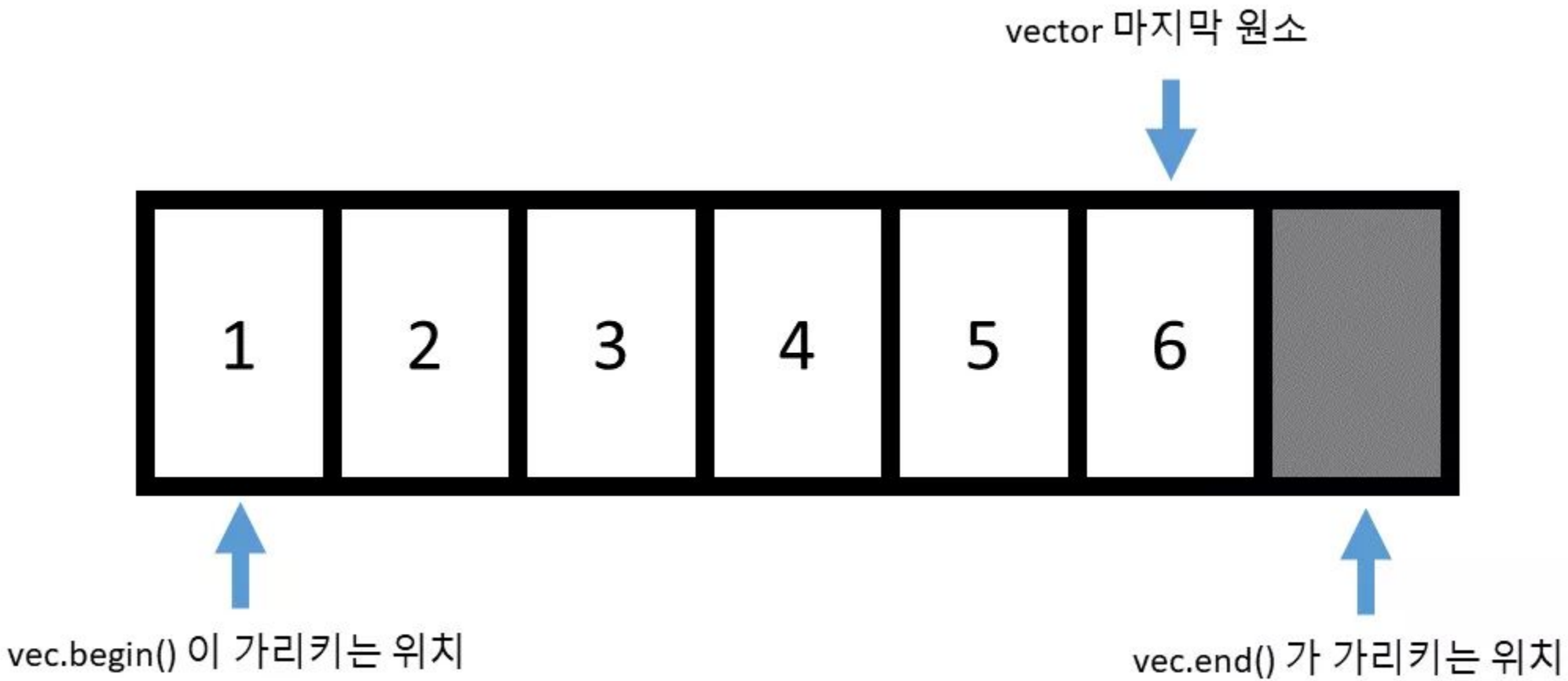
- C++에서 이미 만들어진 표준 템플릿 라이브러리
- 프로그램에 필요한 자료구조 및 알고리즘을 템플릿으로 제공

- Container
- Iterator
- Algorithm
- Functors

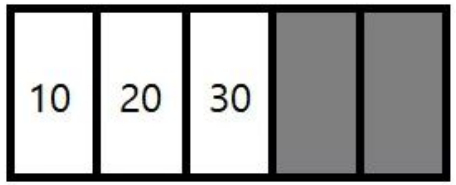
Container

- 순차 컨테이너 (Sequence Container)
  - vector
  - deque
  - list
- 컨테이너 어댑터 (Container Adaptor)
  - stack
  - queue
  - priority_queue
- 연관 컨테이너 (Associative Containers)
  - set
  - map

