

2018년도 여름계절학기

한양대학교  
HANYANG UNIVERSITY

# 창의적 소프트웨어 프로그래밍 (Creative Software Design)

## Template & STL

2018.07.02.

담당교수 이 효 섭

- 객체지향 프로그래밍 언어의 가장 큰 장점 중 하나는 코드의 재사용
- 상속 이외에 템플릿을 이용해 코드 재사용 가능
  - 자료형에 상관없는 일반형으로 함수나 클래스 작성
  - 알고리즘만 정의하고 그 알고리즘에 적용될 자료형은 객체를 선언하거나 함수 호출 시 지정 가능
  - 동일한 알고리즘을 다양한 자료형으로 구현할 때 이용
- 템플릿 함수
  - 함수나 클래스를 만드는 작업을 프로그래머 대신 C++ 컴파일러가 제공
  - 객체지향 프로그래밍의 특징 중 하나인 코드의 재사용을 위한 일종의 기술

## ● 필요성

```
❶ int myabs(int num)
{
    if(num<0)
        num=-num;
    return num;
}
```

```
❷ double myabs(double num)
{
    if(num<0)
        num=-num;
    return num;
}
```

```
❸ long int myabs(long int num)
{
    if(num<0)
        num=-num;
    return num;
}
```

- 형식 매개변수의 자료형과 함수의 반환값에 대한 자료형만 다름
- 알고리즘은 동일
- 함수 3개에서 서로 다른 부분인  
자료형만 T로 바꾸기

```
T myabs(T num)
{
    if(num<0)
        num=-num;
    return num;
}
```

# 템플릿 함수

- T가 무엇인지  
컴파일러에게 알려주지 않았기 때문에 **에러 발생**

- 해결방법

- “자료형 이름(typename)인 T는 자료형을 템플릿화하기 위한 기호” 임을  
컴파일러에게 알려주면 프로그램 내에서 T가 사용되어도 컴파일 에러 발생하지  
않음
- template <typename T> 추가

```
T myabs(T num)
{
    if(num<0)
        num=-num;
    return num;
}
```

```
template <typename T>
```

```
T myabs(T num)
```

```
{
```

```
    if(num<0)
```

```
        num=-num;
```

```
    return num;
```

```
}
```

# 템플릿 함수

- 알고리즘에 적용될 자료형은 함수를 호출할 때 결정해주는데, 다음과 같이 실 매개변수를 기술해 주면 실매개변수의 자료형에 의해서 템플릿 결정

```
int a=-10;  
myabs(a);
```

```
template <typename T>  
T myabs(T num)  
{  
    if(num<0)  
        num=-num;  
    return num;  
}
```

## ● 템플릿 함수의 인스턴스화

- myabs 함수를 호출할 때 정수형 변수를 실 매개변수로 전달해 주면 T가 int형으로 결정되어 컴파일러는 템플릿 함수를 기반으로 각 함수 생성

```
/* 함수 호출*/  
int a=-10;  
abs(a);
```

```
/* 함수 호출*/  
double b=-3.4;  
abs(b);
```

```
/* 함수 호출*/  
long int c=-20L;  
abs(c);
```

```
/* 템플릿 함수 */  
template <typename T>  
T myabs(T num)  
{  
    if(num<0)  
        num=-num;  
    return num;  
}
```

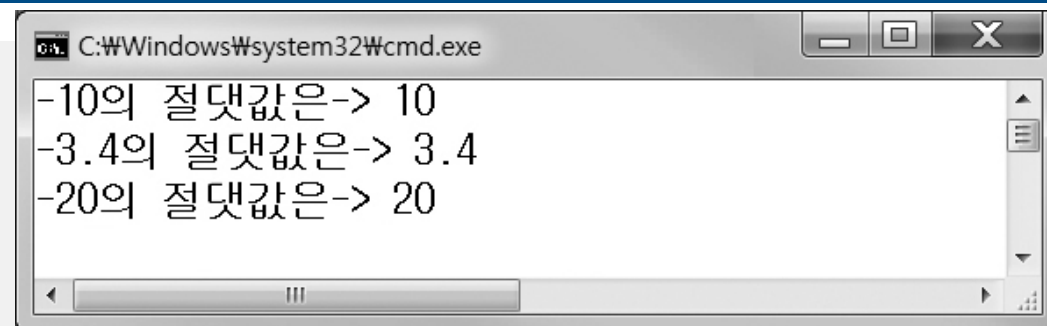
```
int myabs(int num)  
{  
    if(num<0) num=-num;  
    return num;  
}
```

```
double myabs(double num)  
{  
    if(num<0) num=-num;  
    return num;  
}
```

```
long int myabs(long int num)  
{  
    if(num<0) num=-num;  
    return num;  
}
```

# 절댓값을 구하는 함수의 템플릿화

```
01 #include <iostream>
02 using namespace std;
03 template <typename T>
04 T myabs(T num)
05 {
06     if(num<0)
07         num=-num;
08     return num;
09 }
10
11 void main()
12 {
13     int a=-10;
14     cout << a <<"의 절댓값은 -> " << myabs(a) << endl;
15
16     double b=-3.4;
17     cout << b <<"의 절댓값은-> " << myabs(b) << endl;
18
19     long int c=-20L;
20     cout << c <<"의 절댓값은 -> " << myabs(c) << endl;
21 }
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the output of the C++ program, showing the absolute values of -10, -3.4, and -20. The output is as follows:

```
-10의 절댓값은-> 10
-3.4의 절댓값은-> 3.4
-20의 절댓값은-> 20
```

## ● 필요성

- 다양한 자료형이 처리되는 클래스가 필요할 때 여러 번 클래스 정의는 번거로움
- 기본 형식

```
template <typename DATATYPE>
class 클래스명{
    ...
}
```

주의 :  
템플릿 template <typename DATATYPE>  
덧붙이기

- DATATYPE은 식별자로 나중에 특정 자료형으로 변경됨
- 템플릿 클래스를 정의하는 문장 내에서도 변수 선언문 중에서 나중에 자료형이 변경되어야 하는 부분을 DATATYPE으로 지정

```
template <typename DATATYPE>
class Test{
    DATATYPE value;
    . . . . .
}
```



## ● 주의할 점

- 각 멤버함수를 정의할 때마다 `template <typename DATATYPE>` 기술
- 스코프 연산자 앞의 클래스명 다음에도 `<DATATYPE>` 붙이기

```
template <typename DATATYPE>
클래스명<DATATYPE>::멤버함수{
    . . . . .
}
```

## ● 예제

```
template <typename DATATYPE>
void Test<DATATYPE>::SetValue(DATATYPE v)
{
    value = v;
};
```

```
class Test{
    int value;
    void SetValue(int v)
};
void Test::SetValue(int v)
{
    value = v;
};
```

```
Test <int>obj1(10);
Test <double>obj2(5.7);
```

```
Test <int>obj1(10);
```

## ● 예제

```
template <typename DATATYPE>
void Test<DATATYPE>::SetValue(DATATYPE v)
{
    value = v;
};
```

```
class Test{
    double value;
    void SetValue(double v)
};
void Test::SetValue(double v)
{
    value = v;
};
```

```
Test <int>obj1(10);
Test <double>obj2(5.7);
```

```
Test <double>obj2(5.7);
```

# STL

(Standard Template Library)

- C++에서는 유용한 템플릿을 모아놓은 표준 템플릿 라이브러리 제공
- STL
  - 컨테이너(container)와 알고리즘(algorithm)을 일반화한 자료구조 라이브러리
  - 일반화된 라이브러리(generic library)의 사용을 위해 Template(템플릿) 제공
  - 핵심 구성요소
    - 컨테이너 (container)
    - 알고리즘 (algorithm)
    - 반복자 (iterator)

- STL 핵심 구성요소

- 컨테이너(container)
  - 객체들을 담아둘 수 있는 객체. 예) vector
- 반복자(iterator)
  - 컨테이너와 알고리즘 사이에서 연결해 주는 역할
  - 프로그래머가 반복적인 작업을 할 때 사용하는 패턴 중 하나
  - 반복자를 통해 컨테이너 내의 요소들을 다양한 방법으로 다룰 수 있어 꼭 필요
- 알고리즘(algorithm)
  - 자료들을 가공하고 적절히 사용할 수 있는 방법. 예) sort
  - STL에서는 알고리즘을 컨테이너에서 완전히 분리시켜 컨테이너 종류와는 무관하게 어떤 컨테이너에도 적용할 수 있도록 이식성을 높임

## ● STL 컨테이너

- 벡터(vector)는 배열처럼 유사한 값들의 집합으로 임의 접근할 수 있는 자료형

