

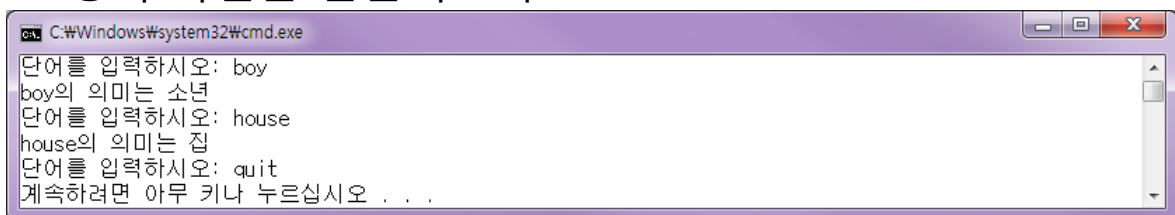
## 제15장 STL과 람다식

1. STL의 개념을 이해하고 사용할 수 있다.
2. 람다식을 이해하고 사용할 수 있다.
3. 각종 STL 알고리즘을 이해하고 사용할 수 있다.

1

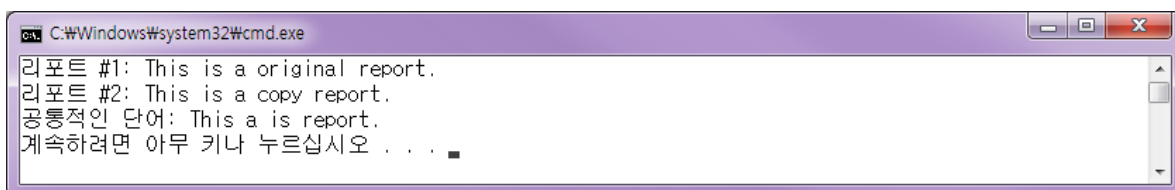
## 이번 장에서 만들어 볼 프로그램

(1) 영어 사전을 만들어보자.



```
C:\Windows\system32\cmd.exe
단어를 입력하시오: boy
boy의 의미는 소년
단어를 입력하시오: house
house의 의미는 집
단어를 입력하시오: quit
계속하려면 아무 키나 누르십시오 . . .
```

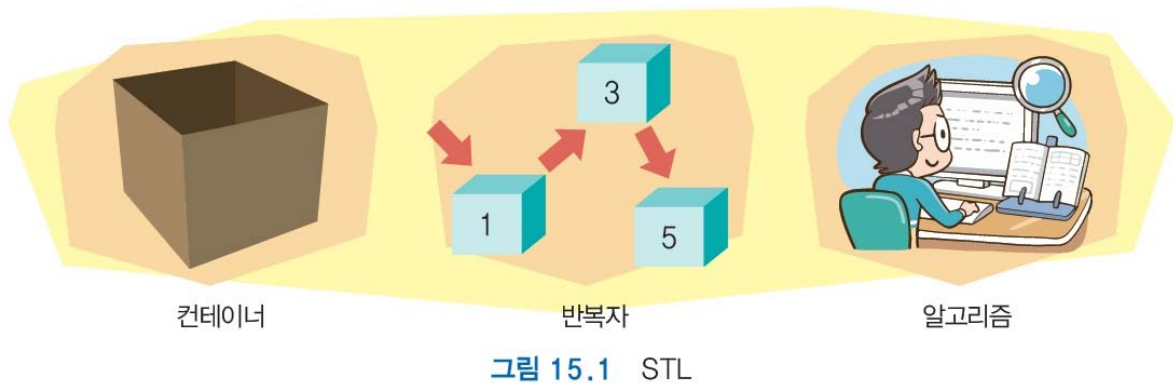
(2) 리포트 복사를 검사하는 프로그램을 작성해보자.



```
C:\Windows\system32\cmd.exe
리포트 #1: This is a original report.
리포트 #2: This is a copy report.
공통적인 단어: This a is report.
계속하려면 아무 키나 누르십시오 . . .
```

2

## 15.2 표준 템플릿 라이브러리(STL)



3

## STL

### □ STL

- 데이터 집합(다양한 **collection** 클래스)을 효율적으로 다루기 위한 **generic**한 라이브러리
- 여러 종류의 컨테이너 클래스, 그것과 동작하기 위한 알고리즘을 제공하기 위해 프레임워크를 제공
- 자료구조(Container) + 알고리즘 + 함수객체 + 반복자(Iterator)

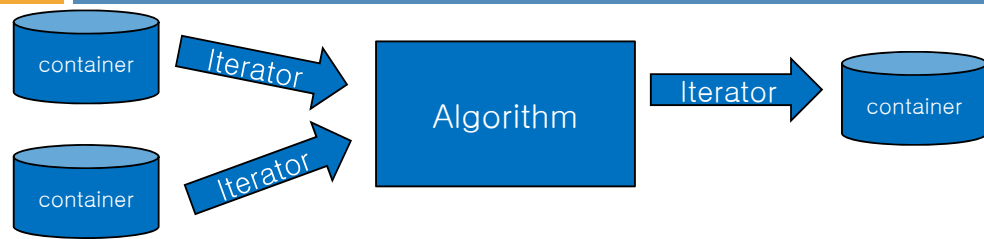
### □ STL 컴포넌트

#### ■ STL

- 서로 다르지만, 잘 구조화된 컴포넌트(컨테이너, 반복자, 함수객체, 알고리즘)간의 협력에 기반한 템플릿 라이브러리

4

# STL 컴포넌트



- 컨테이너 : 특정 타입의 원소들의 집합을 다룸
- 반복자 : 컬렉션 객체가 소유한 원소를 순회하는데 사용
- 알고리즘 : 반복자를 사용하여 컬렉션 객체의 원소를 처리
  - ▣ STL의 모든 컴포넌트는 템플릿 (모든 타입에 대해서 제네릭)
    - 더욱 **generic**한 컴포넌트들: 어댑터, 함수-객체
  - ▣ 객체지향 프로그래밍 아이디어 (데이터와 알고리즘 조합)와 모순

