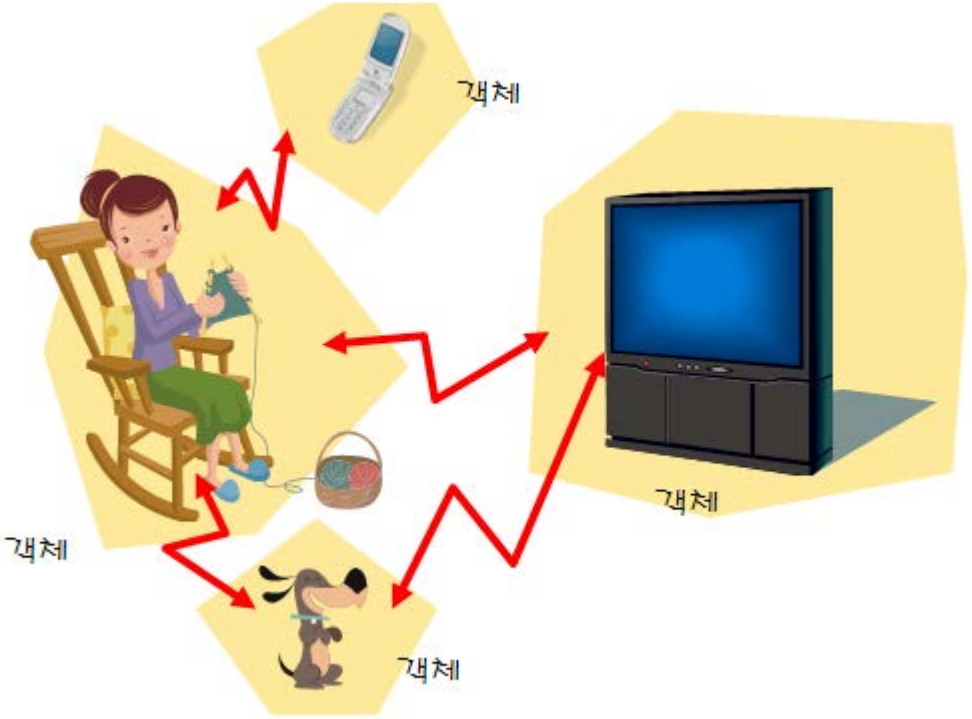
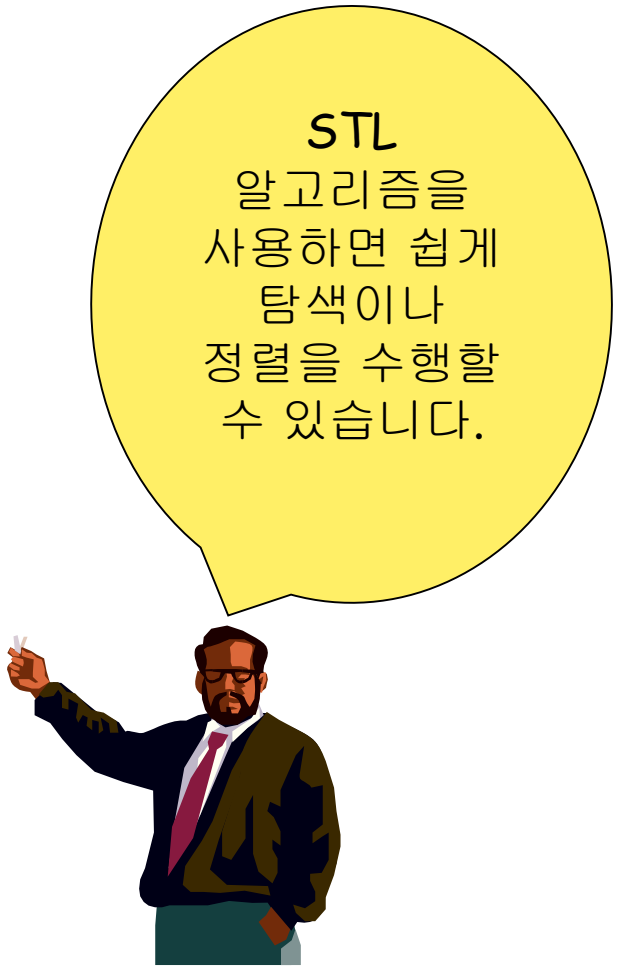
Power C++

