

STL

pair

- 정의 : 이름이 first, second인 두 개의 변수를 저장할 수 있는 struct
- 용도 :
 - 이차원 배열의 인덱스
 - 이차원 좌표 평면에서의 좌표
 - 정점 번호와 해당 정점 번호까지의 최단거리를 묶어서 저장해야하는 경우
- 사용법:
 - pair 을 사용하기 위해서는 를 include 해야 한다.
 - pair은 다른 컨테이너들에 비해 간단한 구조이기 때문에 멤버 함수가 적다.
- 예제:

```
//pair 선언
pair<int,int> p;
pair<char,double> p;

//pair 생성
int a = 1, b = 2;
pair<int,int> p = make_pair(a,b);
pair<int,int> p = make_pair(1,2);

//pair의 멤버 변수에 접근
int valA = p.first;
int valB = p.second;
```

vector

- 정의 : 사이즈가 유동적인 배열
- 용도 : 배열을 사용하는 모든 경우
- 사용법:
 - vector를 사용하기 위해서는 을 include 해야한다.
 - vector은 다양한 컨테이너를 가지고 있다.

멤버 함수	기능
v.size()	v의 사이즈 (원소의 개수) 리턴
v.resize(new_size)	v를 new_size로 사이즈를 바꿔줌
v.empty()	v의 사이즈가 0인지 아닌지 확인
v.begin()	v의 0번째 원소를 가리키는 iterator 리턴
v.end()	v의 마지막 원소를 가리키는 iterator 리턴
v.front()	v의 0번째 원소를 리턴
v.back()	v의 마지막 원소를 리턴
v.push_back(val)	v의 끝에 val 추가
v.pop_back()	v의 마지막 원소를 삭제
v.clear()	v의 모든 원소를 삭제

의 reverse()로 vector의 원소를 뒤집을 수 있다.

```
reverse(v.begin(), v.end());
```

의 sort를 이용하여 오름차순 정렬을 할 수 있다.

```
sort(v.begin(), v.end());
```

- 예제

```
vector<int> iv;
vector<pair<int,int>> pv;

//사이즈가 3인 vector 생성
vector<int> myv(3);

//사이즈가 N(5)이고, 각 원소가 2로 초기화된 vector 생성
int N = 5;
vector<int> myv(N, 2);

//vector에 원소 추가
iv.push_back(1);           // iv : 1
iv.push_back(2);           // iv : 1 2
iv.push_back(3);           // iv : 1 2 3

pv.push_back(make_pair(2,4));

//vector의 size 조정
iv.resize(4);
cout << iv.size();         // 4

cout << iv.front();         // 1
```

```
cout << iv.back(); // 0 (resize를 4로 했기 때문에 마지막 원소는
// 자동적으로 0으로 초기화됨)

iv.pop_back(); // iv : 1 2
iv.clear(); // iv :
```

queue

- 정의 : FIFO(First Input First Output)자료구조
- 용도 :
 - BFS(너비 우선 탐색)
 - 특별한 알고리즘을 사용하는것이 아니라 직접 문제 상황을 구현하는 문제들 중 FIFO 구조를 가지는 문제를 풀 때

(Ex. 다리에 올라갈 수 있는 최대 하중과 각 트럭의 무게가 주어질 때, 모든 트럭이 다리를 지나가는 데 걸리는 최소 시간을 구하는 문제. 다리에 먼저 올라간 트럭이 먼저 나오게 되기 때문에 queue를 이용해 구현하면 풀 수 있다.)

- 사용법:
 - queue 를 사용하기 위해서 <queue> 를 include 해야함.

