STL containers (1)
sequential containers

2023

국민대학교 소프트웨어학부

STL

- STL: 표준 템플릿 라이브러리(Standard Template Library)
- 많은 프로그래머들이 공통적으로 사용하는 자료 구조와 알고리즘들을 template 으로 구현한 클래스
- namespace std 에 포함되어 있음

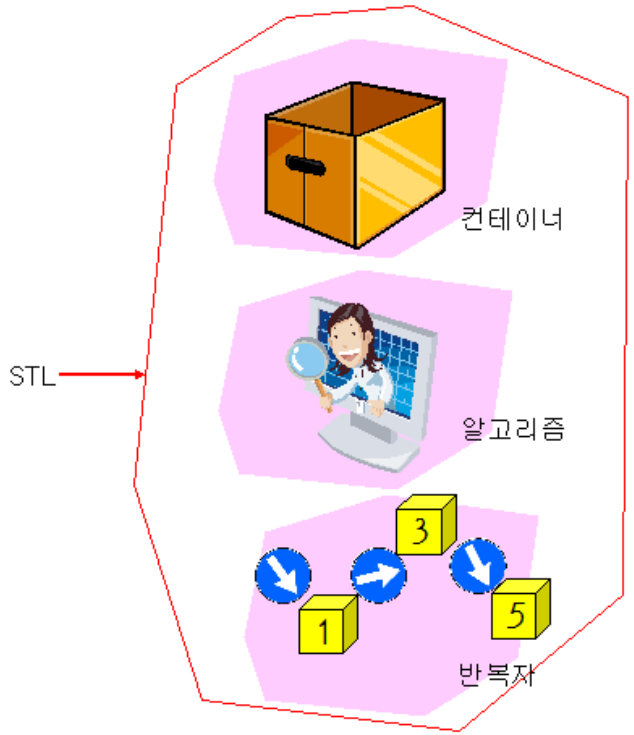


그림 18-1 STL

Standard Template Library

- 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

STL의 장점

- STL은 전문가가 만들어서 테스트를 거친 검증된 라이브러리
- STL은 객체 지향 기법과 일반화 프로그래밍 기법을 적용하여서 만들어졌으므로 어떤 자료형에 대해서도 적용
- STL을 사용하면 개발 기간을 단축할 수 있고 버그가 없는 프로그램을 만들 수 있다.

STL containers

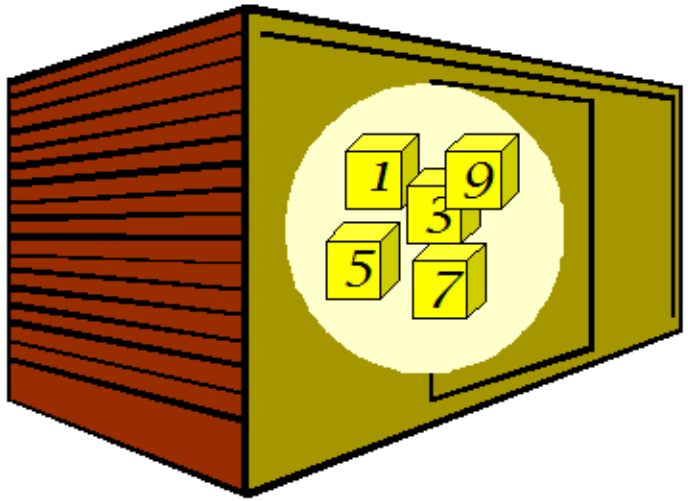


그림 18-2 컨테이너의 개념

순차 컨테이너 :
순차적으로 원소에 접근할 수 있는 자료구조
연관 컨테이너 :
원소의 값(값 자체 혹은 키)으로 접근할 수 있음
검색을 빠르게 할 수 있도록 데이터를 정렬하여 저장하는 자료구조
컨테이너 어댑터:
순차 컨테이너와는 다른 방법으로 원소에 접근하는 자료구조

컨테이너

set :
set의 원소는 정렬되어 있음
map :
key-value 를 값으로 함
key 값으로 정렬되어 있음

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>

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.

벡터

- 벡터 == 동적 배열 == 스마트 배열
- 템플릿으로 설계
- Kvector 와 유사

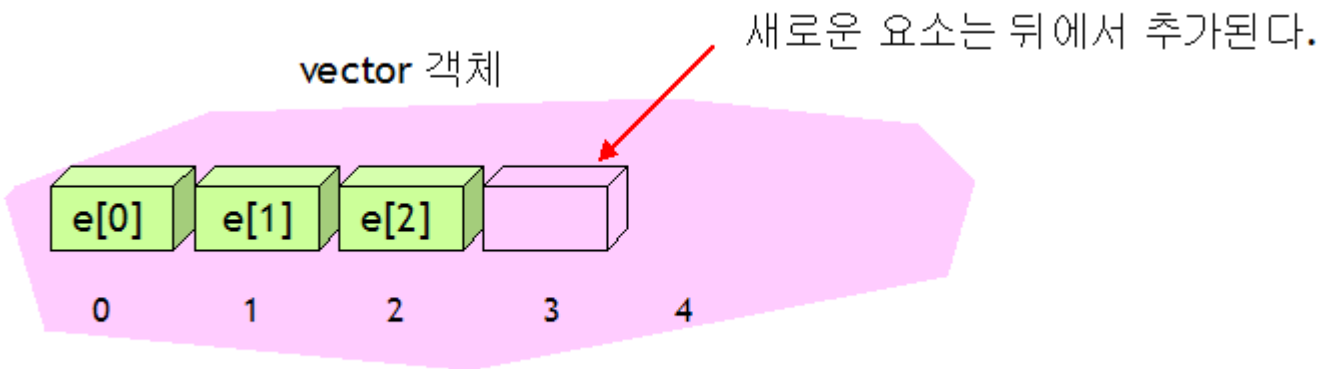


그림 18.5 벡터

empty vector 만들기

```cpp
#include <iostream>
#include <vector>
using namespace std;
//#include "team.h"
int main(){
  vector<int> v1;

  cout << "v1.size() = " << v1.size() << endl;
  cout << "v1.max_size() = " << v1.max_size() << endl;
  cout << "v1.empty() = " << v1.empty() << endl;
  cout << "v1[0] = " << v1[0] << endl;
```

```
v1.size() = 0
v1.max_size() = 4611686018427387903
v1.empty() = 1
Segmentation fault (core dumped)
```

http://www.cplusplus.com/reference/stl/

Legend:

