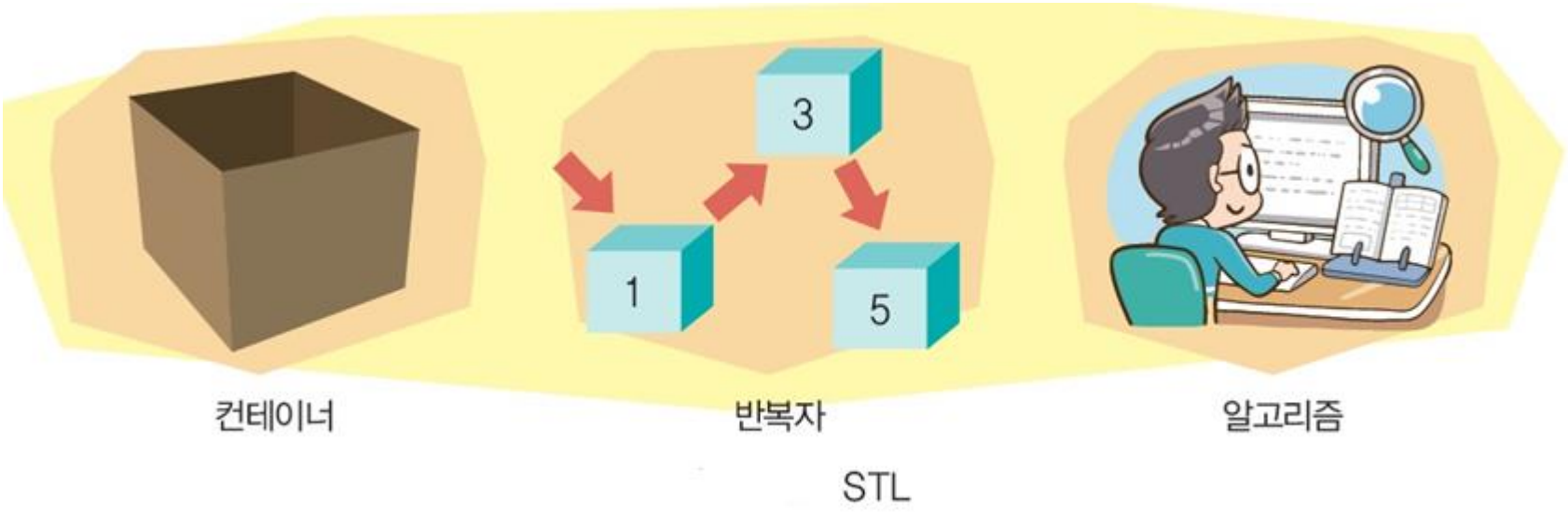
SEOIL UNIVERSITY

STL

Standard Template Library

표준 템플릿 라이브러리(STL)



STL의 3가지 컴포넌트

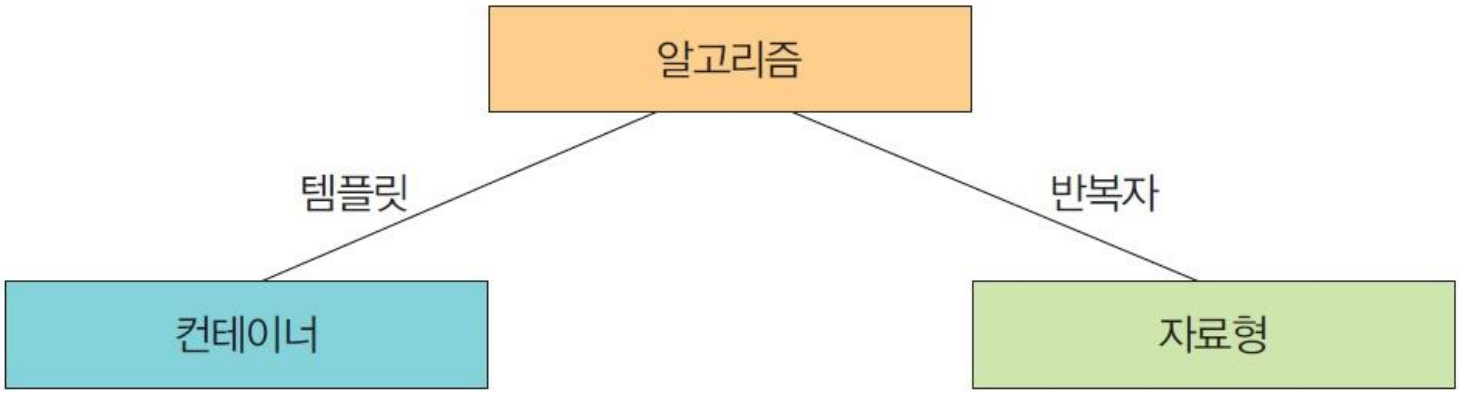
- **컨테이너(container)**
  - 자료를 저장하는 구조이다.
  - 벡터, 리스트, 맵, 집합, 큐, 스택과 같은 다양한 자료 구조들이 제공된다.
- **반복자(iterator)**
  - 컨테이너 안에 저장된 요소들을 순차적으로 처리하기 위한 컴포넌트
- **알고리즘(algorithm)**
  - 정렬이나 탐색과 같은 다양한 알고리즘을 구현

컨테이너

- 컨테이너는 자료를 저장하는 창고와 같은 역할을 하는 구조이다. 즉 배열이나 연결 리스트, 벡터, 집합, 사전, 트리 등이 여기에 해당

반복자

- 반복자(iterator)는 컨테이너의 요소를 가리키는 데 사용
- 반복자는 실제로 컨테이너와 알고리즘 사이의 다리 역할



알고리즘

- 탐색(find): 컨테이너 안에서 특정한 자료를 찾는다.
- 정렬(sort): 자료들을 크기순으로 정렬한다.
- 반전(reverse): 자료들의 순서를 역순으로 한다.
- 삭제(remove): 조건이 만족되는 자료를 삭제한다.
- 변환(transform): 컨테이너의 요소들을 사용자가 제공하는 변환 함수에 따라서 변환한다.

알고리즘의 예

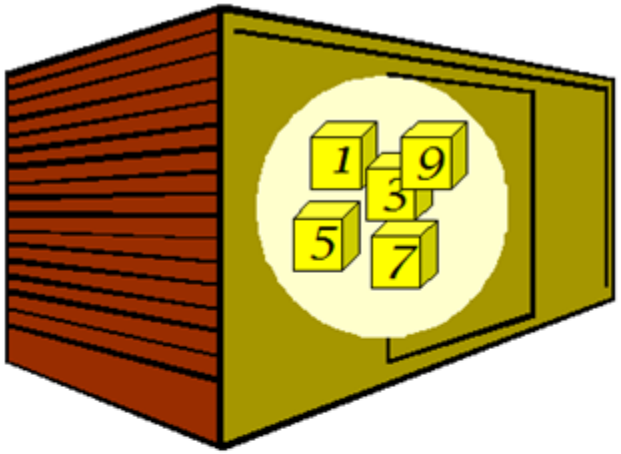
STL의 장점

- STL은 프로그래밍에 매우 유용한 수많은 컨테이너와 알고리즘을 제공
- STL은 객체 지향 기법과 일반화 프로그래밍 기법을 적용하여서 만들어졌으므로 어떤 자료형에 대해서도 사용
- STL은 전문가가 만들어서 테스트를 거친 검증된 라이브러리

컨테이너

- 벡터(vector): 동적 배열처럼 동작한다. 뒤에서 자료들이 추가된다.
- 큐(queue): 데이터가 입력된 순서대로 출력되는 자료 구조
- 스택(stack): 먼저 입력된 데이터가 나중에 출력되는 자료 구조
- 우선 순위큐(priority queue): 큐의 일종으로 큐의 요소들이 우선 순위를 가지고 있고 우선 순위가 높은 요소가 먼저 출력되는 자료 구조
- 리스트(list): 벡터와 유사하지만 중간에서 자료를 추가하는 연산이 효율적이다.
- 집합(set): 중복이 없는 자료들이 정렬되어서 저장된다.
- 맵(map): 키-값(key-value)의 형식으로 저장된다. 키가 제시되면 해당되는 값을 찾을 수 있다.

