소프트웨어 공학

자료구조

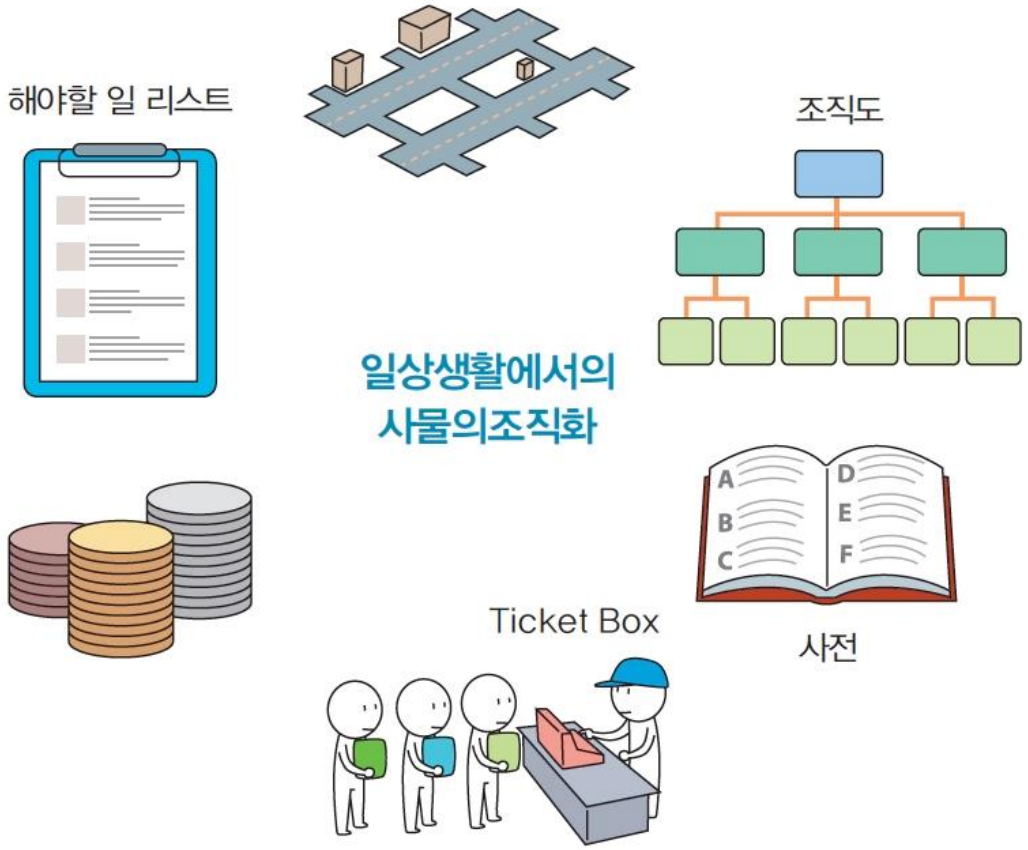
Data Structures

KNU

자료구조란?

- 자료 구조(data structure)
  - 사람이 사물을 정리하는 것과 같이 프로그램에서도 자료를 여러 가지 구조에 따라 정리하는 것
  - 일상생활에서의 정리:
    - 하루에 해야 할 일들을 순차적으로 수첩에 기록하기
    - 책상에 책을 쌓아 놓는 것
    - 상점에서 물건을 구입하기 위해 줄을 서는 것

일상생활에서의 자료구조



자료구조

일상생활 예	자료구조
물건을 쌓아놓는 것	스택
영화관 매표소의 줄	큐
할일 리스트	리스트
영어사전	사전, 탐색 구조
지도	그래프
조직도	트리

자료 구조(Data Structure)

- 기본 자료형(Data type):
  - 정수 자료형(integral) : integer, character, boolean
  - 부동 소수점형(Floating-point): float, double, long double

- 복잡한 데이터를 구성한 자료구조
  - Array
  - List, Stack, Queue, Dequeue
  - Trees: binary tree, binary search tree
  - Priority Queues
  - Hash table
  - Set
  - Graph

기본적인 자료 구조

- 자료형(Data Type): 자료가 가질 수 있는 값과 값에 대한 연산(operation)들의 집합
- C와 C++의 기본 자료형
  - 정수 (int)
    - short int(2 bytes), int(4 bytes), long long (8 bytes)
  - 부동 소수점 (float)
    - float(4 bytes), double(8 bytes)
  - 문자(char)
    - char(0~255)
- C와 C++의 구조체(Structure)

배열(Array)

- 순차적으로 저장된 같은 자료형의 데이터들과 각 데이터에 대응하는 번호(index)로 이루어짐

- 데이터가 N개인 배열을 사용할 때, 일반적으로 0에서 N-1까지의 index를 사용

- 두 가지 생성 방법
  - 정적할당(Static allocation) :
    배열의 크기를 프로그래머가 직접 지정
  - 동적할당(Dynamic allocation) :
    배열의 크기를 실행 중에 사용자가 지정

실습 1: 소수 테이블 구하기

```
#include <iostream>
#define N 1000
using namespace std;

void main( ) {
    int i, a[N];

    for (i = 2; i < N; i++)          /*초기화*/
        a[i] = 1;
    for (i = 2; i < N; i++)
        if (a[i] == 1)
            for(int j = i; j*i < N; j++)    // N-1까지의 i의 배수를 걸러낸다.
                a[i*j] = 0;

    for (i = 2; i < N; i++)
        if (a[i] == 1)
            cout << " " << i;
    cout << endl;
}
```

- **소수의 테이블을 계산하는 방법**
- 기본 개념:

✓ 만약 i 가 소수이면 a[i]에 1을 넣는다.
✓ 만약 i가 소수가 아니면 0을 넣는다.

리스트

- 순서가 있는 항목들의 모임, 즉 “목록”

- 리스트(list)의 예:
  - 숫자들: {1, 2, 3, 4, 5, …, 100}
  - 요일들: {일요일, 월요일, …, 토요일}
  - 매 월: {1월, 2월, 3월, …., 12월}
  - 카드 한 벌의 값: {Ace, 2, 3, …, King}

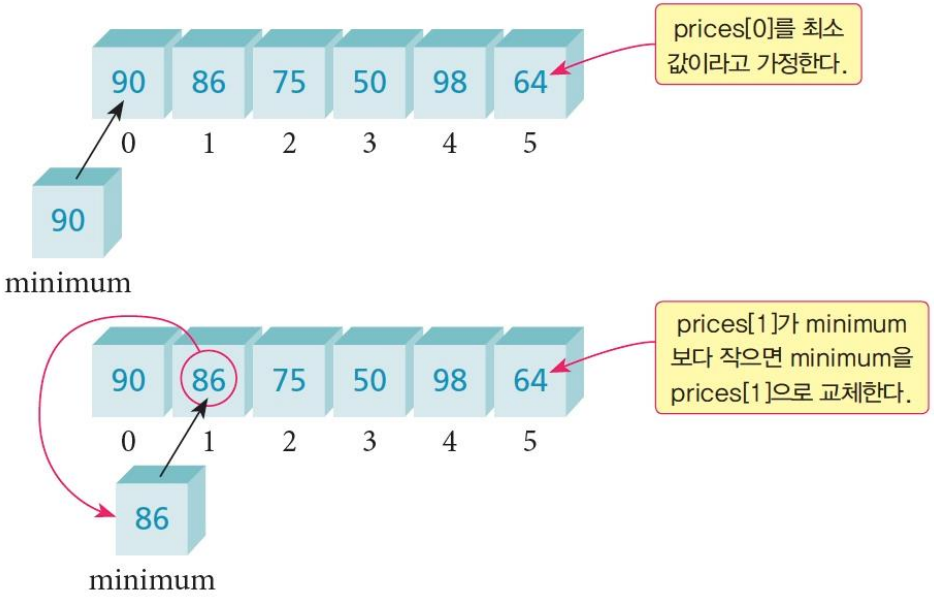
리스트

- 리스트(list)의 예: 최대 흥행 영화 리스트

순위	영화 제목	인덱스
1	아바타(Avatar)	0
2	타이타닉(Titanic)	1
3	스타워즈(Star Wars: The Force Awakens)	2
4	주라기 월드(Jurassic World)	3
5	어벤저스(Marvel's The Avengers)	4