C++98	Available since C++98
C++11	New in C++11

Sequence containers

Headers		<array>	<vector>	<deque>	<forward_list>	<list>
Members		**array**	**vector**	**deque**	**forward_list**	**list**
	constructor	*implicit*	vector	deque	forward_list	list
	destructor	*implicit*	~vector	~deque	~forward_list	~list
	operator=	*implicit*	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin before_begin	begin
	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin		rbegin
	rend	rend	rend	rend		rend
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin cbefore_begin	cbegin
	cend	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin		crbegin
	crend	crend	crend	crend		crend
capacity	size	size	size	size		size
	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty
	resize		resize	resize	resize	resize
	shrink_to_fit		shrink_to_fit	shrink_to_fit		
	capacity		capacity			
	reserve		reserve			
element access	front	front	front	front	front	front
	back	back	back	back		back
	operator[]	operator[]	operator[]	operator[]		
	at	at	at	at		
modifiers	assign		assign	assign	assign	assign
	emplace		emplace	emplace	emplace_after	emplace
	insert		insert	insert	insert_after	insert
	erase		erase	erase	erase_after	erase
	emplace_back		emplace_back	emplace_back		emplace_back
	push_back		push_back	push_back		push_back
	pop_back		pop_back	pop_back		pop_back
	emplace_front			emplace_front	emplace_front	emplace_front
	push_front			push_front	push_front	push_front
	pop_front			pop_front	pop_front	pop_front
	clear		clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap

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

int main(){
  vector<int> v1(3);

  cout << "v1.size() = " << v1.size() << endl;
  cout << "v1.max_size() = " << v1.max_size() << endl;
  cout << "v1.empty() = " << v1.empty() << endl;
  cout << "v1.capacity() = " << v1.capacity() << endl;
  cout << "v1[0] = " << v1[0] << endl;
```

```
v1.size() = 3
v1.max_size() = 4611686018427387903
v1.empty() = 0
v1.capacity() = 3
v1[0] = 0
```

```cpp
#include <iostream>
#include <vector>
using namespace std;
#define N 3

int main(){
  vector<int> v1(N, 5);

  cout << "v1.size() = " << v1.size() << endl;
  cout << "v1.max_size() = " << v1.max_size() << endl;
  cout << "v1.empty() = " << v1.empty() << endl;
  cout << "v1.capacity() = " << v1.capacity() << endl;
  cout << "v1 = ";
  for (int i=0; i<N; i++)
     cout << v1[i] << " ";
  cout << endl;
```

```
v1.size() = 3
v1.max_size() = 4611686018427387903
v1.empty() = 0
v1.capacity() = 3
v1 = 5 5 5
```

```cpp
#include <iostream>
#include <vector>
using namespace std;
#define N 3

int main(){
  vector<int> v1(N, 5);

  cout << "v1.size() = " << v1.size() << endl;
  cout << "v1.max_size() = " << v1.max_size() << endl;
  cout << "v1.empty() = " << v1.empty() << endl;
  cout << "v1.capacity() = " << v1.capacity() << endl;
  v1[0] = 1;
  v1[1] = 2;
  v1[4] = 3;
  cout << "v1 = ";
  for (int i=0; i<N; i++)
     cout << v1[i] << " ";
  cout << endl;
```

```
v1.size() = 3
v1.max_size() = 4611686018427387903
v1.empty() = 0
v1.capacity() = 3
v1 = 1 2 5
```

operator [n]
  index n이 vector의 크기 범위 내에 있는지를 검사하지 않음
  index 가 out-of-bound 인 경우
    : behavior is undefined(어떤 일이 일어날 지 예측할 수 없음)
      프로그램이 종료될 수도 있고, 정상적으로 동작할 수도 있음

```cpp
v1.at(4) = 3;
```

멤버 함수 at(n)
  항상 index n이 vector의 크기 범위 내에 있는지를 검사함
  index 가 out-of-bound 인 경우
    : out-of-range exception이 발생함

```cpp
#include <iostream>
#include <vector>
using namespace std;
#define N 3

int main(){
  vector<int> v1(N, 5);

  v1.resize(6);
  cout << "v1.size() = " << v1.size() << endl;
  cout << "v1.max_size() = " << v1.max_size() << endl;
  cout << "v1.empty() = " << v1.empty() << endl;
  cout << "v1.capacity() = " << v1.capacity() << endl;
  v1[0] = 1;
  v1[1] = 2;
  v1[4] = 3;
  cout << "v1 = ";
  for (int i=0; i<6; i++)
     cout << v1[i] << " ";
  cout << endl;
```

5 5 5 0 0 0

```
v1.size() = 6
v1.max_size() = 4611686018427387903
v1.empty() = 0
v1.capacity() = 6
v1 = 1 2 5 0 3 0
```

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;
#define N 3
template <class T>
void print_vector(vector<T> v, string s=""){
  cout << s;
  for (int i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;
}
int main(){
  vector<int> v1(N, 5);

  v1.resize(6);
  cout << "v1.size() = " << v1.size() << endl;
  cout << "v1.capacity() = " << v1.capacity() << endl;
  v1[0] = 1;
  v1[1] = 2;
  v1[4] = 3;
  print_vector(v1, "v1= ");
  v1.clear();
  print_vector(v1, "v1= ");
  cout << "v1.size() = " << v1.size() << endl;
  cout << "v1.capacity() = " << v1.capacity() << endl;
```

```
v1.size() = 6
v1.capacity() = 6
v1= 1 2 5 0 3 0
v1=
v1.size() = 0
v1.capacity() = 6
```

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;
#define N 3
template <class T>
void print_vector(vector<T> v, string s=""){
  cout << s << "(" << v.capacity() << ")";
  for (int i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;
}
int main(){
  vector<int> v1(N, 5);

  v1.resize(6);
  print_vector(v1, "v1= ");
  v1[0] = 1;
  v1[1] = 2;
  v1[4] = 3;
  print_vector(v1, "v1= ");
  cout << "v1.front() = " << v1.front() << endl;
  cout << "v1.back() = " << v1.back() << endl;
```

```
v1= (6)5 5 5 0 0 0
v1= (6)1 2 5 0 3 0
v1.front() = 1
v1.back() = 0
```

push_back()과 pop_back()

- push_back()
  - 새로운 데이터를 벡터의 끝에 추가하고 벡터의 크기를 1만큼 증가
- pop_back()
  - 벡터의 끝에서 요소를 제거하고 벡터의 크기를 하나 감소

```cpp
#define N 3
template <class T>
void print_vector(vector<T> v, string s=""){
  cout << s << "(" << v.capacity() << ")";
  for (int i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;
}
int main(){
  vector<int> v1(N, 5);

  v1.resize(6);
  print_vector(v1, "v1= ");
  v1[0] = 1;
  v1[1] = 2;
  v1[4] = 3;
  print_vector(v1, "v1= ");
  cout << "v1.front() = " << v1.front() << endl;
  cout << "v1.back() = " << v1.back() << endl;
  v1.push_back(100);
  print_vector(v1, "v1= ");
  v1.pop_back();
  print_vector(v1, "v1= ");
```