5

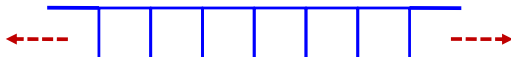
## 컨테이너

### 시퀀스 컨테이너(Sequence Container)

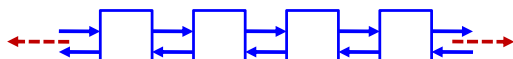
vector



deque



list



<array> <forward\_list>

### 비정렬 연관 컨테이너 (Unordered Associative Container)

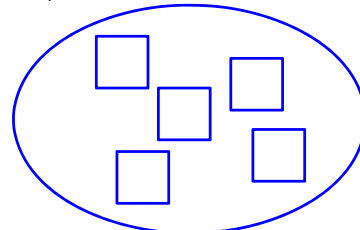
<unordered\_map>

<unordered\_set>

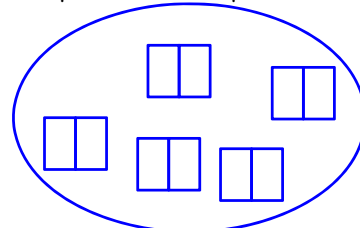
### 연관 컨테이너

(Associative Container)

set/multiset



map/multimap



6

# 컨테이너

## □ 시퀀스 컨테이너

- ▣ vector, deque, list, array, forward\_list
- ▣ 원소들이 자신만의 위치와 순서를 가지는 컬렉션

## □ 연관 컨테이너

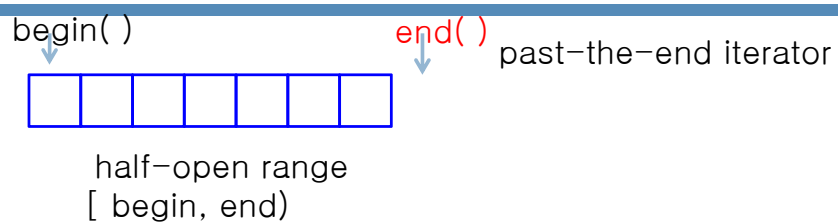
- ▣ set/multiset, map/multimap
- ▣ 원소들이 자동정렬 됨 (검색에 좋음)
  - 원소 순서는 값에만 의존
  - 삽입된 순서와는 무관
  - binary tree로 구현됨

## □ 비정렬 연관 컨테이너

- ▣ unordered\_set/multiset, unordered\_map/multimap
- ▣ hash 함수를 이용하여 저장

7

# 반복자



- 컨테이너가 소유한 객체를 순회하는 스마트 포인터
  - 모든 컨테이너는 자신만의 반복자를 제공
- 기본 동작 : \*, ++, ==, !=, = 연산자
- 함수 : for (pos=coll.begin( ); pos != coll.end( ); ++pos)
- 반-개방 범위 (half-open range) 장점
  1. 원소를 순회하는 루프가 간단: end( ) 까지 순회
  2. 빈 범위 처리가 간단: 빈 컨테이너는 begin( ) == end( )
- 모든 컨테이너의 두 종류 반복자
  1. `container::iterator` : 원소를 읽기/쓰기 모드로 순회
  2. `container::const_iterator` : 원소를 읽기 전용 모드로 순회

8

## 반복자(계속)

### □ 반복자의 종류

- ▣ `containter::iterator` // RW
- ▣ `containter::const_iterator` // Read Only

```
list<char>::iterator pos;  
for (pos=coll.begin( ); pos != coll.end( ); ++pos) {  
    *pos = toupper(*pos);  
}
```

```
for (pos=coll.begin( ); pos != coll.end( ); pos++) // OK but 느리다
```

반복자의 이전 값을  
반환하므로 임시객체  
생성

9

## 알고리즘

- 컨테이너 원소에 대해 검색, 복사, 수정, 수치처리 등 제공
- 반복자와 함께 동작하는 전역함수 (멤버함수 아님)
  - ▣ 모든 컨테이너에 대해서 단 하나의 알고리즘 함수만이 존재
  - ▣ 데이터와 동작을 분리 (generic programming)
  - ▣ 사용자가 정의한 컨테이너 타입에 대해서도 동작
- 단점
  - ▣ 사용법이 직관성과는 거리가 멀다
  - ▣ 특정 자료구조와 알고리즘의 결합이 제대로 동작 안함
- 예
  - ▣ `min_element`, `max_element`, `sort`, `find`, `reverse`

# 범위

- 모든 알고리즘은 하나 이상의 반-개방 범위 처리
- 위험성: 범위의 유효함은 사용자가 보장해야 함
- pos25, pos35 순서를 모를 때

- ▣ 랜덤 액세스 반복자

```
if (pos25 < pos35)
```

```
// < 비교 연산 사용
```

- ▣ 양방향 반복자

```
pos25 = find(coll.begin(), coll.end(), 25);
```

```
pos35 = find(coll.begin(), pos25, 35);
```

```
if (pos35 != coll.end() && pos35 != pos25) // pos35 가 pos25 앞에 위치
```

```
// [pos35, pos25) 만 유효
```

11

```
pos = find_if(coll.begin(), coll.end(), // 범위
```

```
compose_f_gx_hx(logical_or<bool>(), //검색 기준
```

```
bind2nd(equal_to<int>(), 25),
```

```
bind2nd(equal_to<int>(), 35)));
```

```
switch (*pos) {
```

```
case 25: pos25 = pos;
```

```
pos35 = find(++pos, coll.end(), 35);
```

```
... break;
```

```
case 35: pos35 = pos;
```

```
pos25 = find(++pos, coll.end(), 25);
```

```
... break;
```

```
default: ... break;
```

```
}
```

12

## 컨테이너

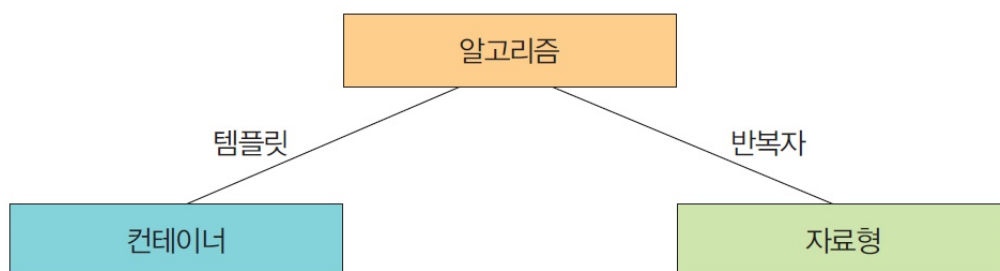
- 컨테이너는 자료를 저장하는 창고와 같은 역할을 하는 구조이다. 즉 배열이나 연결 리스트, 벡터, 집합, 사전, 트리 등이 여기에 해당한다.



