게임 자료구조와 알고리즘

-CHAPTER20-

SOULSEEK

목차

원소를 수정하는 알고리즘

1. 원소를 수정하는 알고리즘

알고리즘	설명(설명에 사용되는 p는 구간[b,e)의 반복자)
p=copy(b,e,t)	구간**[b,e)**의 원소를 **[t,p)**로 모두 복사한다**.**
p=copy_backward(b,e,t)	구간**[b,e)**의 원소를 마지막 원소부터 **[b,f)**로 모두 복사한다**.**
fill(b,e,x)	구간**[b,e)**의 모든 원소를 **x**로 채운다**.**
fill_n(b,n,x)	구간**[b,b+n)**의 모든 원소를 **x**로 채운다**.**
f=for_each(b,e,f)	구간**[b,e)**의 모든 원소에 **f(*p)**동작을 적용한다**.f**를 다시 되돌려 준다**.**
generate(b,e,f)	구간**[b,e)**의 모든 원소를 **f()**로 채운다**.**
generate_n(b,n,f)	구간**[b,b+n)**의 모든 원소를 **f()**로 채운다**.**
iter_swap(p,q)	반복자 **p,q**가 가리키는 ***p**와 ***q**의 원소를 교환한다**.**
p=merge(b,e,b2,e2,t)	정렬된 순차열 **[b,e)**와 **[b2,e2)**를**[t,p)**로 병합 정렬한다**.**
p=merge(b,e,b2,e2,t,f)	정렬된 순차열**[b,e)**와**[b2,e2)**를 **[t,p)**로 병합 정렬한다**. F**사용해서 비교
replace(b,e,x,x2)	구간**[b,e)**의 **x**인 원소를 **x2**로 수정한다**.**
replace_if(b,e,f,x2)	구간**[b,e)**의 **f(*p)**가 참인 원소를 **x2**로 수정한다**.**
p=replace_copy(b,e,t,x,x2)	구간**[b,e)**의 **x**인 원소를 **x2**로 수정한다**.**
p=replace_copy_if(b,e,t,f,x2)	구간**[b,e)**의 **f(*p)**가 참인 원소를 **x2**로 수정하여**[t,p)**로 복사한다**.**

1. 원소를 수정하는 알고리즘

알고리즘	설명(설명에 사용되는 p는 구간[b,e)의 반복자)
swap(a,b)	**a**와 **b**를 교환한다**.**
swap_ranges(b,e,b2)	구간**[b,e)**의 원소와 구간**[b2,b2+(e-b))**의 원소를 교환
p=transform(b,e,t,f)	구간**[b,e)**의 모든 원소를 **f(*p)**하여 **[t,t+(e-b))**에 저장한다**. P**는 저장된 마지막 원소의 반복자**(t+(e-b))**다
p=transform(b,e,b2,t,f)	구간**[b,e)**와**[b2,b2+(e-b))**의 두 순차열의 반복자**p,q**일 때 모든 원소를 **f(*p,*q)**하여**[t,t+(e-b))**에 저장한다**. P**는 저장된 마지막 원소의 반복자**(t+(e-b))**다**.**

1. 원소를 수정하는 알고리즘

copy() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(50);

	vector<int> v2(5); //size 5인 vector 생성

	vector<int>::iterator iter;
	// 구간 [v1.begin(), v1.end())의 모든 원소를 [v2.begin(), iter)의 순차열로 복사.
	iter = copy(v1.begin(), v1.end(), v2.begin());
	cout << "v2 마지막 원소: " << *(iter - 1) << endl;

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
		cout << v2[i] << " ";
	cout << endl;
}
```

주의사항 : **[v2.begin(),v2.end())**의 순차열은
[v1.begin(),v1.end()) 순차열 이상의 원소를 가져야 한다.

1. 원소를 수정하는 알고리즘

copy_backward() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(50);


	vector<int> v2(10); //size 10인 vector 생성

	vector<int>::iterator iter;
	// 구간 [v1.begin(), v1.end())의 모든 원소를
	// [iter, v2.end())의 순차열로 마지막 원소부터 복사.
	iter = copy_backward(v1.begin(), v1.end(), v2.end());
	+cout << "v2 첫 원소: " << *iter << endl;+
	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
		cout << v2[i] << " ";
	cout << endl;
}-
```

1. 원소를 수정하는 알고리즘

fill(), fill_n() 사용 예제..

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	// 구간 [v.begin(), v.end())의 모든 원소를 0으로 채운다.
	fill(v.begin(), v.end(), 0);
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	// 구간 [v.begin(), v.begin()+3)의 모든 원소를 55로 채운다.
	fill_n(v.begin(), 3, 55);
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