```
v1= (6)5 5 5 0 0 0
v1= (6)1 2 5 0 3 0
v1.front() = 1
v1.back() = 0
v1= (7)1 2 5 0 3 0 100
v1= (6)1 2 5 0 3 0
```

반복자

- 반복자(iterator)
  - 현재 처리하고 있는 자료의 위치를 기억하는 객체
  - 포인터와 유사
  - * 연산자 사용 가능
  - ++ 연산자 사용 가능

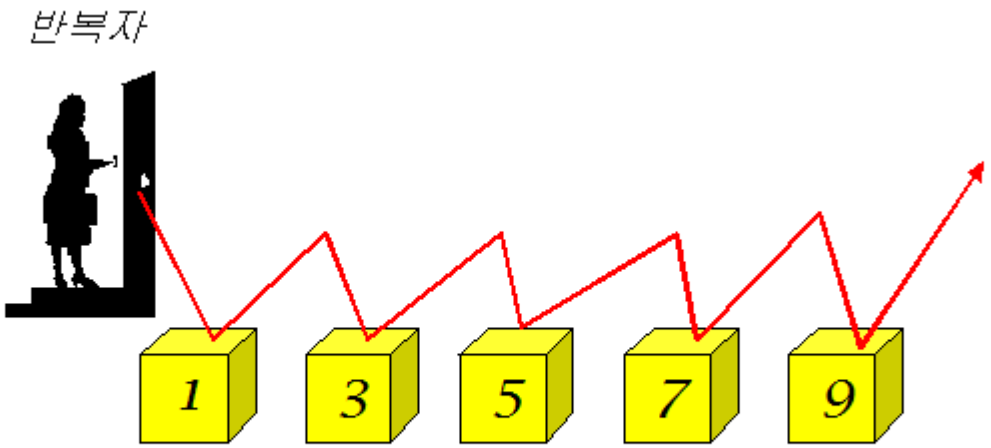
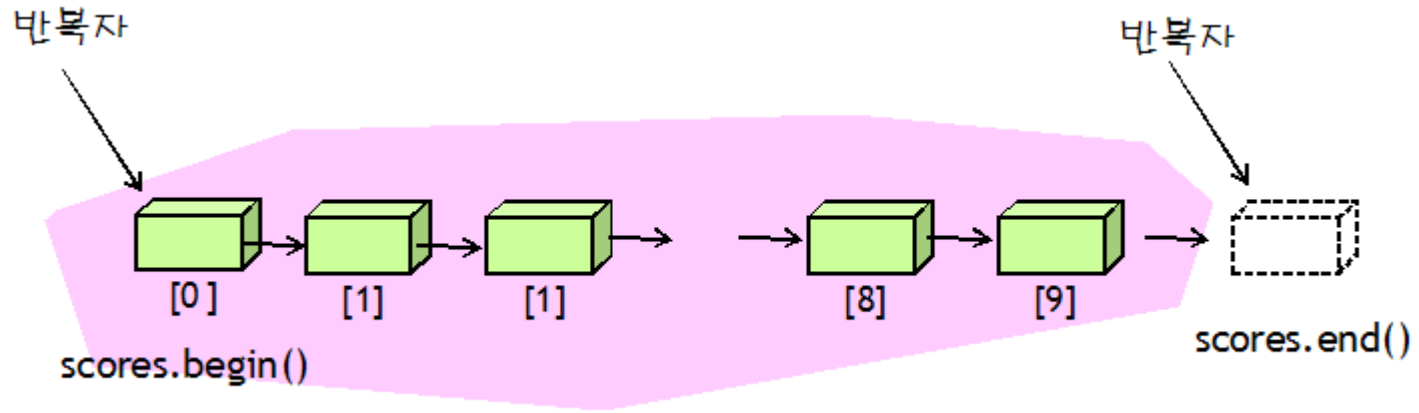


그림 18.3 반복자는 컨테이너에 저장된 요소들을 순차적으로 방문한다.

반복자 iterator 의 사용



- v.begin() 함수는 컨테이너 v의 첫 번째 요소를 가리키는 반복자를 반환.
- v.end() 함수는 컨테이너 v의 마지막 요소의 (가상의) 다음 요소를 가리키는 반복자를 반환.

반복자에 사용 가능한 연산자들

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

```
13  int main(){
14    vector<int> v1(N, 5);
15
16    print_vector(v1, "v1= ");
17    vector<int>::iterator it = v1.begin();
18    for (; it != v1.end(); it++) *it = 10;
19    print_vector(v1, "v1= ");
```

```
v1= (3)5 5 5
v1= (3)10 10 10
```

print_vector() 함수의 변경

```
5  #define N 3
6  template <class T>
7  void print_vector(vector<T> v, string s=""){
8    cout << s << "(" << v.capacity() << ")";
9    auto it = v.begin();
10   for (; it != v.end(); it++) cout << *it << " ";
11   cout << endl;
12 }
13 int main(){
14   vector<int> v1(N, 5);
15
16   print_vector(v1, "v1= ");
17   vector<int>::iterator it = v1.begin();
18   for (; it != v1.end(); it++) *it = 10;
19   print_vector(v1, "v1= ");
```

vector<T>::iterator it=v.begin();

변수 type 을 auto 로 선언하려면
초기화문이 있어야만 함

http://www.cplusplus.com/reference/stl/

Legend:

C++98	Available since C++98
C++11	New in C++11

Sequence containers

Headers		<array>	<vector>	<deque>	<forward_list>	<list>
Members		**array**	**vector**	**deque**	**forward_list**	**list**
	constructor	*implicit*	vector	deque	forward_list	list
	destructor	*implicit*	~vector	~deque	~forward_list	~list
	operator=	*implicit*	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin before_begin	begin
	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin		rbegin
	rend	rend	rend	rend		rend
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin cbefore_begin	cbegin
	cend	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin		crbegin
	crend	crend	crend	crend		crend
capacity	size	size	size	size		size
	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty
	resize		resize	resize	resize	resize
	shrink_to_fit		shrink_to_fit	shrink_to_fit		
	capacity		capacity			
	reserve		reserve			
element access	front	front	front	front	front	front
	back	back	back	back		back
	operator[]	operator[]	operator[]	operator[]		
	at	at	at	at		
modifiers	assign		assign	assign	assign	assign
	emplace		emplace	emplace	emplace_after	emplace
	insert		insert	insert	insert_after	insert
	erase		erase	erase	erase_after	erase
	emplace_back		emplace_back	emplace_back		emplace_back
	push_back		push_back	push_back		push_back
	pop_back		pop_back	pop_back		pop_back
	emplace_front			emplace_front	emplace_front	emplace_front
	push_front			push_front	push_front	push_front
	pop_front			pop_front	pop_front	pop_front
	clear		clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap

http://www.cplusplus.com/reference/vector/vector/rbegin/

public member function

std::vector::rbegin

<vector>

C++98 C++11