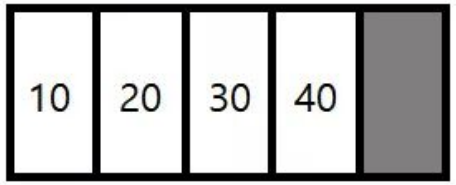
Vector



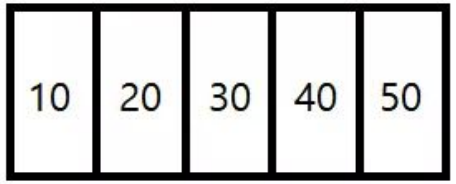
Vector



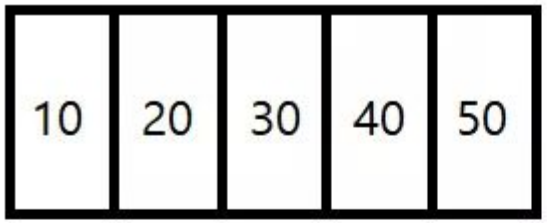
현재 벡터 길이보다 좀 더 많은 공간이 미리 할당되어 있다!



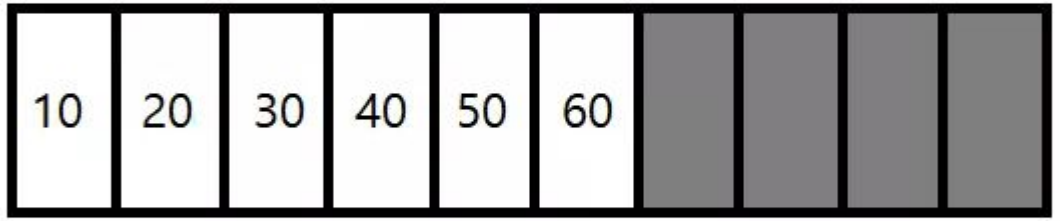
vec.push_back(40);
을 수행하면 그냥 미리 할당된 공간에 복사하면 된다. O(1)



vec.push_back(50); 까지는 문제없다!



기존 메모리는 파괴됨!



vec.push_back(60);
을 수행하면 기존의 할당된 공간이 다 차기 때문에 새롭게 더 큰 메모리를 할당하고 복사를 수행해야한다 O(n)

#include <vector>

```
vector<int> v1; //1차원 배열 생성
vector<int> v1_init(10,987654321); //1차원 배열 [10]생성, 각 배열의 원소를 987654321로 초기화
vector<vector<int>> v2; //2차원 배열 생성
vector<vector<int>> v2_init(10,vector<int>(11,987654321)); // [10][11] 생성, 987654321로 초기화


v1[9]=987654321;
v2_init[3][4]=145; //원소 접근은 배열처럼
v1.push_back(155); //push_back()으로 원소의 개수를 하나 늘릴 수 있다.
v1.pop_back();     //제일 마지막 원소 제거
v2_init[3].push_back(1616); // 이런 경우에는 v2[3]만 원소의 개수가 12개가 된다
```