1. 원소를 수정하는 알고리즘

for_each()알고리즘을 사용한 원소의 수정 예제..

```
void Func(int& r)
{
	r += 5;
}

void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	for_each(v.begin(), v.end(), Func);
		cout << "v : ";

	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

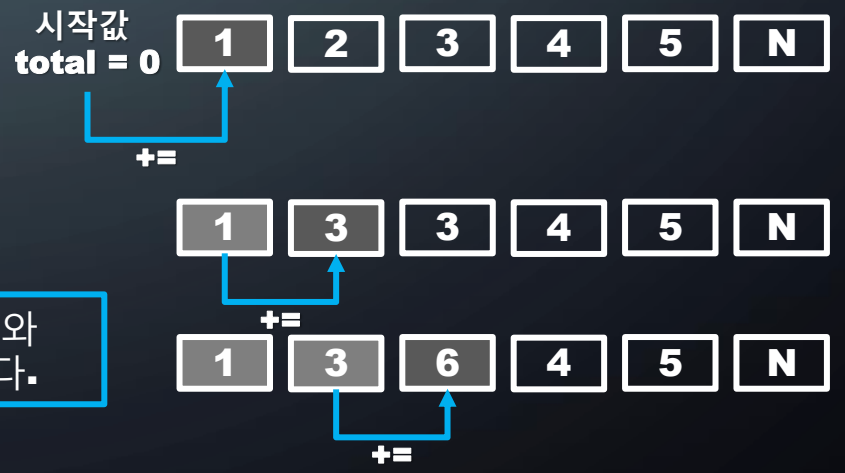
1. 원소를 수정하는 알고리즘

함수자를 사용한 **for_each()** 알고리즘 예제..

```
void main()
{
	vector<int> v;
	v.push_back(1);
	v.push_back(2);
	v.push_back(3);
	v.push_back(4);
	v.push_back(5);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	//[v.begin(), v.end()) 모든 원소를 초기값 0부터 시작하여 *p += *(p-1)를 적용합니다.
	for_each(v.begin(), v.end(), Accumulation(0));
		cout << "v : ";

	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

```
class Accumulation
{
	int total;
public:
	explicit Accumulation(int init = 0) :total(init) { }
	void operator()(int& r)
	{
		total += r;
		r = total;
	}
};
```

누적 값을 저장하고 현재 원소와
이전 원소와의 누적을 적용한다.



1. 원소를 수정하는 알고리즘

generate(), generate_n()알고리즘 사용 예제..

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	// [v.begin(), v.end())의 원소를 1~5로 채운다.
	generate(v.begin(), v.end(), Integer(1));
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	// [v.begin(), v.end())의 원소를 100~104로 채운다.
	generate(v.begin(), v.end(), Integer(100));
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
```

```
	// [v.begin(), v.begin()+3)의 원소를 1~3로 채운다.
	generate_n(v.begin(), 3, Integer(1));
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}

class Integer
{
	int data;
public:
	explicit Integer(int d = 0) :data(d) { }
	int operator()()
	{
		return data++;
	}
};
```

1. 원소를 수정하는 알고리즘

swap(), iter_swap()알고리즘 사용 예제..

```
void main()
{
	int a = 10, b = 20;

	vector<int> v;
	v.push_back(10);
	v.push_back(20);

	cout << "a= " << a << ", " << "b= " << b << endl;

	swap(a, b); //정수 a, b가 바뀐다.

	cout << " a= " << a << " , " << " b= " << b << endl;

	Vector<int>::iterator p = v.begin();
	Vector<int>::iterator q = v.begin() + 1;

	cout << " v[0]= " << v[0] << " , " << " v[1]= " << v[1] << endl;

	Iter_swap(p, q); //반복자가 가리키는 부분이 바뀐다.

	cout << "v[0]= " << v[0] << ", " << "v[1]= " << v[1] << endl;
}
```

1. 원소를 수정하는 알고리즘

merge()알고리즘 사용 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(30);
	v1.push_back(50);
	v1.push_back(60);
	v1.push_back(80);

	vector<int> v2;
	v2.push_back(20);
	v2.push_back(40);
	v2.push_back(70);

	vector<int> v3(10); //size 10인 vector 생성
	vector<int>::iterator iter_end;
	iter_end = merge(v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin());

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
		cout << v2[i] << " ";
	cout << endl;

	cout << "v3 : ";
	for (vector<int>::size_type i = 0; i < v3.size(); ++i)
		cout << v3[i] << " ";
	cout << endl;
	cout << "v3(합병 순차열): " << *v3.begin() << '~'
	<< *(iter_end - 1) << endl;
}
```

1. 원소를 수정하는 알고리즘

1. 원소를 수정하는 알고리즘

merge()알고리즘의 정렬 기준으로 greater를 사용한 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(80);
	v1.push_back(60);
	v1.push_back(50);
	v1.push_back(30);
	v1.push_back(10);

	vector<int> v2;
	v2.push_back(70);
	v2.push_back(40);
	v2.push_back(20);

	vector<int> v3(10); //size 10인 vector 생성
	vector<int>::iterator iter_end;
	//iter_end = merge(v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin(), greater<int>() );
	iter_end = merge(v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin(), Greater<int>());

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
		cout << v2[i] << " ";
	cout << endl;

	cout << "v3 : ";
	for (vector<int>::size_type i = 0; i < v3.size(); ++i)
		cout << v3[i] << " ";
	cout << endl;
	cout << "v3(합병 순차열): “;
	cout << *v3.begin() << '~' << *(iter_end - 1) << endl;
}

template <typename T>
struct Greater
{
	bool operator()(const T& left, const T& right) const
	{
		return left > right;
	}
};
```

1. 원소를 수정하는 알고리즘

replace() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v;

	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(30);
	v.push_back(30);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	//반복자 구간의 원소 30을 0으로 수정한다.
	replace(v.begin(), v.end(), 30, 0);
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

1. 원소를 수정하는 알고리즘

replace_if() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v;

	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);
	v.push_back(60);
	v.push_back(70);
	v.push_back(80);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	//반복자 구간내에 조건이 참인 원소를 모두 0으로 수정한다.
	replace_if(v.begin(), v.end(), Pred, 0);
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