컨테이너

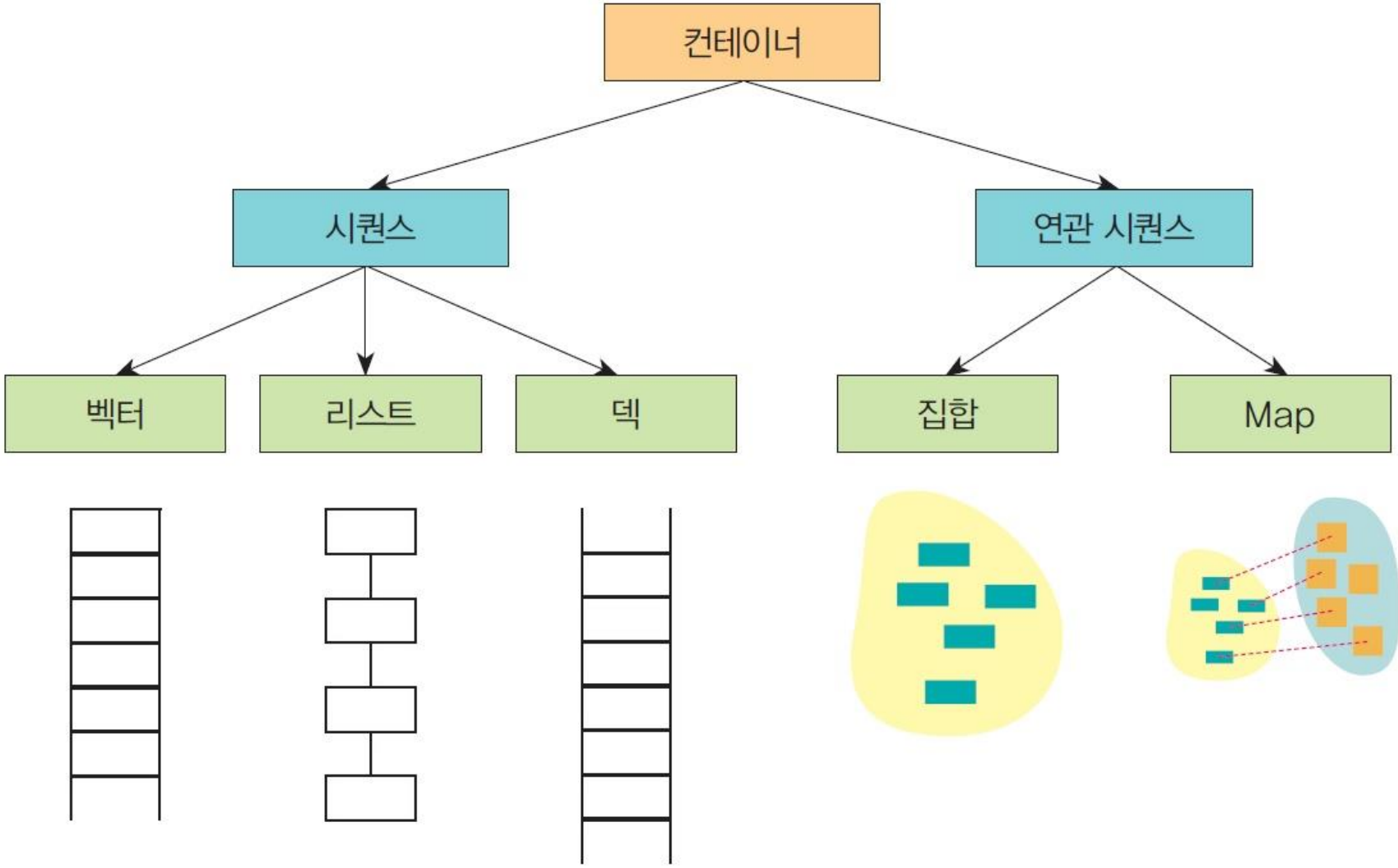


컨테이너

컨테이너의 개념

분류	컨테이너 클래스	설명	헤더 파일
순차 컨테이너	vector	벡터처럼 입력된 순서대로 저장	<vector>
	list	순서가 있는 리스트	<list>
	deque	양끝에서 입력과 출력이 가능	<deque>
연관 컨테이너	set	수학에서의 집합 구현	<set>
	multiset	다중 집합(중복을 허용)	<set>
	map	사전과 같은 구조	<map>
	multimap	다중 맵(중복을 허용)	<map>
컨테이너 어댑터	stack	스택(후입선출)	<stack>
	queue	큐(선입선출)	<queue>
	priority_queue	우선순위큐(우선순위가 높은 원소가 먼저 출력)	<queue>

컨테이너의 분류



순차 컨테이너

순차 컨테이너:

- 자료를 순차적으로 저장

- 벡터(vector): 동적 배열처럼 동작한다. 뒤에서 자료들이 추가된다.
- 데크(deque): 벡터와 유사하지만 앞에서도 자료들이 추가될 수 있다.
- 리스트(list): 벡터와 유사하지만 중간에서 자료를 추가하는 연산이 효율적이다.

연관 컨테이너

연관 컨테이너

- 사전과 같은 구조를 사용하여서 자료를 저장
- 원소들을 검색하기 위한 키(key)
- 자료들은 정렬

- 집합(set): 중복이 없는 자료들이 정렬되어서 저장된다.
- 맵(map): 키-값(key-value)의 형식으로 저장된다. 키가 제시되면 해당되는 값을 찾을 수 있다.
- 다중-집합(multiset): 집합과 유사하지만 자료의 중복이 허용된다.
- 다중-맵(multimap): 맵과 유사하지만 키가 중복될 수 있다.

컨테이너 어댑터

컨테이너 어댑터

- 순차 컨테이너에 제약을 가해서 데이터들이 정해진 방식으로만 입출력

- 스택(stack): 먼저 입력된 데이터가 나중에 출력되는 자료 구조
- 큐(queue): 데이터가 입력된 순서대로 출력되는 자료 구조
- 우선 순위큐(priority queue): 큐의 일종으로 큐의 요소들이 우선 순위를 가지고 있고 우선 순위가 높은 요소가 먼저 출력되는 자료 구조

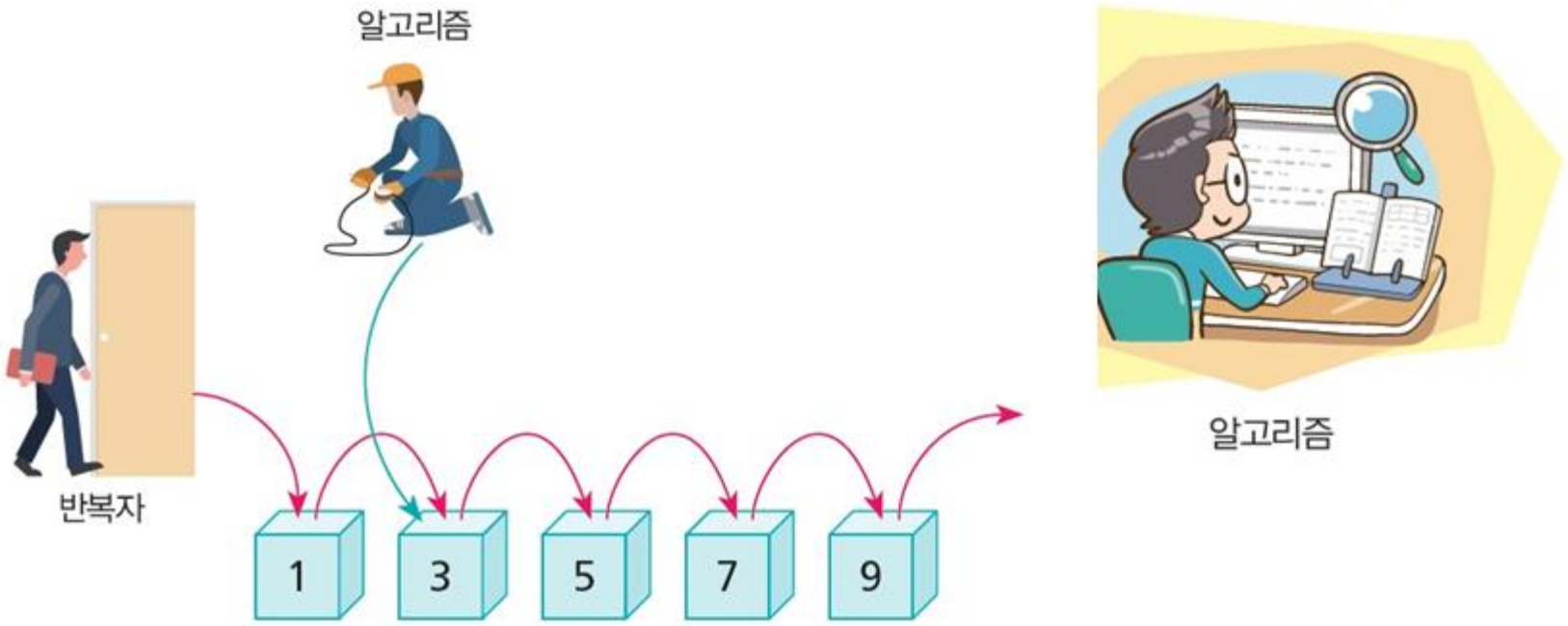
```
#include <iostream>
#include <time.h>
#include <list>
using namespace std;

int main()
{
	list<int> values;

	srand(time(NULL));
	for (int i = 0; i < 10; i++) {
		values.push_back(rand()%100);
	}
	values.sort();

	for (auto& e: values){
		std::cout << e << " ";
	}
	std::cout << endl;
	return 0;
}
```


C:₩Windows₩system32₩cmd.exe
17 23 27 34 39 41 58 74 97 97
계속하려면 아무 키나 누르십시오 . . .