```
      reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
```

Return reverse iterator to reverse beginning

```
1  // vector::rbegin/rend
2  #include <iostream>
3  #include <vector>
4
5  int main ()
6  {
7    std::vector<int> myvector (5);  // 5 default-constructed ints
8
9    int i=0;
10
11   std::vector<int>::reverse_iterator rit = myvector.rbegin();
12   for (; rit!= myvector.rend(); ++rit)
13     *rit = ++i;
14
15   std::cout << "myvector contains:";
16   for (std::vector<int>::iterator it = myvector.begin(); it != myvector.end(); ++it)
17     std::cout << ' ' << *it;
18   std::cout << '\n';
19
20   return 0;
21 }
```

Edit & Run

Output:

```
myvector contains: 5 4 3 2 1
```

여러 형태의 vector<> constructors

#define N 3

```
vector<int> v1(N, 5);
print_vector(v1, "v1= ");
int a[10] = {0,1,2,3,4,5,6,7,8,9};
vector<int> v2(&a[0], &a[5]);
print_vector(v2, "v2= ");
vector<int> v3(v2);
print_vector(v3, "v3= ");
vector<int> v4(&v3[1], &v3[4]);
print_vector(v4, "v4= ");
vector<int> v5(v3.begin()+1, v3.begin()+4);
print_vector(v5, "v5= ");
```

int * (배열 원소에 대한 pointer

copy constructor

container class 의 iterator

```
v1= (3)5 5 5
v2= (5)0 1 2 3 4
v3= (5)0 1 2 3 4
v4= (3)1 2 3
v5= (3)1 2 3
```

std::vector::insert

<vector>

C++98 C++11

```
single element (1)  iterator insert (iterator position, const value_type& val);
fill (2)                 void insert (iterator position, size_type n, const value_type& val);
range (3)           template <class InputIterator>
                         void insert (iterator position, InputIterator first, InputIterator last);
```

Insert elements

The vector is extended by inserting new elements before the element at the specified *position*, effectively increasing the container size by the number of elements inserted.

This causes an automatic reallocation of the allocated storage space if -and only if- the new vector size surpasses the current vector capacity.

Because vectors use an array as their underlying storage, inserting elements in positions other than the vector end causes the container to relocate all the elements that were after *position* to their new positions. This is generally an inefficient operation compared to the one performed for the same operation by other kinds of sequence containers (such as list or forward_list).

The parameters determine how many elements are inserted and to which values they are initialized:

Parameters

`position`
: Position in the vector where the new elements are inserted.
`iterator` is a member type, defined as a random access iterator type that points to elements.

`val`
: Value to be copied (or moved) to the inserted elements.
Member type `value_type` is the type of the elements in the container, defined in deque as an alias of its first template parameter (`T`).

`n`
: Number of elements to insert. Each element is initialized to a copy of *val*.
Member type `size_type` is an unsigned integral type.

`first, last`
: Iterators specifying a range of elements. Copies of the elements in the range `[first,last)` are inserted at *position* (in the same order).
Notice that the range includes all the elements between *first* and *last*, including the element pointed by *first* but not the one pointed by *last*.
The function template argument `InputIterator` shall be an input iterator type that points to elements of a type from which `value_type` objects can be constructed.

`il`
: An initializer_list object. Copies of these elements are inserted at *position* (in the same order).
These objects are automatically constructed from *initializer list* declarators.
Member type `value_type` is the type of the elements in the container, defined in vector as an alias of its first template parameter (`T`).

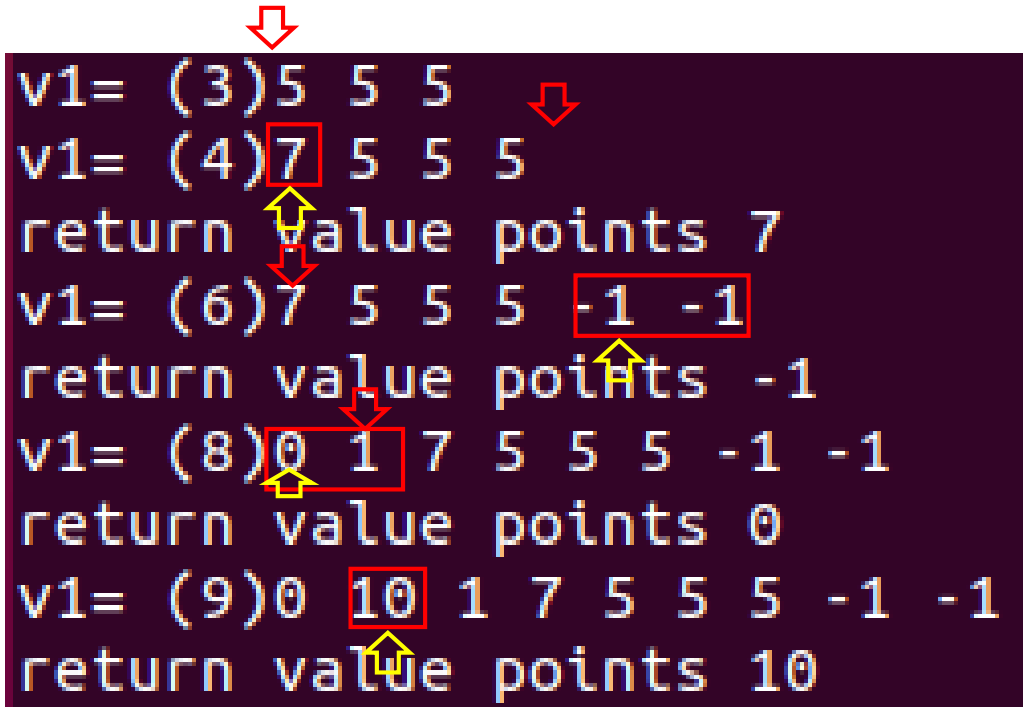
Return value

An iterator that points to the first of the newly inserted elements.

#define N 3

```
13  int main(){
14    vector<int> v1(N, 5);
15    print_vector(v1, "v1= ");
16    int a[10] = {0,1,2,3,4,5,6,7,8,9};
17    vector<int>::iterator it;
18    it = v1.insert(v1.begin(),7);
19    print_vector(v1, "v1= ");
20    cout << "return value points " << *it << endl;
21    it = v1.insert(v1.end(), 2, -1);
22    print_vector(v1, "v1= ");
23    cout << "return value points " << *it << endl;
24    it = v1.insert(v1.begin(), &a[0], &a[2]);
25    print_vector(v1, "v1= ");
26    cout << "return value points " << *it << endl;
27    it = v1.insert(v1.begin()+1,10);
28    print_vector(v1, "v1= ");
29    cout << "return value points " << *it << endl;
```

노란 화살표⇧는 return 값이 가리키는 위치
빨간 화살표⇩는 함수의 1st parameter 가 가리키는 위치



std::vector::erase

<vector>

C++98 C++11