```
vector<int> v2(10, 2);
```

- <int>는 벡터 객체가 갖는 요소의 자료형이 정수형이라는 의미
- vector 객체를 생성 시 생성자는  
첫 번째 매개변수가 벡터의 크기, 두 번째 매개변수는 벡터의 초기값임

- vector 클래스 생성자

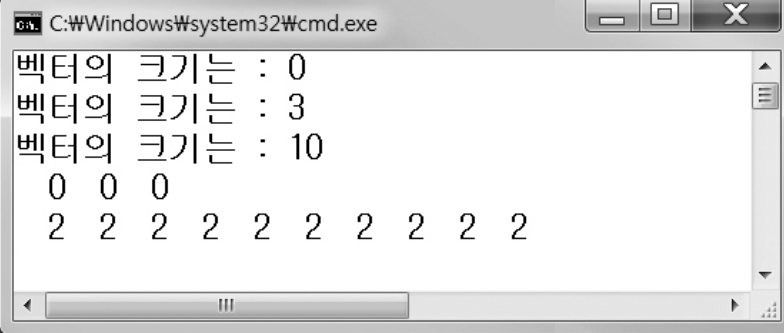
```
vector(); // 빈 벡터(empty vector) 생성  
vector(size_type _Count); // 벡터의 크기만 지정  
vector(size_type _Count, const Type& _Val); // 벡터의 크기와 요소의 초기값 지정
```

- 벡터는 요소의 개수를 알려주는 size라는 멤버함수 제공

```
size_type size() const;
```

# vector 컨테이너 사용하기

```
01 #include <vector>
02 #include <iostream>
03 using namespace std;
04 void main()
05 {
06     vector<int> v0;
07     vector<int> v1(3);
08     vector<int> v2(10, 2);
09
10     cout<<"벡터의 크기는 : " << v0.size()<<endl;
11     cout<<"벡터의 크기는 : " << v1.size()<<endl;
12     cout<<"벡터의 크기는 : " << v2.size()<<endl;
13
14     for(int i=0; i<v1.size(); i++)
15         cout<<" "<<v1[i];
16     cout<<endl;
17
18     for(int i=0; i<v2.size(); i++)
19         cout<<" "<<v2[i];
20     cout<<endl;
21 }
```



```
C:\Windows\system32\cmd.exe
벡터의 크기는 : 0
벡터의 크기는 : 3
벡터의 크기는 : 10
 0 0 0
2 2 2 2 2 2 2 2 2 2
```



## ● 반복자

- 목적 : 컨테이너 종류에 따라 접근 방식이 다른데, 이를 모두 익혀 사용하는 불편함 해소
- 함수 : begin( )과 end( )
- 반복자를 사용하려면 iterator 클래스로 객체 선언 필요

```
vector<int>::iterator iter;
```

- iter가 반복자이며 int형 요소를 저장하고 있는 vector 요소 지시 가능

```
vector<int> v;      // int형 요소를 저장하는 벡터 객체 v 선언  
iter = v.begin();  // 첫 번째 요소 지시  
iter = v.end();    // 마지막 요소를 통과하고 난 다음 요소 지시
```

## ● 반복자

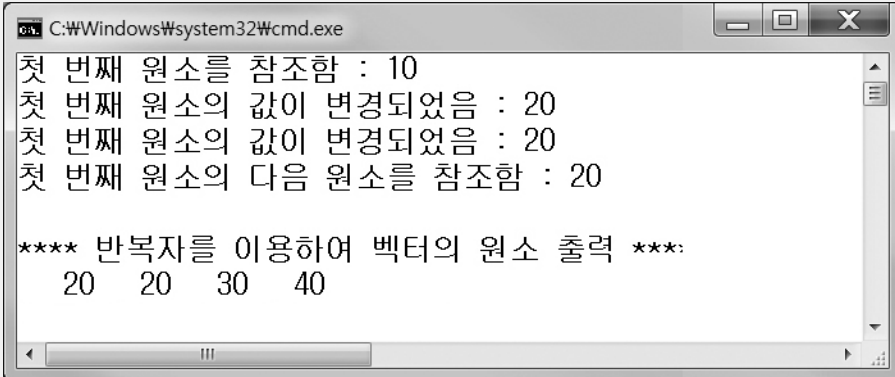
- 반복자는 포인터를 일반화했다고 생각할 수 있음
- 특정 위치를 지시하고 있는 반복자는 요소의 값을 참조하려면 \* 연산자 사용
- 다음 요소를 지시하려고 이동하려면 ++ 연산자 사용

```
vector<int> v;    // int형 요소를 저장하는 벡터 객체 v 선언
iter = v.begin(); // 첫 번째 요소 지시
*iter = 80;       // iter가 참조하는 요소에 특정 값 대입
iter++;           // 다음 요소 지시
```

- push\_back( )
  - STL 컨테이너에서 공통적으로 사용할 수 있는 함수
  - 요소를 벡터 끝에 추가

```
vector<int> v;    // 빈 벡터 생성.
v.push_back( 10 ); // 요소가 추가되어 크기가 1인 벡터가 됨
```

```
01 #include <vector>
02 #include <iostream>
03 using namespace std;
04 void main()
05 {
06     vector<int>::iterator iter;
07     vector<int> v;
08
09     v.push_back( 10 );
10     v.push_back( 20 );
11     v.push_back( 30 );
12     v.push_back( 40 );
13
14     iter = v.begin();
15
16     cout << "첫 번째 요소를 참조함 : " << *iter << endl;
17
18     *iter = 20;
19
20     cout << "첫 번째 요소의 값이 변경되었음 : " << v[0] << endl;
21     cout << "첫 번째 요소의 값이 변경되었음 : " << *iter << endl;
22
23     *iter++;
24     cout << "첫 번째 요소의 다음 요소를 참조함 : " << *iter << endl;
25
26     cout<< "\n**** 반복자를 이용하여 벡터의 요소 출력 ****"<< endl;
27     for(iter=v.begin(); iter != v.end(); iter++)
28         cout<<" " << *iter;
29     cout<<"\n";
30 }
```



```
C:\Windows\system32\cmd.exe
첫 번째 요소를 참조함 : 10
첫 번째 요소의 값이 변경되었음 : 20
첫 번째 요소의 값이 변경되었음 : 20
첫 번째 요소의 다음 요소를 참조함 : 20

**** 반복자를 이용하여 벡터의 요소 출력 ****:
20 20 30 40
```

## ● 알고리즘

- 알고리즘은 자료들을 가공하고 적절하게 사용할 수 있는 방법
- 예) 지정된 범위 내의 요소들을 오름차순으로 정렬하는 `sort( )` 함수

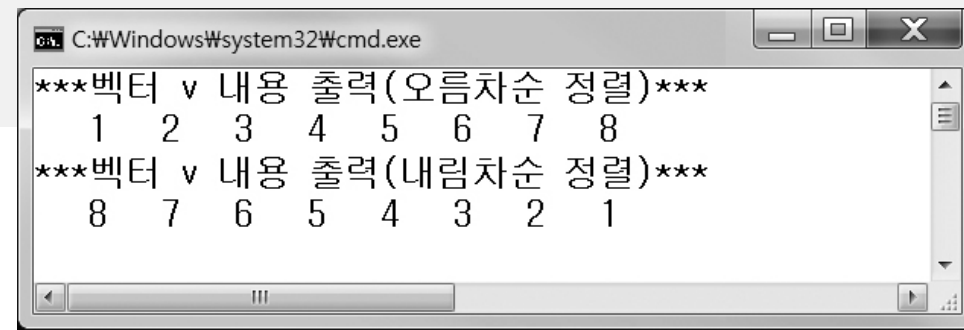
```
sort(start, end);
```

- 내림차순으로 정렬 시 `greater<int>( )` 함수를 `sort( )` 함수 마지막 매개변수에 기술
- 주의사항 : `greater<int>( )` 함수를 사용하려면 `#include <functional>` 을 기술

# sort() 함수 사용하기

```
01 #include <vector>
02 #include <algorithm>
03 #include <functional> // For greater<int>()
04 #include <iostream>
05 using namespace std;
06
07 void main()
08 {
09     vector<int> v(8);
10
11     vector<int>::iterator start, end, iter;
12
13     for (int i= 0; i < v.size(); i++)
14         v[i] = i + 1 ;
15
16     start = v.begin();
17     end = v.end();
18
19     sort(start, end);
```

```
20
21     cout << "***벡터 v 내용 출력
        (오름차순 정렬)***" << endl ;
22     for(iter=v.begin(); iter!=v.end(); iter++)
23         cout<<" "<< *iter;
24
25     cout<<"\n";
26     sort(start, end, greater<int>() );
27
28     cout << "***벡터 v 내용 출력
        (내림차순 정렬)***" << endl ;
29     for( iter=v.begin(); iter!=v.end(); iter++)
30         cout<<" "<< *iter;
31     cout<<"\n";
32 }
```