반복자

- 반복자: 일반화된 포인터(generalized pointer)

반복자

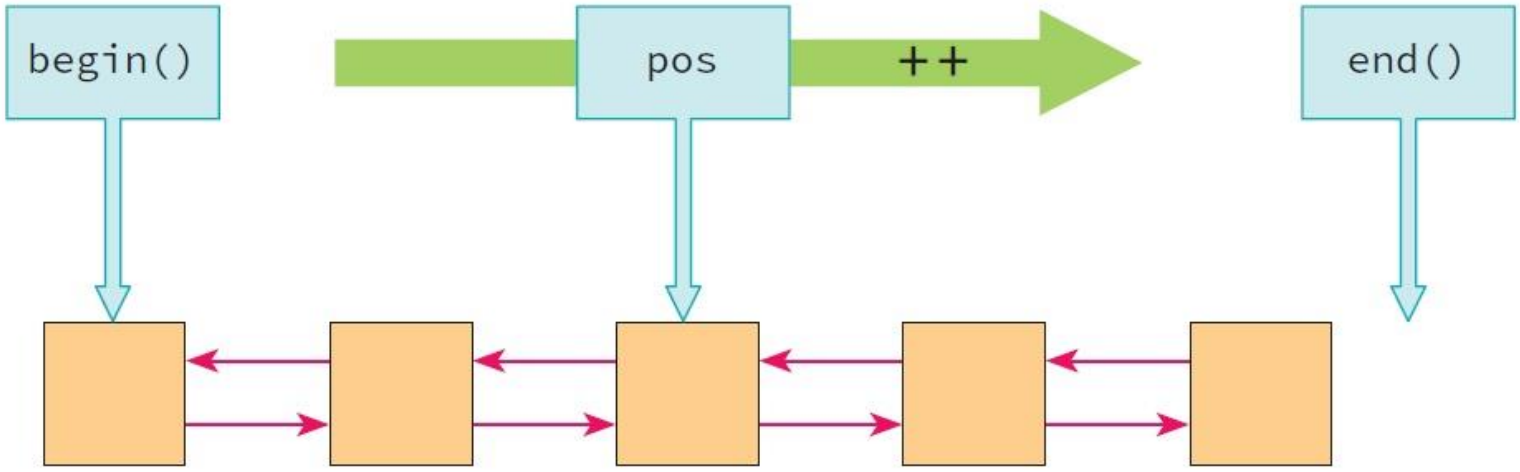


STL 알고리즘은 반복자를 통하여 컨테이너에 접근하여 작업을 한다.

시퀀스

반복자의 연산자

- 컨테이너에서 다음 요소를 가리키기 위한 ++ 연산자
- 컨테이너에서 이전 요소를 가리키기 위한 -- 연산자
- 두개의 반복자가 같은 요소를 가리키고 있는 지를 확인하기 위한 ==와 != 연산자
- 반복자가 가리키는 요소의 값을 추출하기 위한 역참조 연산자 *



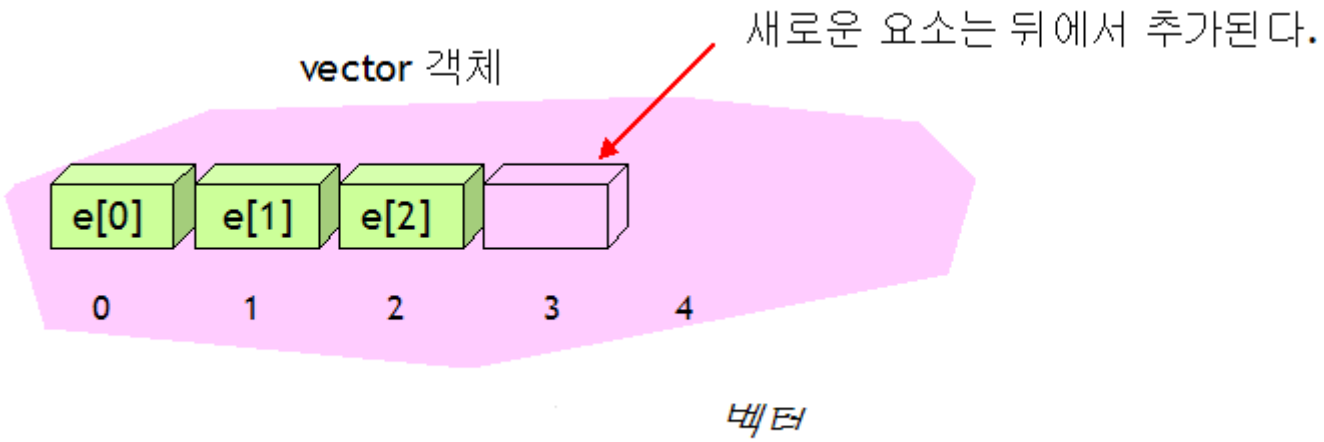
알고리즘

- 탐색(find): 컨테이너 안에서 특정한 자료를 찾는다.
- 정렬(sort): 자료들을 크기순으로 정렬한다.
- 반전(reverse): 자료들의 순서를 역순으로 한다.
- 삭제(remove): 조건이 만족되는 자료를 삭제한다.
- 변환(transform): 컨테이너의 요소들을 사용자가 제공하는 변환 함수에 따라서 변환한다.

벡터

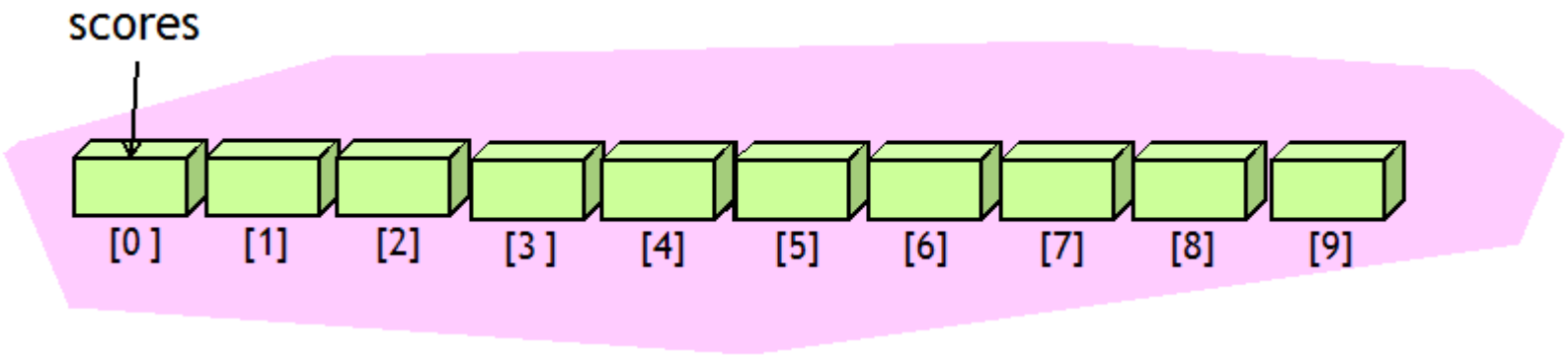
벡터 == 동적 배열 == 스마트 배열
템플릿으로 설계

```
① vector<int> vec;                          // 벡터 선언
② vector<int>::iterator it;// 반복자 선언
③ for(it = vec.begin(); it != vec.end(); it++)
④         cout << *it << " ";
```



벡터

학생들의 성적을 저장하는 벡터를 생성



```
#include <iostream>
#include <vector>                    // 벡터를 사용하려면 이 헤더 파일을 포함하여야 한다
using namespace std;

int main()
{
   vector<double> scores(10);        // 벡터를 생성한다.

   for(int i = 0; i < scores.size() ; i++)
   {
         cout << "성적을 입력하시오: ";
         cin >> scores[i];
   }

   double highest = scores[0];
   for(int i = 1; i < scores.size() ; i++)
         if( scores[i] > highest )
                  highest = scores[i];
   cout << "최고 성적은 " << highest << "입니다.\n";

   return 0;
}
```

```
성적을 입력하시오: 10
성적을 입력하시오: 20
...
성적을 입력하시오: 90
성적을 입력하시오: 100
최고 성적은 100입니다.
```

push_back()과 pop_back()

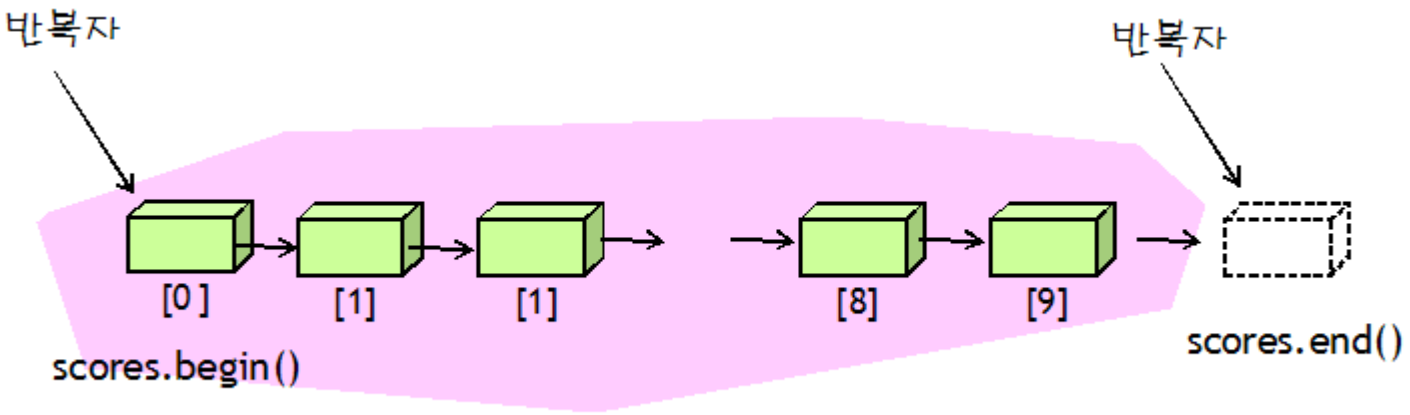
- **push_back()**
  - 새로운 데이터를 벡터의 끝에 추가하고 벡터의 크기를 1만큼 증가