```
iterator erase (iterator position);
iterator erase (iterator first, iterator last);
```

Erase elements

Removes from the vector either a single element (*position*) or a range of elements (`[first,last)`).

This effectively reduces the container size by the number of elements removed, which are destroyed.

Because vectors use an array as their underlying storage, erasing elements in positions other than the vector end causes the container to relocate all the elements after the segment erased to their new positions. This is generally an inefficient operation compared to the one performed for the same operation by other kinds of sequence containers (such as list or forward_list).

Parameters

`position`

Iterator pointing to a single element to be removed from the vector.
Member types `iterator` and `const_iterator` are random access iterator types that point to elements.

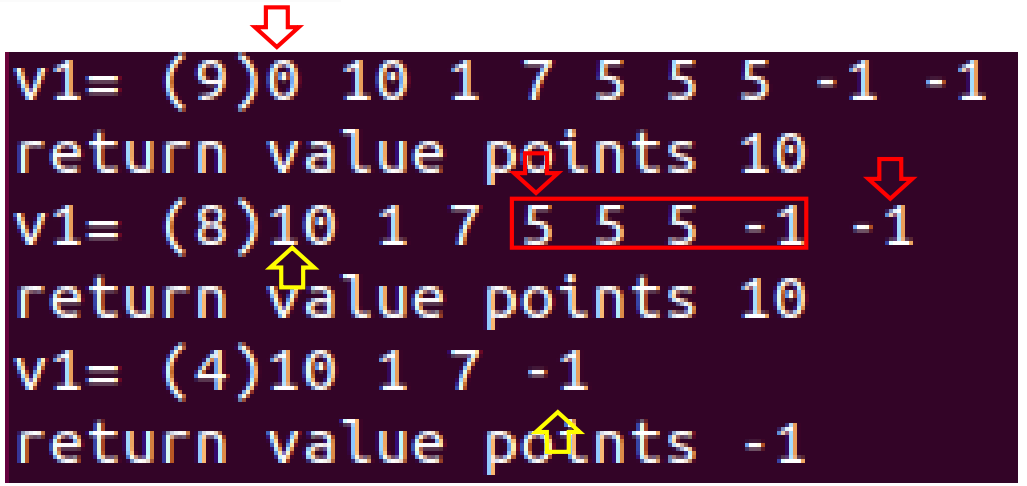
`first, last`

Iterators specifying a range within the vector] to be removed: `[first,last)`. i.e., the range includes all the elements between *first* and *last*, including the element pointed by *first* but not the one pointed by *last*.
Member types `iterator` and `const_iterator` are random access iterator types that point to elements.

Return value

An iterator pointing to the new location of the element that followed the last element erased by the function call. This is the container end if the operation erased the last element in the sequence.

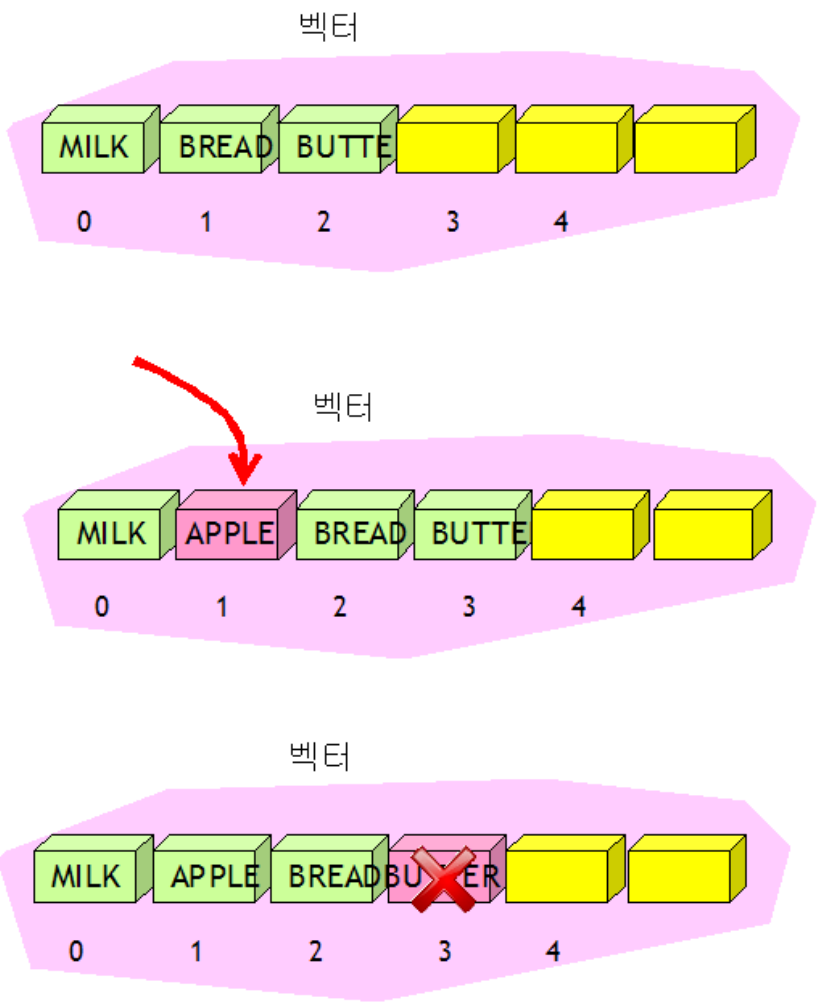
```
28      print_vector(v1, "v1= ");
29      cout << "return value points " << *it << endl;
30      it = v1.erase(v1.begin());
31      print_vector(v1, "v1= ");
32      cout << "return value points " << *it << endl;
33      it = v1.erase(v1.begin()+3, v1.end()-1);
34      print_vector(v1, "v1= ");
35      cout << "return value points " << *it << endl;
```



```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;
template <class T>
void print_vector(vector<T> v, string s=""){
  cout << s << "(" << v.capacity() << ")";
  auto it = v.begin();
  for (; it != v.end(); it++) cout << *it << " ";
  cout << endl;
}
int main(){
  vector<string> vec;

  print_vector(vec, "vec= ");
  vec.push_back("MILK");
  vec.push_back("BREAD");
  vec.push_back("BUTTER");
  print_vector(vec, "vec= ");
  vec.insert(vec.begin()+1, "APPLE");
  print_vector(vec, "vec= ");
  vec.pop_back();
  print_vector(vec, "vec= ");
```