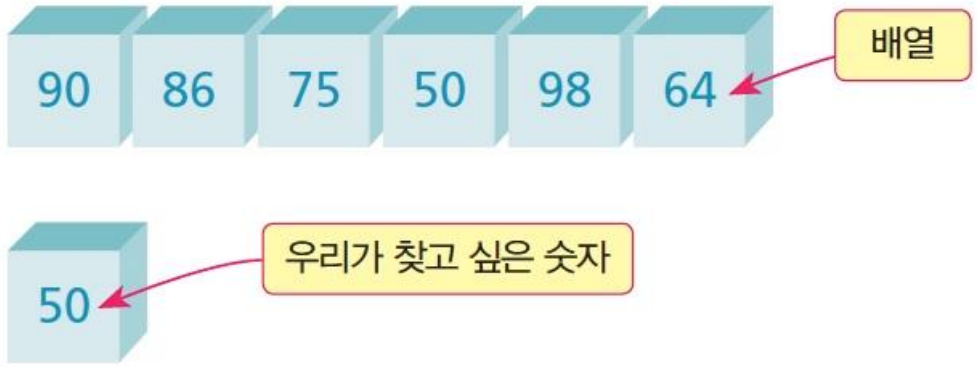
리스트 사용 예

- 리스트에 저장된 값들의 최소값 구하기
  - 리스트의 첫 번째 요소를 최소값으로 가정
  - 리스트의 두 번째 요소부터 마지막 요소까지 이 최소값과 비교
  - 만약 어떤 요소가 현재의 최소값보다 작다면 이것을 새로운 최소값으로 변경
  - 모든 요소들의 검사가 종료되면 최소값을 찾을 수 있다



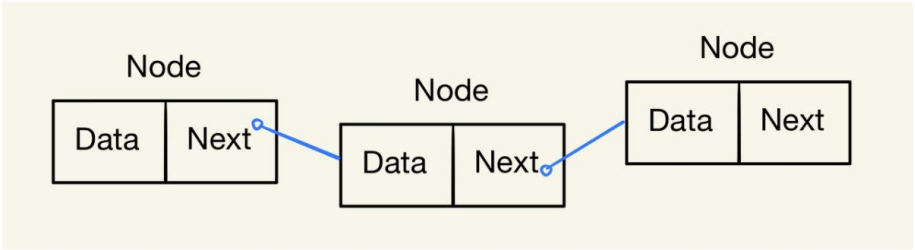
리스트

- 특정값 탐색: 가장 흔한 예->인터넷에서 정보검색
  - 예) 리스트에 숫자들이 저장되어 있고, 이 중에서 하나의 숫자를 찾을 때...



연결 리스트

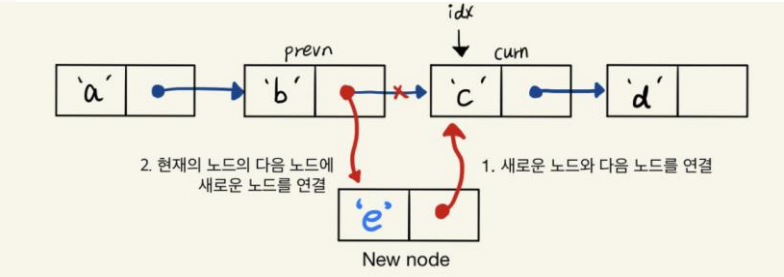
- 연결 리스트
  - 자료들이 연결된 것



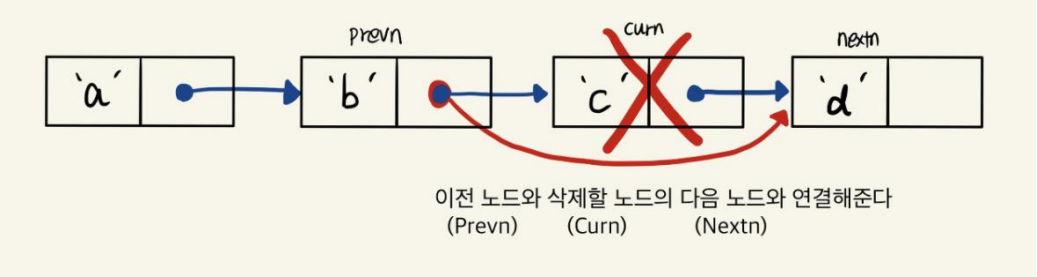
  - 끝말 잇기
    - 기차 → 차고 → 고장 → 장난감 → 감꽃 → 꽃잎

연결 리스트

- 장점
  - 길이를 동적으로 조절 가능
  - 데이터의 삽입과 삭제가 쉬움
- 단점
  - 임의의 요소에 바로 접근할 수 없음
  - 추가 공간 필요
  - 리스트의 뒤에서부터 탐색하기 어려움