- **pop_back()**
  - 벡터의 끝에서 요소를 제거하고 벡터의 크기를

```
#include <iostream>
#include <vector>                    // 벡터를 사용하려면 이 헤더 파일을 포함하여야 한다
using namespace std;

int main()
{
   vector<double> scores;            // 벡터를 생성한다.

   while(true)
   {
         double value = 0.0;
         cout << "성적을 입력하시오(종료는 -1): ";
         cin >> value;
         if( value < 0.0 ) break;
         scores.push_back(value);
   }
   double highest = scores[0];
   for(int i = 1; i < scores.size() ; i++)
         if( scores[i] > highest )
                  highest = scores[i];
   cout << "최고 성적은 " << highest << "입니다.\n";

   return 0;
}
```

```
성적을 입력하시오(종료는 -1): 10
성적을 입력하시오(종료는 -1): 20
성적을 입력하시오(종료는 -1): 30
성적을 입력하시오(종료는 -1): -1
최고 성적은 30입니다.
```

반복자 사용



```
#include <iostream>
#include <vector>                    // 벡터를 사용하려면 이 헤더 파일을 포함하여야 한다
using namespace std;

int main()
{
   vector<double> scores;            // 벡터를 생성한다.

   while(true)
   {
         double value = 0.0;
         cout << "성적을 입력하시오(종료는 -1): ";
         cin >> value;
         if( value < 0.0 ) break;
         scores.push_back(value);
   }

   double highest = -100;
   vector<double>::iterator it;
   for(it = scores.begin(); it < scores.end() ; it++)
         if( *it > highest )
                  highest = *it;


   cout << "최고 성적은 " << highest << "입니다.\n";

   return 0;
}
```

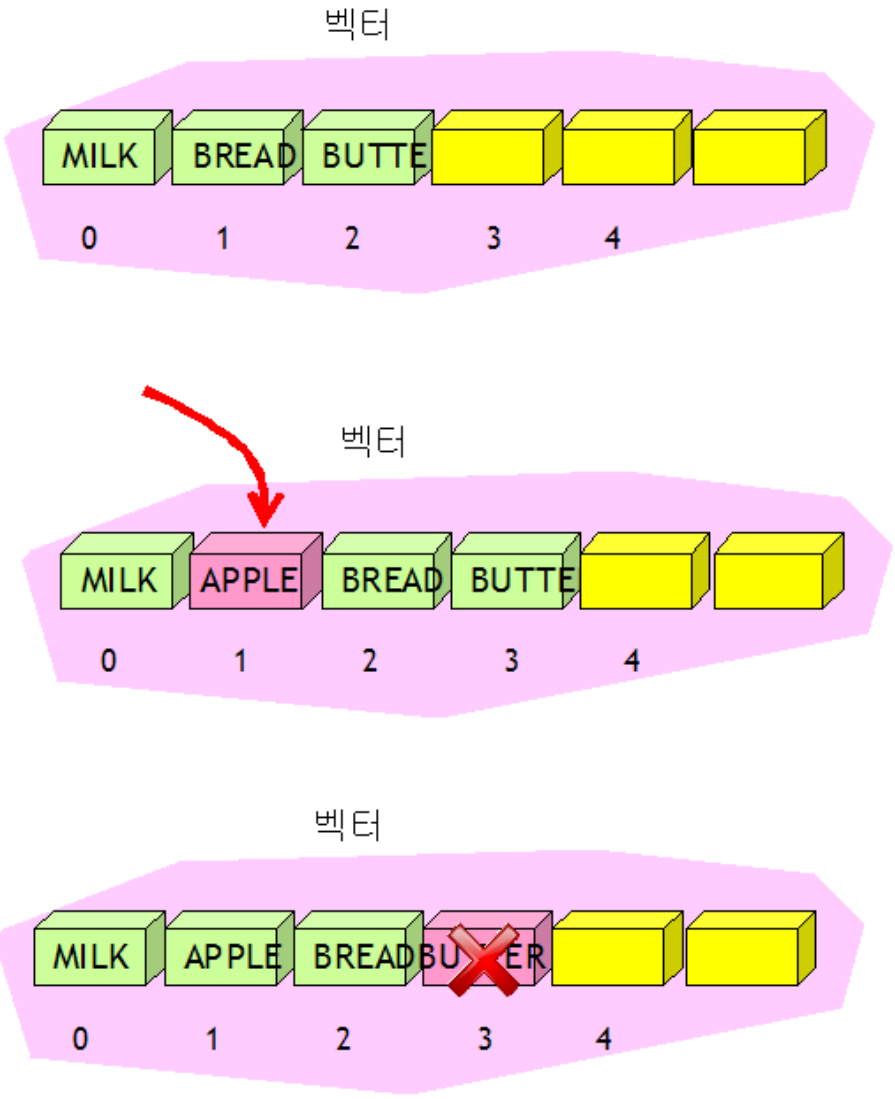
```
성적을 입력하시오(종료는 -1): 10
성적을 입력하시오(종료는 -1): 20
성적을 입력하시오(종료는 -1): 30
성적을 입력하시오(종료는 -1): -1
최고 성적은 30입니다.
```

벡터와 연산자

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

int main()
{
   vector<string> vec;              // 벡터를 생성한다.

   vec.push_back("MILK");                  // 벡터의 끝에 자료를 저장한다.
   vec.push_back("BREAD");
   vec.push_back("BUTTER");

   vector<string>::iterator it;     // 벡터를 순회하기 위해여 반복자를 선언한다.
   for(int i=0;i<vec.size();i++)
        cout << vec[i] << " ";      // [] 연산자 사용
   cout << endl;

   vec.insert(vec.begin()+1, "APPLE");     // 벡터의 첫부분에 자료를 저장한다.
   vec.pop_back();                  // 벡터의 끝에서 자료를 삭제한다.

   for (it = vec.begin(); it != vec.end(); ++it)
        cout << *it << " ";
   cout << endl;

   return 0;
}
```

```
MILK BREAD BUTTER
MILK APPLE BREAD
```

벡터와 연산자


벡터
MILK
BREAD
BUTTE
0
1
2
3
4
벡터
MILK
APPLE
BREAD
BUTTE
0
1
2
3
4
벡터
MILK
APPLE
BREADBU ER
0
1
2
3
4

컨테이너 공통 함수

함수	설명
Container()	기본 생성자
Container(size)	크기가 size인 컨테이너 생성
Container(size, value)	크기가 size이고 초기값이 value인 컨테이너 생성
Container(iterator, iterator)	다른 컨테이너로부터 초기값의 범위를 받아서 생성
begin()	첫 번째 요소의 반복자 위치
clear()	모든 요소를 삭제
empty()	비어있는지를 검사
end()	반복자가 마지막 요소를 지난 위치
erase(iterator)	컨테이너의 중간 요소를 삭제
erase(iterator, iterator)	컨테이너의 지정된 범위를 삭제
front()	컨테이너의 첫 번째 요소 반환
insert(iterator, value)	컨테이너의 중간에 value를 삽입
pop_back()	컨테이너의 마지막 요소를 삭제
push_back(value)	컨테이너의 끝에 데이터를 추가
rbegin()	끝을 나타내는 역반복자
rend()	역반복자가 처음을 지난 위치
size()	컨테이너의 크기
operator=(Container)	할당 연산자의 중복 정의

Old C++ 버전


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

int main()
{
        vector<int> v;
        v.push_back(1);
        v.push_back(2);
        v.push_back(3);

        for (vector<int>::iterator p = v.begin(); p != v.end(); ++p)
                cout << *p << endl;
        return 0;
}
```

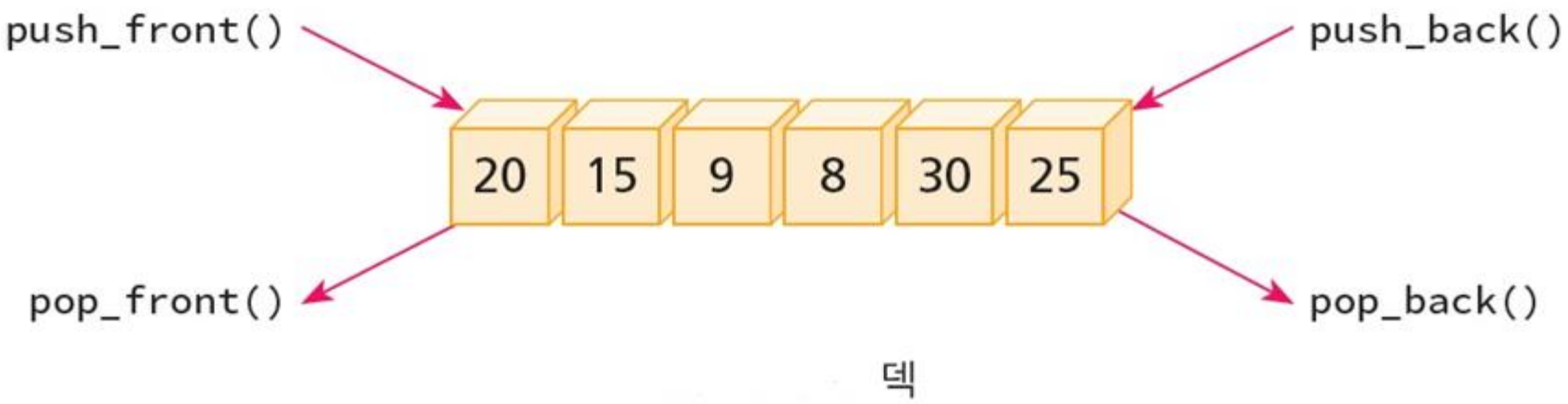
C++14 버전


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

int main()
{
        vector<int> v;
        v.push_back(1);
        v.push_back(2);
        v.push_back(3);

        for (auto p = v.begin(); p != v.end(); ++p)
                cout << *p << endl;
        return 0;
}
```

C++17 버전


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

int main()
{
        vector<int> v;
        v.push_back(1);
        v.push_back(2);
        v.push_back(3);

        for (auto& n : v )
                cout << n << endl;
        return 0;
}
```