```
vec= (0)
vec= (3)MILK BREAD BUTTER
vec= (4)MILK APPLE BREAD BUTTER
vec= (3)MILK APPLE BREAD
```



컨테이너의 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

http://www.cplusplus.com/reference/stl/

Legend:

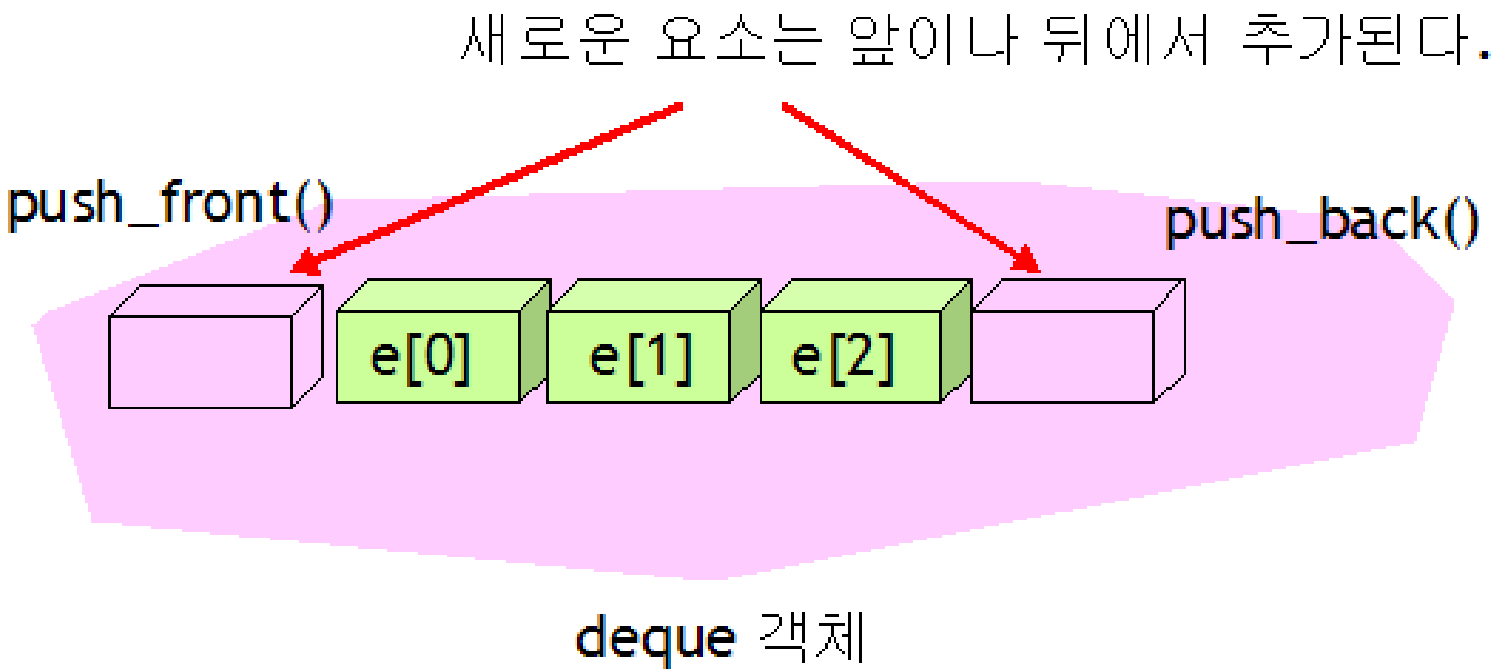
C++98	Available since C++98
C++11	New in C++11

Sequence containers

Headers		<array>	<vector>	<deque>	<forward_list>	<list>
Members		**array**	**vector**	**deque**	**forward_list**	**list**
	constructor	*implicit*	vector	deque	forward_list	list
	destructor	*implicit*	~vector	~deque	~forward_list	~list
	operator=	*implicit*	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin before_begin	begin
	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin		rbegin
	rend	rend	rend	rend		rend
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin cbefore_begin	cbegin
	cend	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin		crbegin
	crend	crend	crend	crend		crend
capacity	size	size	size	size		size
	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty
	resize		resize	resize	resize	resize
	shrink_to_fit		shrink_to_fit	shrink_to_fit		
	capacity		capacity			
	reserve		reserve			
element access	front	front	front	front	front	front
	back	back	back	back		back
	operator[]	operator[]	operator[]	operator[]		
	at	at	at	at		
modifiers	assign		assign	assign	assign	assign
	emplace		emplace	emplace	emplace_after	emplace
	insert		insert	insert	insert_after	insert
	erase		erase	erase	erase_after	erase
	emplace_back		emplace_back	emplace_back		emplace_back
	push_back		push_back	push_back		push_back
	pop_back		pop_back	pop_back		pop_back
	emplace_front			emplace_front	emplace_front	emplace_front
	push_front			push_front	push_front	push_front
	pop_front			pop_front	pop_front	pop_front
	clear		clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap

deque

- vector class 의 전단과 후단에서 모두 요소를 추가하고 삭제하는 것을 허용

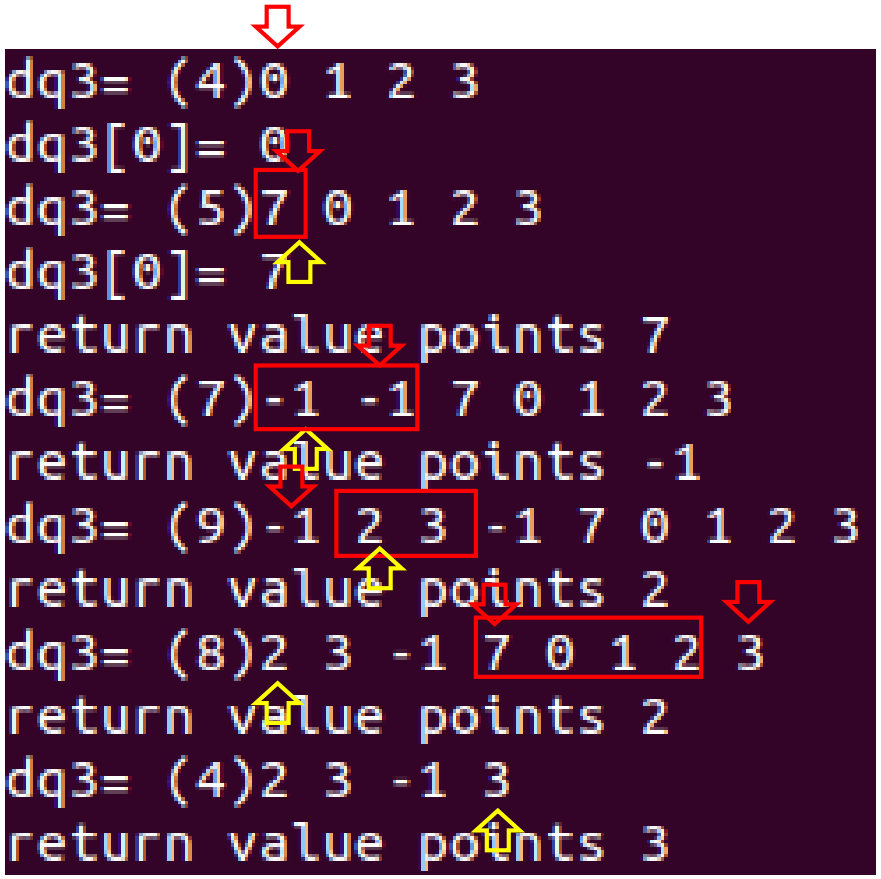