제19장 STL 알고리즘



이번 장에서 학습할 내용

- 반복자
- 탐색 알고리즘
- 비교 알고리즘
- 초기화 알고리즘
- 수치 알고리즘
- 수치 알고리즘
- 함수 객체



STL 알고리즘

- STL 기반의 탐색(searching), 정렬(sorting), 계수(counting) 알고리즘

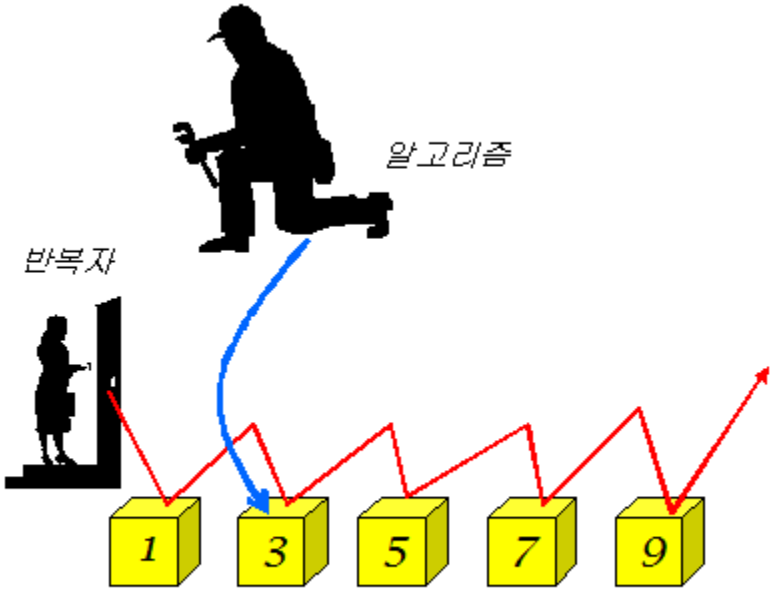


그림 19.1 STL 알고리즘은 반복자를 통하여 컨테이너에 접근하여 작업을 한다.