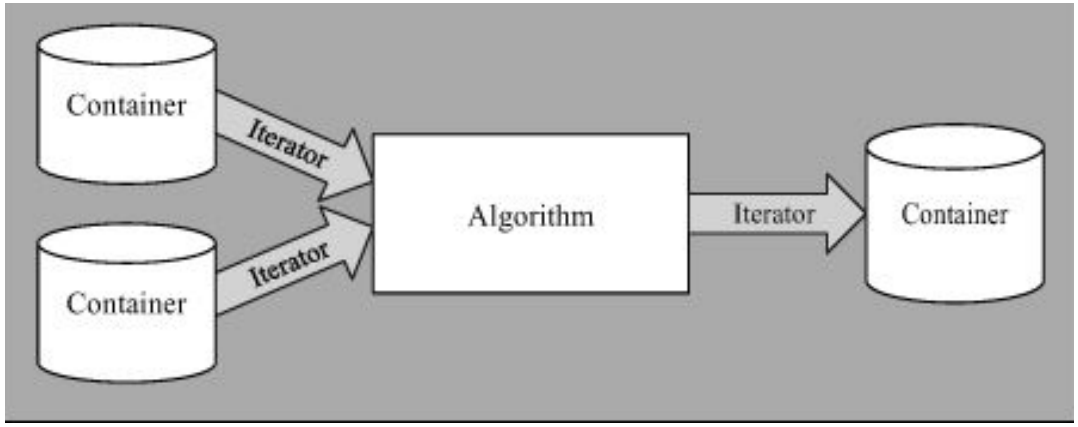
삽입



삭제

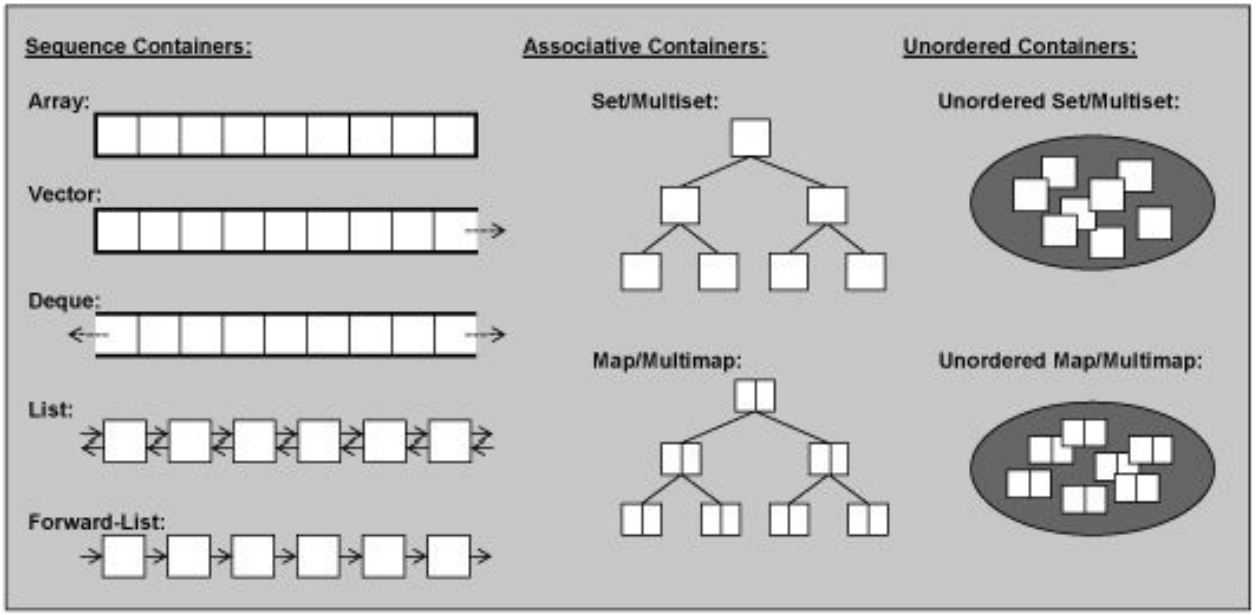
Standard Template Library

- Two important types of data structures in the STL:
  - containers: classes that store data and impose some organization on it
  - iterators: like pointers; mechanisms for accessing elements in a container



Containers

- Two types of container classes in STL:
  - sequence containers: organize and access data sequentially. **`vector`**, **`dequeue`**, and **`list`**
  - associative containers: use keys to allow data for quick access. **`set`**, **`multiset`**, **`map`**, and **`multimap`**



Container Objects 생성

- For example, to create a list of **`int`**, write

```
list<int> mylist;
```

- To create a vector of **`string`** objects, write

```
vector<string> myvector;
```

Iterators

- Generalization of pointers, used to access information in containers
- Four types:
  - forward (uses **++**)
  - bidirectional (uses **++** and **--**)
  - random-access
  - input (can be used with **cin** and **istream** objects)
  - output (can be used with **cout** and **ostream** objects)

Containers and Iterators

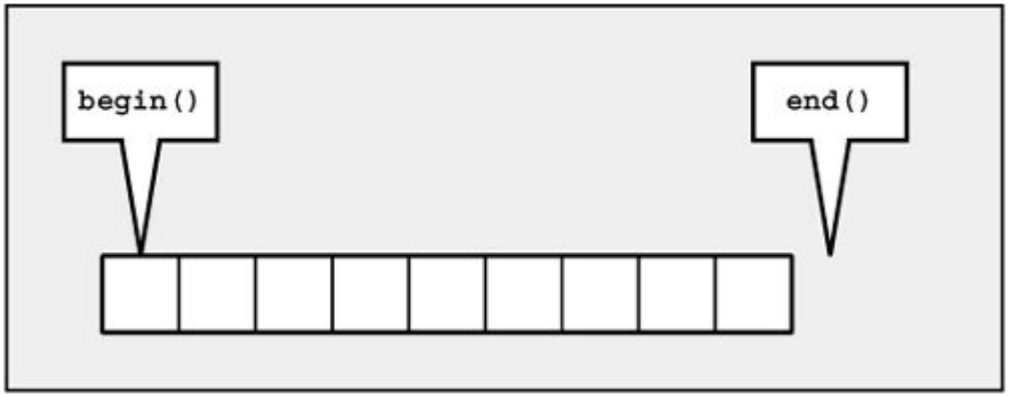
- Each container class defines an iterator type, used to access its contents
- The type of an iterator is determined by the type of the container:

```
list<int>::iterator x;
list<string>::iterator y;
```

`x` is an iterator for a container of type `list<int>`

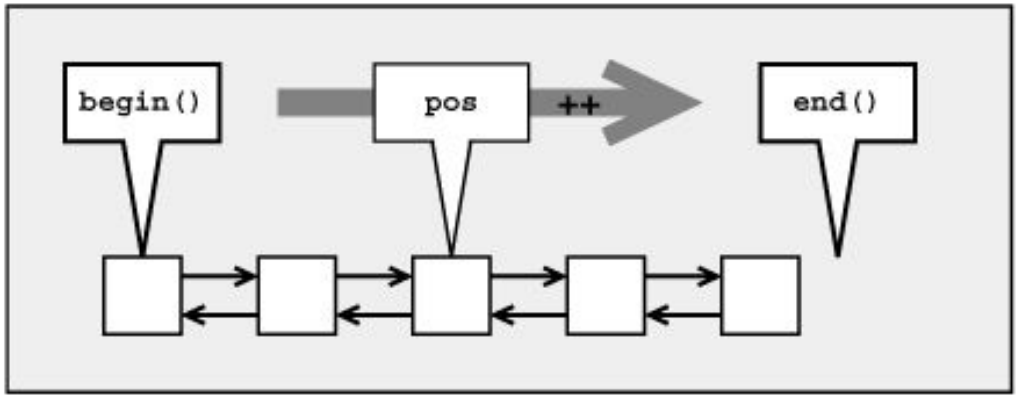
Containers and Iterators

- Each container class defines functions that return iterators:
  - **`begin()`:** returns iterator to item at start
  - **`end()`:** returns iterator denoting end of container



Containers and Iterators

- Iterators support pointer-like operations:
  - `*iter` is the item it points to: this dereferences the iterator
  - `iter++` advances to the next item in the container
  - `iter--` backs up in the container
  - `iter += 2 // impossible`
- The `end()` iterator points to past the end: it should never be dereferenced

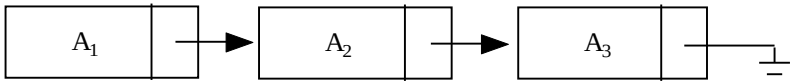


리스트(List)

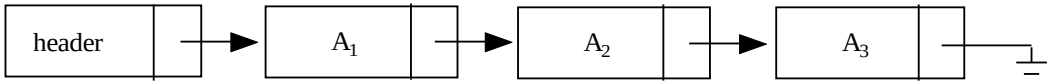
- 사용 가능한 연산(operation):
  - insert: 데이터를 추가
  - remove: 특정 데이터 값을 삭제
  - erase : 특정 iterator의 위치를 삭제
  - sort : 데이터를 정렬함
- 각 데이터는 고정된 위치를 가짐
- 두 가지 구현 방법:
  - 배열 기반 리스트(Array-based list)
  - 연결 리스트(Linked list)

리스트(List)

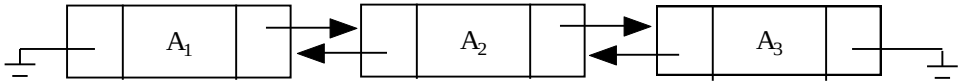
Linked list:



Linked list with a header:



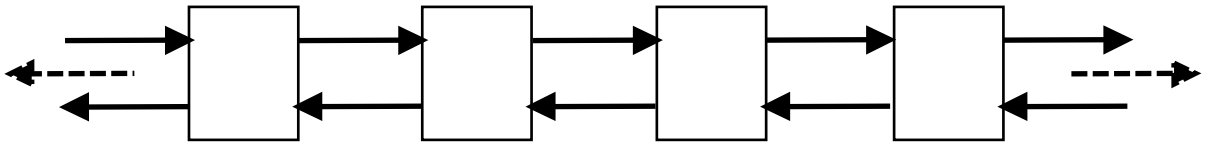
Doubly linked list:



C++의 리스트(List)

- 이중(doubly) 연결 리스트의 형태로 구현
  - 리스트의 각 노드(node)는 자체적인 메모리 공간에 자신의 앞과 뒤의 노드에 대한 링크를 가짐
  - 단점: 임의 접근(random access)를 제공할 수 없음
    일반적으로 임의의 노드에 접근하기 위해서는
    직선적 시간(linear time)이 걸림
    - 따라서 [ ] 연산자를 지원하지 않음
  - 장점: 어떠한 위치에서든 특정 노드의 삽입 또는 삭제가 빠름

- http://www.cplusplus.com/reference/stl/list/



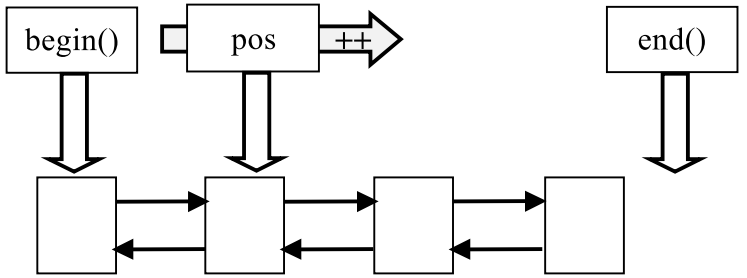
리스트(List) 예제

```
// list1.cpp
#include <iostream>
#include <list>
using namespace std;

void main() {
    list<char> coll;                    // char 타입의 리스트 coll 생성

    for (char c = 'a'; c <= 'z'; c++)
        coll.push_back(c);              // 리스트의 끝에 c 추가

    while (! coll.empty() ) {           // empty() : 리스트가 비었으면 true
        cout << coll.front() << ' ';    // front() : 리스트의 첫번째 데이터 반환
        coll.pop_front();               // pop_front() : 첫번째 데이터 삭제
    }

    cout << endl;
}
```

출력:
a b c d e f g h i j k l m n o p q r s t
u v w x y z

리스트(List) 예제 2

```
// list2.cpp
#include <iostream>
#include <list>
using namespace std;

void main() {
    list<char> coll;

    for (char c='a'; c<='z'; ++c)
        coll.push_back(c);

    list<char>::const_iterator pos;    // list의 iterator 객체인 pos 선언

    for (pos = coll.begin(); pos != coll.end(); ++pos) // coll의 처음부터 끝까지 루프
        cout << *pos << ' ';
    cout << endl;
}
```



출력:
a b c d e f g h i j k l m n o p q r s t u v w x y z

리스트(List) 예제 3

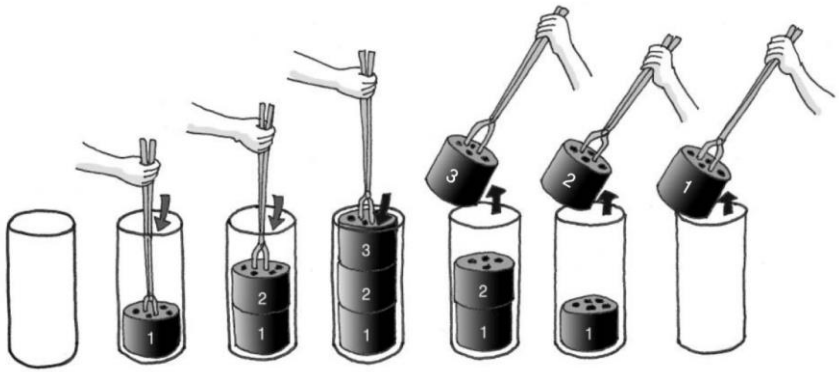
```
// list3.cpp
#include <iostream>
#include <list>
using namespace std;

void main() {
    list<char> coll;

    for (char c='a'; c<='z'; ++c)
        coll.push_back(c);

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

실습 : List로 최대/최소 구하기

- 10개의 자료를 입력하기
  - 이미 입력된 값이 있으면 값을 리스트에 넣지 않음
  - 최대값 및 최소값을 구함

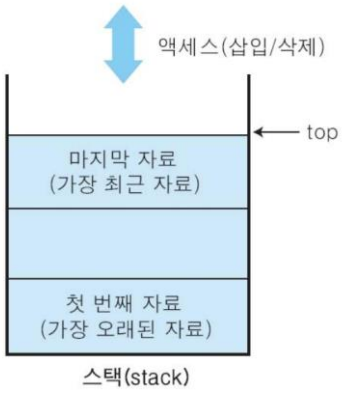
스택

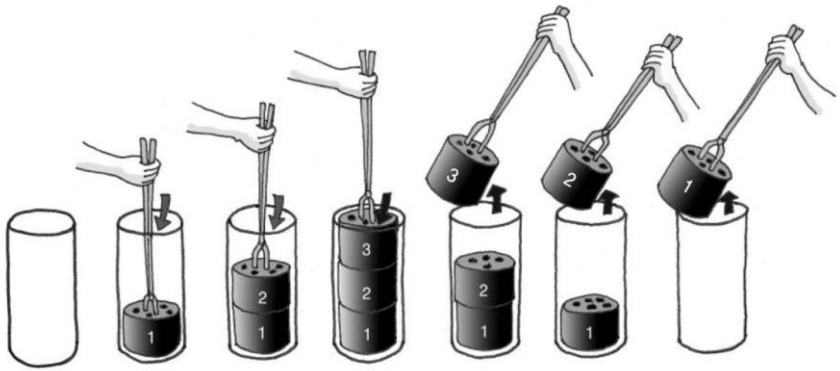
- 자료를 차곡차곡 쌓아 올린 형태의 자료 구조
  - 예: 연탄 아궁이



스택

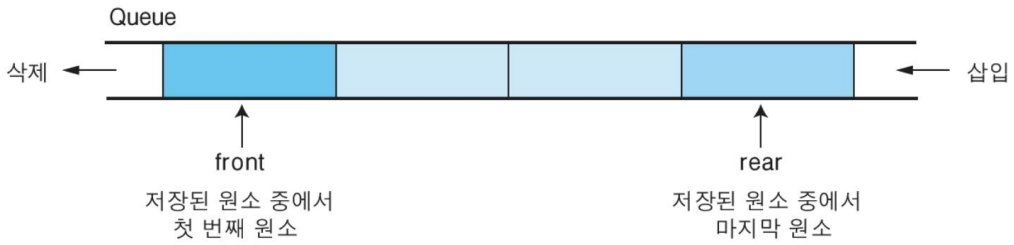
- 자료를 차곡차곡 쌓아 올린 형태의 자료 구조로 자료의 삽입/삭제가 스택의 한 위치에서만 일어남
  - 예: 연탄 아궁이





큐

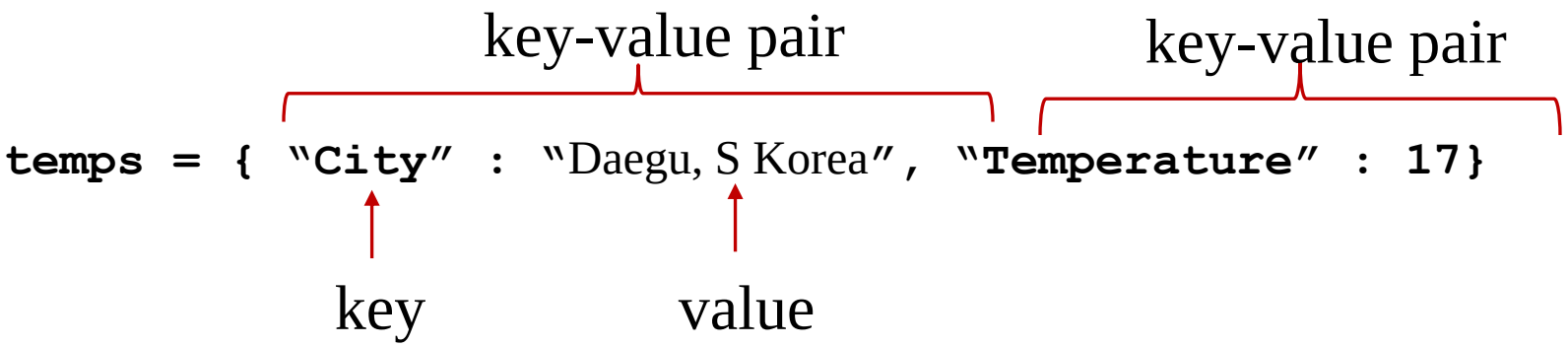
- 자료의 삽입과 삭제가 서로 다른 위치로 제한된 자료구조



사전 (Dictionary)

- 사전 (**dictionary**)은 속성/값 (property/value)의 쌍으로 이루어진 자료구조
- 파이썬에서는 키/값 (key/value)으로 이루어짐

City	Temperature
“Daegu, S Korea”	17



사전 (Dictionary)