```
bool Pred(int n)
{
	return 30 <= n && n <= 50;
}
```

1. 원소를 수정하는 알고리즘

replace_copy(), replace_copy_if() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(30);
	v1.push_back(50);

	vector<int> v2(6); //size: 6인 vector 생성
	vector<int> v3(6); //size: 6인 vector 생성

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
	cout << v1[i] << " ";
	cout << endl;
	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
	cout << v2[i] << " ";
	cout << endl;
	cout << "v3 : ";
	for (vector<int>::size_type i = 0; i < v3.size(); ++i)
	cout << v3[i] << " ";
	cout << endl;

	vector<int>::iterator iter_end;
	// 30인 원소를 모두 0으로 변환하여 [v2.begin(), iter_end) 순차열에 저장한다.
	iter_end = replace_copy(v1.begin(), v1.end(), v2.begin(), 30, 0);
	// 30이하인 원소를 모두 0으로 변환하여 [v3.begin(), iter_end) 순차열에 저장한다.
	iter_end = replace_copy_if(v1.begin(), v1.end(), v3.begin(), Pred, 0);

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
		cout << v2[i] << " ";
	cout << endl;

	cout << "v3 : ";
	for (vector<int>::size_type i = 0; i < v3.size(); ++i)
		cout << v3[i] << " ";
	cout << endl;
}

bool Pred(int n)
{
	return n <= 30;
}
```

1. 원소를 수정하는 알고리즘

swap_ranges() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(50);

	vector<int> v2;
	v2.push_back(11);
	v2.push_back(22);
	v2.push_back(33);
	v2.push_back(44);
	v2.push_back(55);

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
	cout << v1[i] << " ";
	cout << endl;
	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
	cout << v2[i] << " ";
	cout << endl;

	// 반복자 구간내의 원소들을 교환한다.
	// 교환할 벡터의 반복자 구간보다 대상의 전체 원소의 수가
	// 적으면 안된다.
	swap_ranges(v1.begin(), v1.end(), v2.begin());

	cout << endl << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
	cout << v1[i] << " ";
	cout << endl;
	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
	cout << v2[i] << " ";
	cout << endl;
}
```

1. 원소를 수정하는 알고리즘

transform() 알고리즘 사용 예제..

```
int Func(int n)
{
	return n + 5;
}

void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	transform(v.begin(), v.end(), v.begin(), Func);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

1. 원소를 수정하는 알고리즘

transform() 알고리즘의 반환 반복자 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(50);

	vector<int> v2(10); //size: 10 vector 생성

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;
	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	vector<int>::iterator iter_end;
	iter_end = transform(v1.begin(), v1.end(), v2.begin(), Func);

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;
	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
		cout << v2[i] << " ";
	cout << endl;
	cout << "[v2.begin(), iter_end) 순차열: " << *v2.begin()
	<< " ... " << *(iter_end - 1) << endl;
}

int Func(int n)
{
	return n + 5;
}
```

v.begin() ~ v.end() 구간의 원소들을 조건자의 조건을 통해서 변경하고 **v.begin()**이 **가리키는 지점부터 교환**하는 즉, 스스로 값을 변경하는 상황이 된다.

1. 원소를 수정하는 알고리즘

두 순차열의 연산이 가능한 transform() 알고리즘 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(50);

	vector<int> v2;
	v2.push_back(1);
	v2.push_back(2);
	v2.push_back(3);
	v2.push_back(4);
	v2.push_back(5);

	vector<int> v3(5); //size: 5인 vector 생성

	vector<int>::iterator iter_end;
	//iter_end = transform(v1.begin(), v1.end(), v2.begin(), v3.begin(), plus<int>());
	iter_end = transform(v1.begin(), v1.end(), v2.begin(), v3.begin(), Plus);
	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
		cout << v2[i] << " ";
	cout << endl;

	cout << "v3 : ";
	for (vector<int>::size_type i = 0; i < v3.size(); ++i)
		cout << v3[i] << " ";
	cout << endl;

	cout << "[v3.begin(), iter_end) 순차열: ";
	for (vector<int>::iterator iter = v3.begin(); iter != iter_end; ++iter)
		cout << *iter << " ";
	cout << endl;
}

int Plus(int left, int right)
{
	return left + right;
}
```

제거 알고리즘

2. 제거 알고리즘

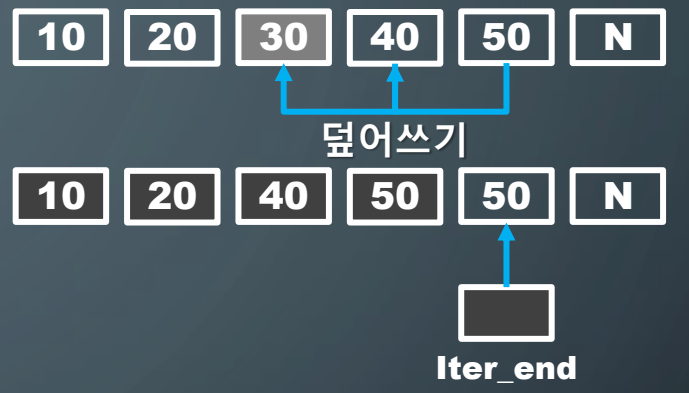
- 원소를 수정하는 알고리즘의 특수한 형태
- 원소를 실제로 제거하지 않고 논리적으로 제거(다음 원소로 덮어쓰기)하기 때문에 순차열의 **size**를 실제로는 변경하지 않는다.
- **iter_begin**과 **iter_end**간의 반복과 실제 **size**만큼 반복이 다르다.