STL 알고리즘의 분류

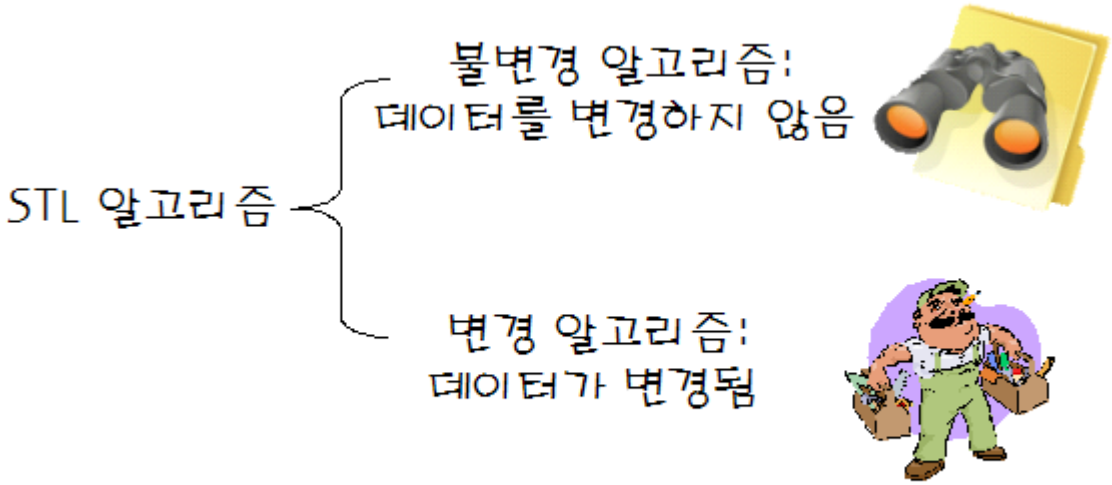


그림 19.2 STL 알고리즘의 분류

불변경 알고리즘

function pointer로 접근

표 19.1 불변경 알고리즘

분류	알고리즘	설명
계수 알고리즘	count()	주어진 값과 일치하는 요소들의 개수를 센다.
	count_if()	주어진 조건에 맞는 요소들의 개수를 센다.
탐색 알고리즘	search()	주어진 값과 일치하는 첫번째 요소를 반환한다.
	search_n()	주어진 값과 일치하는 n개의 요소를 반환한다.
	find()	주어진 값과 일치하는 첫번째 요소를 반환한다.
	find_if()	주어진 조건에 일치하는 첫번째 요소를 반환한다.
	find_end()	주어진 조건에 일치하는 마지막 요소를 반환한다.
	binary_search()	정렬된 컨테이너에 대하여 이진 탐색을 수행한다.
비교 알고리즘	equal()	두개의 요소가 같은지 비교한다.
	mismatch()	두개의 컨테이너를 비교하여서 일치하지 않는 첫번째 요소를 반환한다.
	lexicographical_compare()	두개의 순차 컨테이너를 비교하여서 사전적으로 어떤 컨테이너가 작은지를 반환한다.

변경 알고리즘

표 19.2 변경 알고리즘

분류	알고리즘	설명
초기화 알고리즘	fill()	지정된 범위의 모든 요소를 지정된 값으로 채운다.
	generate()	지정된 함수의 반환값을 할당한다.
변경 알고리즘	for_each()	지정된 범위의 모든 요소에 대하여 연산을 수행한다.
	transform()	지정된 범위의 모든 요소에 대하여 함수를 적용한다.
복사 알고리즘	copy()	하나의 구간을 다른 구간으로 복사한다.
삭제 알고리즘	remove()	지정된 구간에서 지정된 값을 가지는 요소들을 삭제한다.
	unique()	구간에서 중복된 요소들을 삭제한다.
대치 알고리즘	replace()	지정된 구간에서 요소가 지정된 값과 일치하면 대치값으로 바꾼다.
정렬 알고리즘	sort()	지정된 정렬 기준에 따라서 구간의 요소들을 정렬한다.
분할 알고리즘	partition()	지정된 구간의 요소들을 조건에 따라서 두개의 집합으로 나눈다.

반복자

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

- v.begin() 함수는 컨테이너 v에서 첫 번째 요소를 반환한다.
- v.end() 함수는 컨테이너 v에서 마지막 요소를 지났는지를 나타내는 값을 반환한다.

예제

```
int main()
{
    vector<int> vec;                    // 정수형 벡터 생성
    for(int i=0;i<10;i++)
        vec.push_back(i);

    vector<int>::iterator it;           // 반복자 객체 생성
    for(it=vec.begin(); it != vec.end() ; it++)          // 컨테이너의 모든 요소를 출력
        cout << *it << " ";
    cout << endl;

    for(it=vec.begin(); it != vec.end() ; it++)          // 컨테이너의 모든 요소를 -1로 설정
        *it = 0;

    for(it=vec.begin(); it != vec.end() ; it++)          // 컨테이너의 모든 요소를 출력
        cout << *it << " ";
    cout << endl;
    return 0;
}
```

실행 결과

실행 결과

```
0 1 2 3 4 5 6 7 8 9
0 0 0 0 0 0 0 0 0 0
계속하려면 아무 키나 누르십시오 . . .
```

반복자의 종류

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

예제

```
int main()
{
    vector<int> vec;                    // 정수형 벡터 생성
    for(int i=0;i<10;i++)
        vec.push_back(i);

    vector<int>::iterator it;           // 반복자 객체 생성
    it = vec.begin();
    cout << it[2] << " " << endl;
    cout << *(it+2) << " " << endl;
    it = it + 5;
    it--;
    cout << *it << " " << endl;
    return 0;
}
```