13

## 반복자

- 반복자(iterator)는 컨테이너의 요소를 가리키는 데 사용된다. 반복자는 실제로 컨테이너와 알고리즘 사이의 다리 역할을 한다.



14

## 알고리즘

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

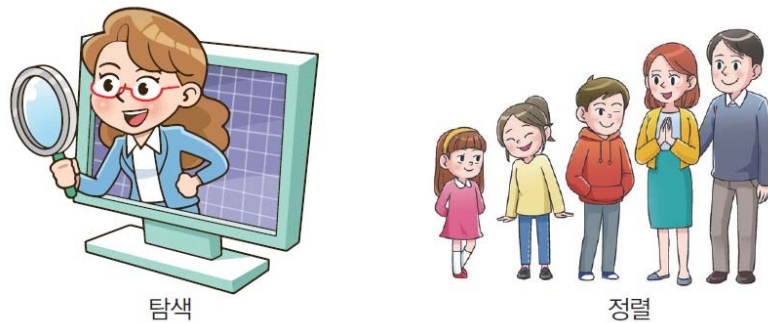


그림 15.2 알고리즘의 예

15

## STL의 장점

1. STL은 프로그래밍에 매우 유용한 수많은 컨테이너와 알고리즘을 제공한다.
2. STL은 객체 지향 기법과 일반화 프로그래밍 기법을 적용하여서 만들어졌으므로 어떤 자료형에 대해서도 사용할 수 있다.
3. STL은 전문가가 만들어서 테스트를 거친 검증된 라이브러리이다.

16

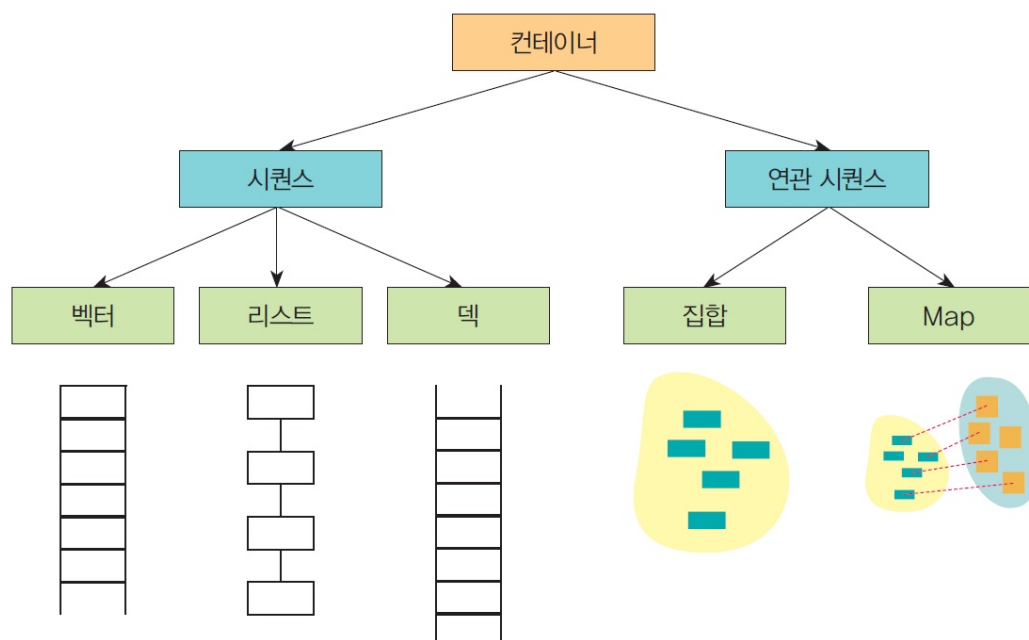


## 15.3 컨테이너

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

17

## 컨테이너의 분류



18

## 예제

```
#include <iostream>
#include <time.h>
#include <list>
using namespace std;
int main()
{
    list<int> values;
    srand(time(NULL));
    for (int i = 0; i < 10; i++) {
        values.push_back(rand()%100);
    }
    values.sort();
    for (auto& e: values){
        std::cout << e << ' ';
    }
    std::cout << endl;
    return 0;
}
```

19

## 예제



20

## 15.4 반복자

- 반복자: 일반화된 포인터(generalized pointer)



21

## 반복자

- 반복자: 일반화된 포인터(generalized pointer)

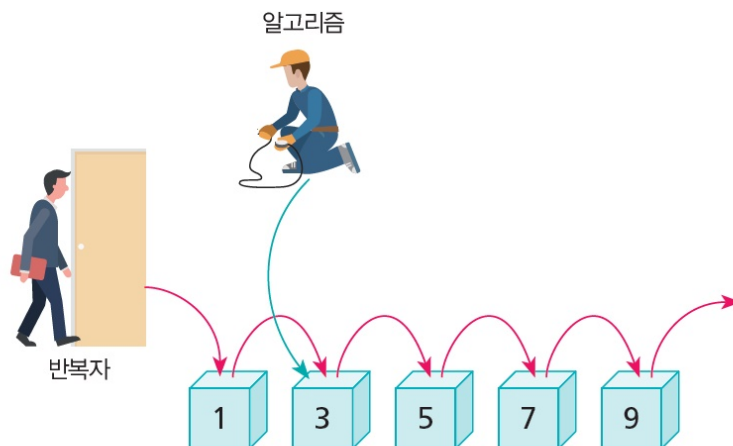


그림 15.10 STL 알고리즘은 반복자를 통하여 컨테이너에 접근하여 작업을 한다.