#include <vector>

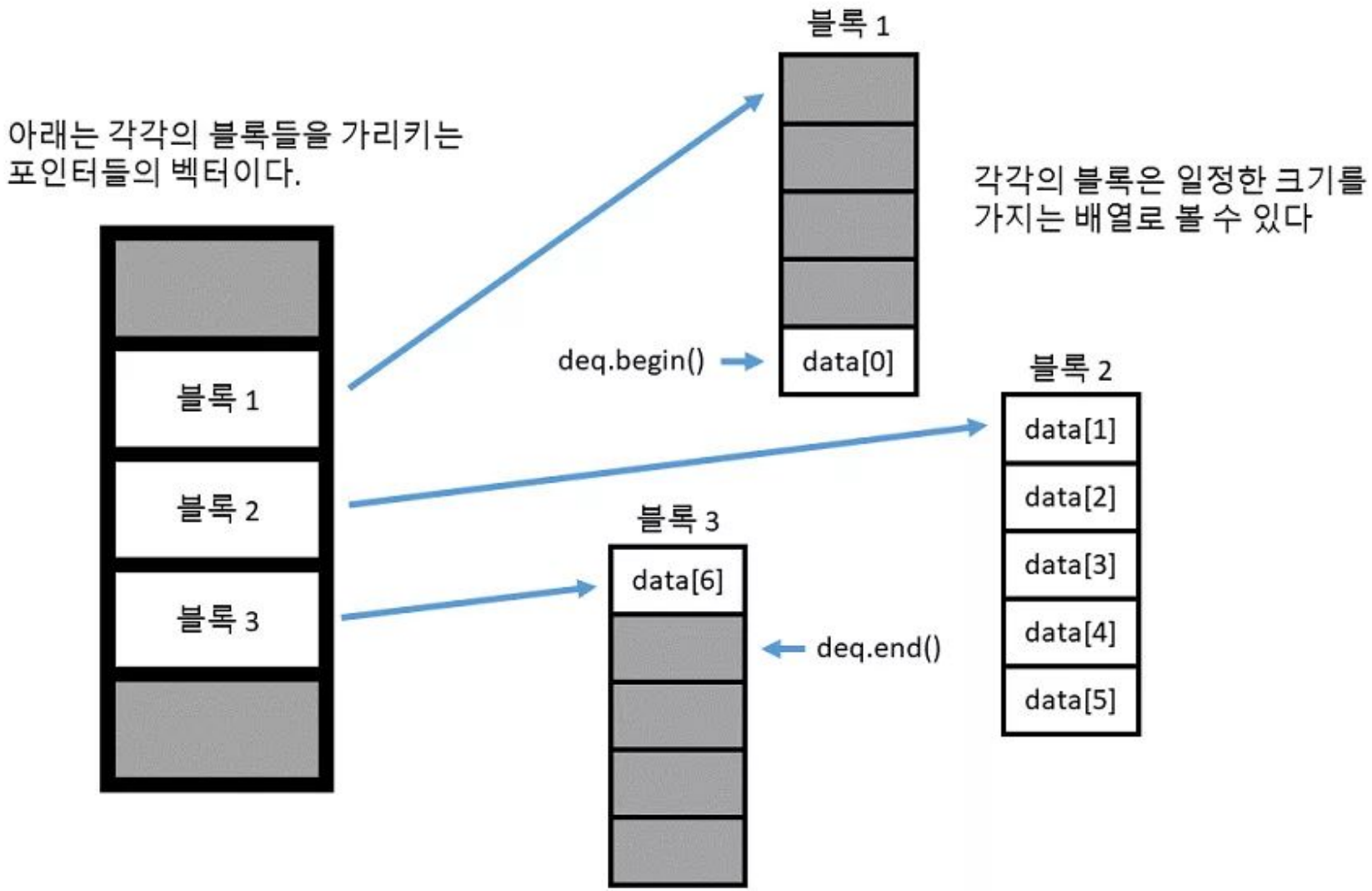
```
vector<int> v;
v.resize(10);
vector<vector<int>> v2;
v2.resize(10,vector<int>());
vector<vector<int>> v3;
v3.resize(10,vector<int>(11));
```

```
vector<int> v1(10,0);
v1.begin(); //v1의 제일 첫 주소
v1.end();   //v1의 가장 마지막 인덱스+1의 주소
v1.back();  //v1의 가장 마지막 인덱스의 원소
v1.pop_back();     //제일 마지막 원소 제거
cout<<v1.size();
v1.clear(); //v1의 모든 원소 삭제
v1.empty(); //비었는지 확인
```

