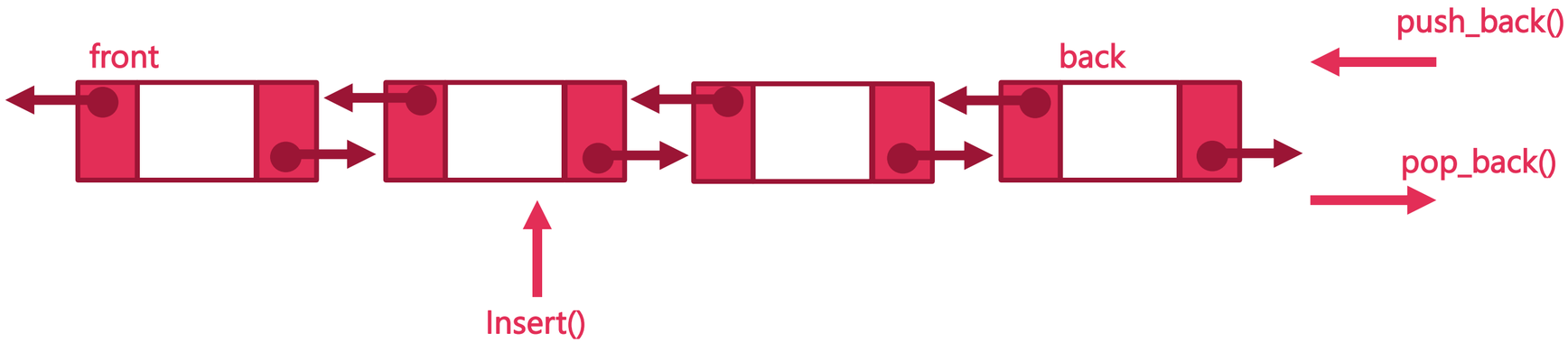
STL

- list, set, multiset, map, multimap -

list란?

노드 기반, 시퀀스 컨테이너로 각각의 객체(원소)들을 각각의 노드에 순차적으로 저장하는 이중 연결 리스트



list에 추가된 멤버함수

- remove(x)
  : x 원소를 모두 제거한다.

- remove_if(pred)
  : pred(단항 조건자)가 '참'인 모든 원소를 제거한다.

- splice(iter1,it2, iter2)
  : iter1이 가리키는 위치에 iter2가 가리키는 위치의 it2의 원소를 잘라 붙인다.

- reverse()
  : 모든 원소의 순서를 반대로 뒤집는다.

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

bool Predicate(int n)
{
    return 10<=n && n<=30;
}
int main()
{
    list<int> it1;
    // 30 50 10 20 40 30 출력
    it1.push_back(30);
    it1.push_back(50);
    it1.push_back(10);
    it1.push_back(20);
    it1.push_back(40);
    it1.push_back(30);

    // 50 10 20 40 출력
    it1.remove(30);
    // 50 40 출력
    it1.remove_if(Predicate);
    // 50 300 40 출력
    list<int> it2={300, 500, 100, 200, 400};
    list<int>::iterator iter1=it1.begin();
    ++iter1;
    list<int>::iterator iter2=it2.begin();
    it1.splice(iter1, it2, iter2);
    // 40 300 50 출력
    it1.reverse();
    return 0;
}
```

list에 추가된 멤버함수

- unique()
  : 인접한 원소를 중복되지 않게 하나씩만 남긴다.

- sort()
  : 자체 정렬 함수로, 기본 정렬은 오름차순이다.

- merge()
  : 정렬된 두 list를 하나의 정렬된 list로 합병해준다.

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

int main()
{
    list<int> it1;
    // 30 50 10 30 30 30 출력
    it1.push_back(30);
    it1.push_back(50);
    it1.push_back(10);
    it1.push_back(30);
    it1.push_back(30);
    it1.push_back(30);

    // 30 50 10 30 출력
    it1.unique();
    // 10 30 30 50 출력
    it1.sort();

    list<int> it2={300, 500, 100, 200, 400};
    it2.sort();
    // it1: 10 30 30 50 100 200 300 400 500 출력
    // it2:
    it1.merge(it2);

    return 0;
}
```

list vs vector

- **배열 기반 컨테이너가 아닌 노드 기반 컨테이너**
- **중간에 원소를 삽입, 제거 시 효율적으로 동작한다.**
- **at(), [] 연산자는 존재하지 않는다.**
- **양방향 반복자를 사용한다.**
  - 양방향 반복자: ++, --을 이용하여 순방향, 역방향으로 이동 가능한 반복자
  - 임의 접근 반복자: 양방향 반복자 기능에 +, -, +=, -=, [] 연산이 가능한 반복자

list vs vector

- 어느 위치에서도 삽입/제거가 빠르다.
- 개별 원소에 index로 접근이 불가능하다.