알고리즘	설명(설명에 사용되는 p는 구간[b,e)의 반복자)
p=remove(b,e,x)	구간**[b,e)**의 순차열을 **x** 원소가 남지 않도록 덮어쓰기로 이동한다. 알고리즘 수행후 순차열은**[b,e)**가 된다.
p=remove(b,e,f)	구간**[b,e)**의 순차열을 **f(*p)**가 참인 원소가 남지 않도록 덮어쓰기로 이동한다. 알고리즘 수행후 순차열은**[b,p)**가 된다.
p=remove_copy(b,e,t,x)	구간**[b,e)**의 순차열에서 ***p==x**가 아닌 원소만 순차열**[t,p)**에 복사한다.
p=remove_copy_if(b,e,t,f)	구간**[b,e)**의 순차열에서 **f(*p)**가 참이 아닌 원소만 순차열**[t,p]**에 복사
p=unique(b,e)	구간**[b,e)**의 순차열을 인접한 중복 원소(값이 같은 원소**:==)**가 남지 않게 덮어 쓰기로 이동한다. 알고리즘 수행후 순차열은 **[b,p)**가 된다.
p=unique(b,e,f)	구간**[b,e)**의 순차열을 **f(*p)**가 참인 인접한 중복 원소가 남지 않게 덮어쓰기로 이동한다. 알고리즘 수행후 순차열은**[b,p)**가 된다.
p=unique_copy(b,e,f)	구간**[b,e)**의 순차열에서 인접한 중복 원소(값이 같은 원소**:==)**가 아닌 원소를 순차열**[t,p)**에 복사한다.
p=unique_copy(b,e,t,f)	구간**[b,e)**의 순차열에서 **f(*p)**가 참인 인접한 중복 원소가 아닌 원소를 순차열**[t,p)**에 복사한다.

2. 제거 알고리즘

remove() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	vector<int>::iterator iter_end;
	iter_end = remove(v.begin(), v.end(), 30);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	cout << "remove 후 [v.begin(), iter_end) 순차열: ";
	for (vector<int>::iterator iter = v.begin(); iter != iter_end; ++iter)
		cout << *iter << " ";
	cout << endl;
}
```

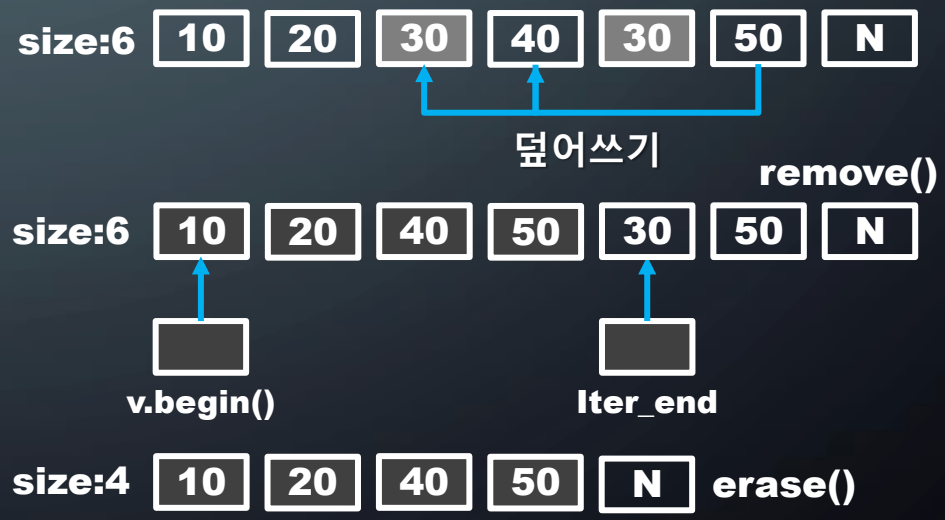


덮어쓰기를 한 후 마지막 반복자를 반환한다.

2. 제거 알고리즘

remove()후의 erase()멤버 함수 사용

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(30);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	vector<int>::iterator iter_end;
	iter_end = remove(v.begin(), v.end(), 30);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	// 구간 [iter_end, v.end())의 원소를 실제 제거(삭제)합니다.
	v.erase(iter_end, v.end());
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

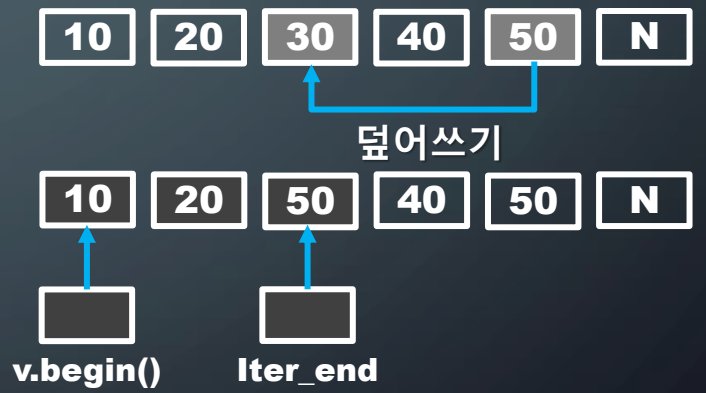


2. 제거 알고리즘

remove_if() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";

	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	vector<int>::iterator iter_end;
	iter_end = remove_if(v.begin(), v.end(), Pred);

	cout << "[v.begin(), iter_end) : ";

	for (vector<int>::iterator iter = v.begin(); iter != iter_end; ++iter)
		cout << *iter << " ";
	cout << endl;
}
```