#include <vector>

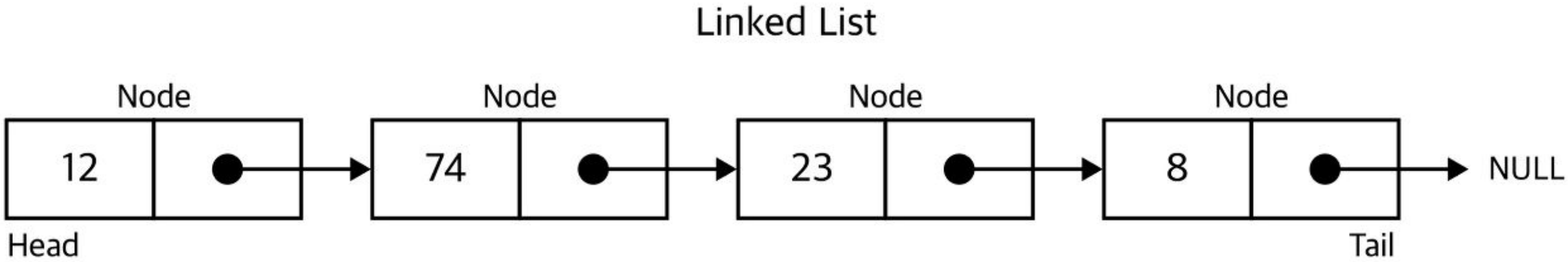
```
vector<int> table(10,5);
 for(int i = 0; i < 5; ++i) table[i]= i;
sort(table.begin(),table.end(),greater<>()); //table 내림차순 정렬
sort(table.begin(),table.end()); //table 오름차순 정렬
table.erase(unique(table.begin(),table.end()),table.end());
//table의 중복 원소 제거(정렬 필수)
```

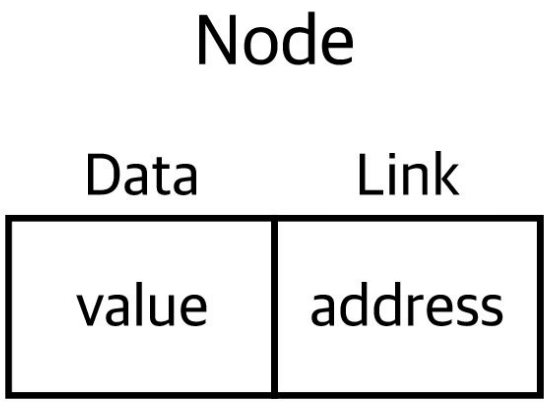
Deque


아래는 각각의 블록들을 가리키는
포인터들의 벡터이다.
블록 1
블록 2
블록 3
블록 1
deq.begin()
data[0]
각각의 블록은 일정한 크기를
가지는 배열로 볼 수 있다
블록 2
data[1]
data[2]
data[3]
data[4]
data[5]
블록 3
data[6]
deq.end()

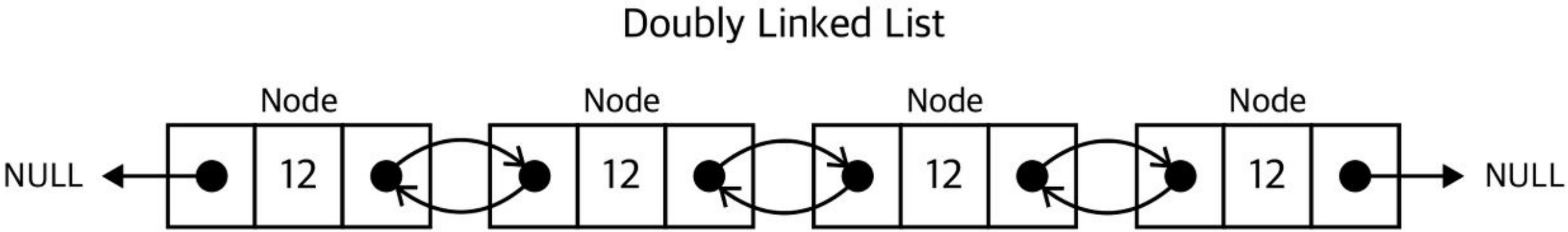
```
#include <deque>

    deque<int> dq;
    dq.push_back(1);
    dq.push_front(2);
    dq.pop_back();
    dq.pop_front();
```