- 끝 쪽에 삽입/제거가 빠르다.
- 개별 원소에 index로 접근이 가능하다.

연관 컨테이너란?

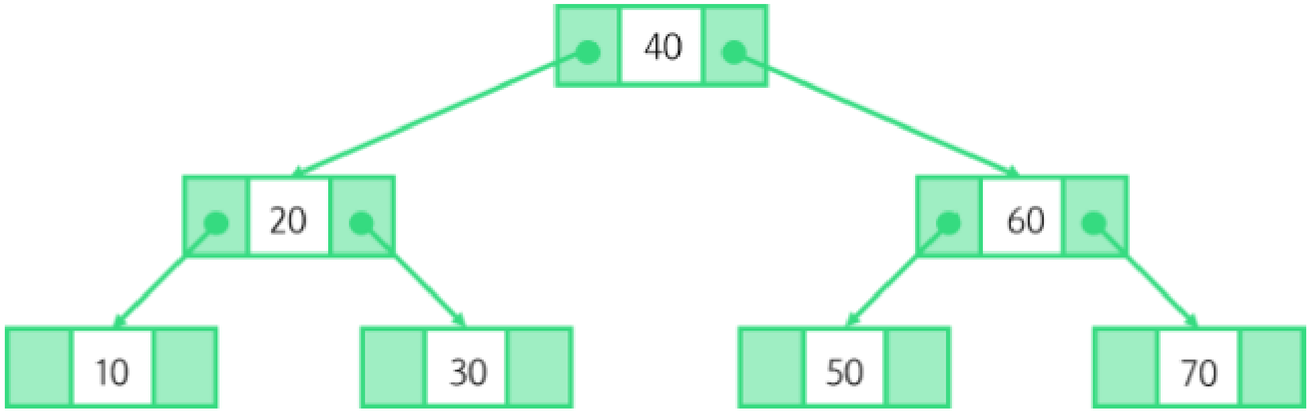
시퀀스 컨테이너와 달리 컨테이너의 원소들을 순차적으로 삽입하지 않고,
원소 추가 시 특정 정렬 기준(디폴트 less)에 의해 자동 정렬되는 컨테이너

**** 연관 컨테이너는 균형 이진 트리를 사용하며,**

원소를 빠르게 찾을 수 있습니다.(로그 시간 복잡도) **

set이란?

원소(key)로 이루어진 컨테이너로, 원소의 중복은 허용하지 않는다.



- 기본 정렬 기준: less
- 탐색: 중위순회(inorder)

set 사용법

set의 초기화(생성자)

- set <type, 조건자> s
  : 빈 컨테이너

- set <type, 조건자> s2(s)
  : s2는 s 컨테이너의 복사본

- set <type, 조건자> s(b,e)
  : 반복자 구간 [b,e)로 초기화된 원소를 갖는다.

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

int main()
{
    // s 출력: 10 20 30
    set<int> s={10,20,30};
    // s2 출력: 10 20 30
    set<int> s2(s);
    // s3 출력: 60 50 40
    set<int, greater<int>> s3={40,50,60};