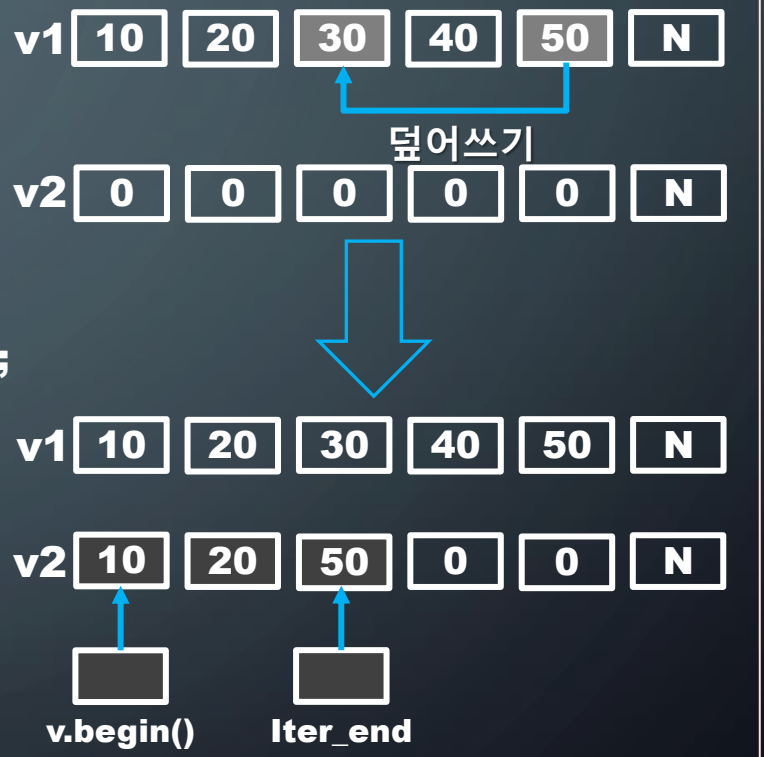
```
bool Pred(int n)
{
	return 30 <= n && n <= 40;
}
```



2. 제거 알고리즘

remove_copy() 알고리즘 사용 예제..

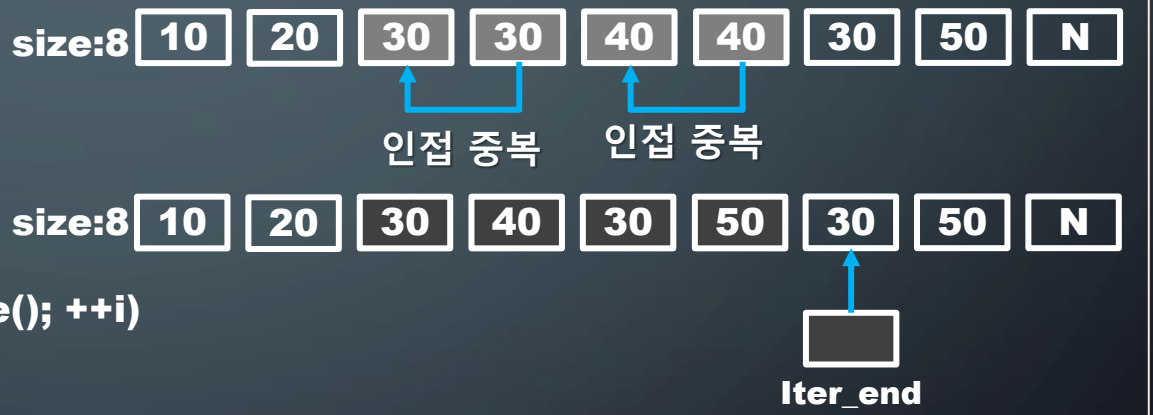
```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(50);

	vector<int> v2(5);
	vector<int>::iterator iter_end;
	iter_end = remove_copy(v1.begin(), v1.end(), v2.begin(), 30);

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;
	cout << "v2 : ";
	for (vector<int>::size_type i = 0; i < v2.size(); ++i)
		cout << v2[i] << " ";
	cout << endl;

	cout << "[v2.begin(), iter_end) : ";
	for (vector<int>::iterator iter = v2.begin(); iter != iter_end; ++iter)
		cout << *iter << " ";
	cout << endl;
}
```



2. 제거 알고리즘

unique() 알고리즘 사용 예제..

```
void main()
{
        vector<int> v;
        v.push_back(10);
        v.push_back(20);
        v.push_back(30);
        v.push_back(30);
        v.push_back(40);
        v.push_back(40);
        v.push_back(30);
        v.push_back(50);

        cout << "v : ";
        for (vector<int>::size_type i = 0; i < v.size(); ++i)
                cout << v[i] << " ";
        cout << endl;

        vector<int>::iterator iter_end;
        iter_end = unique(v.begin(), v.end());

        cout << "v : ";
        for (vector<int>::size_type i = 0; i < v.size(); ++i)
                cout << v[i] << " ";
        cout << endl;

        cout << "[v.begin(), iter_end) : ";
        for (vector<int>::iterator iter = v.begin(); iter != iter_end; ++iter)
                cout << *iter << " ";
        cout << endl;
}
```



인접하지 않은 중복 원소를 제거하지 않는다.