```cpp
#include <iostream>
#include <vector>
#include <deque>
#include <string>
using namespace std;
template <class T>
void print_container(const T& container, string s=""){
  cout << s << "(" << container.size() << ")";
  auto it = container.begin();
  for (; it != container.end(); it++) cout << *it << " ";
  cout << endl;
}
```

```cpp
int main(){
  int a[10] = {0,1,2,3,4,5,6,7,8,9};
  vector<int> v(&a[0], &a[4]);
  deque<int> dq1(3, 5);
  deque<int> dq2(&a[0], &a[4]);
  deque<int> dq3(v.begin(), v.end());
  print_container(v, "v= ");
  print_container(dq1, "dq1= ");
  print_container(dq2, "dq2= ");
  print_container(dq3, "dq3= ");
  dq3.push_front(-1);
  print_container(dq3, "dq3= ");
  dq3.push_back(100);
  print_container(dq3, "dq3= ");
  dq3.pop_front();
  print_container(dq3, "dq3= ");
  dq3.pop_back();
  print_container(dq3, "dq3= ");
```

```
v= (4)0 1 2 3
dq1= (3)5 5 5
dq2= (4)0 1 2 3
dq3= (4)0 1 2 3
dq3= (5)-1 0 1 2 3
dq3= (6)-1 0 1 2 3 100
dq3= (5)0 1 2 3 100
dq3= (4)0 1 2 3
```

```
30    print_container(dq3, "dq3= ");
31    cout << "dq3[0]= " << dq3[0] << endl;
32    auto it = dq3.insert(dq3.begin(),7);
33    print_container(dq3, "dq3= ");
34    cout << "dq3[0]= " << dq3[0] << endl;
35    cout << "return value points " << *it << endl;
36    it = dq3.insert(dq3.begin(),2,-1);
37    print_container(dq3, "dq3= ");
38    cout << "return value points " << *it << endl;
39    it = dq3.insert(dq3.begin()+1,v.begin()+2, v.begin()+4);
40    print_container(dq3, "dq3= ");
41    cout << "return value points " << *it << endl;
42    it = dq3.erase(dq3.begin());
43    print_container(dq3, "dq3= ");
44    cout << "return value points " << *it << endl;
45    it = dq3.erase(dq3.begin()+3, dq3.end()-1);
46    print_container(dq3, "dq3= ");
47    cout << "return value points " << *it << endl;
```



반복자의 종류

- 전향 반복자(forward iterator): ++ 연산자만 가능하다.
- 양방향 반복자(bidirectional iterator): ++ 연산자와 -- 연산자가 가능하다.
- 무작위 접근 반복자(random access iterator): +, -, <, > 연산자와 [ ] 연산자가 가능하다.

- http://www.cplusplus.com/reference/iterator/

The properties of each iterator category are:

category				properties	valid expressions
all categories				*copy-constructible*, *copy-assignable* and *destructible*	`X b(a);` `b = a;`
				Can be incremented	`++a` `a++`
Random Access	Bidirectional	Forward	Input	Supports equality/inequality comparisons	`a == b` `a != b`
				Can be dereferenced as an *rvalue*	`*a` `a->m`
			Output	Can be dereferenced as an *lvalue* (only for *mutable iterator types*)	`*a = t` `*a++ = t`
				default-constructible	`X a;` `X()`
				Multi-pass: neither dereferencing nor incrementing affects dereferenceability	`{ b=a; *a++; *b; }`

				Can be decremented	`--a`
					`a--`
					`*a--`

				Supports arithmetic operators + and –	`a + n`
					`n + a`
					`a - n`
					`a - b`

				Supports inequality comparisons (<, >, <= and >=) between iterators	`a < b`
					`a > b`
					`a <= b`
					`a >= b`

				Supports compound assignment operations += and -=	`a += n` `a -= n`
				Supports offset dereference operator ([])	`a[n]`

list<>::iterator
set<>::iterator
map<>::iterator

vector<>::iterator
deque<>::iterator

Where `X` is an iterator type, `a` and `b` are objects of this iterator type, `t` is an object of the type pointed by the iterator type, and `n` is an integer value.

http://www.cplusplus.com/reference/vector/vector/

Member types

C++98 C++11

member type	definition	notes
value_type	The first template parameter (T)	
allocator_type	The second template parameter (Alloc)	defaults to: allocator<value_type>
reference	value_type&	
const_reference	const value_type&	
pointer	allocator_traits<allocator_type>::pointer	for the default allocator: value_type*
const_pointer	allocator_traits<allocator_type>::const_pointer	for the default allocator: const value_type*
iterator	a random access iterator to value_type	convertible to const_iterator
const_iterator	a random access iterator to const value_type	
reverse_iterator	reverse_iterator<iterator>	
const_reverse_iterator	reverse_iterator<const_iterator>	
difference_type	a signed integral type, identical to: iterator_traits<iterator>::difference_type	usually the same as ptrdiff_t
size_type	an unsigned integral type that can represent any non-negative value of difference_type	usually the same as size_t

http://www.cplusplus.com/reference/list/list/

Member types

C++98 C++11

member type	definition	notes
value_type	The first template parameter (T)	
allocator_type	The second template parameter (Alloc)	defaults to: allocator<value_type>
reference	allocator_type::reference	for the default allocator: value_type&
const_reference	allocator_type::const_reference	for the default allocator: const value_type&
pointer	allocator_type::pointer	for the default allocator: value_type*
const_pointer	allocator_type::const_pointer	for the default allocator: const value_type*
iterator	a bidirectional iterator to value_type	convertible to const_iterator
const_iterator	a bidirectional iterator to const value_type	
reverse_iterator	reverse_iterator<iterator>	
const_reverse_iterator	reverse_iterator<const_iterator>	
difference_type	a signed integral type, identical to: iterator_traits<iterator>::difference_type	usually the same as ptrdiff_t
size_type	an unsigned integral type that can represent any non-negative value of difference_type	usually the same as size_t

```
void print_container(const T& container, string s=""){
  cout << s << "(" << container.size() << ")";
  auto it = container.begin();
  for (; it != container.end(); it++) cout << *it << " "
  cout << endl;
}
```