22

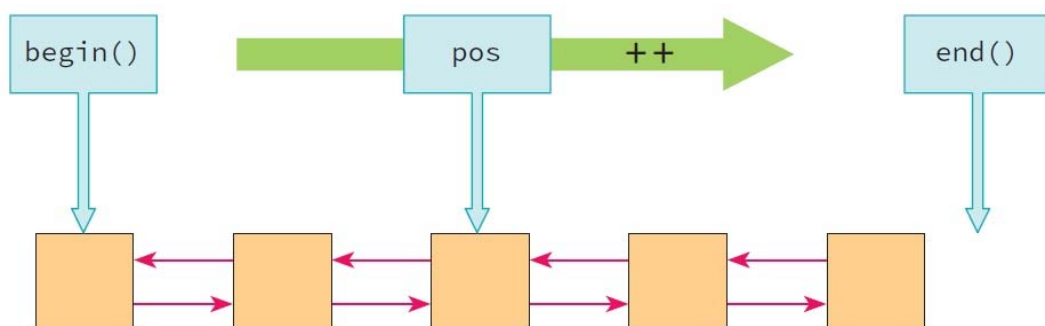
## 시퀀스



23

## 반복자의 연산자

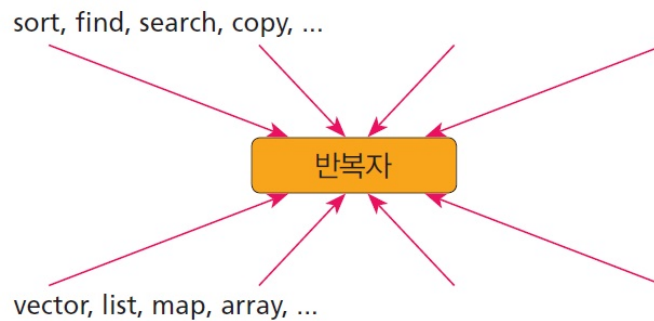
- 컨테이너에서 다음 요소를 가리키기 위한 ++ 연산자
- 컨테이너에서 이전 요소를 가리키기 위한 -- 연산자
- 두개의 반복자가 같은 요소를 가리키고 있는 지를 확인하기 위한 ==와 != 연산자
- 반복자가 가리키는 요소의 값을 추출하기 위한 역참조 연산자 \*



24

## 예제

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



25

## 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.

26

## Old C++ 버전

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> v;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    for (vector<int>::iterator p = v.begin(); p != v.end(); ++p)
        cout << *p << endl;
    return 0;
}
```

27

## C++14 버전 #1

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> v;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    for (auto p = v.begin(); p != v.end(); ++p)
        cout << *p << endl;
    return 0;
}
```

28

## C++14 버전 #2

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> v;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    for (auto& n : v)
        cout << n << endl;
    return 0;
}
```

29

## 15.5 컨테이너의 공통 멤버 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)	대입 연산자의 중복 정의

30

## 15.7 덱

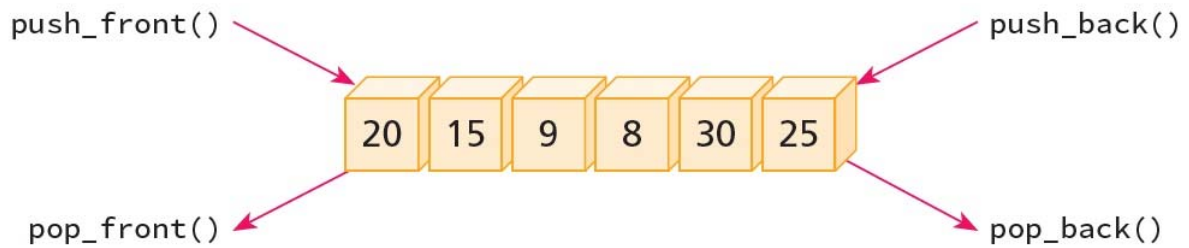


그림 15.4 덱

31

## 예제

```
#include <iostream>
#include <deque>
using namespace std;
int main()
{
    deque<int> dq = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    dq.pop_front();           // 앞에서 삭제
    dq.push_back(11);         // 끝에서 추가
    for (auto& n : dq)
        cout << n << " ";
    cout << endl;
    return 0;
}
```



32



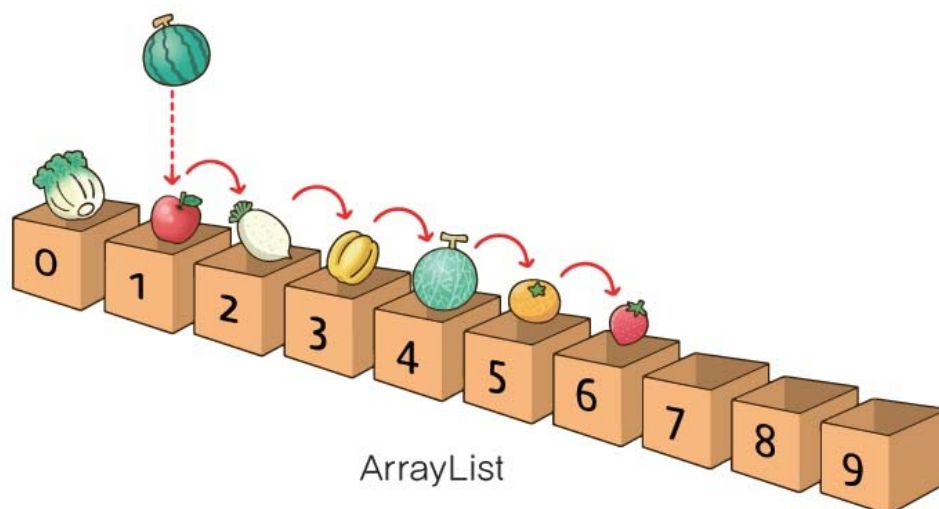
## 예제