2. 제거 알고리즘

조건자 버전 **unique()** 알고리즘 예제..

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(11);
	v.push_back(30);
	v.push_back(32);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
	cout << v[i] << " ";
	cout << endl;

	vector<int>::iterator iter_end;
	iter_end = unique(v.begin(), v.end(), Pred);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
	cout << "[v.begin(), iter_end) : ";
	for (vector<int>::iterator iter = v.begin(); iter != iter_end; ++iter)
		cout << *iter << " ";
	cout << endl;
}
```

```
bool Pred(int left, int right)
{
	return abs(right - left) < 10;
}
```

변경 알고리즘

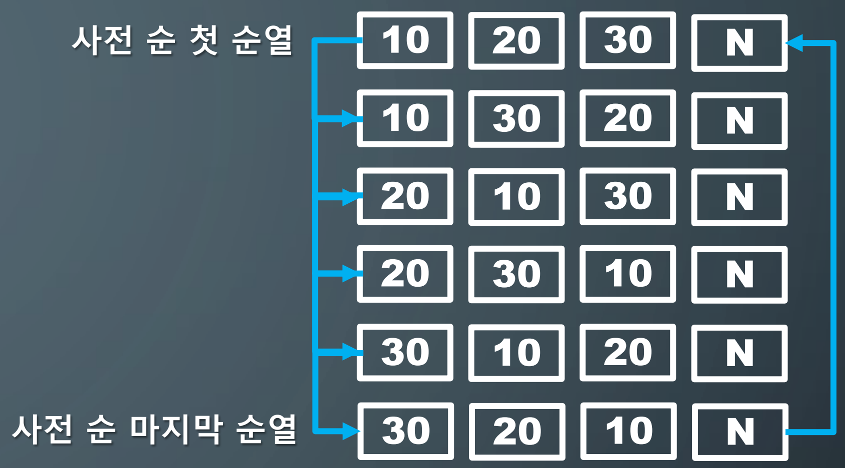
3. 변경 알고리즘

- 순차열의 원소가 서로 교환하거나 이동하여 순차열 원소의 **순서**를 변경하는 알고리즘

알고리즘	설명(설명에 사용되는 p는 구간[b,e)의 반복자)
bool=next_permutation (b,e)	구간**[b,e)**의 순차열을 사전순 다음 순열이 되게 한다**.** 더 이상 다음 순열이 없는 마지막 순열이라면 **bool**은 **false**이다**.**
bool=next_permutation (b,e,f)	구간**[b,e)**의 순차열을 사전순 다음 순열이 되게 한다**.** 비교에 **f**를 사용한다**.** 더 이상 다음 순열이 없는 마지막 순열이라면 **bool**은 **false**이다**.**
bool=prev_permutation (b,e)	구간**[b,e)**의 순차열을 사전순 이전 순열이 되게 한다**.** 더 이상 이전 순열이 없는 첫 순열이라면 **bool**은 **false**이다**.**
bool=prev_permutation (b,e,f)	구간**[b,e)**의 순차열을 사전순 이전 순열이 되게 한다**.** 비교에 **f**를 사용한다**.** 더 이상 이전 순열이 없는 첫 순열이라면 **bool**은 **false**이다**.**
p=partition(b,ef)	구간**[b,e)**의 순차열 중 **if(*p)**가 참인 원소는**[b,p)**의 순차열에 거짓인 원소는 **[p,e)**의 순차열로 분류한다**.**
random_shuffle(b,e)	구간**[b,e)**의 순차열을 랜덤**(**기본 랜덤기**)**으로 뒤섞는다**.**
random_shuffle(b,e,f)	구간**[b,e)**의 순차열을 **f**를 랜덤기로 사용하여 뒤섞는다**.**
reverse(b,e)	구간**[b,e)**의 순차열을 뒤집는다**.**
p=reverse_copy(b,e,t)	구간**[b,e)**의 순차열을 뒤집어 목적지 순차열**[t,p)**에 복사한다**.**
rotate(b,m,e)	구간**[b,e)**의 순차열을 왼쪽으로 회전시킨다**.** 첫 원소와 마지막 원소가 연결된 것처럼 모든 원소가 왼쪽으로 **(m-b)**만큼씩 이동한다**.**
p=rotate_copy(b,m,e,t)	구간**[b,e)**의 순차열을 회전하여 목적지 순차열**[t,p)**에 복사한다**.**
stable_partition(b,e,f)	**Patrition()**알고리즘과 같고 원소의 상대적인 순서를 유지한다**.**

3. 변경 알고리즘

next_permutation() 알고리즘 사용 예제..




