원리를 알면 IT가 맛있다
IT COOKBOOK

IT CookBook, C++ 하이킹 객체지향과 만나는 여행

[강의교안 이용 안내]

- 본 강의교안의 저작권은 한빛아카데미㈜에 있습니다.
- 이 자료를 무단으로 전제하거나 배포할 경우 저작권법 136조에 의거하여 최고 5년 이하의 징역 또는 5천만 원 이하의 벌금에 처할 수 있고 이를 병과(倂科)할 수도 있습니다.

C++ 하이킹

객체지향과 만나는 여행

Chapter 15. STL

한빛아카데미 Hanbit Academy, Inc.

목차

학습목표

- STL의 개념을 말할 수 있다.
- STL의 동작원리를 살펴본다.

01 STL의 이해

- STL은 컨테이너(container)와 알고리즘(algorithm)을 일반화한 자료구조 라이브러리로서 일반화된 라이브러리(generic library)의 사용을 위해서 Template(템플릿)을 제공한다.
- STL의 핵심 구성요소는 컨테이너, 알고리즘, 반복자, 3가지로 대표된다.
  - ❶ 컨테이너(container) : 객체들을 담아둘 수 있는 객체를 말한다. 예를 들면 배열과 유사한 vector를 들 수 있다.
  - ❷ 반복자(iterator) : 컨테이너와 알고리즘 사이에서 이를 연결해 주는 역할을 한다. 반복자는 컨테이너가 관리하는 요소에 접근할 수 있게 해 주는 추상화된 개념으로서, 프로그램을 하는 데 있어 반복적인 작업에 이름을 붙여서 프로그래머가 반복적인 작업을 할 때 사용하는 패턴 중 하나다. 반복자를 통해 컨테이너 내의 요소들을 다양한 방법으로 다룰 수 있기 때문에 알고리즘과 컨테이너를 연결하는 매체가 된다. 컨테이너에 알고리즘을 적용하려면 반복자가 꼭 필요하다.
  - ❸ 알고리즘(algorithm) : 자료들을 가공하고 적절히 사용할 수 있는 방법을 말한다. 예를 들면 컨테이너를 정렬하기 위한 sort를 들 수 있다. STL에서는 알고리즘을 컨테이너에서 완전히 분리시켜 컨테이너 종류와는 무관하게 어떤 컨테이너에도 적용할 수 있도록 이식성을 높였다.

02 vector 템플릿 클래스로 이해하는 구성 요소

STL 컨테이너

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

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

- vector 클래스명 다음에 기술한 <int>는 벡터 객체가 갖는 요소의 자료형이 정수형이라는 의미다. vector 객체를 생성할 때 호출되는 생성자는 첫 번째 매개변수가 벡터의 크기가 되며, 두 번째 매개변수는 벡터의 초깃값이다.
- 다음은 벡터 클래스의 생성자다.

```
vecor();                                    // 빈 벡터(empty vector)를 생성한다.
vector(size_type _Count);                   // 벡터의 크기만 지정한다.
vector(size_type _Count, const Type& _Val); // 벡터의 크기와 요소의 초깃값을 지정한다.
```

- 벡터는 요소의 개수를 알려주는 size라는 멤버함수를 제공한다.

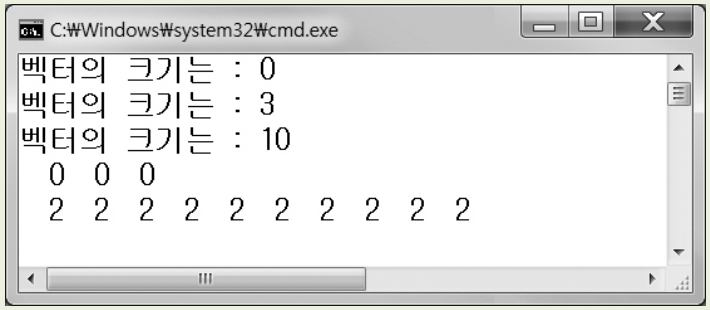
```
size_type size() const;
```

반복자

- 자주 사용하는 함수로 begin( )과 end( )가 있다. 이 두 함수는 반복자와 관련된 함수이다. 컨테이너 종류에 따라서 접근 방식이 다른데, 이를 모두 익혀서 사용하는 불편함을 해소하고자 나온 개념이 반복자다.

예제 15-1. 벡터 컨테이너 사용하기(15_01.cpp)


```
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 }
```



02 vector 템플릿 클래스로 이해하는 구성 요소

- 반복자를 사용하려면 iterator 클래스로 객체 선언을 해야 한다.

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

- iter가 반복자이며 이는 int형 요소를 저장하고 있는 vector의 요소를 지시할 수 있다.

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

- 반복자는 포인터를 일반화했다고 생각할 수 있다. 특정 위치를 지시하고 있는 반복자는 요소의 값을 참조하려면 * 연산자를 사용하고 다음 요소를 지시하려고 이동하려면 ++ 연산자를 사용한다.