멤버 함수	기능
q.size()	q의 사이즈(원소의 개수)를 리턴
q.empty()	q의 사이즈가 0인지 아닌지 확인
q.front()	q에 가장 먼저 들어간 원소를 리턴
q.back()	q의 가장 나중에 들어간 원소를 리턴
q.push(val)	q의 위 (마지막에) val 추가
q.pop()	q에 가장 먼저 들어간 원소를 삭제

- 예시

```
queue<pair<int,int>> q;

q.push(make_pair(1,2));

int a = 2, b = 3;
pair<int,int> p = make_pair(a,b);
q.push(p);

cout << q.front().first << ' ' << q.front().second; // 1 2
cout << q.back().first << ' ' << q.back().second; // 2 3
```

```

//q가 비어있지 않은 동안
while(!q.empty())
{
    pair<int,int> n = q.front();
    q.pop();

    cout << n.first << ' ' << n.second << '\n';           // 1 2
                                                         // 2 3
}

cout << q.size();                                         // 0

q.push(make_pair(4,5));
q.push(make_pair(5,6));

queue<pair<int,int>> emt;
swap(q, emt);      //clear이 없어서 빈 queue와 swap해줌

cout << q.size();                                         // 0

```

stack

- 정의 : LIFO (Last Input First Output) 자료구조
- 용도:
 - DFS(깊이 우선 탐색)
 - 특별한 알고리즘을 사용하는 것이 아니라 직접 문제 상황을 구현하는 문제들 중 LIFO의 구조를 가지는 문제를 풀 때.

(Ex. 만약 어떤 일의 실행순서를 기록해두었다가 나중에 일의 순서를 반대로 출력하고 싶은 경우, 일을 실행할 때마다 stack에 차례대로 push 해두었다가 모든 일을 실행하고 나서 stack이 빌 때까지 pop하면 된다)

- 사용법
 - stack을 사용하기 위해서는 `<stack>`을 include 해야 한다.
 - queue와 맥락이 비슷하다.

멤버 함수	기능
s.size()	s의 사이즈(원소의 개수)
s.empty()	s의 사이즈가 0인지 아닌지를 확인
s.top()	s의 가장 나중에 들어간 원소를 리턴
s.push(val)	s의 뒤에 val 추가
s.pop()	s에 가장 나중에 들어간 원소를 삭제

- 예제

```
stack<int> s;

for(int i=1; i<=5; i++)
    s.push(i);


cout << s.size();                // 5

//s가 비어있지 않은 동안
while(!s.empty())
{
    int n = s.top();
    s.pop();

    cout << n << '\n';          // 5 4 3 2 1
}

cout << s.size();
```

map

- 정의:
 - 인덱스로 int가 아닌 다른 자료형을 사용할 수 있는 배열.
 - 편의상 배열이라 칭함  실제로는 트리 구조임
 - map의 내부적인 구조는 각 노드가 key와 value 쌍으로 이루어진 트리이다. 특히 검색, 삽입, 삭제 등의 속도를 빠르게 하기 위해 균형 이진 트리 중의 하나인 '레드 블랙 트리'로 구현되어 있다."

- 용도:

- 연관 있는 두 값을 함께 묶어서 관리하되, 검색을 빠르게 하고 싶은 경우

(Ex. 만약 SNS상 사람들의 친구 관계를 그래프를 이용해 나타내고, 이 그래프에 여러가지 알고리즘들을 적용해 멋진 일들을 하고 싶다고 합시다. 일반적인 경우라면, 사람을 정점으로, 사람들 간의 친구 관계를 간선으로 나타내게 되겠죠?? 문제는! 그래프의 연결 관계를 배열에 저장하려면, 각 정점을 사람 이름으로 나타내는 것이 아니라 번호로 나타내야 한다는 점입니다. 그러려면, 어떤 과정이 필요할까요?? 사람의 **이름**을 정점의 **번호**로 **변환**해주는 과정이 필요하겠죠?? 이 때, map을 이용해 문제를 해결할 수 있습니다. key를 사람의 이름으로, value를 정점 번호로 묶어서 저장해놓는다면 나중에 몇 번 정점에 해당하는지 알고 싶은 사람의 이름이 있을 때, 이름을 검색해 해당 정점 번호를 빠르게 알아낼 수 있습니다.