```
C:\Windows\system32\cmd.exe
***벡터 v 내용 출력(오름차순 정렬)***
1 2 3 4 5 6 7 8
***벡터 v 내용 출력(내림차순 정렬)***
8 7 6 5 4 3 2 1
```

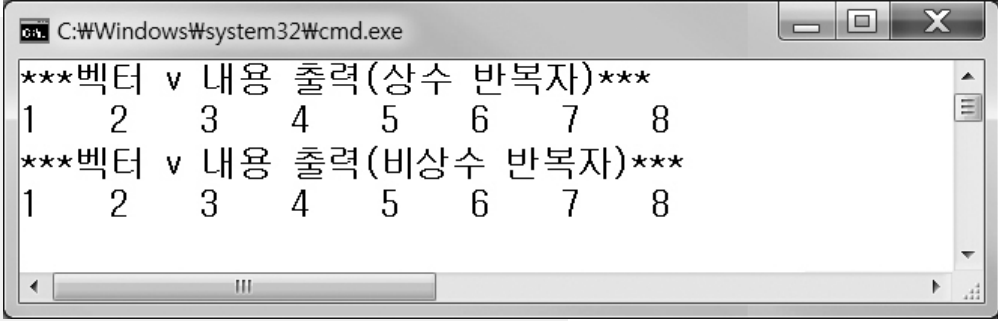
## ● 동작 방식에 따른 분류

반복자	설명
입력 반복자 (input iterator)	반복자가 가리키는 위치의 데이터를 읽기만 가능한 반복자다. 다음 위치로 이동할 수 있다.
출력 반복자 (output iterator)	반복자가 가리키는 위치의 데이터를 쓰기만 가능한 반복자다. 다음 위치로 이동할 수 있다.
순방향 반복자 (forward iterator)	입력 반복자 + 출력 반복자다. +가 가리키는 위치의 데이터를 여러번 읽거나 쓸 수 있고 현재의 위치를 저장해서 동일한 위치 에서 순회를 다시 시작할 수 있다.
양방향 반복자 (bidirectional iterator)	순방향 반복자 + 반대 방향으로 이동 가능한 반복자다.
임의 접근 반복자 (random access iterator)	양방향 반복자 + 임의의 위치로 상수 시간에 이동할 수 있도록 산술연산(+, -)이 가능하다.

- **비상수 반복자와 상수 반복자**
  - **값 변경 여부에 따른 분류**
    - 비상수 반복자(mutable)/ 상수 반복자(constant)
  - 비상수 반복자는 operator \* 연산자에 의해서 참조값을 반환하고, 상수 반복자는 상수 참조값을 결과값으로 반환
  - 상수 반복자는 컨테이너의 요소를 변경할 수 있기 때문에 상수 컨테이너의 요소는 상수 반복자로만 접근 가능

# 비상수 반복자와 상수 반복자 사용하기

```
01 #include <vector>
02 #include <algorithm>
03 #include <iostream>
04 using namespace std;
05
06 void main()
07 {
08     vector<int> v(8) ;
09
10     for (int i = 0; i < v.size(); i++)
11         v[i] = i + 1 ;
12
13     vector<int>::iterator iter; // 비상수 반복자
14     vector<int>::const_iterator citer; // 상수 반복자
15
16     cout << "***벡터 v 내용 출력(비상수 반복자)***" << endl ;
17     for(iter = v.begin(); iter != v.end(); iter++)
18         cout << *iter << " ";
19     cout << endl;
20
21     cout << "***벡터 v 내용 출력(상수 반복자)***" << endl ;
22     for(citer = v.begin(); citer != v.end(); citer++)
23         cout << *citer << " ";
24     cout << endl;
25 }
```



```
C:\Windows\system32\cmd.exe
***벡터 v 내용 출력(상수 반복자)***
1 2 3 4 5 6 7 8
***벡터 v 내용 출력(비상수 반복자)***
1 2 3 4 5 6 7 8
```



# 비상수 반복자와 상수 반복자의 차이점

```
01 #include <vector>
02 #include <algorithm>
03 #include <iostream>
04 using namespace std;
05
06 void main()
07 {
08     vector<int> v(8) ;
09
10     for (int i = 0; i < v.size(); i++)
11         v[i] = i + 1 ;
12
13     vector<int>::iterator iter; // 비상수 반복자
14     vector<int>::const_iterator citer; // 상수 반복자
15
16     iter = v.begin();
17     *iter = 100; // 가능!
18     citer = v.begin();
19     *citer = 100; // 에러 발생!
20 }
```

오류 목록 - 문서 열기



2(오류 2개)

1(경고 1개)

0(메시지 0개)

설명



1 warning C4018: '<' : signed 또는 unsigned가 일치하지 않습니다.



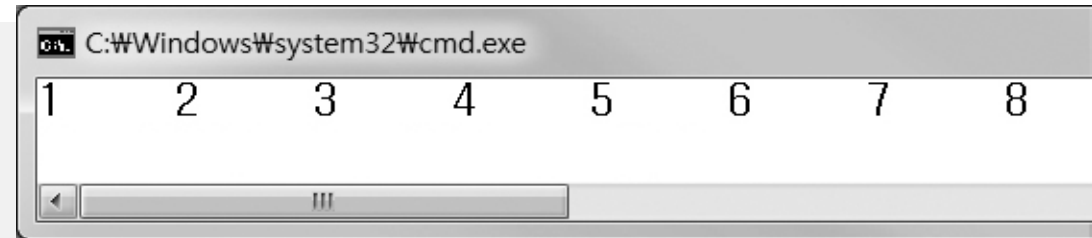
2 error C3892: 'citer' : const인 변수에 할당할 수 없습니다.



3 IntelliSense: 식이 수정할 수 있는 lvalue여야 합니다.

# 상수 컨테이너와 상수 반복자 사용하기

```
01 #include <iostream>
02 #include <vector>
03 #include <algorithm>
04 using namespace std;
05 void PrintVector(const vector<int> & v)
06 {
07     // vector<int>::iterator iter = v.begin(); // 컴파일 에러
08     vector<int>::const_iterator citer = v.begin();
09
10     for( ; citer != v.end(); citer++)
11         cout << *citer << " ";
12     cout << endl;
13 }
14 void main()
15 {
16     vector<int> v(8) ;
17
18     for (int i = 0; i < v.size(); i++)
19         v[i] = i + 1 ;
20
21     PrintVector( v );
22 }
```

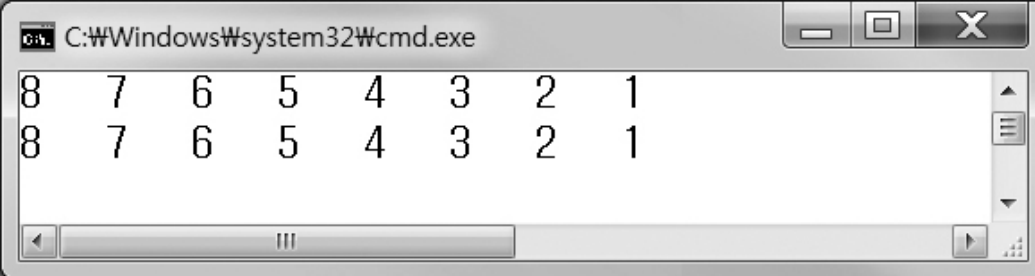


## ● 역방향 반복자

- STL 컨테이너는 사용 편의성과 효율성을 위해 역방향 반복자(reverse\_iterator) 제공
- 순방향 반복자(iterator)와 반대로 동작
- 역방향 반복자의 상수 역방향 반복자(const\_reverse\_iterator) 제공

# 역방향 반복자와 상수 역방향 반복자 사용하기

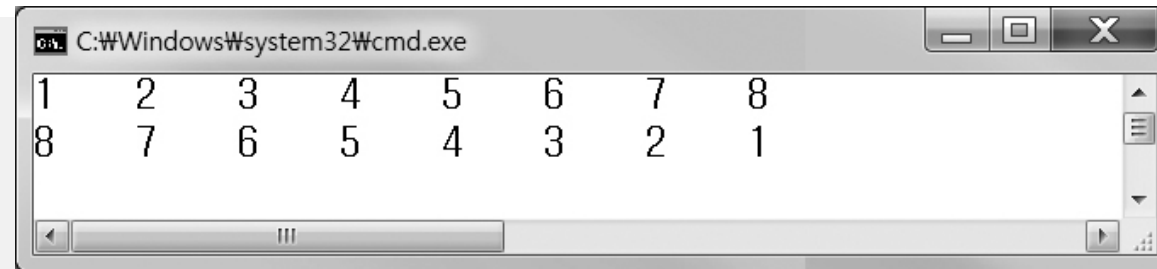
```
01 #include <vector>
02 #include <algorithm>
03 #include <iostream>
04 using namespace std;
05
06 void main()
07 {
08     vector<int> v(8);
09
10     for (int i = 0; i < v.size(); i++)
11         v[i] = i + 1;
12
13     vector<int>::reverse_iterator riter; // 역방향 반복자
14     vector<int>::const_reverse_iterator criter; // 상수 역방향 반복자
15
16     for(riter = v.rbegin(); riter != v.rend(); riter++)
17         cout << *riter << " ";
18     cout << endl;
19
20     for(criter = v.rbegin(); criter != v.rend(); criter++)
21         cout << *criter << " ";
22     cout << endl;
23 }
```



```
C:\Windows\system32\cmd.exe
8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
```

# 역방향 반복자 사용하기