```
int main()
{
    deque<string> dq = { "naver", "daum", "cnn", "yahoo", "google" };
    dq.push_front("infinity");      // 앞에서 추가
    dq.pop_back();                  // 끝에서 삭제
    for (auto& e : dq)
        cout << e << " ";
    cout << endl;
    return 0;
}
```



33

## 15.8 리스트



**그림 15.5** 배열의 중간에 삽입하려면 원소들을 이동하여야 한다.

34

# 리스트

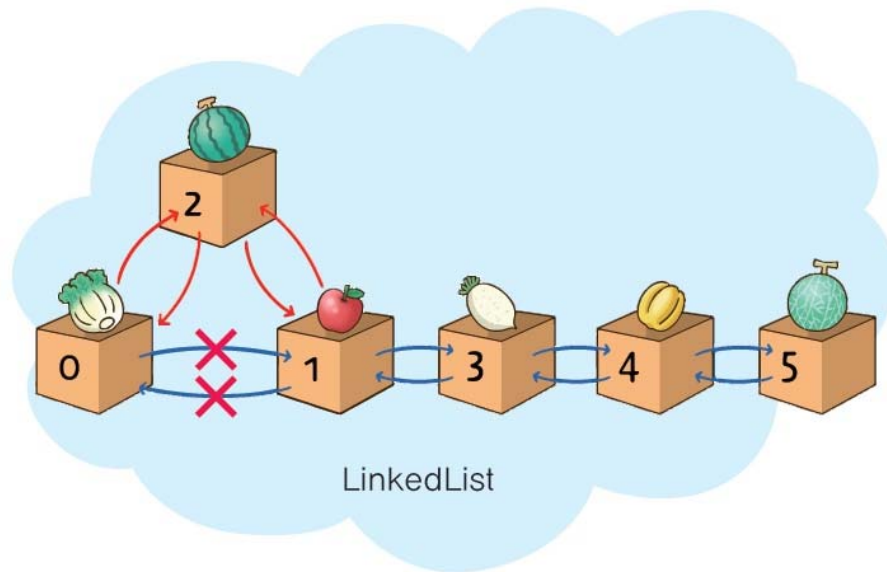


그림 15.6 연결 리스트 중간에 삽입하려면 링크만 수정하면 된다.

35

## 예제

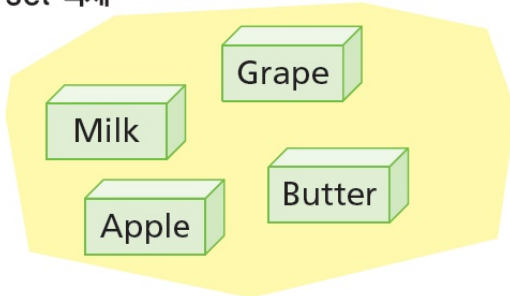
```
#include <list>
using namespace std;
int main()
{
    list<int> my_list={ 10, 20, 30, 40 };
    auto it = my_list.begin();
    it++;
    it++;
    my_list.insert(it, 25);
    for (auto& n : my_list)
        cout << n << " ";
    cout << endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe  
10 20 25 30 40  
계속하려면 아무 키나 누르십시오 . . .

36

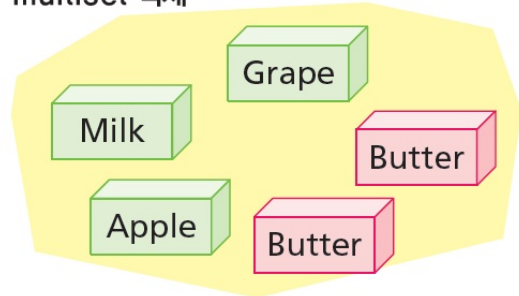
## 15.9 집합

set 객체



집합은 순서가 없고  
중복을 허용하지 않음

multiset 객체



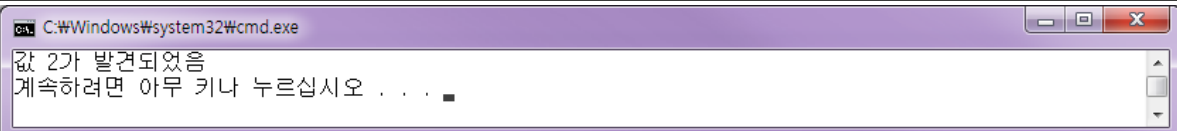
다중집합은  
중복을 허용함

그림 15.7 집합과 다중집합

37

## 예제

```
int main()
{
    set<int> my_set;
    my_set.insert(1);
    my_set.insert(2);
    my_set.insert(3);
    auto pos = my_set.find(2);
    if (pos != my_set.end())
        cout << "값 " << *pos << "가 발견되었음" << endl;
    else
        cout << "값이 발견되지 않았음" << endl;
    return 0;
}
```



38

## Lab: 리포트 복사 검사 프로그램



39

## 예제

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
#include <sstream>
#include <string>
#include <set>
using namespace std;
int main()
{
    string report1 = "This is a original report.";
    string report2 = "This is a copy report.";
    vector<string> v1;
    vector<string> v2;
    // 첫 번째 리포트를 단어로 분리하는 과정
    istringstream iss1(report1);
    for (string s; iss1 >> s; )
        v1.push_back(s);
```

40

## 예제

```
// 두 번째 리포트를 단어로 분리하는 과정
istringstream iss2(report2);
for (string s; iss2 >> s; )
    v2.push_back(s);
sort(v1.begin(), v1.end());
sort(v2.begin(), v2.end());
vector<string> common;
set_intersection(v1.begin(), v1.end(),
                 v2.begin(), v2.end(),
                 back_inserter(common));
cout << "report1=" << report1 << endl;
cout << "report2=" << report2 << endl << endl;
```

41

## 예제

```
cout << "공통적인 단어: ";
for (string e : common)
    cout << e << ' ';
cout << endl;
return 0;
}
```

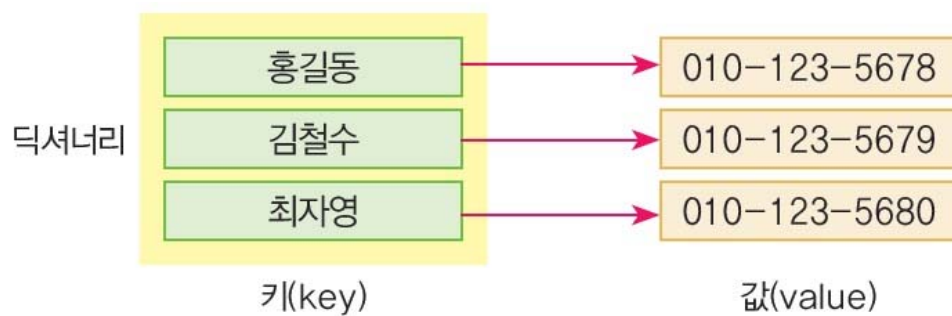
42

- 
- 두 개의 파일이 서로 일치하는지 비교하는 프로그램을 작성하라. report\_check.cpp를 참조한다.

43

## 15.10 Map

---



44

## 예제

```
#include <iostream>
#include <map>
#include <string>
#include <iterator>
using namespace std;
int main()
{
    map<string, string> myMap;
    myMap.insert(make_pair("김철수", "010-123-5678"));
    myMap.insert(make_pair("홍길동", "010-123-5679"));
    myMap["최자영"] = "010-123-5680";
}
```

45

## 예제

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



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The window contains the following text output from the program:

```
김철수 :: 010-123-5678
최자영 :: 010-123-5680
홍길동 :: 010-123-5679
단어 '김영희'는 발견되지 않았습니다.
계속하려면 아무 키나 누르십시오 . . .
```

46

## Lab: 영어사전

- Map을 가지고 영어 사전을 구현해보자. 사용자로부터 단어를 받아서 단어의 설명을 출력한다.



47

## 예제

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

48



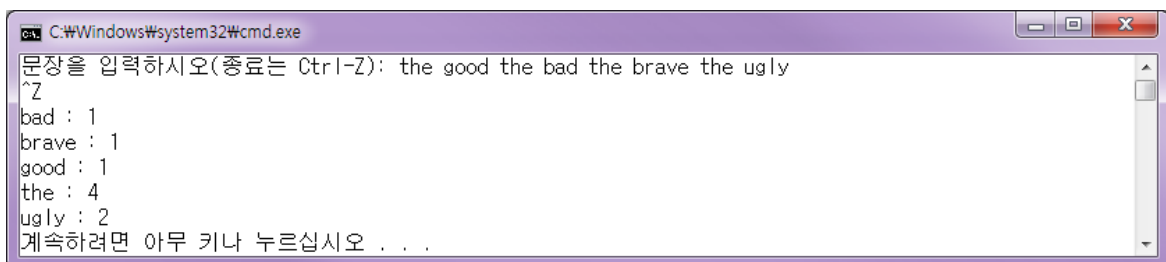
## 예제

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

49

## Lab: 단어 빈도 계산하기

- Map을 이용하여서 사용자로부터 문장을 받아들이고 각 단어가 나오는 빈도를 계산하는 프로그램을 작성하여 보자.



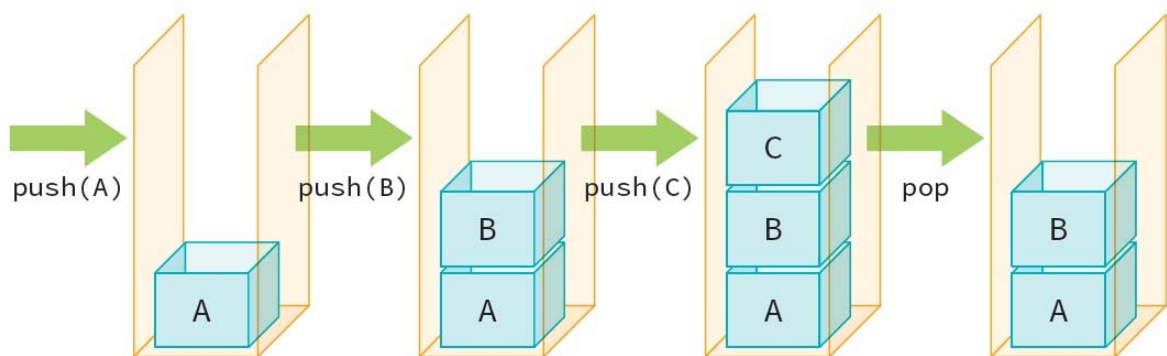
50

## 예제

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main()
{
    map<string, int> table;
    string s;
    cout << "문장을 입력하시오(종료는 Ctrl-Z): ";
    while (true) {
        cin >> s;
        table[s]++;
        if (cin.eof()) break;
    }
    for (auto& iter = table.begin(); iter != table.end(); iter++) {
        cout << iter->first << " : " << iter->second << endl;
    }
    return 0;
}
```

51

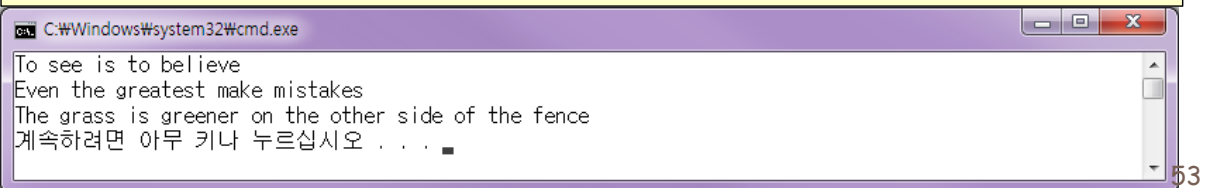
## 15.11 컨테이너 어댑터: 스택



52

## 예제

```
int main()
{
    stack<string> st;
    string sayings[3] =
    { "The grass is greener on the other side of the fence",
      "Even the greatest make mistakes",
      "To see is to believe" };
    for (auto& s : sayings)
        st.push(s);
    while (!st.empty()) {
        cout << st.top() << endl;
        st.pop();
    }
    return 0;
}
```



C:\Windows\system32\cmd.exe

To see is to believe  
Even the greatest make mistakes  
The grass is greener on the other side of the fence  
계속하려면 아무 키나 누르십시오 . . .

53

## 큐

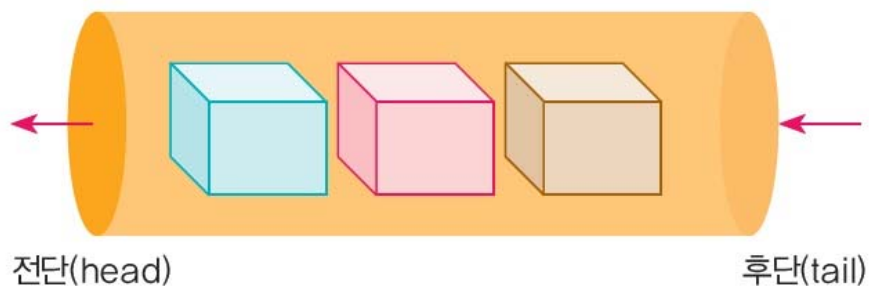
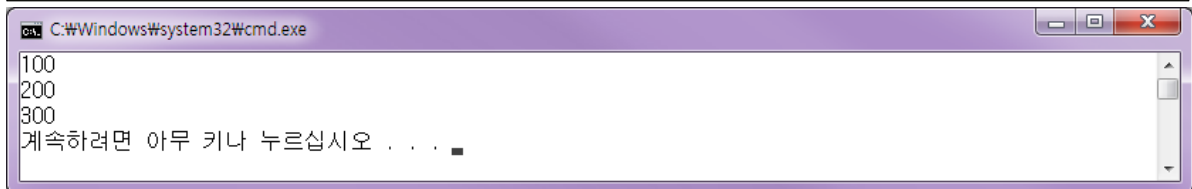