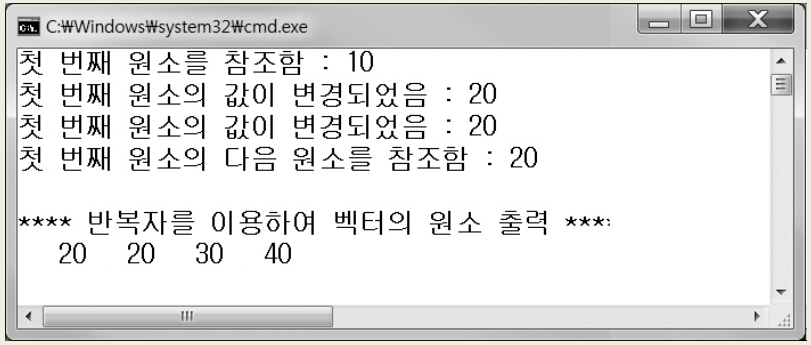
```
vector <int> v;      // int형 요소를 저장하는 벡터 객체 v를 선언한다.
iter = v.begin();    // 첫 번째 요소를 지시한다.
*iter = 80;          // iter가 참조하는 요소에 특정 값을 대입한다.
iter++;              // 다음 요소를 지시한다.
```

- STL 컨테이너에서 공통적으로 사용할 수 있는 함수로 push_back( )이 있다. 이 함수는 요소를 벡터 끝에 추가한다.

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

예제 15-2. 반복자 사용하기(15_02.cpp)


```
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 }
```



02 vector 템플릿 클래스로 이해하는 구성 요소

알고리즘

- 알고리즘은 이 자료들을 가공하고 적절하게 사용할 수 있는 방법을 말한다. 알고리즘의 예로 지정된 범위 내의 요소들을 오름차순으로 정렬하는 sort( ) 함수를 사용해 보자.

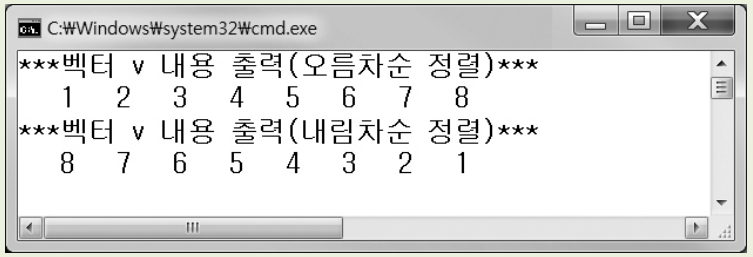
```
sort(start, end);
```

- 만일 내림차순으로 정렬하려면 greater<int>( ) 함수를 sort( ) 함수의 마지막 매개변수에 기술해 주어야 한다. 주의할 점은 greater<int>( ) 함수를 사용하려면 #include <functional>을 프로그램 문두에 기술해야 한다는 것이다.

예제 15-3. sort( ) 함수 사용하기(15_03.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
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 cout<<"₩n";
25
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 }
```



03 반복자

- 반복자는 동작 방식에 따라 5가지로 분류할 수 있다.

[표 15-1] 반복자의 종류

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 위치에서 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 산술 연산(+, -)이 가능하다.

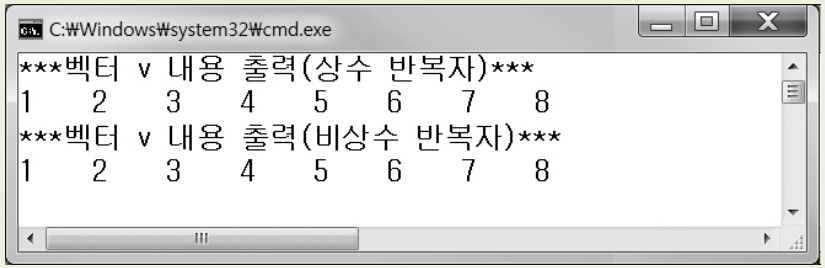
비상수 반복자와 상수 반복자

- 반복자는 반복자의 동작 방식으로 반복자를 분류하는 것 외에 반복자가 가리키는 값을 변경할 수 있는지 없는지에 따라 '비상수 반복자(mutable)'와 '상수 반복자(constant)'로 분류 할 수 있다.
- 비상수 반복자는 operator * 연산자에 의해서 참조값을 상수 반복자는 상수 참조값을 결과값으로 반환한다.
- 상수 반복자는 컨테이너의 요소를 변경할 수 있기 때문에 상수 컨테이너의 요소는 상수 반복자로만 접근할 수 있다.

예제 15-4. 비상수 반복자와 상수 반복자 사용하기(15_04.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
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 }
```



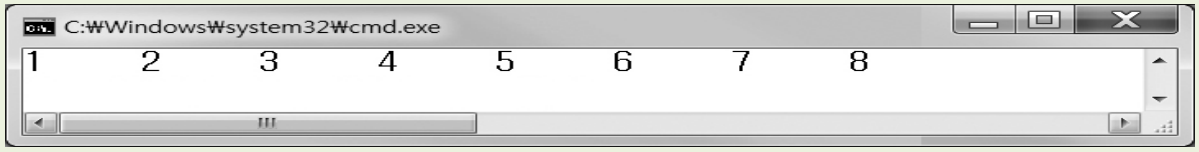
예제 15-5. 비상수 방복자와 상수 반복자의 차이점(15_05.cpp)


```
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 }
```