데큐(Deque)

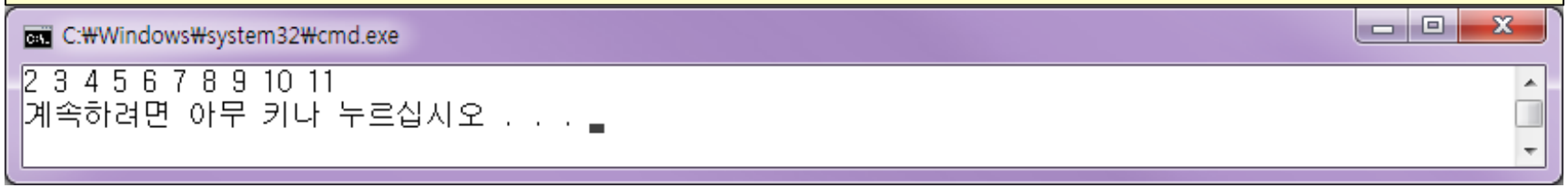


덱

데큐(Deque) : 정수

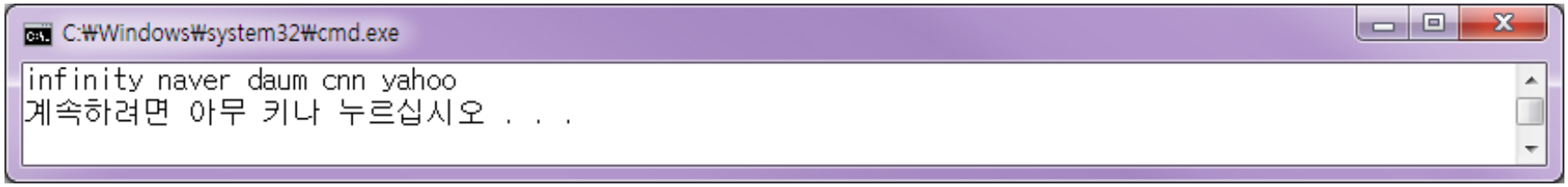
SEOIL UNIVERSITY

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

int main()
{
	deque<int> dq = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

	dq.pop_front();			// 앞에서 삭제
	dq.push_back(11);			// 끝에서 추가
	for (auto& n : dq)
		cout << n << " ";
	cout << endl;

	return 0;
}
```

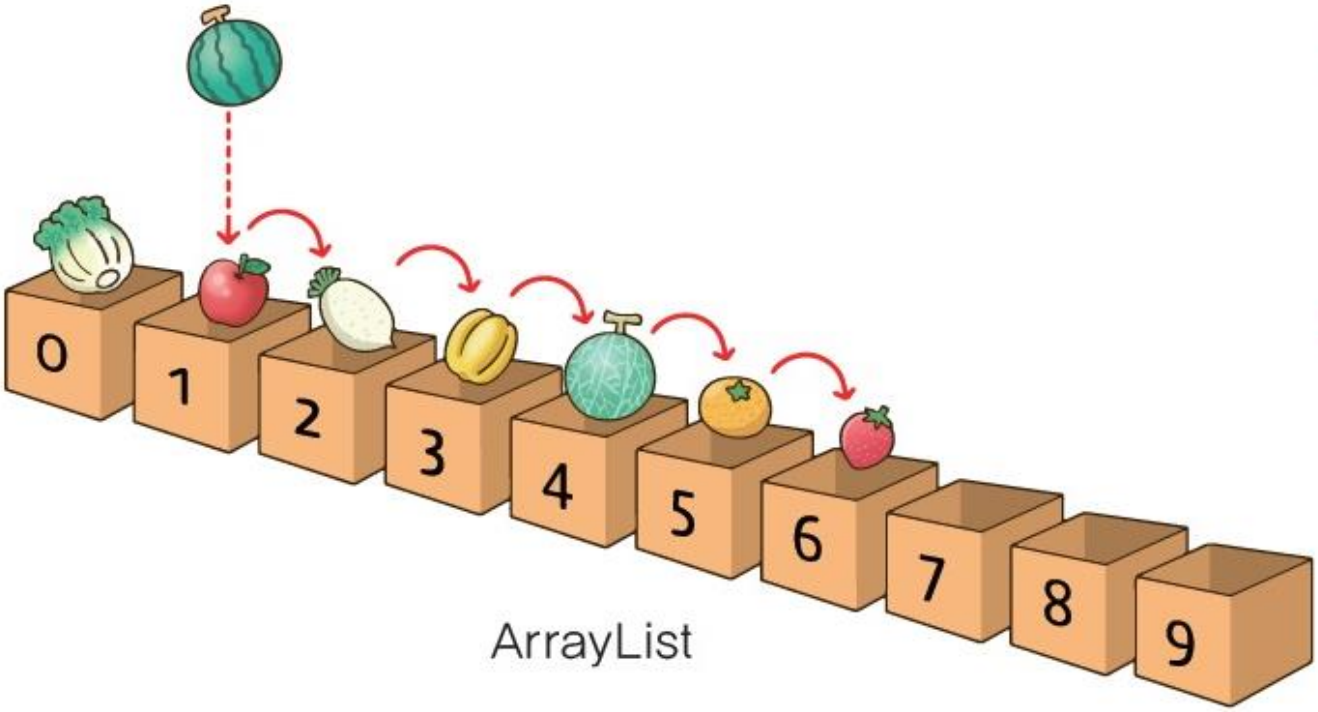


데큐(Deque) : 문자열


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

int main()
{
	deque<string> dq = { "naver", "daum", "cnn", "yahoo", "google" };

	dq.push_front("infinity");		// 앞에서 삭제
	dq.pop_back();				// 끝에서 추가
	for (auto& e : dq)
		cout << e << " ";
	cout << endl;

	return 0;
}
```

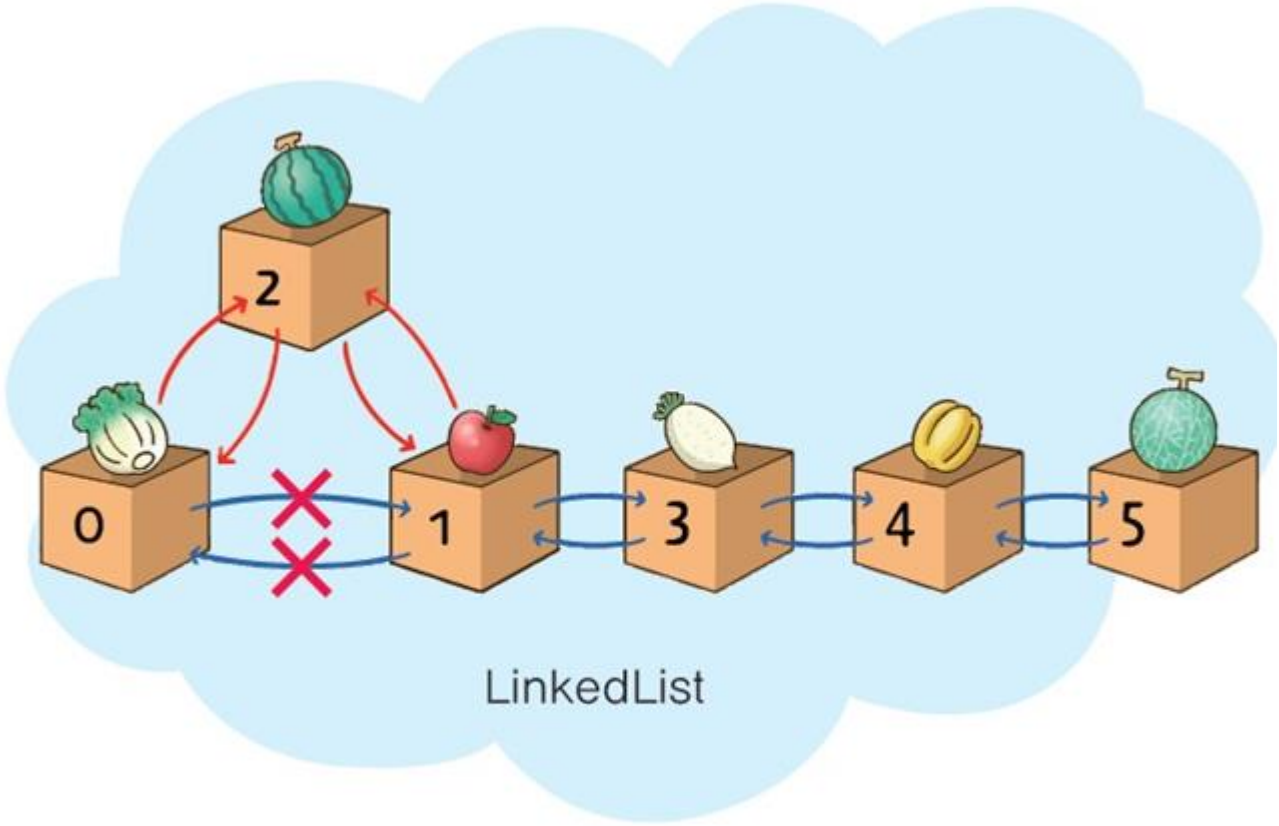


리스트



- 배열의 중간에 삽입하려면 원소들을 이동하여야 한다.