```
01 #include <vector>
02 #include <algorithm>
03 #include <iostream>
04 using namespace std;
05
06 void main()
07 {
08     vector<int> v(8) ;
09
10     for (int i = 0; i < v.size(); i++)
11         v[i] = i + 1 ;
12
13     vector<int>::iterator iter;
14     vector<int>::reverse_iterator riter;
15
16     for(iter = v.begin(); iter != v.end(); iter++)
17         cout << *iter << " ";
18     cout << endl;
19
20     for(riter = v.rbegin(); riter != v.rend(); riter++)
21         cout << *riter << " ";
22     cout << endl;
23 }
```



```
C:\Windows\system32\cmd.exe
1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1
```

- STL의 컨테이너는 형식이 같은 객체의 집합을 저장하고 관리
- STL 컨테이너의 3가지 분류
  - 시퀀스 컨테이너(sequence container)
    - 자료의 선형적인 집합. 자료를 저장하는 기본 임무에 충실한 가장 일반적인 컨테이너
    - 삽입된 자료를 무조건 저장하며 입력되는 자료에 특별한 제약/관리 규칙 없음
    - 사용자는 시퀀스의 임의 위치에 원하는 요소를 마음대로 삽입, 삭제가능
    - STL제공 : 벡터(vector), 리스트(list), 덱(deque)
  - 연관 컨테이너(associative container)
    - 자료 저장과 일정한 규칙에 따라 자료 조직화 및 관리
    - 정렬이나 해시 등의 방법을 통해 삽입되는 자료를 항상 일정한 기준(오름차순, 해시 함수)에 맞는 위치에 저장. 검색 속도가 빠른 장점
    - STL제공 : 정렬 연관 컨테이너인 셋(set), 맵(map) 등

- STL의 컨테이너는 형식이 같은 객체의 집합을 저장하고 관리
- STL 컨테이너의 3가지 분류
  - 어댑터 컨테이너(adapter container)
    - 시퀀스 컨테이너를 변형해서 자료를 미리 정해진 일정한 방식에 따라 관리
    - STL제공 : 스택(stack), 큐(queue), 우선순위 큐(priority\_queue)
      - 스택은 항상 LIFO의 원리로 동작/ 큐는 항상 FIFO의 원리로 동작
      - 자료를 넣고 빼는 순서를 외부에서 마음대로 조작할 수 없음
      - 컨테이너의 규칙대로 조작해야 함

## ● list

- 리스트는 템플릿 클래스이므로 객체를 생성할 때 요소로 저장할 자료형 언급

```
list<int> li;
```

- 리스트 함수

멤버 함수	설명
push_front	제일 앞에 요소 추가
push_back	제일 뒤에 요소 추가
pop_front	제일 앞의 요소 삭제
pop_back	제일 뒤의 요소 삭제

- 리스트와 벡터 비교

- 리스트는 벡터와 같이 동일한 자료의 집합을 관리
- 벡터가 읽고 쓰기에 강한 컨테이너라면 리스트는 삽입, 삭제에 강한 컨테이너



## ●map

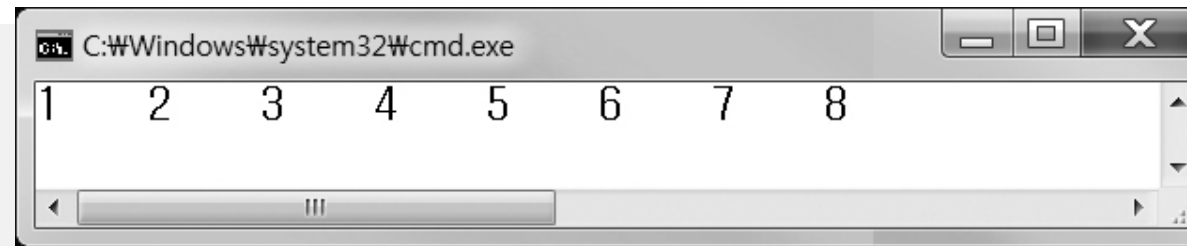
- 키(key), 값(value)의 쌍으로 데이터를 저장
- 키를 이용해서 값을 찾을 때 유용하게 사용
- 키로 원하는 항목을 검색하기 때문에 검색 속도가 빠름

## ●set

- set은 중복되지 않는 요소들을 관리
- 동일한 요소의 중복을 허용하지 않아 이미 저장된 객체를 또 다시 저장하면 추가되지 않음

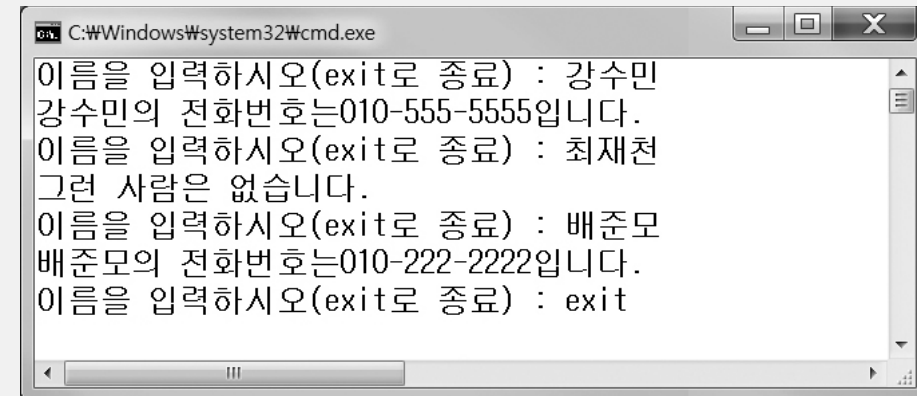
# List 컨테이너 사용하기

```
01 #include <list>
02 #include <iostream>
03 using namespace std;
04
05 void main()
06 {
07     list<int> li;
08
09     for (int i = 0; i < 8; i++)
10         li.push_back(i + 1);
11
12     list<int>::iterator iter;
13
14     for(iter = li.begin(); iter != li.end(); iter++)
15         cout << *iter << " ";
16     cout << endl;
17 }
```



# Map 컨테이너 사용하기

```
01 #include <map>
02 #include <string>
03 #include <iostream>
04 using namespace std;
05
06 void main()
07 {
08     map<string, string> PhoneBooks;
09     map<string, string>::iterator iter;
10
11     string Name;
12
13     PhoneBooks["김선호"]="010-1111-1111";
14     PhoneBooks["배준모"]="010-2222-2222";
15     PhoneBooks["송기수"]="010-3333-3333";
16     PhoneBooks["강수민"]="010-5555-5555";
17
18     for (;;) {
19         cout << "이름을 입력하시오(exit로 종료) : ";
20         cin >> Name;
21         if (Name=="exit")
22             break;
23         iter=PhoneBooks.find(Name);
24         if (iter == PhoneBooks.end()) {
25             cout << "그런 사람은 없습니다.\n";
26         } else {
27             cout<<Name<<"의 전화번호는“
28                 <<iter->second<<"이다.\n";
29         }
30     }
```



```
C:\Windows\system32\cmd.exe
이름을 입력하시오(exit로 종료) : 강수민
강수민의 전화번호는010-555-5555입니다.
이름을 입력하시오(exit로 종료) : 최재천
그런 사람은 없습니다.
이름을 입력하시오(exit로 종료) : 배준모
배준모의 전화번호는010-222-2222입니다.
이름을 입력하시오(exit로 종료) : exit
```

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

void main()
{
    set<int> intSet;
    set<int>::iterator IterPos;

    intSet.insert(90); intSet.insert(95);
    intSet.insert(100); intSet.insert(60);
    intSet.insert(60); intSet.insert(95);
    intSet.insert(85);

    cout <<"총 개수 : "<<intSet.size() <<endl;

    for(IterPos = intSet.begin(); IterPos != intSet.end(); ++IterPos){
        cout << *IterPos <<" ";
    }
    cout << endl;
```

# Set 컨테이너 사용하기

```
set<int>::iterator EraseIter = intSet.find(70);
```

```
if (EraseIter != intSet.end()) {  
    intSet.erase(EraseIter);  
}
```

```
cout << " >> 70을 삭제 후 모든 요소 출력 << " << endl;
```

```
for(IterPos = intSet.begin(); IterPos != intSet.end(); ++IterPos){  
    cout << *IterPos << " ";  
}
```

```
cout << endl;
```

```
cout << " >> 95를 검색 하여 70로 변경한 후 모든 요소 출력 << " << endl;
```

```
set<int>::iterator FindIter = intSet.find(95);
```

```
if(FindIter != intSet.end())  
{
```

```
    /*FindIter = 100;
```

```
    intSet.erase(FindIter);
```

```
    intSet.insert(70);
```

```
}
```

```
for(IterPos = intSet.begin(); IterPos != intSet.end(); ++IterPos){
```

```
    cout << *IterPos << " ";
```

```
}
```

```
cout << endl;
```

```
}
```

```
C:\Windows\system32\cmd.exe  
총 개수 : 5  
60 85 90 95 100  
>> 70을 삭제 후 모든 요소 출력 <<  
60 85 90 95 100  
>> 95를 검색 하여 70로 변경한 후 모든 요소 출력 <<  
60 70 85 90 100
```

- STL 알고리즘은 문제 해결을 위해서 제공되는 STL 함수
- 분류
  - 변경 불가 시퀀스 알고리즘(nonmutating sequence algorithm)
  - 변경 가능 시퀀스 알고리즘(mutaing sequence algorithm)
  - 정렬 관련 알고리즘(sorting-related algorithm)
  - 범용 수치 알고리즘(generalized numeric algorithm)

## ● 변경 불가 시퀀스 알고리즘(nonmutating sequence algorithm)

함수	설명
for_each	주어진 함수를 모든 원소에 적용한다.
find	선형 검색한다.
find_first_of	주어진 집합에 속하는 값들을 찾아낸다.
adjacent_find	동일한 원소가 서로 이웃한 부분을 찾아낸다.
count	주어진 값의 개수를 계산한다.
mismatch	두 시퀀스를 스캔해서 달라지는 원소의 위치를 찾아낸다.
equal	두 시퀀스를 스캔해서 모든 원소가 같은지를 검사한다.
search	원하는 시퀀스를 찾아낸다.
search_n	시퀀스에 주어진 값과 동일한 원소가 연속적으로 이어진 시퀀스를 찾아낸다.
find_end	주어진 시퀀스와 일치하는 맨 마지막 부분을 찾아낸다.