예제 15-6. 상수 컨테이너와 상수 반복자 사용하기(15_06.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
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 }
```



03 반복자

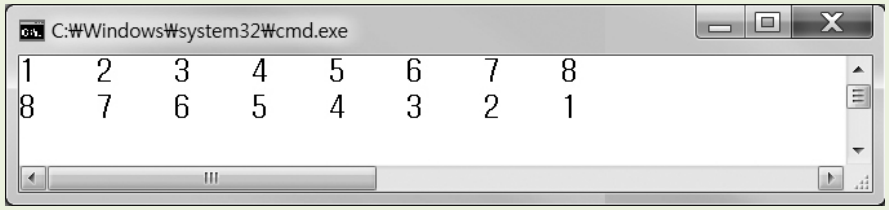
역방향 반복자

- STL 컨테이너는 모두 사용 편의성과 효율성이 좋도록 역방향 반복자(reverse_iterator)를 제공한다. 역방향 반복자를 사용하면 순방향 반복자(iterator)와 반대로 동작시킬 수 있다.
- STL 컨테이너는 역방향 반복자의 상수 역방향 반복자(const_reverse_iterator)도 제공한다.

예제 15-7. 역방향 반복자 사용하기(15_07.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
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 }
```



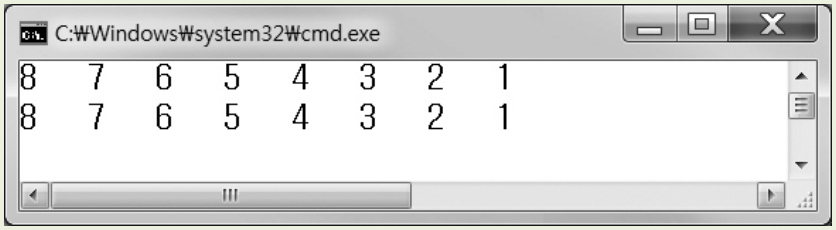
예제 15-8. 역방향 반복자와 상수 역방향 반복자 사용하기(15_08.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
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 }
```



04 컨테이너

- STL의 컨테이너는 형식이 같은 즉, 동질적인 객체의 집합을 저장하고 관리하는 역할을 한다.
- STL 컨테이너는 크게 3가지로 분류할 수 있다.
  1. 시퀀스 컨테이너(sequence container) : 자료의 선형적인 집합이며 자료를 저장하는 기본 임무에 충실한 가장 일반적인 컨테이너다. 삽입된 자료를 무조건 저장하며 입력되는 자료에 특별한 제약이나 관리 규칙은 없다. 사용자는 시퀀스의 임의 위치에 원하는 요소를 마음대로 삽입, 삭제할 수 있다. STL에는 벡터, 리스트, 데크, 3가지의 시퀀스 컨테이너가 제공된다.
  2. 연관 컨테이너(associative ontainer) : 자료를 무조건 저장하기만 하는 것이 아니라 일정한 규칙에 따라 자료를 조직화해서 관리하는 컨테이너다. 정렬이나 해시 등의 방법을 통해 삽입되는 자료를 항상 일정한 기준(오름차순, 해시 함수)에 맞는 위치에 저장해 놓으므로 검색 속도가 빠른 것이 장점이다. 표준 STL에는 정렬 연관 컨테이너인 셋, 맵 등의 컨테이너가 제공된다.
  3. 어댑터 컨테이너(adapter container) : 시퀀스 컨테이너를 변형해서 자료를 미리 정해진 일정한 방식에 따라 관리하는 것이 특징이다. 스택, 큐, 우선순위 큐, 3가지가 있는데, 스택은 항상 LIFO의 원리로 동작하며 큐는 항상 FIFO의 원리로 동작한다. 자료를 넣고 빼는 순서를 외부에서 마음대로 조작할 수 없으며 컨테이너의 규칙대로 조작해야 한다.

04 컨테이너

list

- 리스트는 vector 클래스와 마찬가지로 템플릿 클래스이므로 객체를 생성할 때 요소로 저장할 자료형을 언급해서 다음과 같이 생성한다.

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

[표 15-2] 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

- 리스트는 벡터와 같이 동일한 자료의 집합을 관리한다. 하지만 내부적인 구조가 다르기 때문에 특징이 서로 다르다. 벡터가 읽고 쓰기에 강한 컨테이너라면 리스트는 삽입, 삭제에 강한 컨테이너다.

map

- 맵은 키(key), 값(value)의 쌍으로 데이터를 저장하므로 키를 이용해서 값을 찾을 때 유용하게 사용한다. 그리고 키로 원하는 항목을 검색하기 때문에 검색 속도가 빠르다는 장점이 있다.

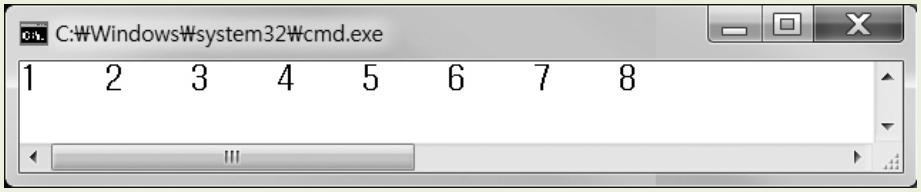
set

- set은 중복되지 않는 요소들을 관리한다. 동일한 요소의 중복을 허용하지 않기 때문에 이미 저장된 객체를 또 다시 저장하면 추가되지 않는다.

예제 15-9. list 컨테이너 사용하기(15_09.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

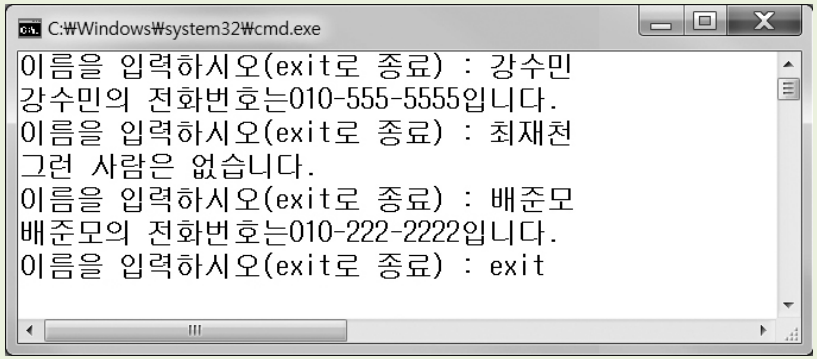
```
01 #include <list>
02 #include <iostream>
03 using namespace std;
04
05 void main()
06 {
07 list<int> li;
08
09 for (int i = 0; i < 8; i++)
10 li.push_back(i + 1);
11
12 list<int>::iterator iter;
13
14 for(iter = li.begin(); iter != li.end(); iter++)
15 cout << *iter <<" ";
16 cout << endl;
17 }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