```
temps = { " City" :"Daegu, S Korea" , "Temperature" : 17}
```

- 추출

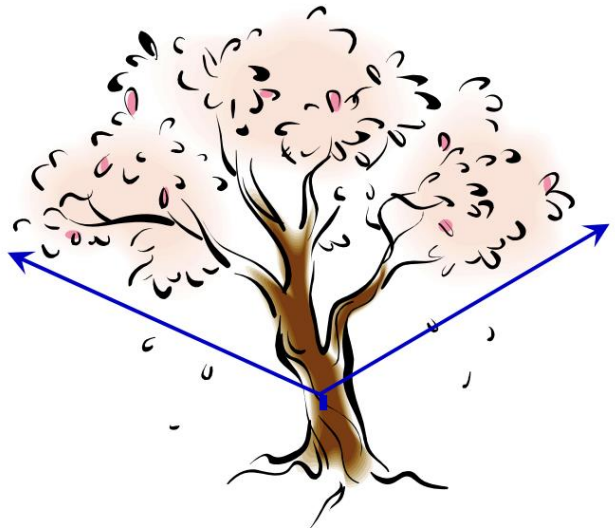
```
cityTemp = temps["Temperature"]
cityName = temps["City"]
```

- 갱신

```
temps["Temperature"] = 20
```

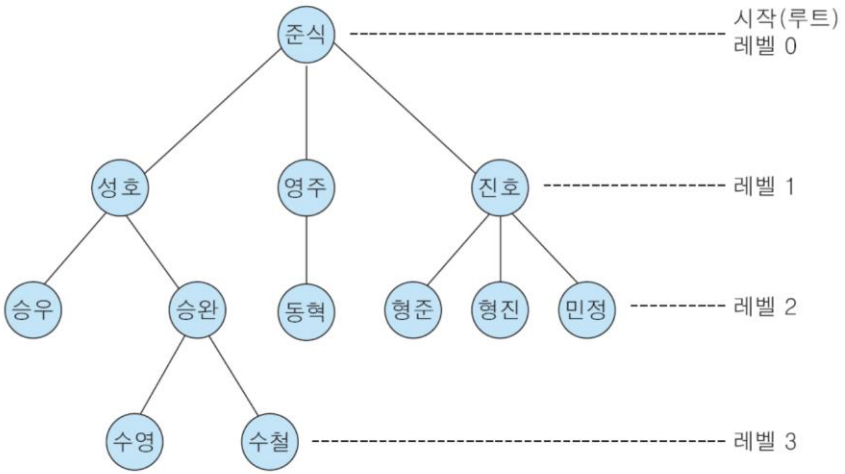
트리

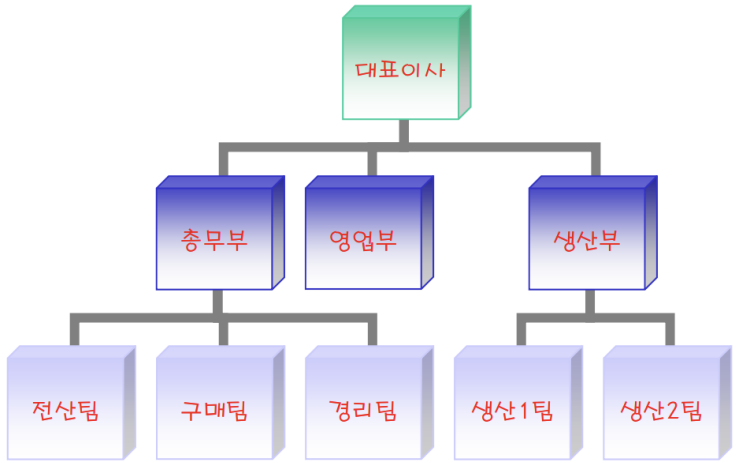
- 원소들 간에 1:N 관계를 가지는 비선형 자료구조
- 원소들 간에 계층관계를 가지는 계층형 자료구조
- 상위 원소에서 하위 원소로 내려가면서 확장되는 나무 모양의 구조



트리

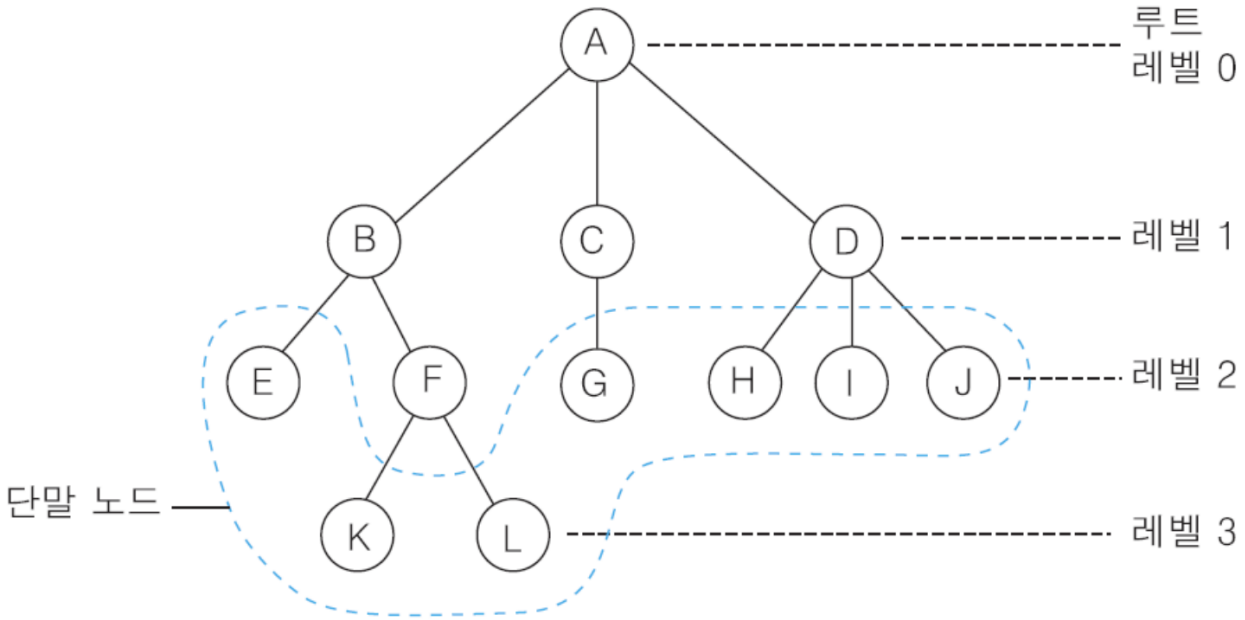
- 트리의 예: 가계도, 조직도





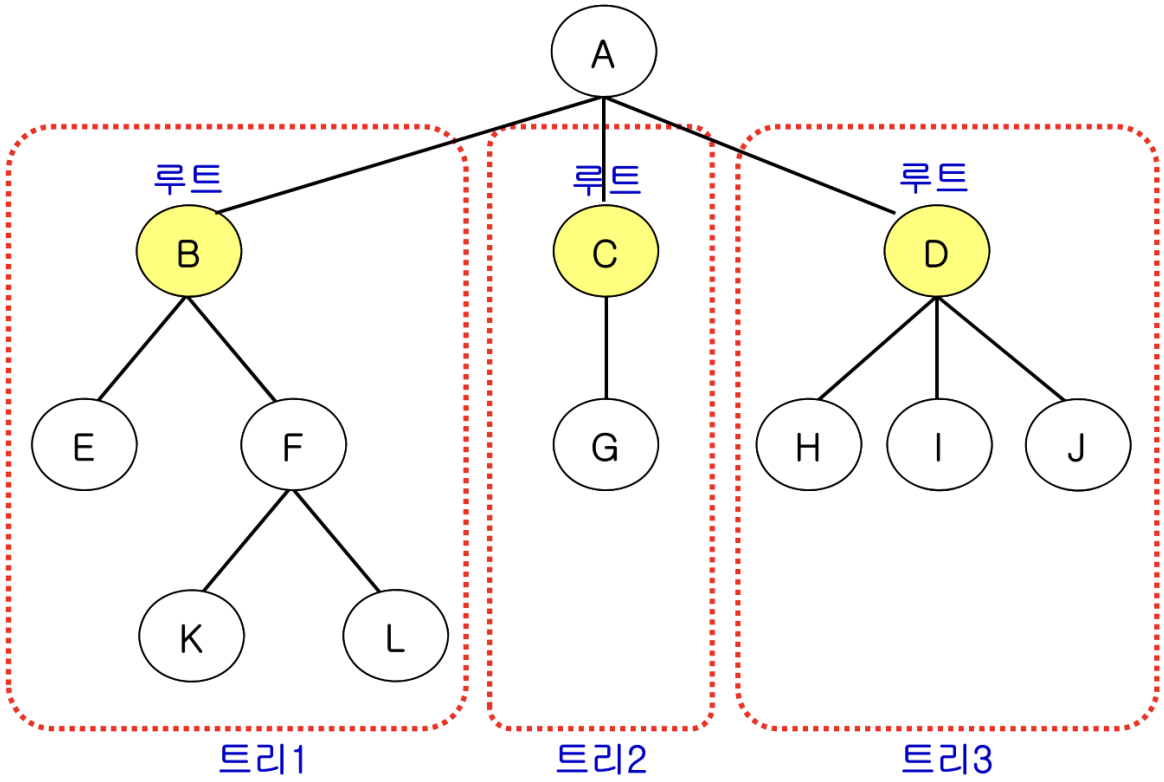