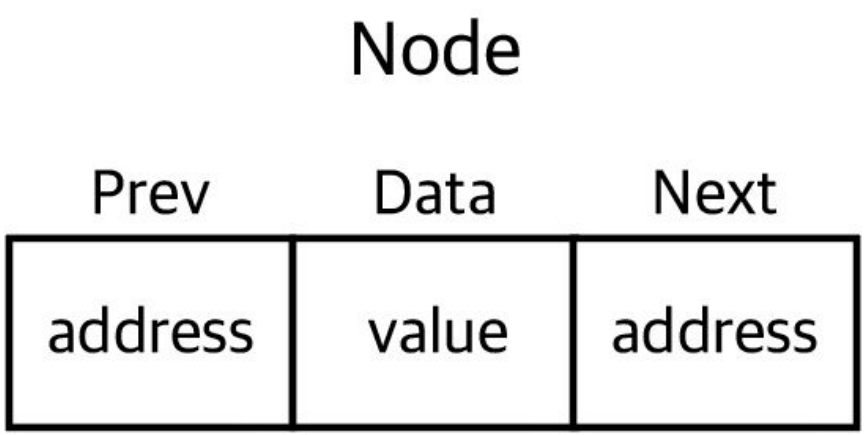
연결 리스트(Linked List)



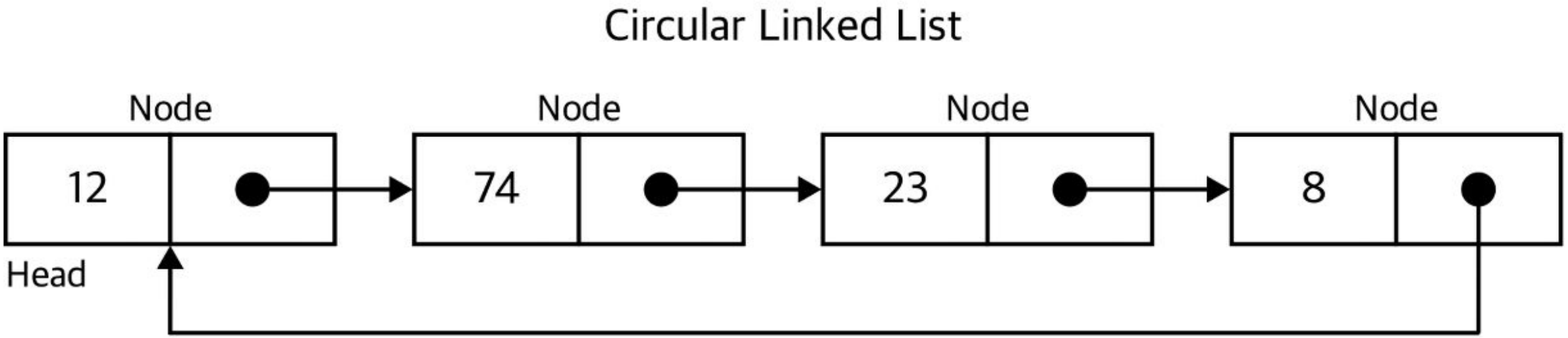


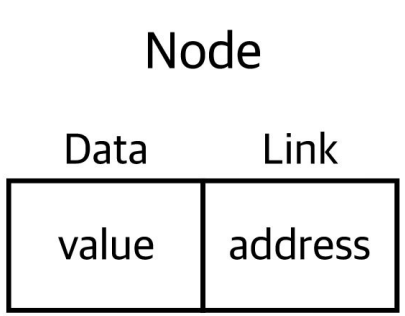
연결 리스트(Linked List)