예제 15-10. map 컨테이너 사용하기(15_10.cpp)

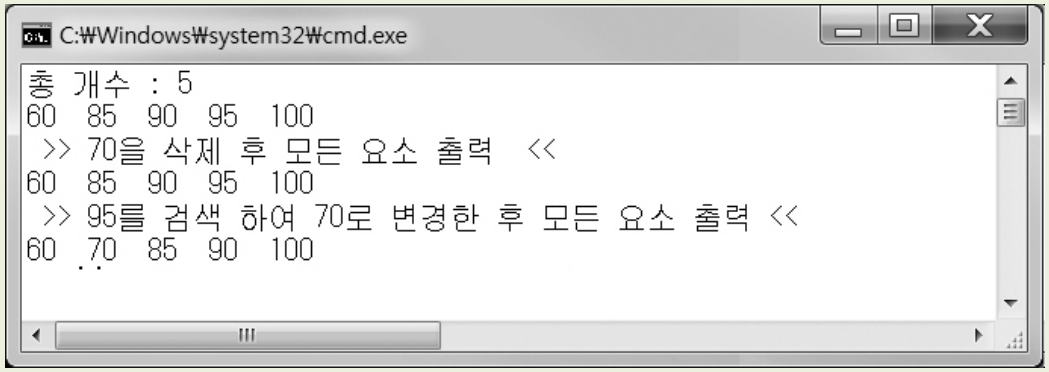
원리를 알면 IT가 맛있다
IT COOKBOOK

```
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"<<iter->second<<"이다.₩n";
28 }
29 }
30 }
```



예제 15-11. set 컨테이너 사용하기(15_11.cpp)

책의 소스코드 참고



05 STL 알고리즘

- STL 알고리즘은 문제 해결을 위해서 제공되는 STL 함수로 크게 4가지로 분류한다.
  1. 변경 불가 시퀀스 알고리즘(nonmutating sequence algorithm)
  2. 변경 가능 시퀀스 알고리즘(mutaing sequence algorithm)
  3. 정렬 관련 알고리즘(sorting-related algorithm)
  4. 범용 수치 알고리즘(generalized numeric algorithm)

변경 불가 시퀀스 알고리즘

[표 15-3] 변경 불가 시퀀스 알고리즘

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.

05 STL 알고리즘

for_each

- for_each는 컨테이너 클래스에서 공통적으로 사용할 수 있는 함수다. for_each는 일정 범위 안에 있는 모든 요소들에 대해서 함수를 호출한다. 아래의 PrintElement는 요소의 값을 출력하는 함수다.

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

- for_each는 매개변수 3개를 사용한다. 첫 번째와 두 번째 매개변수에 의해서 컨테이너의 범위가 지정되고 마지막 매개변수로 지정된 함수가 지정된 범위 내의 모든 요소에 의해서 적용된다.