트리

- 트리의 용어



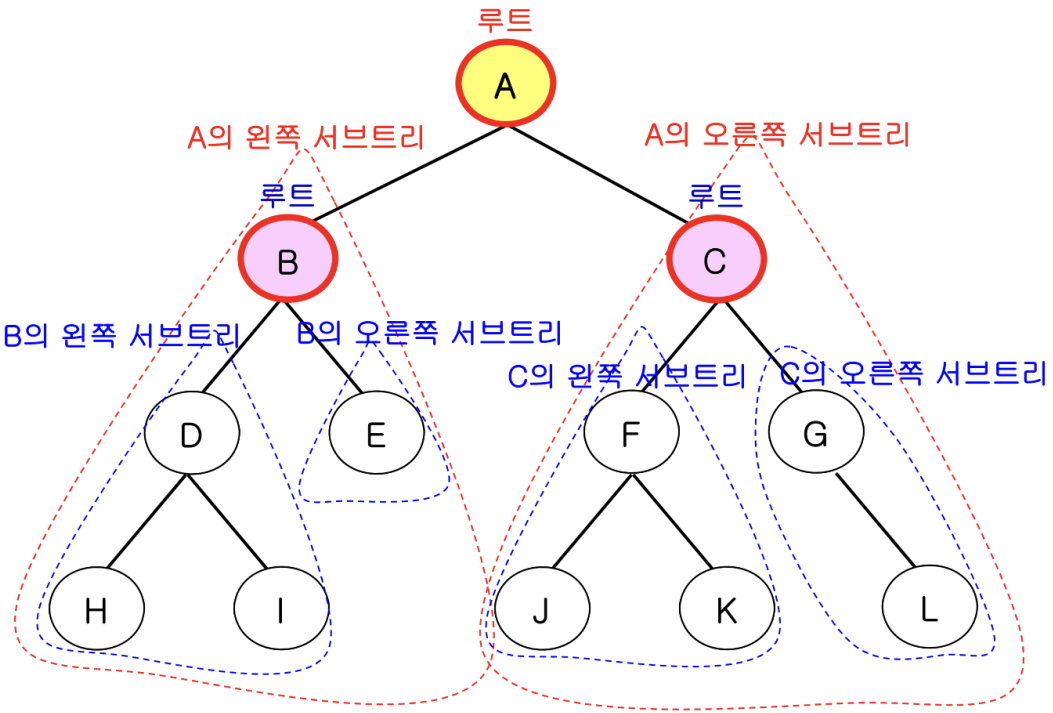
트리

- Forest: 서브트리의 집합



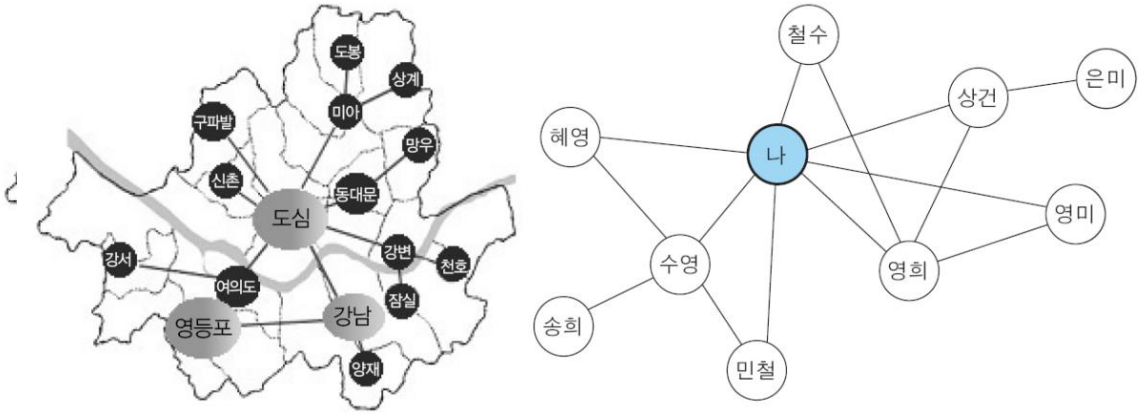
트리

- 이진 트리 (Binary tree): 모든 노드는 왼쪽 자식과 오른쪽 자식 노드만을 가짐



그래프

- 선형 자료구조나 트리 자료구조로 표현하기 어려운 N:M 관계를 표현하기 위한 자료구조
- 그래프의 예



스택(Stack)

- 데이터의 삽입과 삭제가 리스트의 끝(top)에서만 이루어지는 리스트
  - 후입선출(LIFO) : last in, first out
- 가능한 연산(Operations):
  - push(): 요소를 삽입
  - pop(): 첫 요소를 삭제
  - top(): 첫 요소에 접근
  - size(): 요소의 개수를 리턴
  - empty(): stack이 비어있는지 확인

C++의 스택(Stack)