```
void main()
{
        vector<int> v;
        v.push_back(10);
        v.push_back(20);
        v.push_back(30);

        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end()) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end()) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end()) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end()) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end()) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        // 마지막 순열이므로 next_permutation()은 false 반환
        cout << next_permutation(v.begin(), v.end()) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
}
```

3. 변경 알고리즘

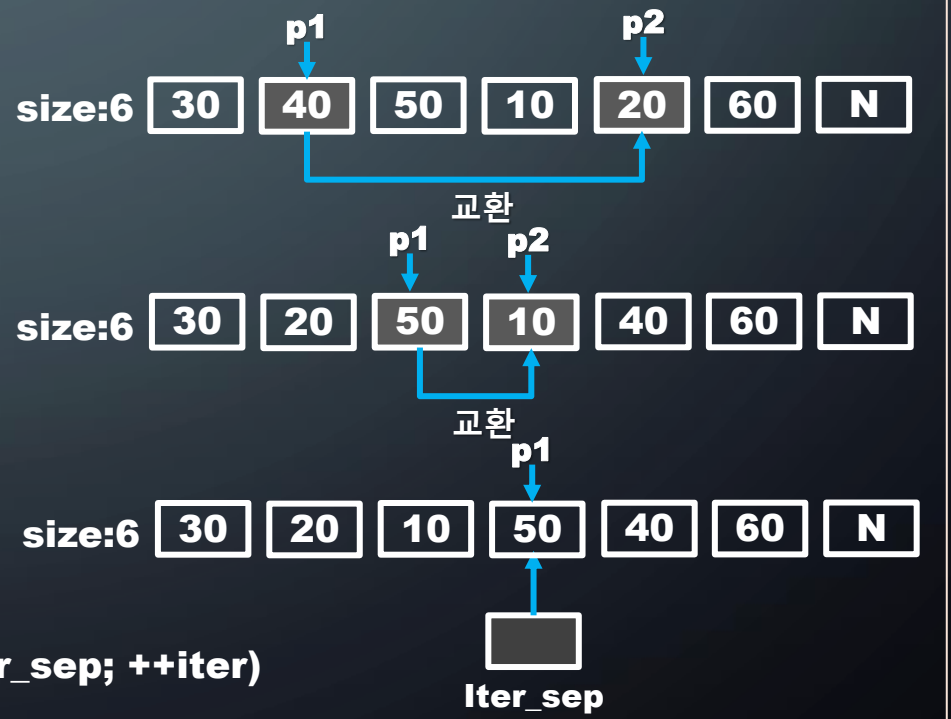
조건자 버전 next_permutation() 알고리즘 예제..

```
class Point
{
        int x, y;
public:
        explicit Point(int _x = 0, int _y = 0) :x(_x), y(_y) { }
        int GetX() const { return x; }
        int GetY() const { return y; }
};
ostream& operator<<(ostream& out, const Point& arg)
{   //Point 객체 cout 출력을 위한 연산자 오버로딩
        out << "(" << arg.GetX() << "," << arg.GetY() << ")";
        return out;
}
bool Pred(const Point& left, const Point& right)
{   //순열의 비교 기준으로 사용될 이항 조건자
        return left.GetY() < right.GetY();
}

void main()
{
        vector<Point> v;
        v.push_back(Point(5, 1));
        v.push_back(Point(7, 2));
        v.push_back(Point(5, 3));

        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end(), Pred) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end(), Pred) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end(), Pred) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end(), Pred) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        cout << next_permutation(v.begin(), v.end(), Pred) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
        // Point의 y를 기준으로 마지막 순열이므로 next_permutation()은 false 반환
        cout << next_permutation(v.begin(), v.end(), Pred) << endl;;
        cout << "v : " << v[0] << " " << v[1] << " " << v[2] << endl;
}
```

3. 변경 알고리즘

partition() 알고리즘 사용 예제..

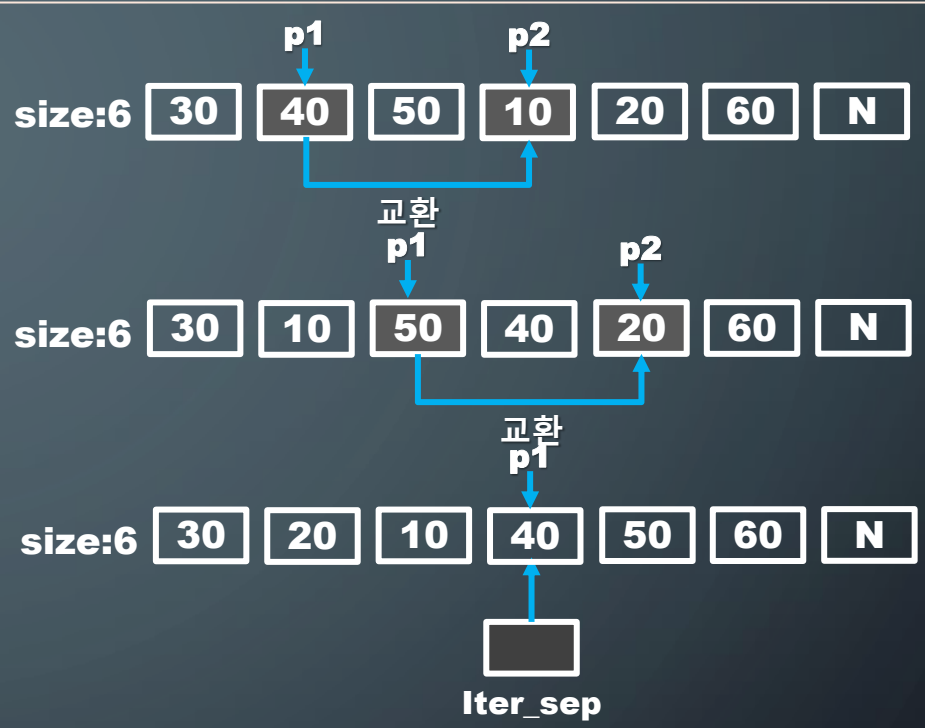
```
void main()
{
	vector<int> v;
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);
	v.push_back(10);
	v.push_back(20);
	v.push_back(60);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	vector<int>::iterator iter_sep;
	// 40 원소를 기준으로 미만과 이상을 분류한다.
	iter_sep = partition(v.begin(), v.end(), Pred);

	cout << "40미만의 순차열: ";
	for (vector<int>::iterator iter = v.begin(); iter != iter_sep; ++iter)
		cout << *iter << " ";
	cout << endl;

	cout << "40이상의 순차열: ";
	for (vector<int>::iterator iter = iter_sep; iter != v.end(); ++iter)
		cout << *iter << " ";
	cout << endl;
}

bool Pred(int n)
{
	return n < 40;
}
```



3. 변경 알고리즘

stable_partition() 알고리즘 사용 예제.