- for\_each

- 컨테이너 클래스에서 공통적으로 사용할 수 있는 함수

```
vector<int> v(8) ;  
for_each(v.begin(), v.end(), PrintElement);
```

- 매개변수 3개 사용

- 첫 번째와 두 번째 매개변수에 의해서 컨테이너의 범위가 지정
- 마지막 매개변수로 지정된 함수가 지정된 범위 내의 모든 요소에 의해서 적용



- find, find\_if

- find

- 주어진 구간 내 주어진 값과 동일 값을 찾아 위치를 반복자 형태로 알림
    - 검색에 실패하면 end 반환

- find\_if

- 마지막 매개변수로 조건자 함수를 넘겨주고 반복자들의 구간 내에서 함수에 제시한 조건을 만족하는 요소 찾기
    - 구간 내 요소에 대해 조건자 함수를 적용해서 결과값으로 true를 반환되는 위치를 반복자 형태 알림
    - 조건에 만족하는 요소를 찾지 못하면 end 반환

- count, count\_if

- count : 주어진 구간 내에서 주어진 값과 동일한 값을 찾는 요소의 개수 알리기
  - count\_if : 마지막 매개변수로 조건자 함수를 넘겨주기

# for\_each() 사용하기

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

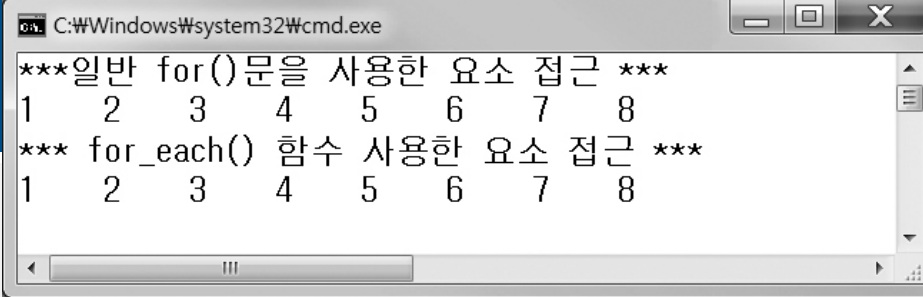
void PrintElement(int n) // 요소의 값을 출력하는 함수 정의
{
    cout << n << "    ";
}

void main()
{
    vector<int> v(8) ;

    for (int i = 0; i < v.size(); i++)
        v[i] = i + 1 ;

    cout << "***일반 for()문을 사용한 요소 접근 ***" << endl ;
    for( int i = 0 ; i < v.size(); i++)
        cout << v[i] << "    ";
    cout << endl;

    cout << "*** for_each() 함수 사용한 요소 접근 ***" << endl ;
    for_each(v.begin(), v.end(), PrintElement) ;
    cout << endl ;
}
```



```
C:\Windows\system32\cmd.exe
***일반 for()문을 사용한 요소 접근 ***
1 2 3 4 5 6 7 8
*** for_each() 함수 사용한 요소 접근 ***
1 2 3 4 5 6 7 8
```

# find() 사용하기

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

void main()
{
    vector<int> v(8) ;

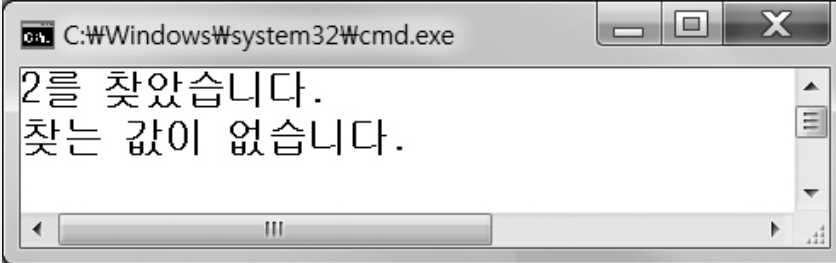
    for (int i = 0; i < v.size(); i++)
        v[i] = i + 1 ;

    vector<int>::iterator iter;
    iter = find(v.begin(), v.end(), 2);

    if( iter != v.end() )
        cout << *iter << "를 찾았습니다." << endl;
    else
        cout << "찾는 값이 없습니다." << endl;

    iter = find(v.begin(), v.end(), 20);

    if( iter != v.end() )
        cout << *iter << "를 찾았습니다." << endl;
    else
        cout << "찾는 값이 없습니다." << endl;
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays two lines of Korean text: "2를 찾았습니다." (Found 2) and "찾는 값이 없습니다." (The value being searched for does not exist). The text is displayed in a monospaced font, and the window has standard Windows window controls (minimize, maximize, close) in the title bar.

# find\_if() 사용하기

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

bool Greater5(int n)
{
    return n > 5;
}

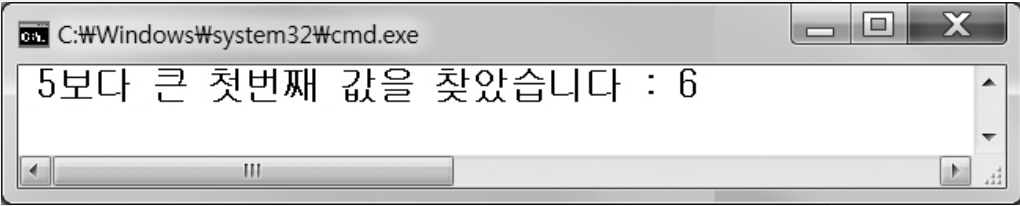
void main()
{
    vector<int> v(8) ;

    for (int i = 0; i < v.size(); i++)
        v[i] = i + 1 ;

    vector<int>::iterator iter;

    iter = find_if(v.begin(), v.end(), Greater5);

    if( iter != v.end() )
        cout << " 5보다 큰 첫번째 값을 찾았습니다 : " << *iter << endl;
    else
        cout << " 찾는 값이 없습니다." << endl;
}
```



C:\Windows\system32\cmd.exe

5보다 큰 첫번째 값을 찾았습니다 : 6

# count(), count\_if() 사용하기

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

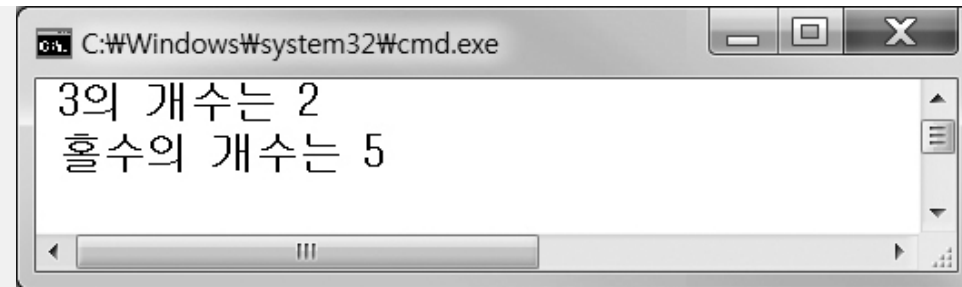
bool is_odd(int num)
{
    return num % 2 == 1 ;
}

void main()
{
    vector<int>  v(8);

    v.push_back(3);    v.push_back(2);
    v.push_back(7);    v.push_back(9);
    v.push_back(4);    v.push_back(1);    v.push_back(3);

    int total;
    total  = count(v.begin(), v.end(), 3);
    cout << " 3의 개수는 " << total << endl;

    total  = count_if(v.begin(), v.end(), is_odd);
    cout << " 홀수의 개수는 " << total << endl;
}
```



C:\Windows\system32\cmd.exe