find, find_if

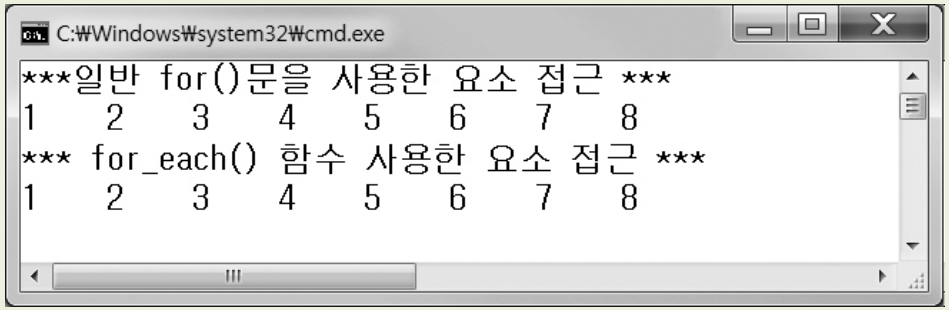
- find는 주어진 구간 내에서 주어진 값과 동일한 값을 찾아서 그 위치를 반복자 형태로 알려준다. 만일 검색에 실패하면 end를 반환한다.
- find_if는 마지막 매개변수로 조건자 함수를 넘겨주고 반복자들의 구간 내에서 함수에 제시한 조건을 만족하는 요소를 찾는다. 그리고 구간 내 요소에 대해서 조건자 함수를 적용해서 결과값으로 true를 반환되는 위치를 반복자 형태로 알려주며 조건에 만족하는 요소를 찾지 못하면 end를 반환한다.

count, count_if

- count는 주어진 구간 내에서 주어진 값과 동일한 값을 찾는 요소의 개수를 알려준다. count_if는 마지막 매개변수로 조건자 함수를 넘겨준다.

예제 15-12. for_each( ) 사용하기(15_12.cpp)

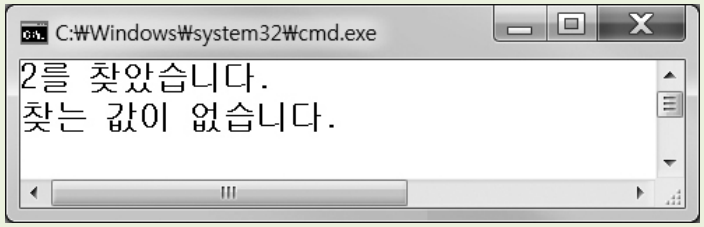
책의 소스코드 참고



예제 15-13. find( ) 사용하기(15_13.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

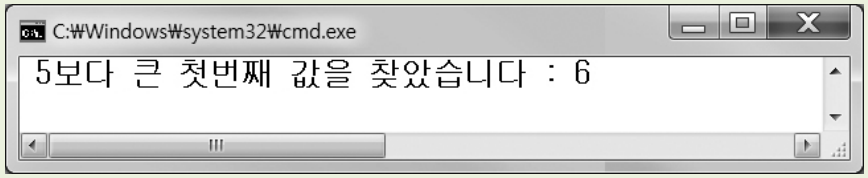
책의 소스코드 참고



예제 15-14. find_if( ) 사용하기(15_14.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

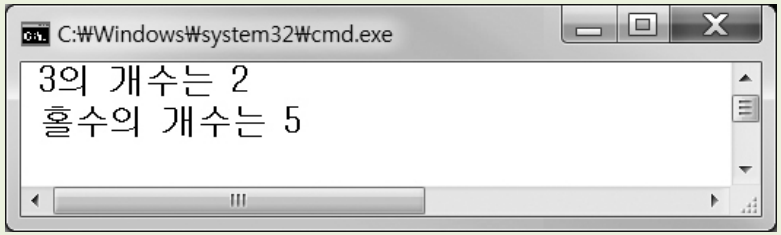
책의 소스코드 참고



예제 15-15. count( ), count_if( ) 사용하기(15_15.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

책의 소스코드 참고



05 STL 알고리즘

equal

- 두 구간 내의 요소들이 일치하는지 조사한다.

search

- find는 일정 구간 내에서 특정한 값 하나에 대해서 일치되는 위치를 찾는다. 반면 search는 연속적인 값들을 탐색할 경우 사용한다. 전체 구간 안(v1)에 부분 구간(v2)을 찾아 그 위치를 반환한다.

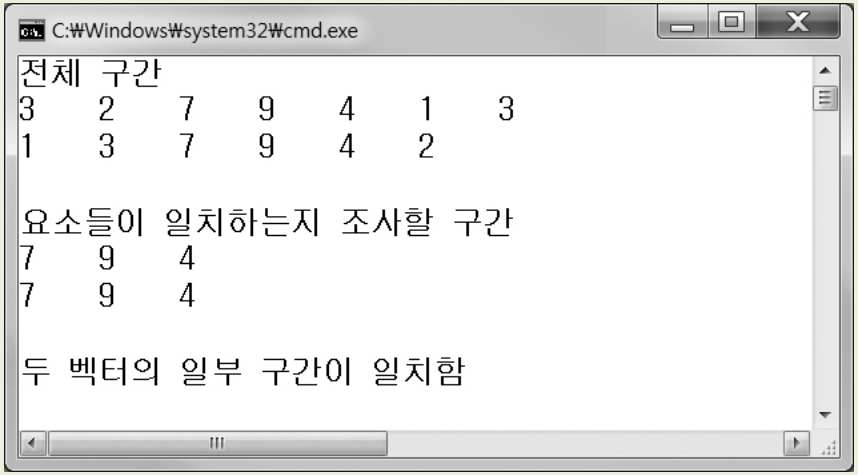
변경 가능 시퀀스 알고리즘

[표 15-4] 변경 가능 시퀀스 알고리즘

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.