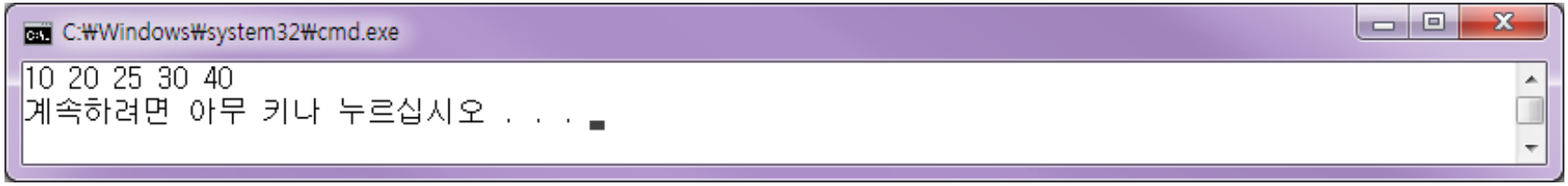
리스트



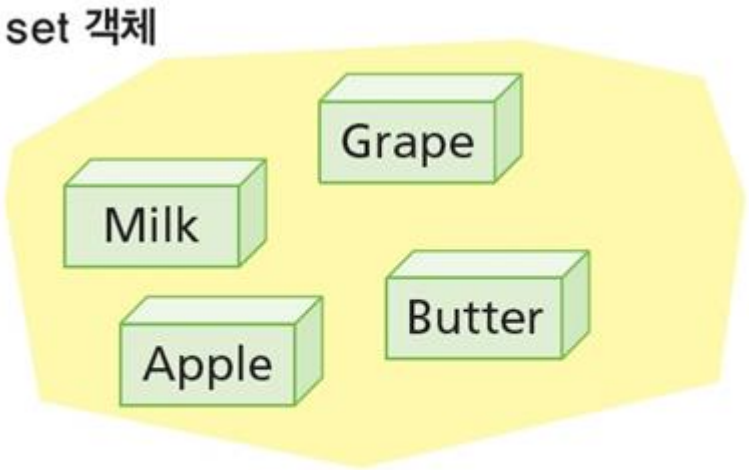
연결 리스트 중간에 삽입하려면 링크만 수정하면 된다.

리스트


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

int main()
{
        list<int> my_list={ 10, 20, 30, 40 };

        auto it = my_list.begin();
        it++;
        it++;
        my_list.insert(it, 25);
        for (auto& n : my_list)
                cout << n << " ";
        cout << endl;
        return 0;
}
```



집합

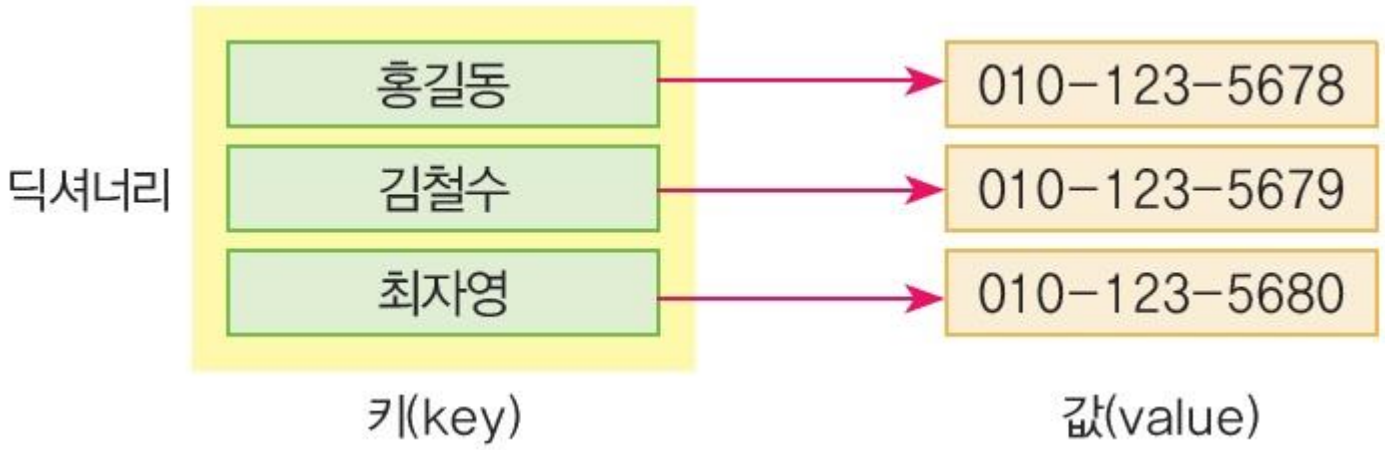


집합은 순서가 없고
중복을 허용하지 않음

집합

```
#include <set>
int main()
{
        set<int> my_set;

        my_set.insert(1);
        my_set.insert(2);
        my_set.insert(3);

        auto pos = my_set.find(2);
        if (pos != my_set.end())
                cout << "값 " << *pos << "가 발견되었음" << endl;
        else
                cout << "값이 발견되지 않았음" << endl;

        return 0;
}
```

Map


딕셔너리
홍길동
김철수
최자영
010-123-5678
010-123-5679
010-123-5680
키(key)
값(value)

Map


```
#include <iostream>
#include <map>
#include <string>
#include <iterator>
using namespace std;

int main()
{
        map<string, string> myMap;

        myMap.insert(make_pair("김철수", "010-123-5678"));
        myMap.insert(make_pair("홍길동", "010-123-5679"));

        myMap["최자영"] = "010-123-5680";
```

Map


```
        // 모든 요소 출력
        for(auto& it : myMap){
                cout << it.first << " :: " << it.second << endl;
        }
        if (myMap.find("김영희") == myMap.end())
                cout << "단어 '김영희'는 발견되지 않았습니다. " << endl;
        return 0;
}
```

Lab: 영어사전

- Map을 가지고 영어 사전을 구현
- 사용자로부터 단어를 받아서 단어의 설명을 출력

Lab: 영어사전

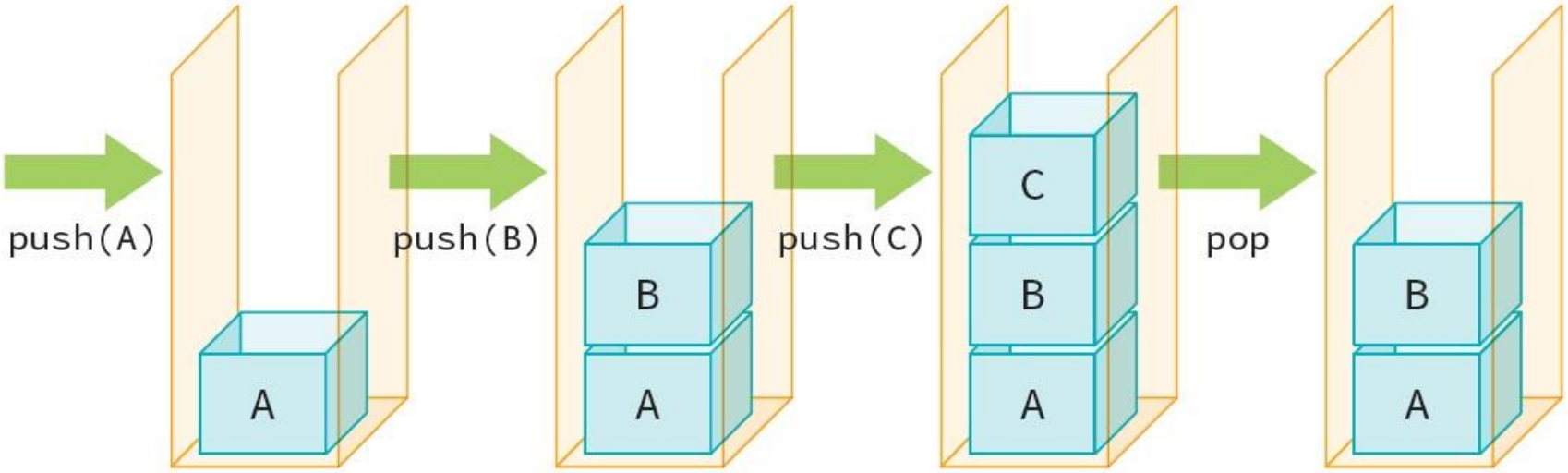
SEOIL UNIVERSITY

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

int main()
{
        string word;
        map<string, string> dic;
        dic["boy"] = "소년";
        dic["school"] = "학교";
        dic["office"] = "직장";
        dic["house"] = "집";
        dic["morning"] = "아침";
        dic["evening"] = "저녁";
```