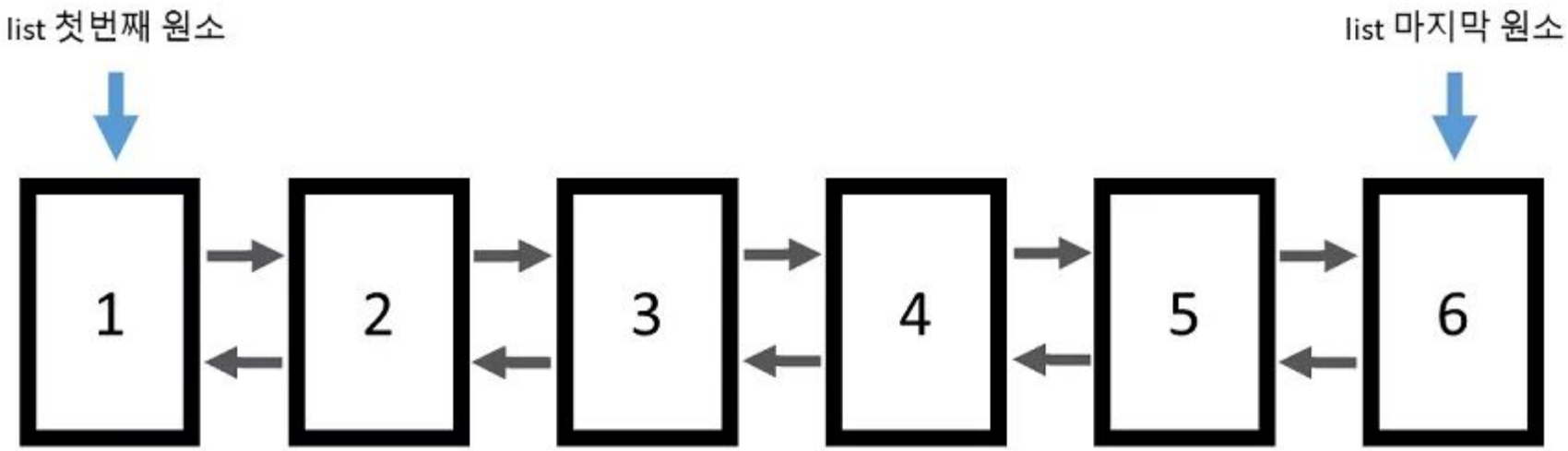


연결 리스트(Linked List)





List



이와 같이 각각의 원소가 앞쪽과 뒤쪽에
오는 원소들을 가리키고 있습니다.

#include <list>

```
list<int> a;
a.push_back(22);
a.push_back(33);
a.push_front(11);
a.push_back(44);
a.push_back(55);

list<int>::iterator iter = a.begin();

for(iter=a.begin(); iter!=a.end(); iter++)
{
    cout << *iter << endl;
}
```

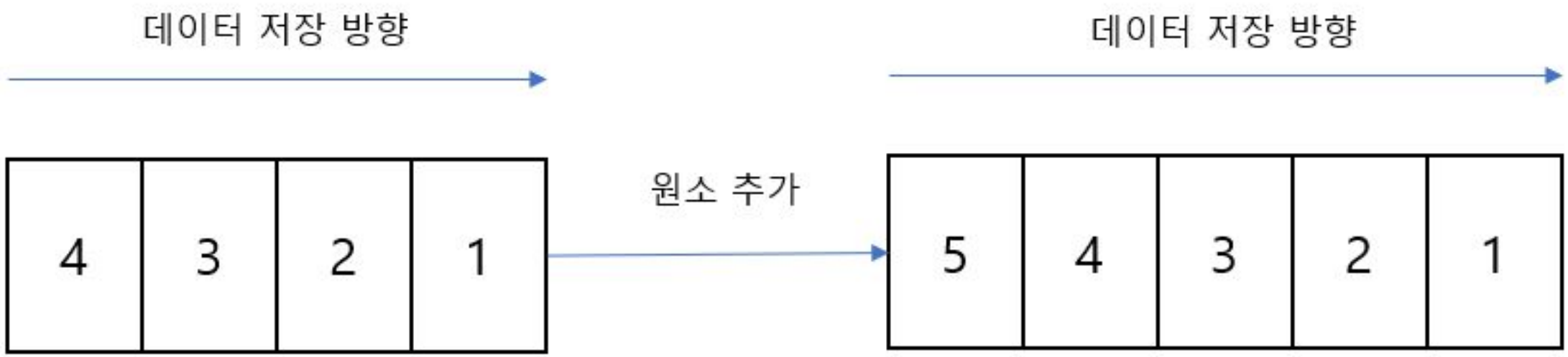
#include <list>

```
a.pop_front();
a.pop_back();

for(iter=a.begin(); iter!=a.end(); iter++)
{
    cout << *iter << endl;
}
cout << a.size() << endl;
cout << a.empty() << endl;
cout << a.front() << endl;
cout << a.back() << endl;
cout <<endl;
```