실행 결과

```
2
2
4
계속하려면 아무 키나 누르십시오 . . .
```

역순 반복자

iterator2.cpp

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

int main()
{
   vector<int> vec;                    // 정수형 벡터 생성
   for(int i=0;i<10;i++)
         vec.push_back(i);
   vector<int>::reverse_iterator rit;                    // 정수형 벡터 생성
   for(rit = vec.rbegin(); rit!= vec.rend() ; rit++)
         cout << *rit << " ";
   cout << endl;
   return 0;
}
```

실행 결과

```
9 8 7 6 5 4 3 2 1 0
계속하려면 아무 키나 누르십시오 . . .
```

공통 알고리즘

myheader.h

```
#ifndef MYHEADER_H
#define MYHEADER_H
template <typename T>
void print(const T& v, const char* message="")
{
    typename T::const_iterator it;
    std::cout << message;
    std::cout << "( ";
    for (it=v.begin(); it!=v.end(); ++it) {
        std::cout << *it << ' ';
    }
    std::cout << " )" << std::endl;
}
#endif
```

탐색 알고리즘: find()

find.cpp

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

int main()
{
   string fruits[5] = { "사과", "토마토", "배", "수박", "키위" };
   vector<string> vec(&fruits[0], &fruits[5]);          // 배열->벡터

   vector<string>::iterator it;                         //반복자 정의
   it = find(vec.begin(), vec.end(), "수박");

   if (it != vec.end())    == 있으면
        cout << "수박이 " << distance(vec.begin(), it) << "에 있습니다." << endl;

   return 0;                배열 포인터로 되어 있으니까 인덱스 접근 가능
}
```

vec의 처음부터 끝까지 "수박"을 탐색

it의 위치를 인덱스로 반환

실행 결과

```
수박이 3에 있습니다.
계속하려면 아무 키나 누르십시오 . . .
```

find_if()

find_if.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;
// 문자열 s가 "김"을 포함하면 true를 반환
bool checkKim(string s)
{
   if( s.find("김") != string::npos)        "김"이 있으면~
        return true;       npos : position /  const size_t(unsigned int) npos = -1;
   else
        return false;
}
```

find_if()

```
int main()
{
   string names[5] = { "김철수", "박문수", "강감찬", "김유신", "이순신" };
   vector<string> vec(&names[0], &names[5]);       // 배열->벡터

   vector<string>::iterator it;                    // 반복자 정의
   it = vec.begin();
   while(true){
        it=find_if(it, vec.end(),checkKim);
        if (it==vec.end())                         // 탐색 실패
                  break;
        cout << "위치 " << distance(vec.begin(), it)<<
                  "에서 " << *it << "를 탐색하였음" << endl;
        it++;
   }
   return 0;
}
```

위치 알려줌 = pointer / 함수의 이름 = 함수가 시작하는 곳의 위치
checkKim 은 function pointer

find에서 발견하면 그 곳이 해당하는 포인터를 반환, 발견 못하면 end() 즉 nullptr

실행 결과

```
위치 0에서 김철수를 탐색하였음
위치 3에서 김유신를 탐색하였음
계속하려면 아무 키나 누르십시오 . . .
```

search()

search.cpp

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

int main()
{
   vector<int> vec;
   for(int i=0;i<10;++i)              // vec = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
         vec.push_back(i);

   list<int> ilist;
   for(int i=3;i<6;++i)               // ilist = (3, 4, 5)
         ilist.push_back(i);
```

search()

```
    list<int> ilist;
    for(int i=3;i<6;++i)                    // ilist = (3, 4, 5)
        ilist.push_back(i);

    vector<int>::iterator it;
    it = search(vec.begin(),vec.end(), ilist.begin(), ilist.end());

    if (it != vec.end()){
        cout << "부분 구간이 " << distance(vec.begin(), it) << "에 있습니다." << endl;
    }

    return 0;
}
```