```
3의 개수는 2
홀수의 개수는 5
```

- equal
  - 두 구간 내의 요소들이 일치하는지 조사
- search
  - find는 일정 구간 내에서 특정한 값 하나에 대해서 일치되는 위치를 찾기
  - search는 연속적인 값들을 탐색할 경우 사용.  
전체 구간 안( $v1$ )에 부분 구간( $v2$ )을 찾아 그 위치를 반환

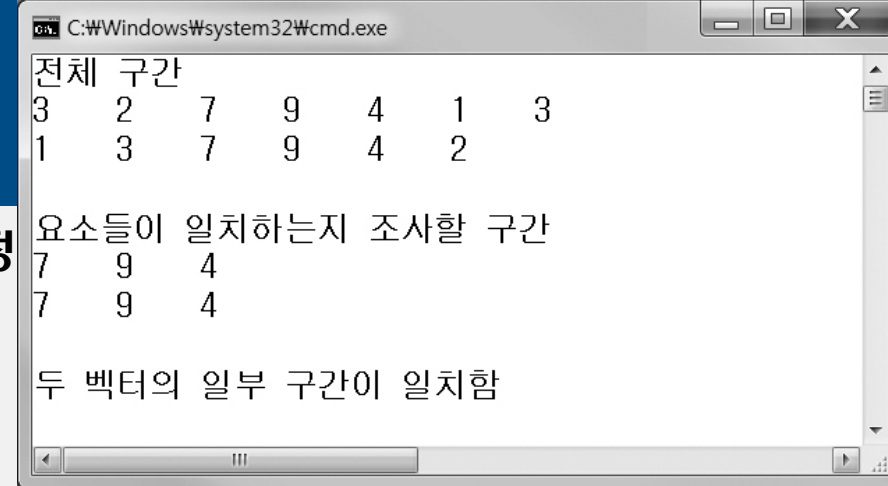
# equal() 사용하기

```
void PrintElement(int n) // 요소의 값을 출력하는 함수 정
{
    cout << n << "    ";
}
```

```
void main()
{
    vector<int> v1;
    v1.push_back(3);      v1.push_back(2);      v1.push_back(7);
    v1.push_back(9);      v1.push_back(1);      v1.push_back(3);
    v1.push_back(4);

    vector<int> v2;
    v2.push_back(1);      v2.push_back(3);      v2.push_back(7);
    v2.push_back(9);      v2.push_back(4);      v2.push_back(2);

    cout << "전체 구간" << endl;
    for_each(v1.begin(), v1.end(), PrintElement) ;      cout<<endl;
    for_each(v2.begin(), v2.end(), PrintElement) ;      cout<<endl;
    cout<<endl;
    cout << "요소들이 일치하는지 조사할 구간" << endl;
    for_each(v1.begin()+2, v1.begin()+5, PrintElement) ; cout<<endl;
    for_each(v2.begin()+2, v2.begin()+5, PrintElement) ; cout<<endl;
    cout<<endl;
    if( equal(v1.begin()+2, v1.begin()+5, v2.begin()+2) == true)
        cout << "두 벡터의 일부 구간이 일치함" << endl;
    else
        cout << "두 벡터의 일부 구간이 일치하지 않음" << endl;
}
```



```
C:\Windows\system32\cmd.exe
전체 구간
3 2 7 9 4 1 3
1 3 7 9 4 2
요소들이 일치하는지 조사할 구간
7 9 4
7 9 4
두 벡터의 일부 구간이 일치함
```

# search() 사용하기

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

void main( )
{
    vector<int> v1;
    vector<int> v2;

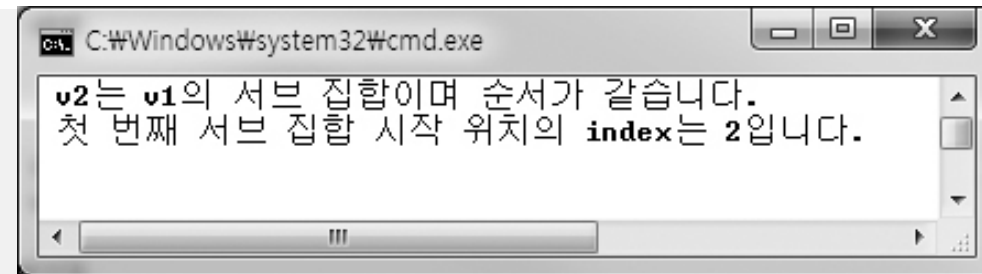
    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);
    v1.push_back(4);
    v1.push_back(5);

    v2.push_back(3);
    v2.push_back(4);

    vector<int>::iterator iter ;

    iter = search(v1.begin(), v1.end(), v2.begin(), v2.end());

    if( iter != v1.end() ) {
        cout << " v2는 v1의 서브 집합이며 순서가 같습니다." << endl;
        cout << " 첫 번째 서브 집합 시작 위치의 index는 "
              << iter-v1.begin() << "입니다." << endl;
    }
}
```





## ● 변경 가능 시퀀스 알고리즘(mutating sequence algorithm)

함수	설명
copy	원소들을 복사한다.
swap	시퀀스의 원소를 맞바꾼다.
transform	주어진 함수를 원소에 적용하고 얻은 리턴 값으로 원소를 교체한다.
replace	주어진 값과 동일한 원소들을 다른 값으로 교체한다.
fill	원소들을 주어진 값으로 교체한다.
generate	원소들을 함수 호출을 통해 얻은 리턴 값으로 교체한다.
remove	주어진 값과 동일한 원소들을 제거한다.
unique	연속적으로 동일한 원소들을 제거한다.
reverse	원소들을 뒤집는다.
rotate	원소들을 회전한다.
random_shuffle	원소들의 순서를 랜덤시킨다.
partition	조건을 만족하는 원소들을 앞 쪽으로 재배치한다.

- **copy : 한 컨테이너에서 다른 컨테이너로 데이터 복사**

```
int arr[10]={3, 5, 1, 4, 8, 7, 0, 9, 2, 6};    // 배열 선언
vector<int> vec(10);                          // 정수형 요소 10개를 갖는 벡터 객체 선언
copy(arr, arr+10, v.begin());                // 배열을 벡터로 복사
```

- copy는 정보를 출력할 목적으로 출력스트림을 대상으로 복사
- STL은 출력 스트림을 나타내는 반복자로 ostream\_iterator 제공

```
#include <iterator>
ostream_iterator<int> out_iter(cout, " : ");
```

- copy에 출력 반복자를 사용해 컨테이너의 내용을 출력

```
copy(v.begin(), v.end(), out_iter);
```

- 만일 역순으로 출력하려면 역방향 반복자를 사용
- rbegin은 맨 마지막 요소 다음을 지시하고, rend는 첫 번째 요소를 지시

```
copy(v.rbegin(), v.rend(), out_iter);
```

# copy() 사용하기

```
#include <iterator> // ostream_iterator
#include <vector>
#include <iostream>
using namespace std;

void main()
{
    int arr[10]={3, 5, 1, 4, 8, 7, 0, 9, 2, 6};

    vector<int> v(10);

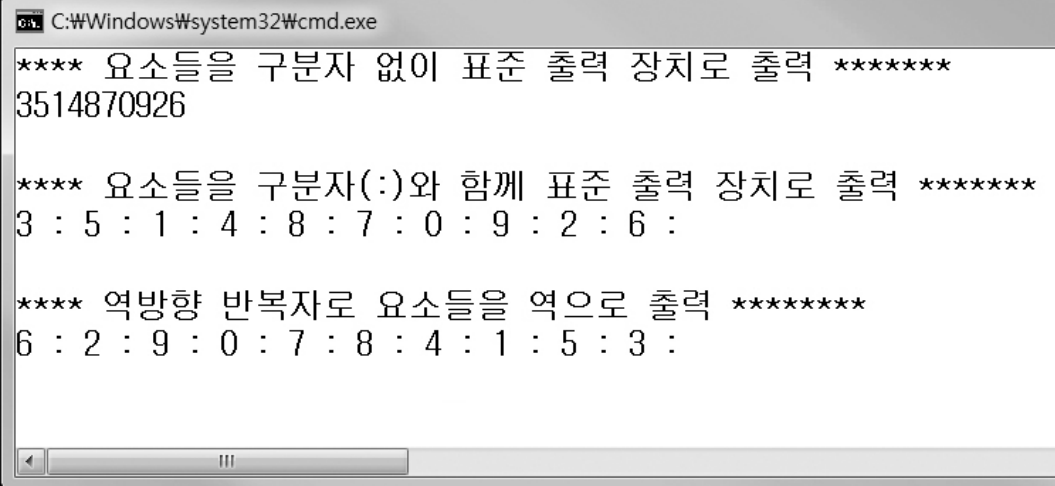
    copy(arr, arr+10, v.begin());

    cout<< " ** 요소들을 구분자 없이 표준 출력 장치로 출력 **"<<endl;
    copy(v.begin(), v.end(), ostream_iterator<int> (cout));
    cout<<"\n\n";

    cout<<" ** 요소들을 구분자(:)와 함께 표준 출력 장치로 출력 **"<<endl;

    ostream_iterator<int> out_iter( cout, " : " );
    copy(v.begin(), v.end(), out_iter);
    cout<<"\n\n";

    cout << "***** 역방향 반복자로 요소들을 역으로 출력 ***** "<< endl;
    copy(v.rbegin(), v.rend(), out_iter);
    cout<<"\n\n";
}
```



```
C:\Windows\system32\cmd.exe
**** 요소들을 구분자 없이 표준 출력 장치로 출력 ****
3514870926

**** 요소들을 구분자(:)와 함께 표준 출력 장치로 출력 ****
3 : 5 : 1 : 4 : 8 : 7 : 0 : 9 : 2 : 6 :

**** 역방향 반복자로 요소들을 역으로 출력 ****
6 : 2 : 9 : 0 : 7 : 8 : 4 : 1 : 5 : 3 :
```

- transform
  - 주어진 구간 내의 요소들을 매개변수로 전달한 함수를 적용해서 변경한다.
- replace
  - 주어진 구간 내의 요소들의 값을 매개변수로 전달한 값으로 대체한다.
- fill
  - 주어진 구간 내의 요소들을 값을 매개변수로 전달한 값으로 할당한다.
- reverse
  - 주어진 구간 내의 요소들을 역순으로 배치한다.
- random\_shuffle
  - 주어진 구간 내의 요소들의 위치를 무작위로 재배치한다.