Lab: 영어사전

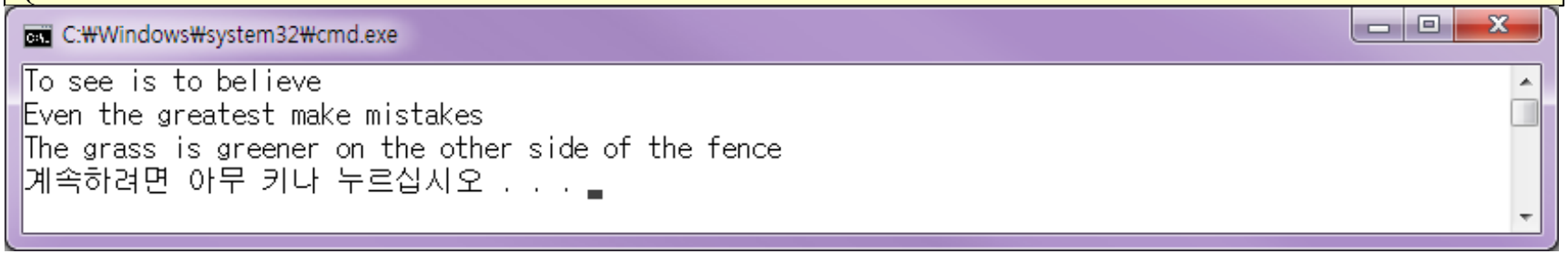

```
        while (true) {
                cout << "단어를 입력하시오: ";
                cin >> word;
                if (word == "quit") break;
                string meaning = dic[word];
                if (meaning != "")
                        cout << word << "의 의미는 " << meaning << endl;
        }
        return 0;
}
```

스택

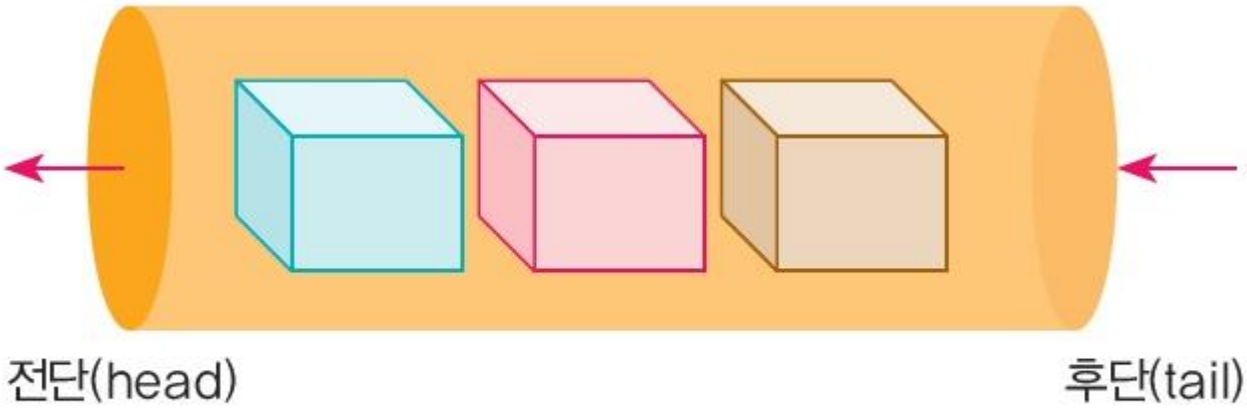


스택

```
#include <stack>
int main()
{
        stack<string> st;
        string sayings[3] =
        { "The grass is greener on the other side of the fence",
                "Even the greatest make mistakes",
                "To see is to believe" };

        for (auto& s : sayings)
                st.push(s);
        while (!st.empty()) {
                cout << st.top() << endl;
                st.pop();
        }
        return 0;
}
```



큐

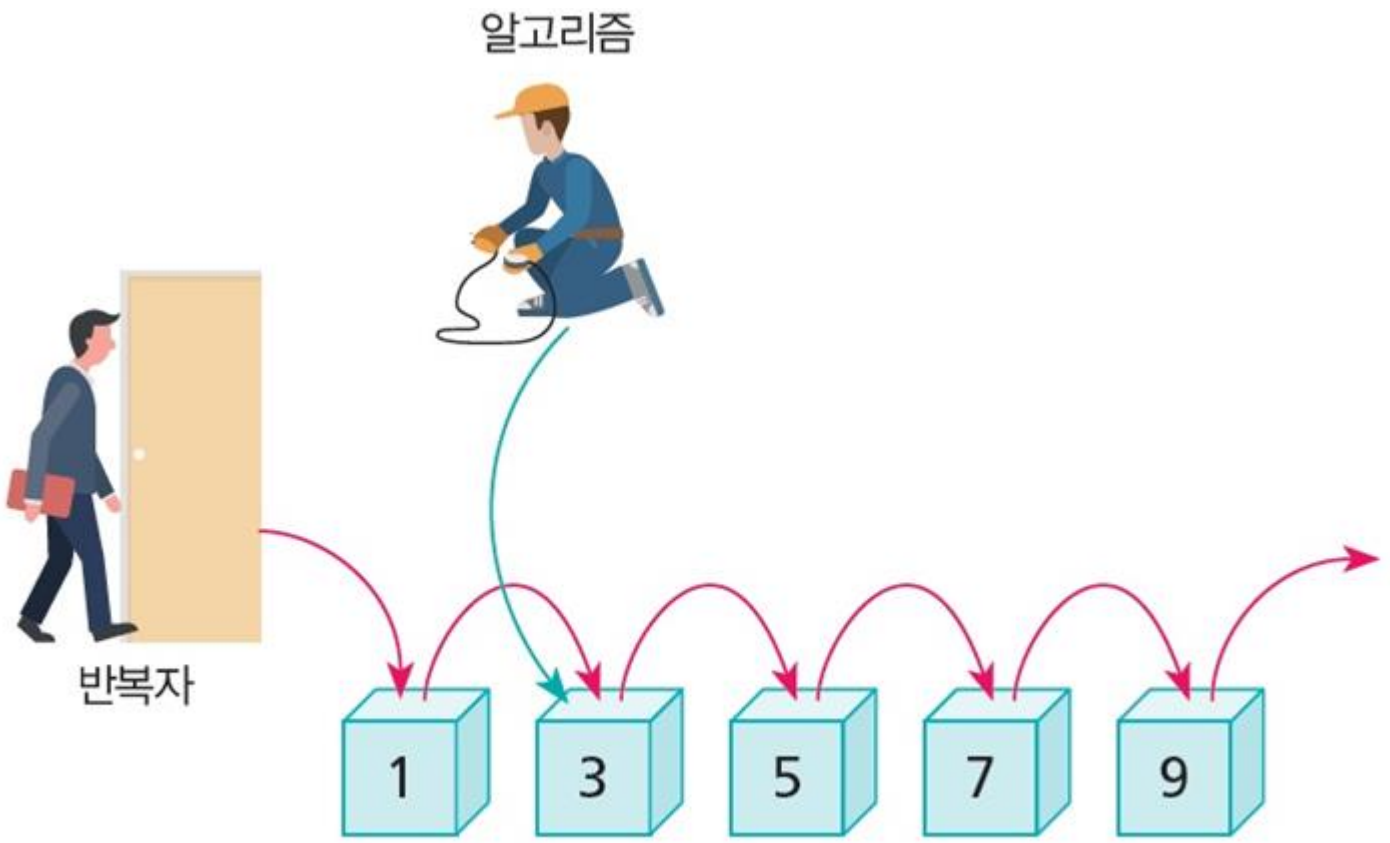


큐

큐

```
include<queue>
int main()
{
        queue<int> qu;
        qu.push(100);
        qu.push(200);
        qu.push(300);
        while (!qu.empty()) {
                cout << qu.front() << endl;
                qu.pop();
        }
        return 0;
}
```

STL 알고리즘



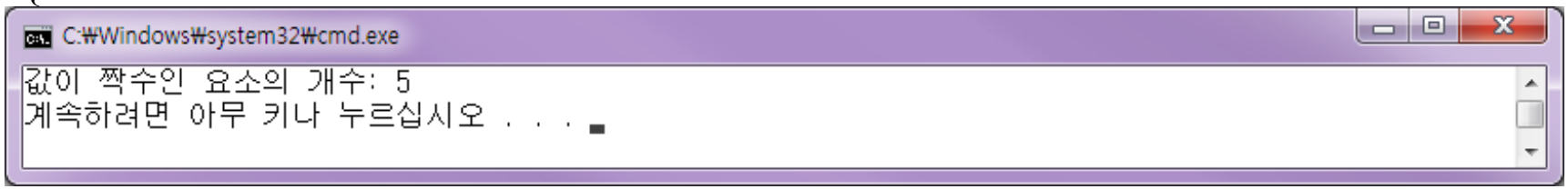
STL 알고리즘은 반복자를 통하여 컨테이너에 접근하여 작업을 한다.