```
// stack.cpp
#include <iostream>
#include <stack>
using namespace std;

void main() {
    stack<int> s;

    for (int i=1; i<=10; ++i)
        s.push(i); //스택에 s 삽입

    while( !s.empty() ) {
        cout << s.top() << endl;
        s.pop(); //스택의 top을 제거
    }
}
```

- push(): 요소를 삽입
- pop(): 첫 요소를 삭제
- top(): 첫 요소에 접근
- size(): 요소의 개수를 리턴
- empty(): stack이 비어있는지 확인

- **주의:** pop() 은 첫번째 요소를 제거하고 아무것도 반환하지 않는다. 그래서 top()으로 첫번째 요소를 얻은 후 pop()을 불러와서 그것을 삭제한다.

스택(Stack) 구현 샘플

- 연결 리스트 사용
  - http://www.comp.hkbu.edu.hk/~chxw/pig/code/stackli.h
  - http://www.comp.hkbu.edu.hk/~chxw/pig/code/stackli.c

- 배열 사용
  - http://www.comp.hkbu.edu.hk/~chxw/pig/code/stackar.h
  - http://www.comp.hkbu.edu.hk/~chxw/pig/code/stackar.c

  - 주의: 스택을 만들 때 최대 스택의 크기를 정의해야 한다

실습 : 스택 활용하기

- 0이 입력될 때까지 양의 정수를 입력받아 짝수이면 그대로 출력하고 홀수이면 스택에 넣어두었다가 마지막에 한꺼번에 출력함
- 예
  - 입력 : 2 3 4 6 7 5 0
  - 출력 : 2 4 6 5 7 3 //5 7 3은 스택에 들어갔다 반대로 출력

큐(Queue)

- 데이터의 삽입이 한 쪽 끝에서만 수행되는 반면 삭제는 다른 한 쪽 끝에서만 수행되는 리스트
  - 선입선출(FIFO): first in, first out

- 가능한 연산(Operations):
  - push(): 노드를 삽입
  - pop(): 첫 노드를 삭제
  - front(): 첫 노드에 접근
  - back(): 마지막 노드에 접근
  - size(): 노드의 개수를 리턴
  - empty(): 큐에 노드가 있는지 확인

C++의 큐(Queue)

```
// queue.cpp
#include <iostream>
#include <queue>
using namespace std;

void main()
{
    queue<int> s;

    for (int i=1; i<=10; ++i)
        s.push(i);

    while( !s.empty() ) {
        cout << s.front() << endl;
        s.pop();
    }

}
```

- push(): 노드를 삽입
- pop(): 첫 노드를 삭제
- front(): 첫 노드에 접근
- back(): 마지막 노드에 접근
- size(): 노드의 개수를 리턴
- empty(): 큐에 노드가 있는지 확인

큐(Queue) 예제 II

```
// queue2.cpp
#include <iostream>
#include <queue>
#include <string>
using namespace std;

void main()
{
    queue<string> q;

    q.push("These ");
    q.push("are ");
    q.push("more than ");


    cout << q.front();        q.pop();
    cout << q.front();        q.pop();
```

```
    q.push("four ");
    q.push("words! ");

    // skip one element
    q.pop();

    cout << q.front();
    q.pop();
    cout << q.front();
    q.pop();

    cout << "number of elements in
       the queue: " << q.size() << endl;

}
```

큐(Queue) 구현 샘플

- circular array로 구현
  - 큐를 만들 때 큐의 최대 크기를 지정
  - 큐의 시작을 가리키는 front(=head)변수와 끝을 가리키는 rear(=tail)변수가 필요

- Sample code
  - http://www.comp.hkbu.edu.hk/~chxw/pig/code/queue.h
  - http://www.comp.hkbu.edu.hk/~chxw/pig/code/queue.c

우선순위 큐(Priority Queues)

- 우선순위를 가지는 데이터를 저장
  - 각 데이터는 우선 순위를 나타내는 키(Key)를 포함

- 가능한 연산(operations):
  - push(): 노드를 삽입
  - pop(): 우선순위 가장 높은 노드를 삭제
  - top(): 우선순위 가장 높은 노드에 접근
  - size(): 노드의 개수를 리턴
  - empty(): 컨테이너가 비어있는지 확인

C++의 우선순위 큐(Priority Queue)

```
// pqueue.cpp
#include <iostream>
#include <queue>
using namespace std;

void main()
{
    priority_queue<int> s;

    s.push(5); s.push(4); s.push(8);
    s.push(9); s.push(2); s.push(7);
    s.push(6); s.push(3);
    s.push(10);

    while( !s.empty() ) {
        cout << s.top() << endl;
        s.pop();
    }

}
```

- push(): 노드를 삽입
- pop(): 우선순위 가장 높은 노드를 삭제
- top(): 우선순위 가장 높은 노드에 접근
- size(): 노드의 개수를 리턴
- empty(): 컨테이너가 비어있는지 확인

- 기본적으로 내림차순으로 정렬됨

실습2 : 스택과 큐를 활용하기

- 0 이 입력될 때까지 양 정수를 입력받아, 홀수이면 스택에 두었다가 거꾸로 출력하고 짝수이면 큐에 넣어 두었다가 홀수 다음에 입력 순서데로 출력함

- 예
  - 입력 : 2 3 4 6 7 5 0
  - 출력 : 5 7 3 2 4 6 //5 7 3은 스택에 들어갔다 반대로 출력

Pair<>

- The class pair treats two values as a single unit.
  - pair<int, int> a(10, 20), b;
  - b.first = 10;
  - b.second = 30;

- The class map, multimap use pairs to manage the key/value pairs

Pair<> 예제1

```
// pair1.cpp
#include <iostream>
using namespace std;

double dist_Points(pair<double, double> p1, pair<double, double> p2)
{
	double res = 0.0;
	res = (p1.first – p2.first) * (p1.first – p2.first);
	res += (p1.second – p2.second) * (p1.second – p2.second);
	return sqrt(res);
}

void main()
{
	pair<double, double> a  (10.0, 10.0);  // C++11 { 10.0, 10.0}
	pair<double, double> b = make_pair<double, double>(20.0, 20.0);

	cout << “Distance “ << dist_Points(a, b) << endl;
}
```

Pair<> 예제2

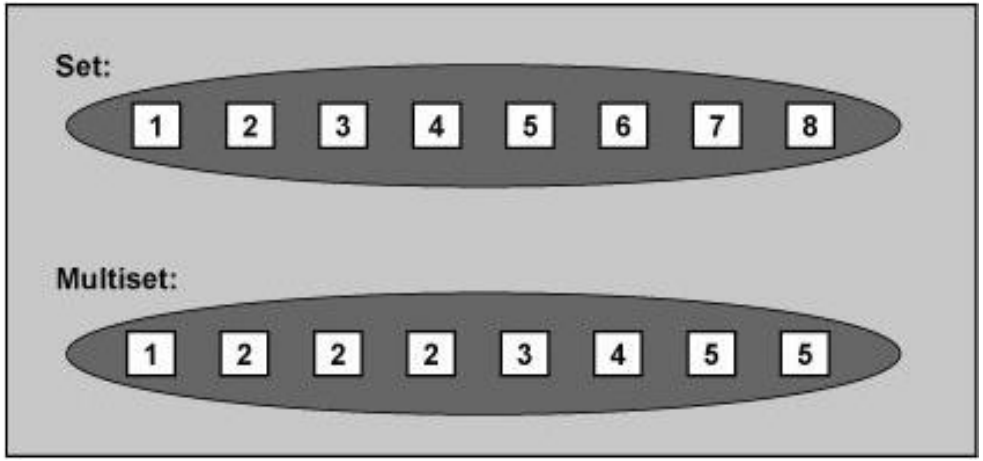
```
// pair2.cpp
#include <iostream>
using namespace std;