예제 15-16. equal( ) 사용하기(15_16.cpp)

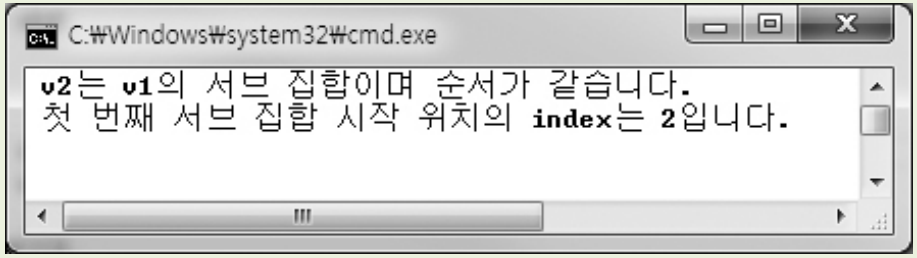
책의 소스코드 참고



예제 15-17. search( ) 사용하기(15_17.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

책의 소스코드 참고



05 STL 알고리즘

copy

- copy는 한 컨테이너에서 다른 컨테이너로 데이터를 복사한다. 다음은 배열을 벡터 객체에 복사하는 예다.

```
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 복사한다. STL은 출력 스트림을 나타내는 반복자로 ostream_iterator를 제공한다.

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

- copy에 출력 반복자를 다음과 같이 사용해서 컨테이너의 내용을 출력할 수 있다.

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

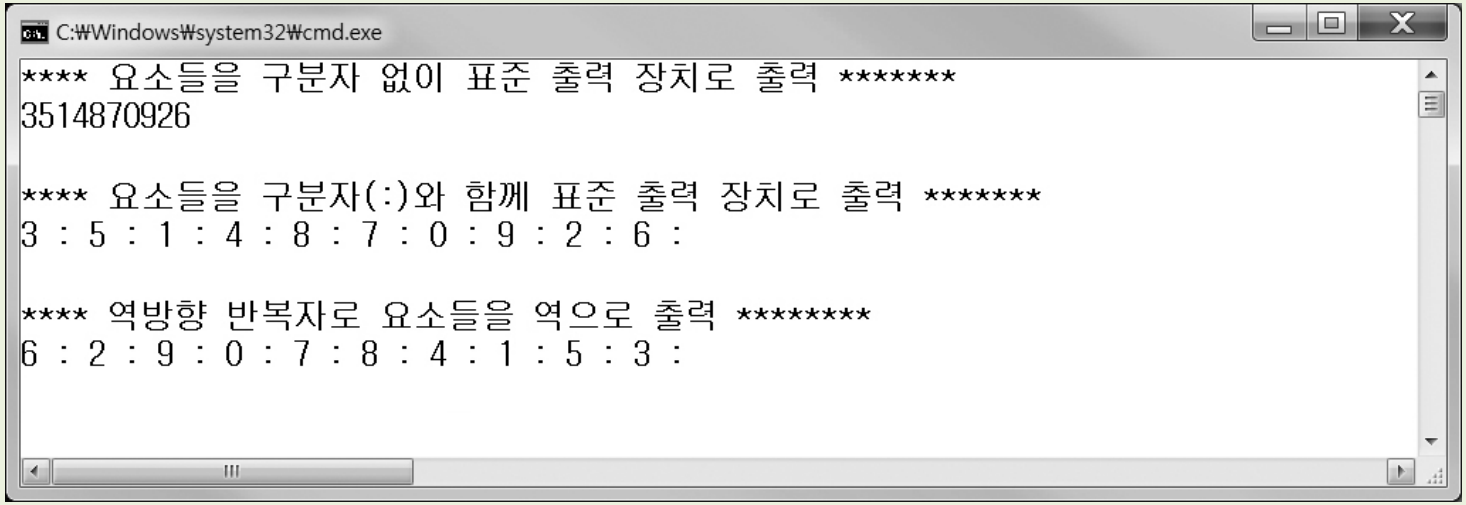
- 만일 역순으로 출력하려면 역방향 반복자를 사용한다. 역방향 반복자로는 rbegin과 rend가 있는데, rbegin은 맨 마지막 요소 다음을 지시하고, rend는 첫 번째 요소를 지시한다.

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

예제 15-18. copy( ) 사용하기(15_18.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

책의 소스코드 참고



05 STL 알고리즘

- **transform**
  - transform은 주어진 구간 내의 요소들을 매개변수로 전달한 함수를 적용해서 변경한다.
- **replace**
  - replace는 주어진 구간 내의 요소들의 값을 매개변수로 전달한 값으로 대체한다.
- **fill**
  - fill은 주어진 구간 내의 요소들을 값을 매개변수로 전달한 값으로 할당한다.
- **reverse**
  - reverse는 주어진 구간 내의 요소들을 역순으로 배치한다.
- **random_shuffle**
  - random_shuffle은 주어진 구간 내의 요소들의 위치를 무작위로 재배치한다.

예제 15-19. transform( ) 사용하기(15_19.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
01 #include <iterator> // ostream_iterator
02 #include <vector>
03 #include <algorithm>
04 #include <iostream>
05 using namespace std;
06
07 int Increment(int n)
08 {
09 return n+1;
10 }
11
12 void main()
13 {
14 vector<int> v(8) ;
15
16 for (int i = 0; i < v.size(); i++)
17 v[i] = i + 1 ;
18
19 ostream_iterator<int> out_iter( cout, " : " );
20
21 cout << "********** 변 경 전 ********** "<< endl;
22 copy(v.begin(), v.end(), out_iter);
23 cout <<endl;
24
25 transform(v.begin(),v.end(), v.begin(), Increment);
26
27 cout << "********** 변 경 후 ********** "<< endl;
28 copy(v.begin(), v.end(), out_iter);
29 cout <<endl;
30 }
```