find()와 find_if() 함수

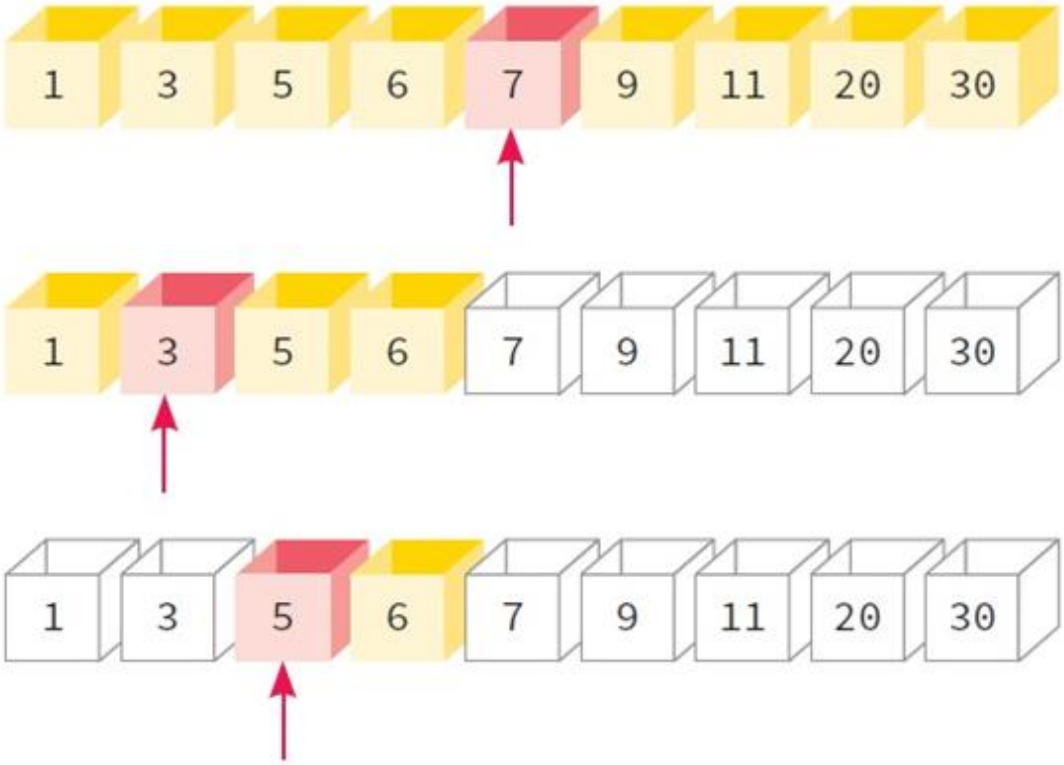

```
#include<algorithm>
int main()
{
        vector<string> vec { "사과", "토마토", "배", "수박", "키위" };

        auto it = find(vec.begin(), vec.end(), "수박");
        if (it != vec.end())
                cout << "수박이 " << distance(vec.begin(), it) << "에 있습니다." << endl;
        return 0;
}
```

count() 함수


```
#include<algorithm>
template <typename T>
bool is_even(const T& num)
{
        return (num % 2) == 0;
}

int main()
{
        vector<int> vec;
        for (int i = 0; i<10; i++)
                vec.push_back(i);

        size_t n = count_if(vec.begin(), vec.end(), is_even<int>);
        cout << "값이 짝수인 요소의 개수: " << n << endl;
        return 0;
}
```



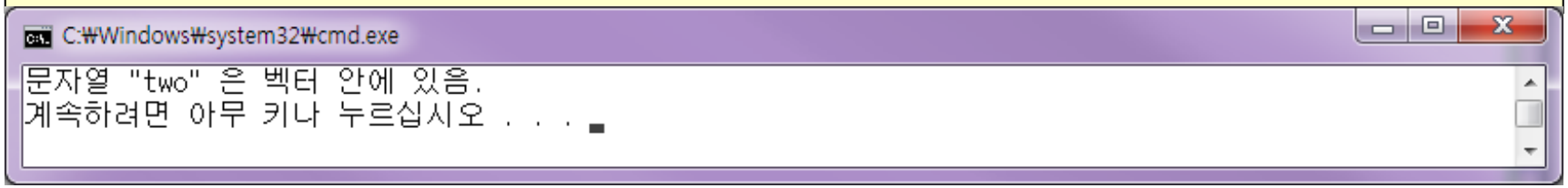
binary_search()



이진 탐색의 개념

binary_search()

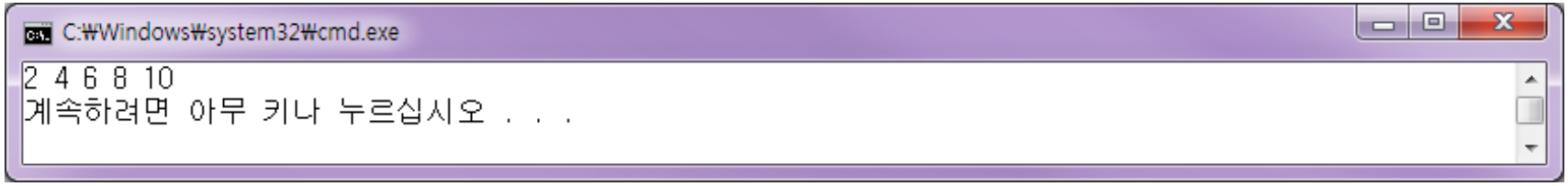

```
bool comp(string s1, string s2) {
        return (s1 == s2);
}

int main(void) {
        vector<string> v = { "one", "two", "three" };
        bool result;

        result = binary_search(v.begin(), v.end(), "two", comp);
        if (result == true)
                cout << "문자열 \"two\" 은 벡터 안에 있음." << endl;
        return 0;
}
```



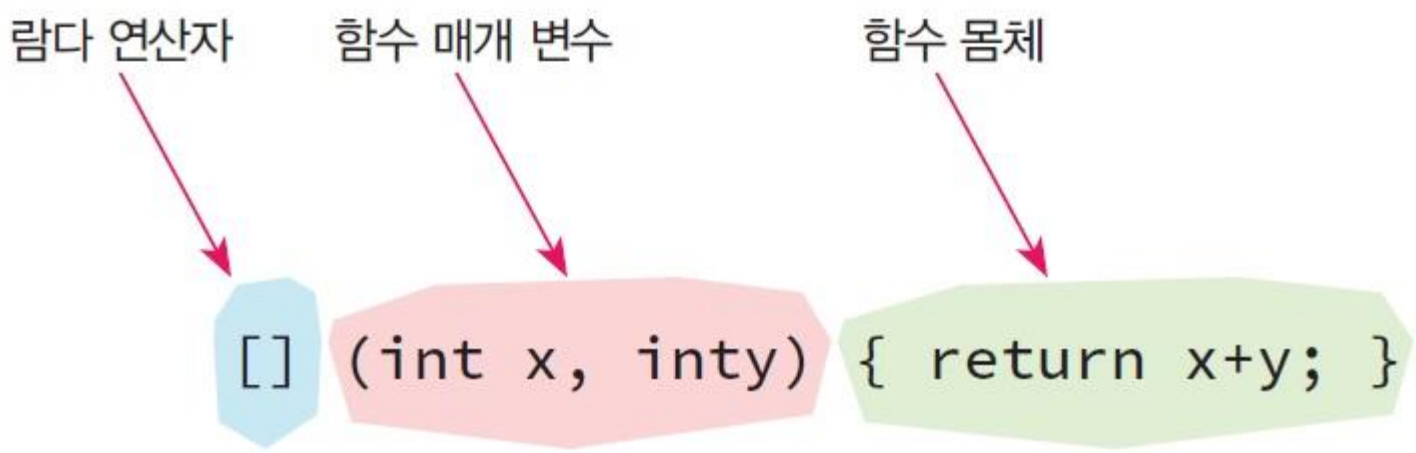
for_each() 함수

SEOIL UNIVERSITY

```
void printEven(int n) {
        if (n % 2 == 0)
                cout << n << ' ';
}

int main(void) {
        vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        for_each(v.begin(), v.end(), printEven);
        cout << endl;
        return 0;
}
```

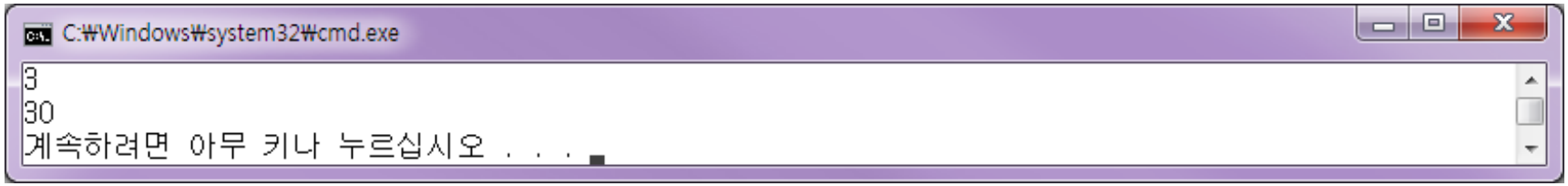


람다식



람다식


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

int main()
{
        auto sum = [](int x, int y) { return x + y; };
        cout << sum(1, 2) << endl;
        cout << sum(10, 20) << endl;
        return 0;
}
```



5보다 큰 정수


```
bool is_greater_than_5(int value)
{
	return (value > 5);
}

int main()
{
	vector<int> numbers{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
	auto count = count_if(numbers.begin(), numbers.end(), is_greater_than_5);

	cout << "5보다 큰 정수들의 개수: " << count << endl;
	return 0;
}
```

SEOIL UNIVERSITY

감사합니다.