```
void print_container(const T& container, string s=""){
  cout << s << "(" << container.size() << ")";
  auto it = container.begin();
  for (; it < container.end(); it++) cout << *it << " ";
  cout << endl;
}
```

T 가 list<> 인 경우 compiler error, vector<>, deque<> 의 경우 가능함

```
template <class T>
void print_vector(vector<T> v, string s=""){
  cout << s;
  for (int i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;
}
```

리스트

- random access 가 안 되는 deque 라고 볼 수 있음
- 즉, [] operator 가 없음
- bidirectional iterator
- 이중 연결 리스트로 구현

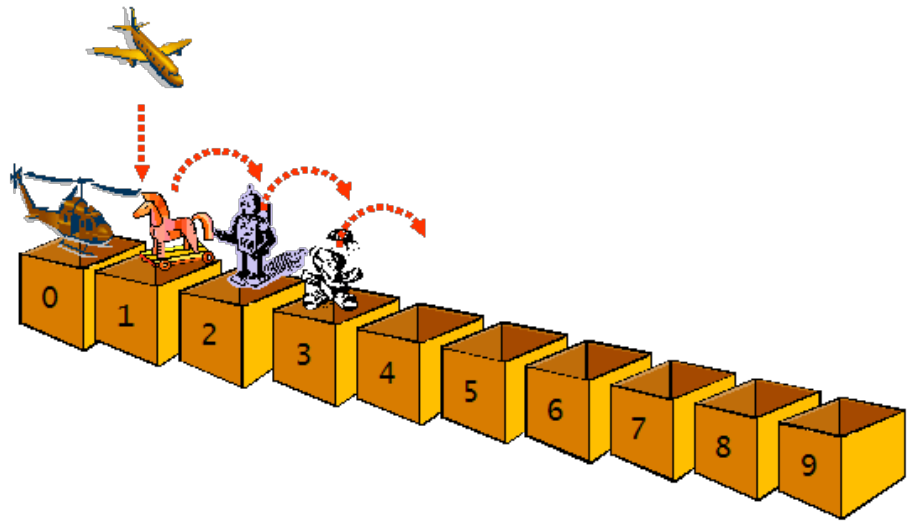


그림 18.7 배열의 중간에 삽입하려면 원소들을 이동하여야 한다.

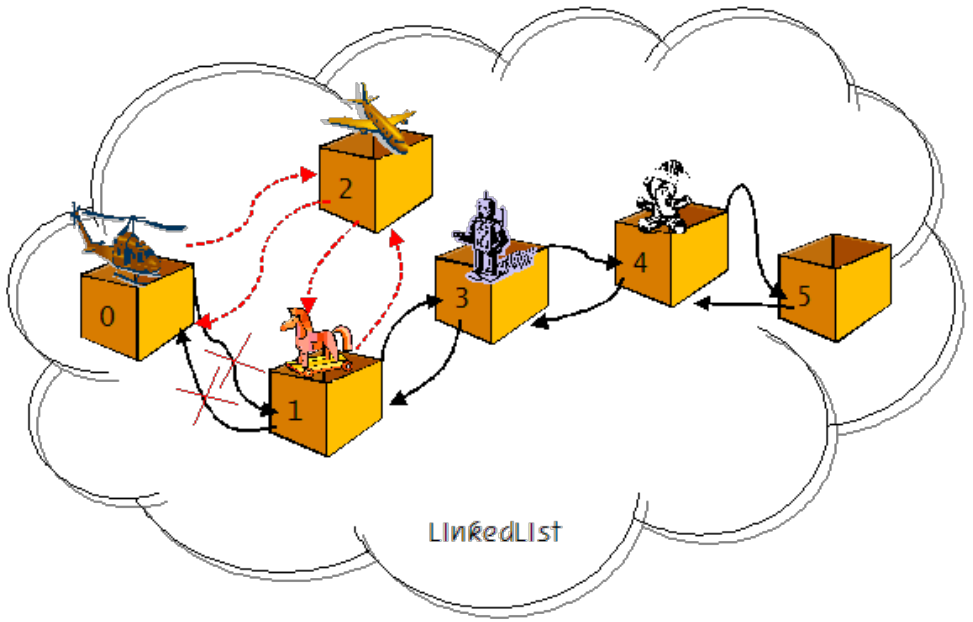


그림 18.8 연결 리스트 중간에 삽입하려면 링크만 수정하면 된다.

```cpp
#include <iostream>
#include <vector>
#include <deque>
#include <list>
#include <string>
using namespace std;
template <class T>
void print_container(const T& container, string s=""){
  cout << s << "(" << container.size() << ")";
  auto it = container.begin();
  for (; it != container.end(); it++) cout << *it << " ";
  cout << endl;
}
```

```
v= (4)0 1 2 3
dq= (4)0 1 2 3
l1= (3)5 5 5
l2= (4)0 1 2 3
l3= (4)0 1 2 3
l3= (5)-1 0 1 2 3
l3= (6)-1 0 1 2 3 100
l3= (5)0 1 2 3 100
l3= (4)0 1 2 3
```

```cpp
int main(){
  int a[10] = {0,1,2,3,4,5,6,7,8,9};
  vector<int> v(&a[0], &a[4]);
  deque<int> dq(v.begin(), v.end());
  list<int> l1(3, 5);
  list<int> l2(&a[0], &a[4]);
  list<int> l3(dq.begin(), dq.end());
  print_container(v, "v= ");
  print_container(dq, "dq= ");
  print_container(l1, "l1= ");
  print_container(l2, "l2= ");
  print_container(l3, "l3= ");
  l3.push_front(-1);
  print_container(l3, "l3= ");
  l3.push_back(100);
  print_container(l3, "l3= ");
  l3.pop_front();
  print_container(l3, "l3= ");
  l3.pop_back();
  print_container(l3, "l3= ");
//  cout << l3[0] << endl;
```

```
33    print_container(l3, "l3= ");
34  //  cout << l3[0] << endl;
35    auto it = l3.insert(l3.begin(),7);
36    print_container(l3, "l3= ");
37    cout << "return value points " << *it << endl;
38    it = l3.insert(l3.begin(),2,-1);
39    print_container(l3, "l3= ");
40    cout << "return value points " << *it << endl;
41  //  it = l3.insert(l3.begin()+1,v.begin()+2, v.begin()+4);
42    it = l3.begin(); it++;
43    it = l3.insert(it,v.begin()+2, v.begin()+4);
44    print_container(l3, "l3= ");
45    cout << "return value points " << *it << endl;
46    it = l3.erase(l3.begin());
47    print_container(l3, "l3= ");
48    cout << "return value points " << *it << endl;
49  //  it = l3.erase(l3.begin()+3, l3.end()-1);
50    it++; it++; it++;
51    auto end_it = l3.end(); end_it--;
52    it = l3.erase(it, end_it);
53    print_container(l3, "l3= ");
54    cout << "return value points " << *it << endl;
```