    return 0;
}
```

set 사용법

set 요소 접근

- begin()
  : 첫 원소를 가리키는 반복자를 반환

- end()
  : 끝을 가리키는 반복자를 반환

- *연산자로 원소에 접근할 수 있다.

set 사용법

set 요소 삽입 및 삭제

- insert(k)
  : k를 삽입하며, 원소를 가리키는 반복자와 성공 여부의 bool 값인 pair 객체를 반환한다.

- Insert(p,k)
  : p가 가리키는 위치부터 빠르게 k를 삽입하며, 삽입한 원소를 가리키는 반복자를 반환한다.

- insert(b,e)
  : 반복자 구간 [b,e)의 원소를 삽입한다.

- erase(p)
  : p가 가리키는 원소를 제거하며, 다음 원소를 가리키는 반복자를 반환한다.

- clear()
  : 모든 원소를 제거한다.

set 사용법

set의 요소 검색

- find(k)
  : k의 원소의 위치를 가리키는 반복자를 반환한다.

- lower_bound(k)
  : k의 시작 구간을 가리키는 반복자를 반환한다.

- upper_bound(k)
  : k의 끝 구간을 가리키는 반복자를 반환한다.

- equl_range(k)
  : k 원소의 반복자 구간인 pair 객체를 반환한다.

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

int main()
{
    // int형의 원소를 가진 빈 set 컨테이너 생성
    set<int> s1;

    s1.insert(30);
    s1.insert(40);
    s1.insert(10);
    s1.insert(50);
    s1.insert(20);

    pair<set<int>::iterator, bool> pr;
    pr=s1.insert(60);
    cout<<"s1에 "<<*pr.first<<" 삽입 성공 여부: "<<pr.second<<endl;
    pr=s1.insert(30);
    cout<<"s1에 "<<*pr.first<<" 삽입 성공 여부: "<<pr.second<<endl;

    set<int>::iterator iter;
    int num=70;
    iter=s1.find(num);
    // 원소를 찾지 못하면 끝 표시 반복자를 반환합니다.
    if(iter!=s1.end())
        cout<<"s1에 "<<num<<"이 존재합니다."<<endl;
    else
        cout<<"s1에 "<<num<<"이 존재하지 않습니다."<<endl;

    return 0;
}
```

multiset이란?

set과 다르게 중복 원소를 컨테이너에 저장할 수 있다.

따라서 lower_bound(), upper_bound(), equal_range() 함수를 유용하게 사용할 수 있다.

- lower_bound(k)
  : k의 시작 구간을 가리키는 반복자를 반환한다.

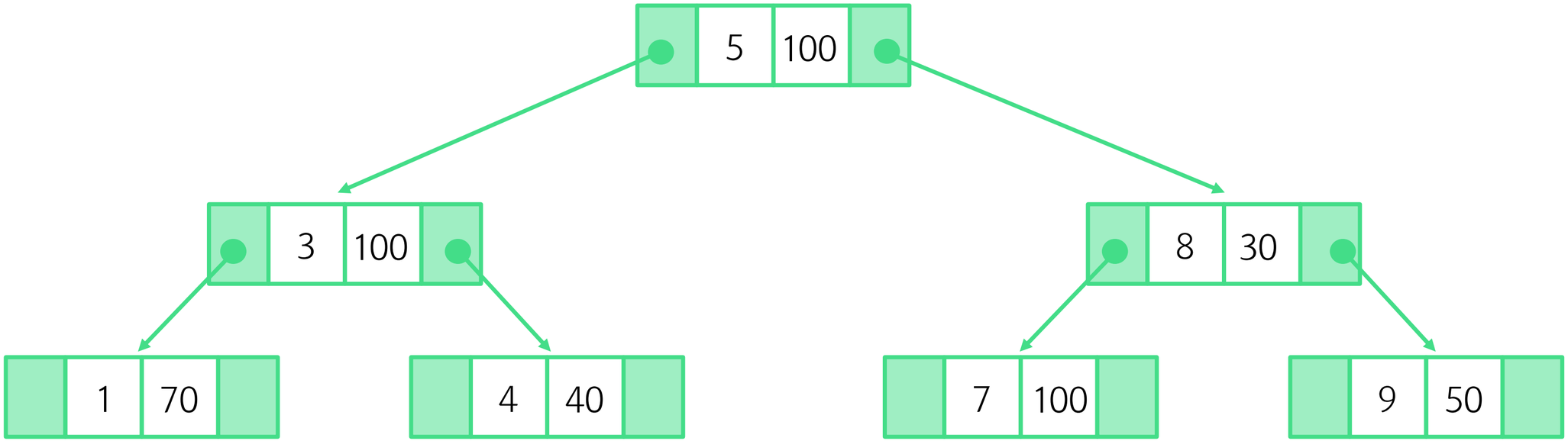
- upper_bound(k)
  : k의 끝 구간을 가리키는 반복자를 반환한다.