typedef  pair<double, double>  Point;

double dist_Points(Point p1, Point p2)
{
      double res = 0.0;
      res = (p1.first – p2.first) * (p1.first – p2.first);
      res += (p1.second – p2.second) * (p1.second – p2.second);
      return sqrt(res);
}

void main()
{
      Point a (10.0, 10.0);                          // C++11 { 10.0, 10.0}
      Point b = make_pair (20.0, 20.0);

      cout << “Distance “ << dist_Points(a, b) << endl;
}
```

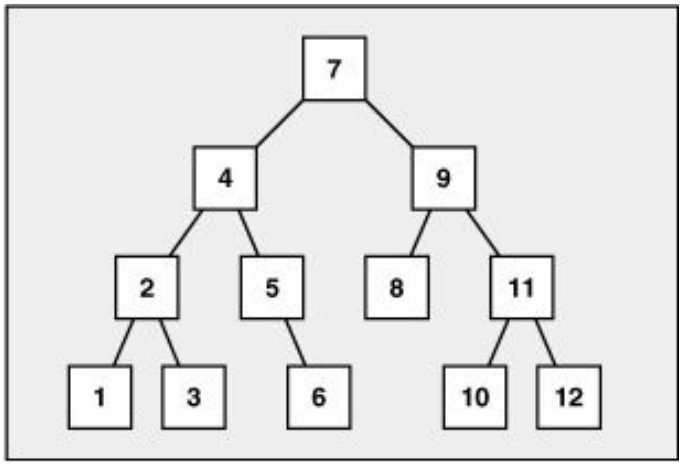
실습 : 원점에서 가장 가까운 점 찾기

- 5개의 좌표 점을 입력받아 vector에 저장하고, 원점에 가장 가까운 점을 찾아 출력해 보자.
  - 입력
    - 9 3 7 5 2 6 5 1 3 3
  - 출력
    - (3,3)

Sets and Multisets

- Set and Multiset containers sort their elements automatically according to a certain sorting criterion
  - Set : no duplicates,
  - Multiset : duplicates allowed
  - Implemented as balanced binary tree





Set 예제

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

void main() {
	set<int> set1, set2;

	for(int i = 0; i < 7; i++) {
		set1.insert(i);
		set2.insert(i+3);
	}
//	for(int data : set1 ) { cout << data << " "; } cout << endl;  // C++11
	set<int>::iterator it;
	for( it = set1.begin(); it != set1.end(); it++) { cout << *it << " "; }  cout << endl;

	set<int> uset(set1);
	uset.insert(set2.begin(), set2.end() );
	cout << "Union : ";
	for( it = uset.begin(); it != uset.end(); it++) { cout << *it << " "; }  cout << endl;
}
```

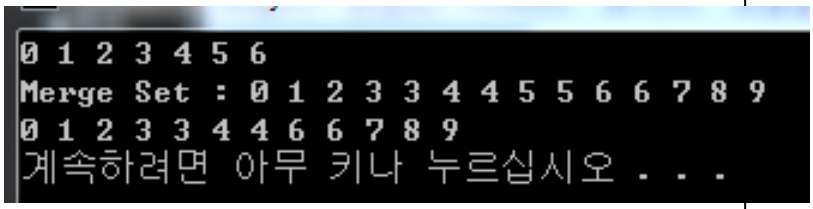
실습4 : 합집합, 교집합, 차집합 구하기

- 0이 입력될 때까지 원소를 입력받아 set<int>에 저장하여 두 집합 A, B를 만든다. 두 집합의 합집합, 교집합, 차집합(A-B) 순으로 출력해보자.

  - 입력
    - 1 2 3 4 5 6 7 0
    - 3 5 9 10 11 0
  - 출력
    - {1 2 3 4 5 6 7 9 10 11} // 합집합
    - {3 5} // 교집합
    - {1 2 4 6 7} // 차집합

Multisets 예제

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

void main() {
    set<int> set1;   multiset<int> mset2;

    for(int i = 0; i < 7; i++) {
        set1.insert(i);  mset2.insert(i+3);
    }
    set<int>::iterator it;
    for( it = set1.begin(); it != set1.end(); it++) { cout << *it << " "; }  cout << endl;

    multiset<int> u_set(mset2);
    u_set.insert(set1.begin(), set1.end() );
    cout << "Merge Set : ";
    multiset<int>::iterator it2;
    for( it2 = u_set.begin(); it2 != u_set.end(); it2++) { cout << *it2 << " "; }
    cout << endl;
    u_set.erase(5);
    for( it2 = u_set.begin(); it2 != u_set.end(); it2++) { cout << *it2 << " "; }
    cout << endl;
}
```



실습5 : 선호도 통계치 구하기

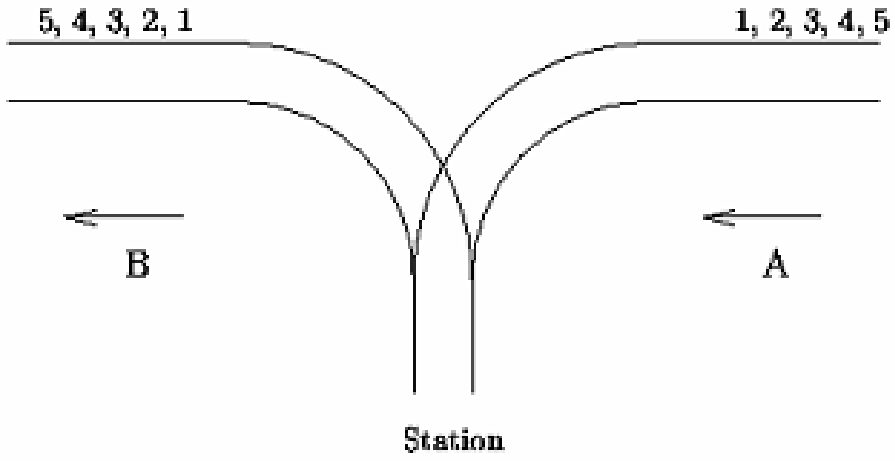
- 식당 메뉴의 선호도 자료를 입력받아 선호도 점수별 카운트를 출력하는 프로그램을 multiset 을 이용하여 작성해보자.
- 선호도는 0 ~ 10 사이의 값으로 -1이 입력될 때까지 입력받음
- 각 선호도 점수별 카운트 값을 출력함

```
1 2 9 8 7 3 7 8 7 0 10 9 9 8 3 2 1 1 0 7 7 -1
0 : 2
1 : 3
2 : 2
3 : 2
7 : 5
8 : 3
9 : 3
10 : 1
계속하려면 아무 키나 누르십시오 . . .
```

실습6 : T-형 간이역 통과 열차번호

- 아래 그림과 같은 간이역으로 통과하였을 때, (1, 2, 3, 4, 5)로 출발한 열차가 종착역 B에 도착 가능한 번호인지 검사하라.

  - 입력
    - 5 4 3 1 2
  - 출력
    - No



실습7 : T-형 간이역 통과 열차번호

- 아래 그림과 같은 간이역으로 통과하였을 때, (1, 2, 3, 4, …, n)로 출발한 n 칸의 열차가 종착역 B에 도착할 수 있는 모든 열차 순서를 나열하시오.

  - 입력(열차의 수)
    - 3
  - 출력
    - 1 2 3
    - 2 1 3
    - 2 3 1
    - 1 3 2
    - 3 2 1