# transform() 사용하기

```
#include <iterator> // ostream_iterator
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;

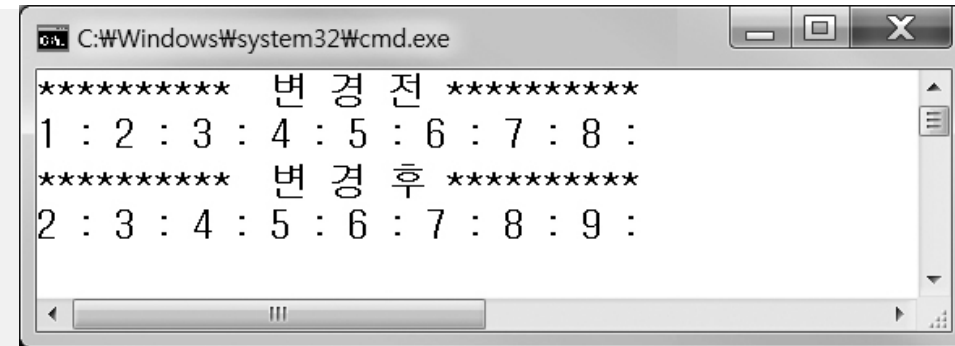
int Increment(int n)
{
    return n+1;
}

void main()
{
    vector<int> v(8) ;
    for (int i = 0; i < v.size(); i++)
        v[i] = i + 1 ;

    ostream_iterator<int> out_iter( cout, " : " );
    cout << "***** 변경 전 *****" << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;

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

    cout << "***** 변경 후 *****" << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;
}
```



```
C:\Windows\system32\cmd.exe
***** 변경 전 *****
1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 :
***** 변경 후 *****
2 : 3 : 4 : 5 : 6 : 7 : 8 : 9 :
```

# replace() 사용하기

```
#include <iterator> // ostream_iterator
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
```

```
void main()
{
    int arr[10]={3, 5, 1, 4, 3, 7, 3, 9, 2, 6};

    vector<int> v(10);

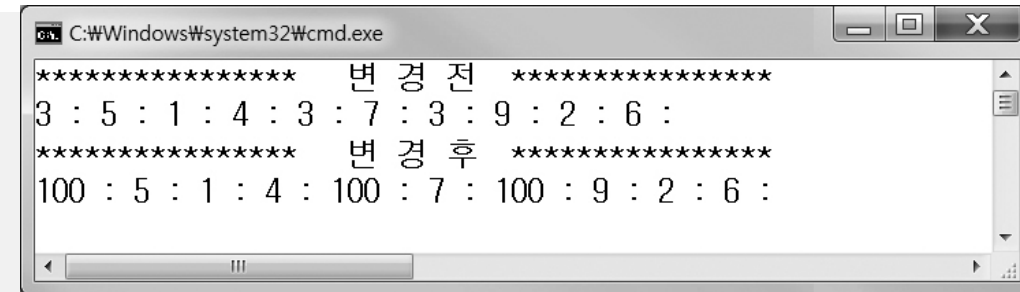
    copy(arr, arr+10, v.begin()); //배열을 벡터로 복사

    ostream_iterator<int> out_iter( cout, " : " );

    cout << "***** 변경 전 *****" << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;

    replace(v.begin(), v.end(), 3, 100);

    cout << "***** 변경 후 *****" << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;
}
```



```
C:\Windows\system32\cmd.exe
***** 변경 전 *****
3 : 5 : 1 : 4 : 3 : 7 : 3 : 9 : 2 : 6 :
***** 변경 후 *****
100 : 5 : 1 : 4 : 100 : 7 : 100 : 9 : 2 : 6 :
```

# fill() 사용하기

```
#include <iterator> // ostream_iterator
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
```

```
void main()
{
    int arr[10]={3, 5, 1, 4, 3, 7, 3, 9, 2, 6};

    vector<int> v(10);

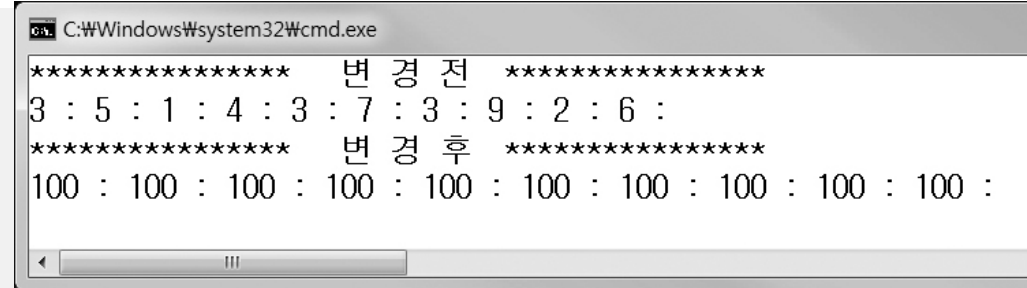
    copy(arr, arr+10, v.begin()); //배열을 벡터로 복사

    ostream_iterator<int> out_iter( cout, " : " );

    cout << "***** 변경 전 *****" << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;

    fill(v.begin(), v.end(), 100);

    cout << "***** 변경 후 *****" << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;
}
```



```
C:\Windows\system32\cmd.exe
***** 변경 전 *****
3 : 5 : 1 : 4 : 3 : 7 : 3 : 9 : 2 : 6 :
***** 변경 후 *****
100 : 100 : 100 : 100 : 100 : 100 : 100 : 100 : 100 : 100 :
```

# reverse() 사용하기

```
#include <iterator> // ostream_iterator
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
```

```
void main()
{
    vector<int>  v(8);    23

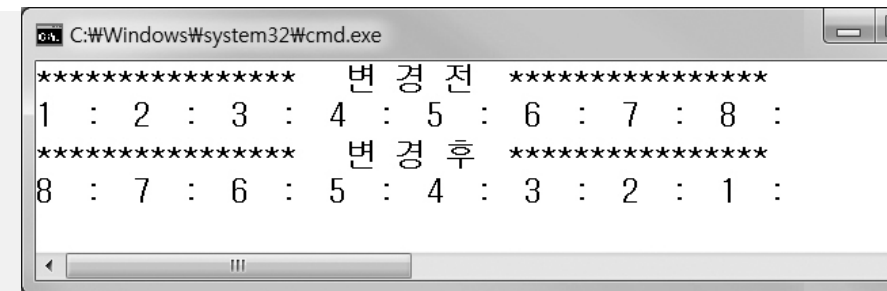
    for (int i= 0; i < v.size(); i++)
        v[i] = i + 1 ;

    ostream_iterator<int>  out_iter( cout, " : " );

    cout << "*****   변   경   전   ***** " << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;

    reverse( v.begin(), v.end() );

    cout << "*****   변   경   후   ***** " << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;
}
```



```
C:\Windows\system32\cmd.exe
*****   변   경   전   *****
1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 :
*****   변   경   후   *****
8 : 7 : 6 : 5 : 4 : 3 : 2 : 1 :
```



# random\_shuffle() 사용하기

```
#include <iterator> // ostream_iterator
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
```

```
void main()
{
    vector<int>  v(8);

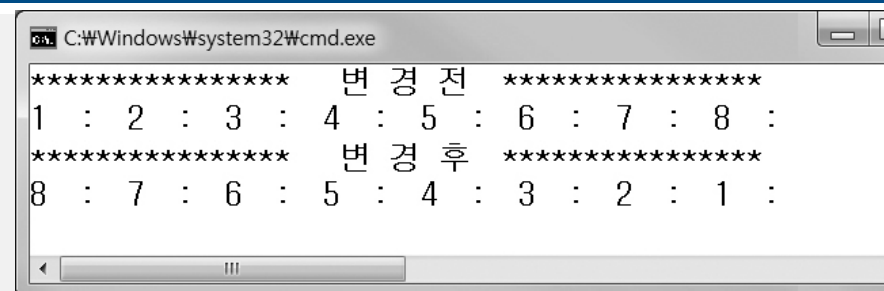
    for (int i= 0; i < v.size(); i++)
        v[i] = i + 1 ;

    ostream_iterator<int>  out_iter( cout, " : " );

    cout << "***** 변경 전 *****" << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;

    random_shuffle( v.begin(), v.end() );

    cout << "***** 변경 후 *****" << endl;
    copy(v.begin(), v.end(), out_iter);
    cout << endl;
}
```



```
C:\Windows\system32\cmd.exe
***** 변경 전 *****
1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 :
***** 변경 후 *****
8 : 7 : 6 : 5 : 4 : 3 : 2 : 1 :
```