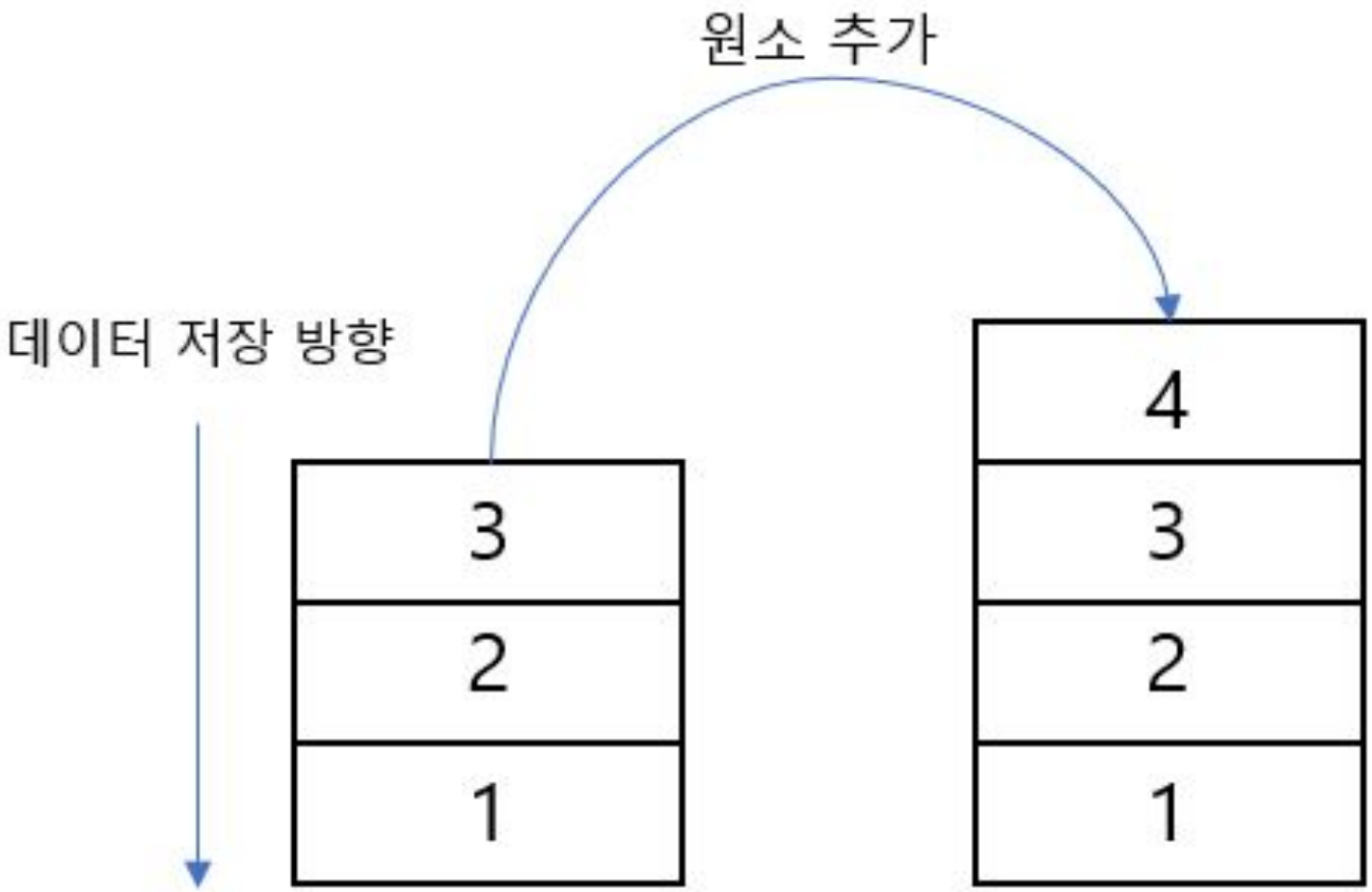
#include <list>

```
iter++;
iter++;
a.insert(iter, 55555);
for(iter=a.begin(); iter!=a.end(); iter++)
{
    cout << *iter << endl;
}
```