실행 결과

```
부분 구간이 3에 있습니다.
계속하려면 아무 키나 누르십시오 . . .
```

count()

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

int main()
{
   char *s = "I go to the school";
   vector<int> vec;
   for(int i=0;i<10;i++)
         vec.push_back(i);

   size_t n1 = count(s, s+strlen(s), 'o');
   size_t n2 = count_if(vec.begin(),vec.end(), is_even<int>);

   cout << "값이 'o'인 요소의 개수: " << n1 << endl;
   cout << "값이 짝수인 요소의 개수: "<< n2 << endl;

   return 0;
}
```

실행 결과

```
값이 'o'인 요소의 개수: 4
값이 짝수인 요소의 개수: 5
```

binary_search()

- 이진 탐색:
  - 정렬된 리스트에서 만약 찾고자 하는 원소가 중간 원소보다 크면 찾고자 하는 원소는 뒷부분에 있고 반대이면 앞부분에 있다.

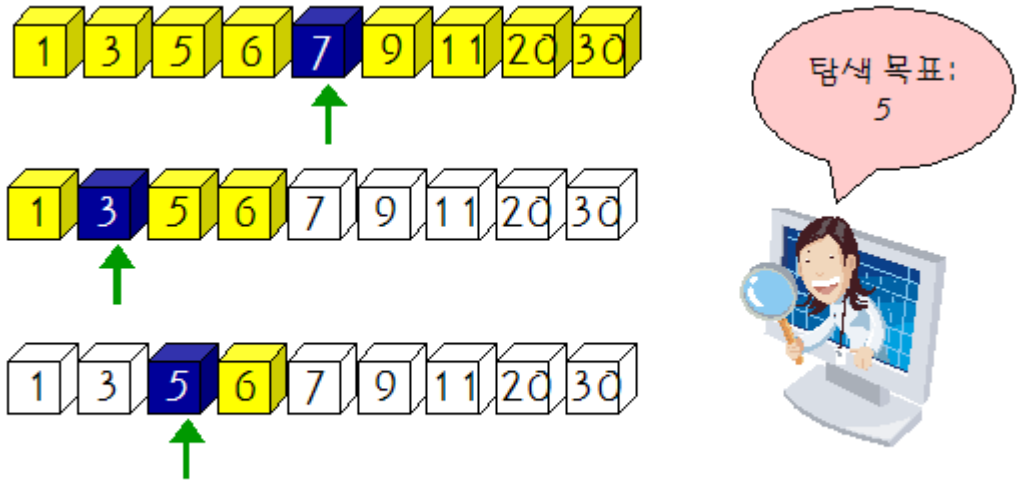


그림 19-3 이진 탐색의 개념

예 제

binary_search.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "myheader.h"
using namespace std;

int main()
{
   const int wanted = 6;
   int values[9] = { 1, 3, 5, 5, 5, 8, 11, 20, 30 };
   vector<int> vec(&values[0], &values[9]);
   vector<int>::iterator it;

   print(vec);
   bool isInIt =binary_search(vec.begin(),vec.end(), wanted);
```