- equl_range(k)
  : k 원소의 반복자 구간인 pair 객체를 반환한다.

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

int main()
{
    multiset<int> s={10, 20, 30, 40, 50};

    multiset<int>::iterator iter;
    s.insert(30);
    iter=s.lower_bound(30);
    cout<<"lower_iter: "<<*iter<<endl;
    iter=s.upper_bound(30);
    cout<<"upper_iter: "<<*iter<<endl;

    pair<set<int>::iterator, set<int>::iterator> pr;
    pr=s.equal_range(30);
    cout<<"구간 [lower_iter, upper_iter)의 순차열: ";
    for(iter=pr.first; iter!=pr.second; ++iter)
        cout<<*iter<<" ";
    cout<<endl;

    return 0;
}
```

map이란?

원소를 key와 value 쌍으로 저장하는 컨테이너로, key의 중복은 허용하지 않는다.



map 사용법

map 요소 접근

- m[k] = v
  : m 컨테이너에 원소(k, v)를 추가하거나,
  key에 해당하는 원소의 value를 v로 갱신한다.

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

int main()
{
    // key, value 모두 정수형인 컨테이너 생성
    // 기본 정렬 기준 less
    map<int,int> m;
    // m: (3,100) (5,100) (8,30) 출력
    m.insert(pair<int,int>(5,100));
    m.insert(pair<int,int>(3,100));
    m.insert(pair<int,int>(8,30));

    // m: (3,100) (4,50) (5,100) (8,50) 출력
    m[4]=50;
    m[8]=50;
    map<int,int>::iterator iter;
    for(iter=m.begin(); iter!=m.end(); iter++)
        cout<<"("<<iter->first<<','<<iter->second<<")"<<" ";
    cout<<endl;

    return 0;
}
```

multimap이란?

map과 다르게 중복 key를 컨테이너에 저장할 수 있다.

따라서 lower_bound(), upper_bound(), equal_range() 함수를 유용하게 사용할 수 있다.

- lower_bound(k)
  : k의 시작 구간을 가리키는 반복자를 반환한다.

- upper_bound(k)
  : k의 끝 구간을 가리키는 반복자를 반환한다.

- equl_range(k)
  : k 원소의 반복자 구간인 pair 객체를 반환한다.

set, map?

빠른 검색을 필요로 할 때 사용

중복되지 않게 원소를 저장하고, key만 필요할 경우 – set

중복되지 않게 원소를 저장하고, key와 value 모두 필요할 경우 – map

중복을 허용하고 key만 필요할 경우 – multiset

중복을 허용하고 key와 value 모두 필요할 경우 – multimap

map 예제

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

int main()
{
    map<string,string> m;
    map<string,string>::iterator iter;
    int Case;
    string name;
    string phonenum;

    while(1){
        cout<<"전화번호부"<<endl<<endl;
        cout<<"1. 전화번호 추가"<<endl;
        cout<<"2. 전화번호 검색"<<endl;
        cout<<"3. 전화번호 삭제"<<endl;
        cout<<"4. 전화번호 목록"<<endl;
        cout<<"5. 종료"<<endl;
        cout<<"선택: ";
        cin>>Case;

        switch(Case)
        {
            case 1:
                cout<<"이름: ";
                cin>>name;
                cout<<"전화번호: ";
                cin>>phonenum;
                m.insert(pair<string,string>(name,phonenum));
                cout<<endl<<endl<<endl;
                break;
            case 2:
                cout<<"전화번호 검색(이름): ";
                cin>>name;
                iter=m.find(name);
                cout<<iter->first<<", "<<iter->second;
                cout<<endl<<endl<<endl;
                break;
            case 3:
                cout<<"전화번호 삭제(이름): ";
                cin>>name;
                iter=m.find(name);
                m.erase(iter);
                cout<<endl<<endl<<endl;
                break;
            case 4:
                cout<<"전화번호 목록"<<endl;
                for(iter=m.begin(); iter!=m.end(); iter++)
                    cout<<iter->first<<", "<<iter->second<<endl;
                cout<<endl<<endl<<endl;
                break;
            case 5:
                return 0;
        }
    }
}
```