그림 15.8 큐

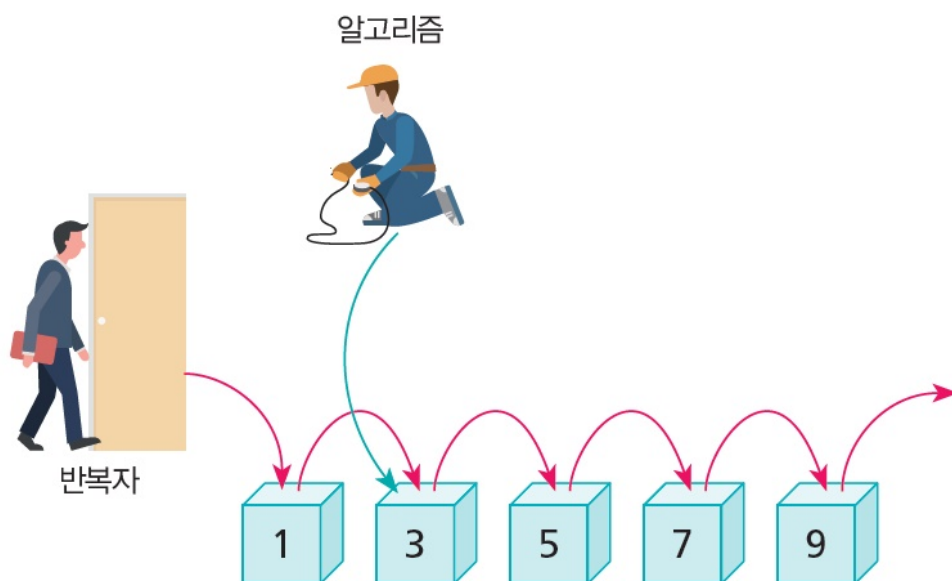
## 예제

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



55

## 15.12 STL 알고리즘



**그림 15.10** STL 알고리즘은 반복자를 통하여 컨테이너에 접근하여 작업을 한다.

56

## find()와 find\_if() 함수

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

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



C:\Windows\system32\cmd.exe  
수박이 3에 있습니다.  
계속하려면 아무 키나 누르십시오 . . .

57

## count() 함수

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

int main()
{
    vector<int> vec;
    for (int i = 0; i < 10; i++)
        vec.push_back(i);
    size_t n = count_if(vec.begin(), vec.end(), is_even<int>);
    cout << "값이 짝수인 요소의 개수: " << n << endl;
    return 0;
}
```



C:\Windows\system32\cmd.exe  
값이 짝수인 요소의 개수: 5  
계속하려면 아무 키나 누르십시오 . . .

58

## binary\_search()

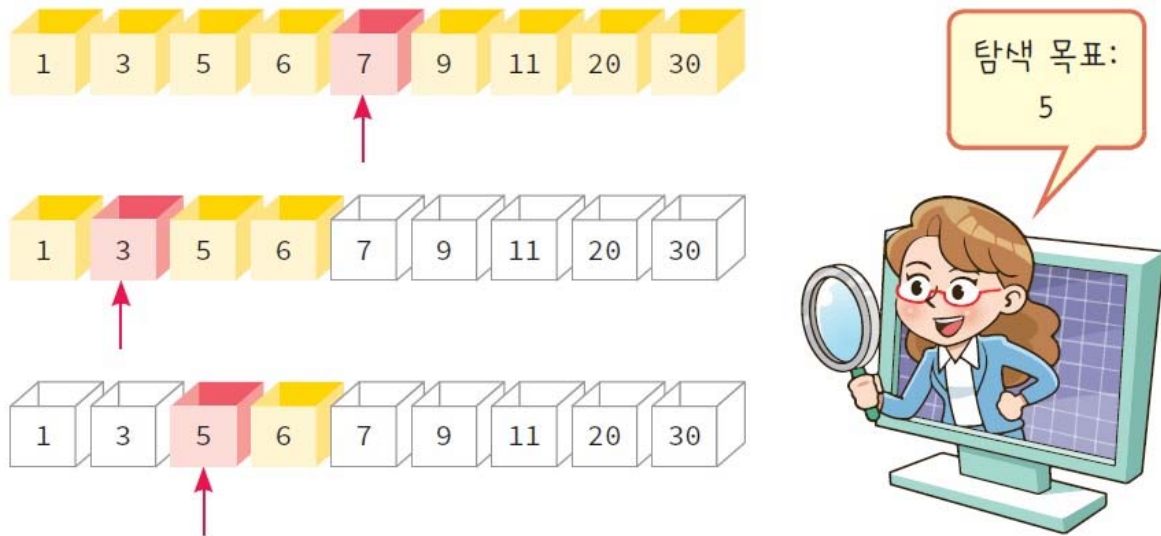
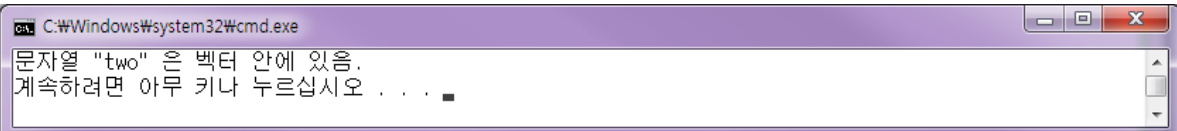


그림 15.11 이진 탐색의 개념

59

```
bool comp(string s1, string s2) {  
    return (s1 == s2);  
}  
  
int main(void) {  
    vector<string> v = { "one", "two", "three" };  
    bool result;  
  
    result = binary_search(v.begin(), v.end(), "two", comp);  
    if (result == true)  
        cout << "문자열 \"two\" 은 벡터 안에 있음." << endl;  
    return 0;  
}
```



60

## for\_each() 함수