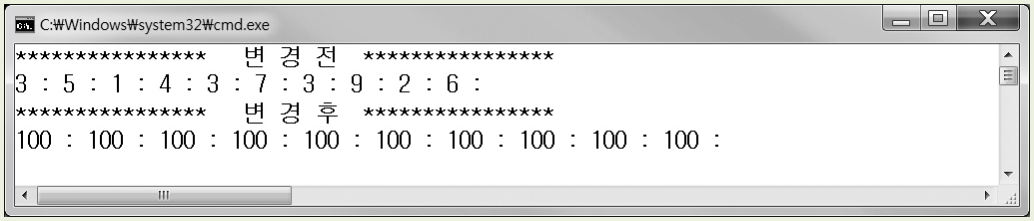
예제 15-20. replace( ) 사용하기(15_20.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
01 #include <iterator> // ostream_iterator
02 #include <vector>
03 #include <algorithm>
04 #include <iostream>
05 using namespace std;
06
07 void main()
08 {
09 int arr[10]={3, 5, 1, 4, 3, 7, 3, 9, 2, 6};
10
11 vector<int> v(10);
12
13 copy(arr, arr+10, v.begin()); // 배열을 벡터로 복사
14
15 ostream_iterator<int> out_iter( cout, " : " );
16
17 cout << "**************** 변 경 전 **************** "<< endl;
18 copy(v.begin(), v.end(), out_iter);
19 cout <<endl;
20
21 replace(v.begin(), v.end(), 3, 100);
22
23 cout << "**************** 변 경 후 **************** "<< endl;
24 copy(v.begin(), v.end(), out_iter);
25 cout <<endl;
26 }
```

예제 15-21. fill( ) 사용하기(15_21.cpp)


```
01 #include <iterator> // ostream_iterator
02 #include <vector>
03 #include <algorithm>
04 #include <iostream>
05 using namespace std;
06
07 void main()
08 {
09 int arr[10]={3, 5, 1, 4, 3, 7, 3, 9, 2, 6};
10
11 vector<int> v(10);
12
13 copy(arr, arr+10, v.begin()); // 배열을 벡터로 복사
14
15 ostream_iterator<int> out_iter( cout, " : " );
16
17 cout << "**************** 변 경 전 **************** "<< endl;
18 copy(v.begin(), v.end(), out_iter);
19 cout <<endl;
20
21 fill(v.begin(), v.end(), 100);
22
23 cout << "**************** 변 경 후 **************** "<< endl;
24 copy(v.begin(), v.end(), out_iter);
25 cout <<endl;
26 }
```



예제 15-22. reverse( ) 사용하기(15_22.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

책의 소스코드 참고

예제 15-23. random_shuffle( ) 사용하기(15_23.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
01 #include <iterator>// ostream_iterator
02 #include <vector>
03 #include <algorithm>
04 #include <iostream>
05 using namespace std;
06
07 void main()
08 {
09 vector<int> v(8);
10
11 for (int i= 0; i < v.size(); i++)
12 v[i] = i + 1 ;
13
14 ostream_iterator<int> out_iter( cout, " : " );
15
16 cout << "**************** 변 경 전 **************** "<< endl;
17 copy(v.begin(), v.end(), out_iter);
18 cout <<endl;
19
20 random_shuffle( v.begin(), v.end() );
21
22 cout << "**************** 변 경 후 **************** "<< endl;
23 copy(v.begin(), v.end(), out_iter);
24 cout <<endl;
25 }
```

원리를 알면 IT가 맛있다
IT COOKBOOK

정렬 관련 알고리즘

[표 15-5] 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

- binary_search는 구간내의 원소를 탐색하는 find와 동일한 기능을 한다. 차이점은 검색 방식에 있는데, find는 선형 검색을 하는 반면 binary_search는 이진 탐색을 한다.

lower_bound

- lower_bound는 주어진 구간 내에서 매개변수로 전달한 값이 있는 첫 번째 위치를 반환하며 만약 값이 없으면 삽입할 수 있는 위치를 반환한다.

범용 수치 알고리즘

- C++에서 자주 사용되는 범용 수치 알고리즘은 다음과 같다.

[표 15-6] 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.

05 STL 알고리즘

accumulate

- accumulate는 일정 구간 내에 속한 값들의 누적합을 구한다.

  accumulate(Iterator s, Iterator e, T value)

- 마지막 매개변수는 누적합의 초깃값인데, 일반적으로 0을 지정한다.