```
void main()
{
        vector<int> v;
        v.push_back(30);
        v.push_back(40);
        v.push_back(50);
        v.push_back(10);
        v.push_back(20);
        v.push_back(60);

        cout << "v : ";
        for (vector<int>::size_type i = 0; i < v.size(); ++i)
                cout << v[i] << " ";
        cout << endl;

        vector<int>::iterator iter_sep;
        // 원소들의 상대적인 순서를 유지하며 40 원소를 기준으로 미만과 이상을 분류한다.
        iter_sep = stable_partition(v.begin(), v.end(), Pred);

        cout << "40미만의 순차열: ";
        for (vector<int>::iterator iter = v.begin(); iter != iter_sep; ++iter)
                cout << *iter << " ";
        cout << endl;
        cout << "40이상의 순차열: ";
        for (vector<int>::iterator iter = iter_sep; iter != v.end(); ++iter)
                cout << *iter << " ";
        cout << endl;
}

bool Pred(int n)
{
        return n < 40;
}
```



40이 기준이므로 기분으로 왼쪽은 40미만이 오른쪽은 40이상이 배치되는 상대적인 순서를 지키게 된다.

3. 변경 알고리즘

random_shuffle() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	//#include <ctime>, #include <cstdlib> 추가 랜덤 초기값
	//srand( (unsigned)time(NULL) );

	// 원소를 뒤섞는다.
	random_shuffle(v.begin(), v.end());
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	// 원소를 뒤섞는다.
	random_shuffle(v.begin(), v.end());
	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

3. 변경 알고리즘

reverse() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	// [v.begin(), v.end()) 순차열 뒤집기.
	reverse(v.begin(), v.end());

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```

3. 변경 알고리즘

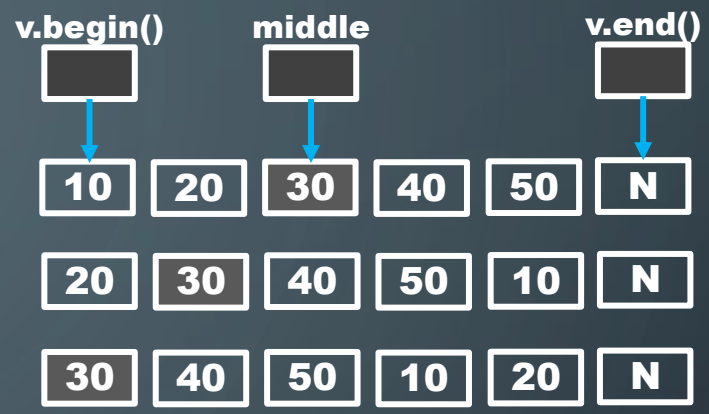
reverse_copy() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(50);

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	vector<int> v2(5); //size : 5 인 vector 생성 (덮어쓰기이므로)

	vector<int>::iterator iter_end;
	iter_end = reverse_copy(v1.begin(), v1.end(), v2.begin());

	cout << "v2 : ";
	for (vector<int>::iterator iter = v2.begin(); iter != iter_end; ++iter)
		cout << *iter << " ";
	cout << endl;
}
```

3. 변경 알고리즘

rotate() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v;
	v.push_back(10);
	v.push_back(20);
	v.push_back(30);
	v.push_back(40);
	v.push_back(50);

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;

	vector<int>::iterator middle = v.begin() + 2;
	// 모든 원소를 왼쪽으로 2만큼씩 회전한다.
	rotate(v.begin(), middle, v.end());

	cout << "v : ";
	for (vector<int>::size_type i = 0; i < v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}
```



v.begin()과 **middl**간의 차이만큼 회전을 실행한다. 즉, **middle begin()**이 될 때까지..

3. 변경 알고리즘

rotate_copy() 알고리즘 사용 예제..

```
void main()
{
	vector<int> v1;
	v1.push_back(10);
	v1.push_back(20);
	v1.push_back(30);
	v1.push_back(40);
	v1.push_back(50);

	cout << "v1 : ";
	for (vector<int>::size_type i = 0; i < v1.size(); ++i)
		cout << v1[i] << " ";
	cout << endl;

	vector<int> v2(5);
	vector<int>::iterator middle = v1.begin() + 2;
	vector<int>::iterator iter_end;
	// 모든 원소를 왼쪽으로 2만큼씩 회전하여 [v2.begin(), iter_end) 순차열에 복사한다.
	iter_end = rotate_copy(v1.begin(), middle, v1.end(), v2.begin());

	cout << "v2 : ";
	for (vector<int>::iterator iter = v2.begin(); iter != iter_end; ++iter)
		cout << *iter << " ";
	cout << endl;
}
```