예 제

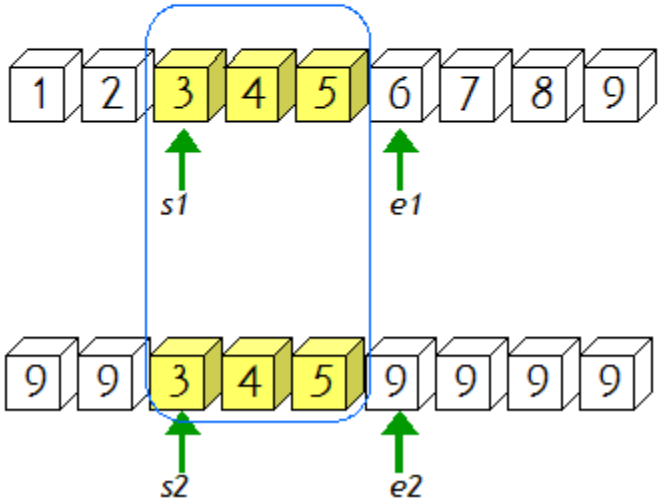
```
    if(isInIt ==true )
        cout << wanted << "을 찾았음 "<< endl;
    else {
        it =lower_bound(vec.begin(), vec.end(), wanted);
        vec.insert(it, wanted);
    }
    print(vec);

    return 0;
}
```

실행 결과

```
( 1 3 5 5 5 8 11 20 30 )
( 1 3 5 5 5 6 8 11 20 30 )
계속하려면 아무 키나 누르십시오 . . .
```

비교 알고리즘: equal()

```
bool
equal(ForwIter s1, ForwIter e1, ForwIter s2, [UnaryPred func]);
```



예 제

```
int main()
{
   int values1[9] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
   int values2[9] = { 9, 9, 3, 4, 5, 9, 9, 9, 9 };
   vector<int> vec1(&values1[0], &values1[9]);
   vector<int> vec2(&values2[0], &values2[9]);
   print(vec1);
   print(vec2);
   bool isEqual =equal(vec1.begin()+2,vec1.begin()+5, vec2.begin()+2);

   if(isEqual ==true )
        cout << "두개의 구간이 동일함" << endl;

   return 0;
}
```

실행 결과

```
( 1 2 3 4 5 6 7 8 9 )
( 9 9 3 4 5 9 9 9 9 )
두개의 구간이 동일함
계속하려면 아무 키나 누르십시오 . . .
```

초기화 알고리즘: fill()

```
int main()
{
    vector<int> v1;
    for( int i = 0; i < 10; i++ ) {
          v1.push_back( i );
    }
    cout << "fill() 이전의 값 ";
    print(v1);
    fill( v1.begin(), v1.end(), 0);

    cout << "fill() 이후의 값 ";
    print(v1);
    return 0;
}
```

실행 결과

```
fill() 이전의 값 ( 0 1 2 3 4 5 6 7 8 9 )
fill() 이후의 값 ( 0 0 0 0 0 0 0 0 0 0 )
계속하려면 아무 키나 누르십시오 . . .
```

copy()

copy.cpp

```
// copy와 reverse의 사용 예
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
using namespace std;

int main()
{
   string names[5] = {"사과","배","키위","레몬","포도"};
   vector<string> fruits(5);
   vector<string>::iterator it;
   copy(&names[0], &names[5], fruits.begin());
   reverse(fruits.begin(), fruits.end());
   for (it = fruits.begin(); it != fruits.end(); ++it)
      cout << *it << " ";
   cout << endl;
}
```

실행 결과

```
포도 레몬 키위 배 사과
계속하려면 아무 키나 누르십시오 . . .
```

for_each()

```
// 각 요소에 대하여 호출되는 함수
void display(int element)
{
   cout << element << ' ';
}
int main()
{
   vector<int> vec;
   for(int i=0;i<10; i++)
         vec.push_back(i);

   for_each (vec.begin(), vec.end(),  display);
   cout << endl;
   return 0;
}
```

실행 결과

```
0 1 2 3 4 5 6 7 8 9
계속하려면 아무 키나 누르십시오 . . .
```

transform()

```
// 각 요소에 대하여 호출되는 함수
int increment(int element)
{
   return ++element;
}
int main()
{
   vector<int> vec;
   vector<int> result(10);
   for(int i=0;i<10; i++)
         vec.push_back(i);

   print(vec);
   transform (vec.begin(), vec.end(), result.begin(), increment);
   print(result);
   return 0;
}
```