```
void printEven(int n) {
    if (n % 2 == 0)
        cout << n << ' ';
}
int main(void) {
    vector<int> v = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    for_each(v.begin(), v.end(), printEven);
    cout << endl;
    return 0;
}
```



61

- 컨테이너의 각각의 요소를 제공하는 기능을 알고리즘 함수를 사용하여 구현하라.

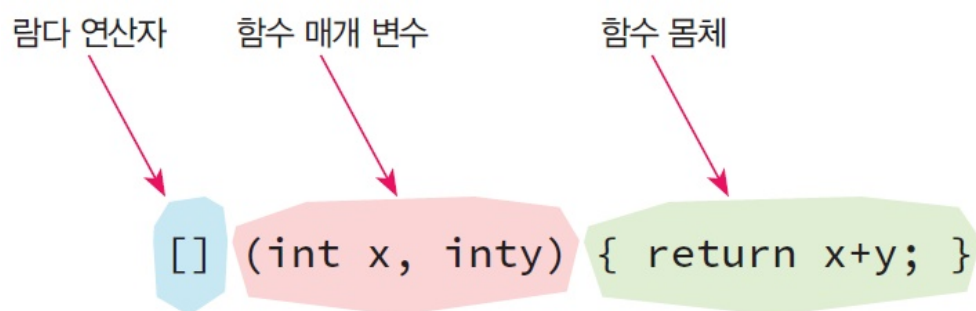
62

## 15.14 람다식



63

## 람다식의 정의



64



```
#include <iostream>
using namespace std;
int main()
{
    auto sum = [](int x, int y) { return x + y; };
    cout << sum(1, 2) << endl;
    cout << sum(10, 20) << endl;
    return 0;
}
```

65

## 예제

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

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

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

66

## 람다식과 함수 객체

---

- 이름 없는 함수
  - ▣ 함수 객체를 대체
  - ▣ [캡처] (매개변수 리스트)mutable throw()->반환형식{구문;}
    - 캡처: 람다식 내부에서 외부에 선언된 요소에 접근
  - ▣ LambdaSample.cpp 실행

```
auto func = [](int nParam) -> int
{
    cout <<"Lambda:" << nParam <<endl;
    return nParam;
};
```

67

- 
- LambdaFunctional.cpp 실행
    - ▣ 람다식을 이용해 함수 자체가 다른 함수의 매개변수
    - ▣ #include <functional>
    - ▣ std::function<int (char\*,int)> param
      - 임의의 함수를 매개변수로 받는 클래스 템플릿

68

## 함수 포인터와 콜백

### □ QuickSortSample.cpp

```
int CompareData(void *pLeft, void *pRight) {  
    return *(int *)pLeft - *(int *)pRight; }  
qsort(aList, 5, sizeof(int), CompareData);  
//정렬 대상에 따라 비교 방법을 사용자가 결정  
//return *(int *)pRight - *(int *)pLeft; 로 바꿔서 실행
```

### □ 콜백(callback)

- 다른 코드의 인수로서 넘겨주는 실행 가능한 코드
- 콜백을 넘겨받는 코드: 콜백을 필요에 따라 즉시 실행할 수도 있고, 아니면 나중에 실행할 수도 있다.
- 콜백수신 코드로 콜백 코드(함수)를 전달할 때는 콜백 함수의 포인터 (핸들), 서브루틴 또는 람다함수의 형태로 넘겨준다.

69

## 함수 객체

### □ 함수 호출 연산자를 다중 정의한 클래스

- Functor
- 클래스이므로 함수의 매개변수로 사용

### □ FunctionObject 실행

```
class Add {  
    int operator()(int a, int b) {return a+b;}  
    double operator()(double a, double b){return a+b;} }  
Add adder;  
cout << adder(3, 4) <<endl;  
cout << adder(3.3, 4.4) <<endl;
```

70

## 함수의 매개변수로 functor 사용

```
void TestFunc(Add &add) {cout <<add(3,4)<<endl;}
int main() {
    Add adder;
    TestFunc(adder);
    return 0;
}
```

71

## 함수 객체의 클래스 상속

### □ FuncObjSort.cpp 실행

```
class CCompareBase {
    virtual int operator()(int a, int b)const = 0; };
class CTestData {
    void Sort(const CCompareBase &cmp) {
        if(cmp(m_array[i], m_array[j]) < 0) ...};
class CMyCmpDesc : public CCompareBase {
    int operator()(int a, int b) const {return a-b;};
class CMyCmpAsce : public CCompareBase {
    int operator()(int a, int b) const {return b-a;};
CTestData data;
CMyCmpDesc desc; data.Sort(desc);
CMyCmpAsce asce; data.Sort(asce);
```

72

## 람다식으로 변경하기

### □ LambdaSort.cpp

- ▣ 함수 객체를 람다식으로 바꾼 정렬
- ▣ 두 항의 비교 방법을 사용자 코드에서 직접 람다식으로 정의해서 Sort() 함수로 전달
  - 별도의 함수 객체를 만들 필요가 없다
  - 상황에 맞게 변경하기 쉬운 유연성

```
void sort(function<int(int, int)> cmp) {  
    if(cmp(m_array[i], m_array[j]) < 0) ...}  
data.Sort([](int a, int b)->int {return a-b;});  
data.Sort([](int a, int b)->int {return b-a;});
```

73

## 람다 캡처

### □ 람다식 내부에서 외부에 선언된 변수에 접근하는 선언

- ▣ 복사 캡처: [사용할 외부의 변수 이름]
- ▣ 참조 캡처: [&사용할 외부의 참조 변수]
- ▣ 디폴트 복사 캡처: 모든 외부 변수사용 [=]
- ▣ 디폴트 참조 캡처: 참조로 모두 사용 [&]

#### ▣ LambdaCap01.cpp

```
int nData = 10;  
auto TestFunc = [nData](void)->void {cout<<nData;}
```

#### ▣ LambdaCap02.cpp

```
auto TestFunc = [nData](void)mutable -> void{++nData;}  
//mutable: 변경가능하나 영향은 람다식 내부까지만
```

74

## 복사 캡처할 변수가 여러 개

- LambdaCap03.cpp

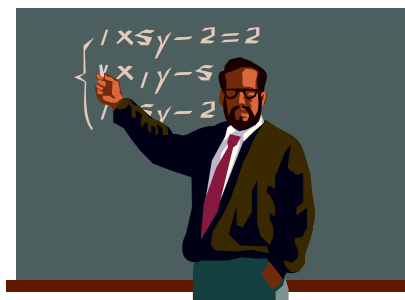
```
int x=10, y=20;
auto TestFunc = [x, y]() -> int {return x+y;}
```
- 일부 참조, 일부는 복사 캡처

```
int x, y, z;
[&, z] {};
[=, &z] {};
```
- 중복 지정은 안됨

```
[x, x] {};
[x, &x] {};
[&, &y] {};
[=, y] {};
```

75

## Q & A



76