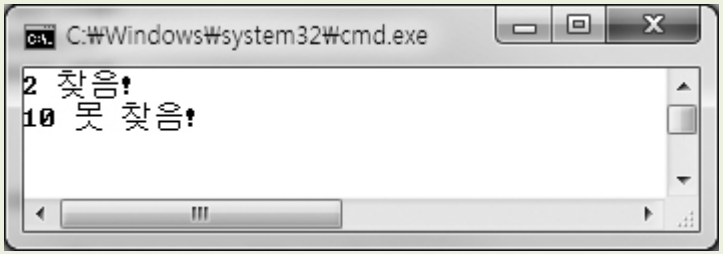
inner_prodect

- inner_product는 벡터의 내적을 구한다.

예제 15-24. binary_search( ) 사용하기(15_24.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

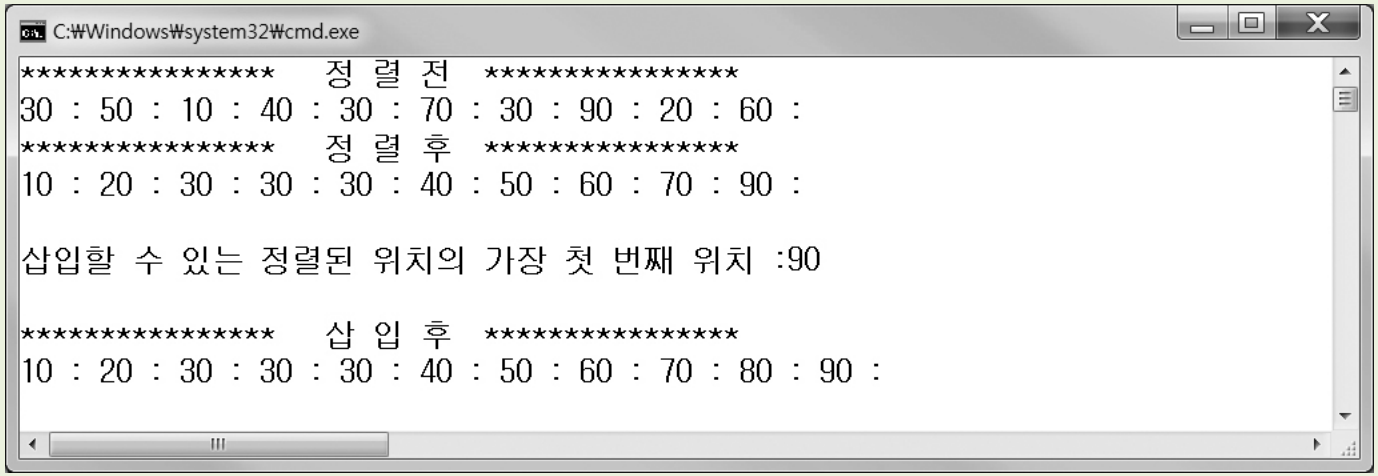
```
01 #include <vector>
02 #include <algorithm>
03 #include <iostream>
04 using namespace std;
05
06 void main()
07 {
08 vector<int> v(8);
09
10 for (int i= 0; i < v.size(); i++)
11 v[i] = i + 1 ;
12
13 if( binary_search(v.begin(), v.end(), 2) )
14 cout << "2 찾음!" << endl;
15 else
16 cout << "2 못 찾음!" << endl;
17
18 if( binary_search(v.begin(), v.end(), 10) )
19 cout << "10 찾음!" << endl;
20 else
21 cout << "10 못 찾음!" << endl;
22 }
```



예제 15-25. lower_bound( ) 사용하기(15_25.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

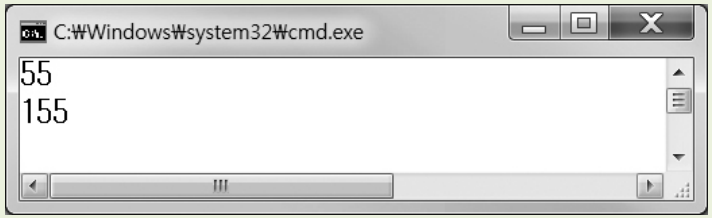
책의 소스코드 참고



예제 15-26. accumulate( ) 사용하기(15_26.cpp)

원리를 알면 IT가 맛있다
IT COOKBOOK

```
01 #include <numeric>
02 #include <vector>
03 #include <iostream>
04 using namespace std;
05
06 void main()
07 {
08 vector<int> v(10);
09
10 for (int i= 0; i < v.size(); i++)
11 v[i] = i + 1 ;
12
13 cout << accumulate(v.begin(), v.end(), 0) << endl;
14 cout << accumulate(v.begin(), v.end(), 100) << endl;
15 }
```



예제 15-27. inner_product( ) 사용하기(15_27.cpp)


```
01 #include <numeric>
02 #include <vector>
03 #include <iostream>
04 using namespace std;
05
06 void main()
07 {
08 vector<int> v1;
09 vector<int> v2;
10
11 v1.push_back(1);
12 v1.push_back(2);
13 v1.push_back(3);
14
15 v2.push_back(2);
16 v2.push_back(2);
17 v2.push_back(2);
18
19 cout << inner_product(v1.begin(), v1.end(), v2.begin(), 0) << endl;
20 }
```