실행 결과

```
( 0 1 2 3 4 5 6 7 8 9 )
( 1 2 3 4 5 6 7 8 9 10 )
계속하려면 아무 키나 누르십시오 . . .
```

remove()

```
int main()
{
    string names[5] = {"사과","배","키위","레몬","포도"};
    vector<string> fruits(5);
    copy(&names[0], &names[5], fruits.begin());
    print(fruits, "삭제전\n");
    vector<string>::iterator it;
    it = remove(fruits.begin(), fruits.end(), "레몬");
    print(fruits, "remove() 후\n");
    fruits.erase(it, fruits.end());
    print(fruits, "erase() 후\n");
    return 0;
}
```

실행 결과

```
삭제전
( 사과 배 키위 레몬 포도 )
remove() 후
( 사과 배 키위 포도 포도 )
erase() 후
( 사과 배 키위 포도 )
```

sort()

sort.cpp

```
#include <iostream>
#include <algorithm>
#include <vector>
#include "myheader.h"
using namespace std;

int main()
{
   int value[10]={ 82, 25, 26, 7, 67, 55, 31, 19, 99 };

   vector<int> v1(&value[0],&value[10]);
   print(v1,"초기 리스트\n");
   sort (v1.begin(), v1.end());
   print(v1, "sort() 적용후 리스트\n");

   vector<int> v2(&value[0],&value[10]);
   stable_sort (v2.begin(), v2.end());
   print(v2, "stable_sort() 적용후 리스트\n");
```

sort()

```
    vector<int> v3(&value[0],&value[10]);
    partial_sort (v3.begin(), v3.begin()+3, v3.end());
    print(v3, "partial_sort() 적용후 리스트\n");

    vector<int> v4(&value[0],&value[10]);
    nth_element(v4.begin(), v4.begin()+5, v4.end());
    print(v4, "nth_element() 적용후 리스트\n");

    return 0;
}
```

실행 결과

```
초기 리스트
( 82 25 26 7 67 55 31 19 99 0 )
sort() 적용후 리스트
( 0 7 19 25 26 31 55 67 82 99 )
stable_sort() 적용후 리스트
( 0 7 19 25 26 31 55 67 82 99 )
partial_sort() 적용후 리스트
( 0 7 19 82 67 55 31 26 99 25 )
nth_element() 적용후 리스트
( 0 7 19 25 26 31 55 67 82 99 )
```

함수 객체

sort.cpp

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <functional>
#include "myheader.h"
using namespace std;

int main()
{
   vector<int> vec;
   for(int i=0;i<10; i++)
         vec.push_back(rand()%100);

   sort (vec.begin(), vec.end(), greater<int>());
   print(vec);
   return 0;
}
```

함수 객체

실행 결과

```
( 78 69 67 64 62 58 41 34 24 0  )
계속하려면 아무 키나 누르십시오 . . .
```

내장된 함수 객체

내장 함수 객체	설명
plus<type>()	e1 + e2
minus<type>()	e1 - e2
multiplies<type>()	e1 * e2
divides<type>()	e1 / e2 (몫)
modulus <type>()	e1 % e2 (나머지)
negate<type>()	- e
equal_to<type>()	e1 == e2
not_equal_to<type>()	e1 ! = e2
less<type>()	e1 < e2
less_equal<type>()	e1 <= e2
greater<type>()	e1 > e2
greater_equal<type>()	e1 >= e2
logical_not<type>()	! e
logical_and<type>()	e1 && e2
logical_or<type> ()	e1 || e2

예제

```
int main()
{
    vector<int> vec;
    for(int i=0;i<10; i++)
        vec.push_back(rand()%100);
    vector<int>::iterator it;

    for (it=vec.begin();;it++) {
        it=find_if(it, vec.end(), bind2nd (greater<int>(), 50)) ;
        if (it==vec.end()) break;
        cout << *it << " 발견" << endl;
    }
    return 0;
}
```

실행 결과

```
67 발견
69 발견
78 발견
58 발견
62 발견
64 발견
```

Q & A