## ● 정렬 관련 알고리즘

함수	설명
sort, stable_sort, partial_sort	원소들을 오름차순으로 정렬한다.
nth_element	N번째로 작은 원소를 찾아낸다.
binary_search, lower_bound, upper_bound, equal_range	정렬 시퀀스를 분할하며 검색한다.
merge	두 정렬 시퀀스를 하나로 합친다.
includes, set_union, set_intersection, set_difference, set_symmetric_difference	정렬 구조에 set 연산을 수행한다.
push_heap, pop_heap, make_heap, sort_heap	힙으로 구성된 시퀀스에 정렬을 수행한다.
min, max, min_element, max_element	시퀀스나 pair의 최솟값, 최댓값을 찾아낸다.
lexicographical_compare	두 시퀀스를 operator <와 operator >를 사용해서 비교한다.
next_permutation, prev_permutation	순열을 생성한다.

- `binary_search`
  - 구간내의 원소를 탐색하는 `find`와 동일 기능
  - 차이점 - 검색 방식
    - `find`는 선형 검색을 하는 반면 `binary_search`는 이진 탐색
- `lower_bound`
  - 주어진 구간 내에서 매개변수로 전달한 값이 있는 첫 번째 위치를 반환
  - 값이 없으면 삽입할 수 있는 위치를 반환

# binary\_search() 사용하기

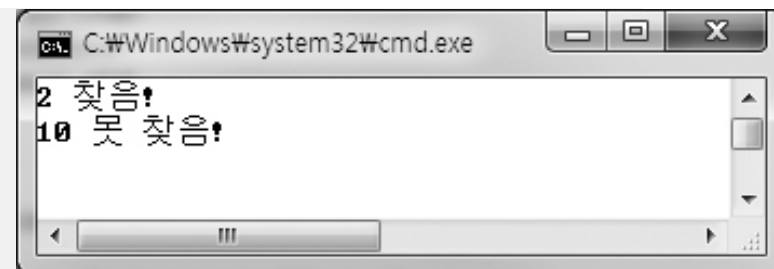
```
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;

void main()
{
    vector<int>  v(8);

    for (int i= 0; i < v.size(); i++)
        v[i] = i + 1 ;

    if( binary_search(v.begin(), v.end(), 2) )
        cout << "2 찾음!" << endl;
    else
        cout << "2 못 찾음!" << endl;

    if( binary_search(v.begin(), v.end(), 10) )
        cout << "10 찾음!" << endl;
    else
        cout << "10 못 찾음!" << endl;
}
```



# lower\_bound() 사용하기

```
{
int arr[10]={30, 50, 10, 40, 30, 70, 30, 90, 20, 10};
vector<int> v(10);
copy(arr, arr+10, v.begin()); //배열을 벡터로 복사

ostream_iterator<int> out_iter( cout, " : " );

cout << "***** 정렬 전 *****" << endl;
copy(v.begin(), v.end(), out_iter);
cout << endl;

sort(v.begin(), v.end());

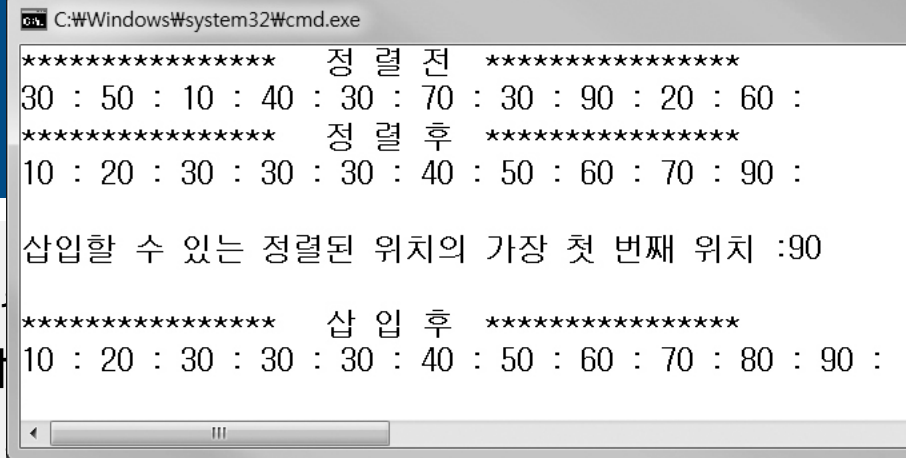
cout << "***** 정렬 후 *****" << endl;
copy(v.begin(), v.end(), out_iter);
cout << endl;

vector<int>::iterator iter;
iter = lower_bound(v.begin(), v.end(), 80);

cout << "\n삽입할 수 있는 정렬된 위치의 가장 첫 번째 위치 : " ;
cout << *iter << "\n\n";

v.insert(iter, 80);

cout << "***** 삽입 후 *****" << endl;
copy(v.begin(), v.end(), out_iter);
cout << endl;
}
```



```
C:\Windows\system32\cmd.exe
***** 정렬 전 *****
30 : 50 : 10 : 40 : 30 : 70 : 30 : 90 : 20 : 60 :
***** 정렬 후 *****
10 : 20 : 30 : 30 : 30 : 40 : 50 : 60 : 70 : 90 :

삽입할 수 있는 정렬된 위치의 가장 첫 번째 위치 :90

***** 삽입 후 *****
10 : 20 : 30 : 30 : 30 : 40 : 50 : 60 : 70 : 80 : 90 :
```

## ● 범용 수치 알고리즘

함수	설명
accumulate	원소들의 합을 계산한다.
inner_product	두 시퀀스의 원소 쌍을 곱해서 더한 값을 계산한다.
partial_sum	원소들의 부분합을 계산한다.
adjacent_difference	인접 원소들의 차를 계산한다.

- accumulate
  - 일정 구간 내에 속한 값들의 누적합을 구한다.  
`accumulate(iterator s, iterator e, T value)`
  - 마지막 매개변수는 누적합의 초기값인데, 일반적으로 0을 지정한다.
- inner\_product
  - inner\_product는 벡터의 내적을 구한다.

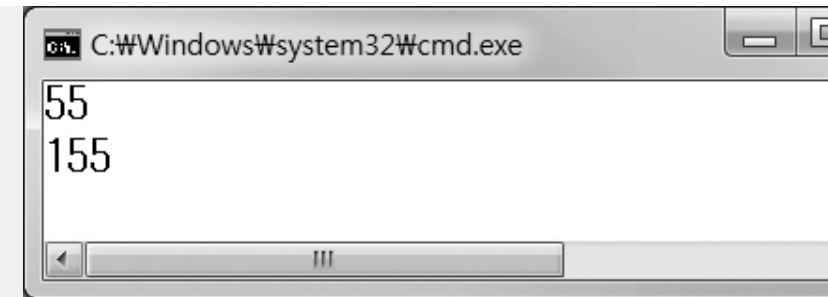
# accumulate() 사용하기

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

void main( )
{
    vector<int>  v(10);

    for (int i= 0; i < v.size(); i++)
        v[i] = i + 1 ;

    cout << accumulate(v.begin(), v.end(), 0) << endl;
    cout << accumulate(v.begin(), v.end(), 100) << endl;
}
```



# inner\_product() 사용하기

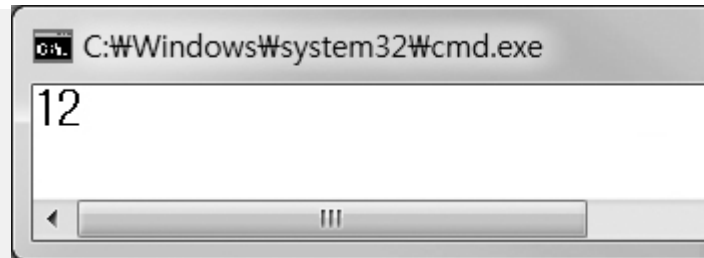
```
#include <numeric>
#include <vector>
#include <iostream>
using namespace std;
```

```
void main( )
{
    vector<int> v1;
    vector<int> v2;

    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);

    v2.push_back(2);
    v2.push_back(2);
    v2.push_back(2);

    cout << inner_product(v1.begin(), v1.end(), v2.begin(), 0) << endl;
}
```





# Thank you!

## Beyond The Engine of Korea

HANYANG UNIVERSITY



한양대학교  
HANYANG UNIVERSITY