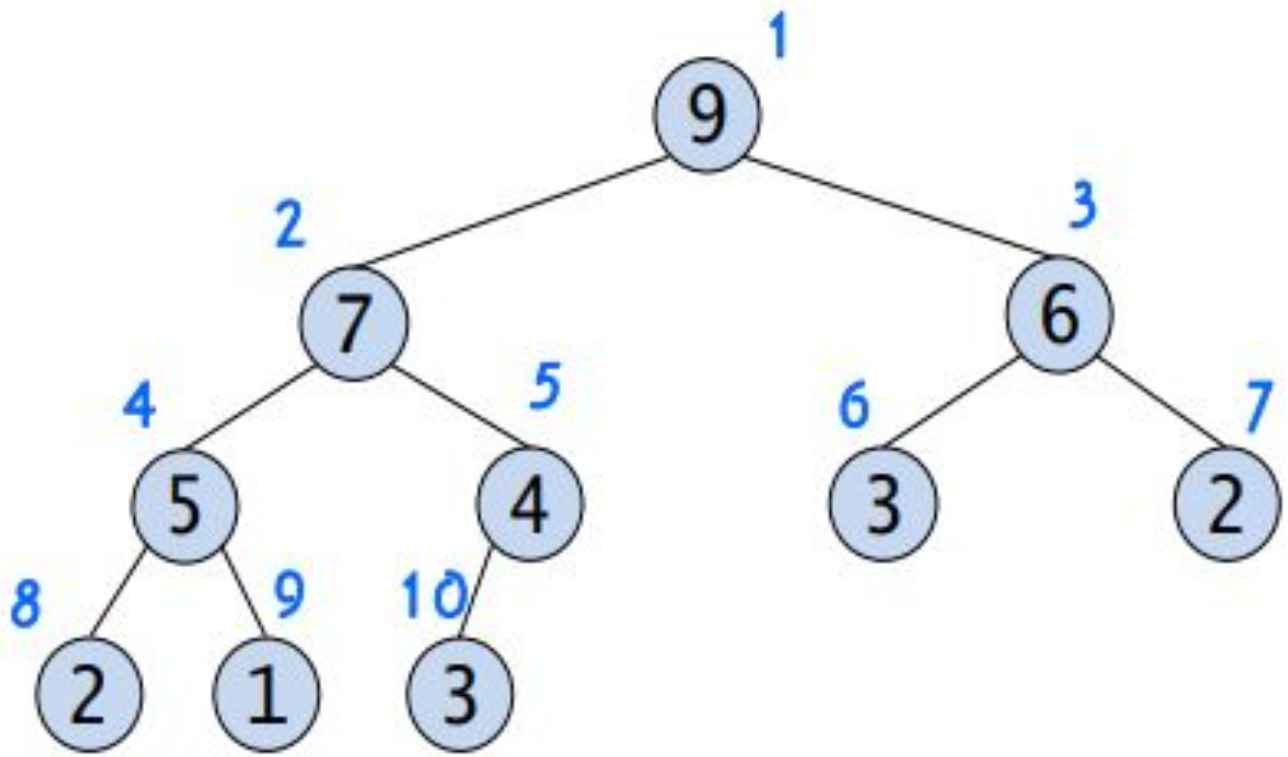
Queue



Stack



Priority Queue



0	
1	9
2	7
3	6
4	5
5	4
6	3
7	2
8	2
9	1
10	3
11	