- 사용법:

- map을 사용하기 위해선, 을 include 해야 한다.
- map은 요소에 접근할 때 iterator을 이용하는 방식과, 인덱스(key)를 이용하는 방식을 사용한다.

멤버 함수	기능
m.size()	m의 노드 개수를 리턴
m.empty()	m의 사이즈가 0인지 아닌지를 확인
m.begin()	m의 첫번째 원소를 가리키는 iterator 리턴
m.end()	m의 마지막 원소를 가리키는 iterator 리턴
m[k] = v	m에 key가 k이고, value가 v인 노드 추가
m.insert(make_pair(k,v))	m에 key가 k이고, value가 v인 노드 추가
m.erase(k)	m에서 key가 k인 노드 삭제
m.find(k)	m에서 key가 k인 노드를 찾아, 해당 노드를 가리키는 iterator 리턴 (만약 key가 k인 노드가 존재하지 않는 경우, m의 마지막 원소를 가리키는 iterator 리턴)
m.count(k)	m에서 key가 k인 노드의 개수를 리턴

cf) insert와 erase함수의 경우, 파라미터로 값 자체가 아닌 **iterator**를 넘겨주는 방식을 사용할 수 있습니다. iterator를 넘겨주는 방식을 통해, array(배열)나 vector(벡터)에 있는 모든 값들을 추가하거나, map의 첫 번째 원소나 마지막 원소를 삭제할 수도 있습니다.

ex1) vector v에 있는 모든 값 추가 -> m.insert(v.begin(), v.end())

ex2) map의 첫 번째 원소 삭제 -> m.erase(m.begin())

- 예제

```
map<char,int> m;
map<char,int>::iterator it;

m['B'] = 2; //m : (B,2)
m.insert(make_pair('A',1)); //m : (A,1) (B,2)
m['C'] = 3; //m : (A,1) (B,2) (C,3)

m.erase('A'); //m : (B,2) (C,3)

//m전체를 순회하며 key와 value 출력
for(it = m.begin(); it != m.end(); it++)
    cout << it->first << ' ' << it->second << '\n';

if(m.find('B') != m.end())
    cout << "key값이 B인 노드가 존재합니다." << '\n';
else
    cout << "key값이 B인 노드가 존재하지 않습니다." << '\n';
```

set

- 정의 : Key만 있는 map 또는 정렬된 집합

(set은 map과 구조가 매우 유사합니다. 단, key만 있고 value가 없는 map이라고 생각할 수 있습니다. 따라서, 사용법도 map과 크게 다르지 않습니다. 정렬된 집합 이라고 정의한 이유는 set이 갖는 값들이 중복을 허락하지 않고 정렬되어 있기 때문입니다. (위의 map에서 설명한 key값의 특성

- 용도
 - 특정 값에 대해 검색을 빠르게 하고 싶은 경우
- 사용법
 - set을 사용하기 위해선 `<set>` 을 include 해야 한다.
 - 기본적인 사용법은 map과 유사하다.

멤버 함수	기능
<code>s.size()</code>	s의 노드 개수를 리턴
<code>s.empty()</code>	s의 사이즈가 0인지 아닌지 확인
<code>s.begin()</code>	s의 첫번째 원소를 가리키는 iterator 리턴
<code>s.end()</code>	s의 마지막 원소를 가리키는 iterator 리턴
<code>s.insert(k)</code>	s에 값이 k인 노드 추가
<code>s.erase(k)</code>	s에서 값이 k인 노드 삭제
<code>s.find(k)</code>	s에서 값이 k인 노드를 찾아, 해당 노드를 가리키는 iterator 리턴 (값이 k인 노드가 존재하지 않는 경우, s의 마지막 원소를 가리키는 iterator 리턴)
<code>s.count(k)</code>	s에서 값이 k인 노드의 개수를 리턴

- 예제

```
set<int> s;
set<int>::iterator it;

s.insert(4);           //s : 4
s.insert(1);           //s : 1 4
s.insert(2);           //s : 1 2 4

vector<int> v;
v.push_back(3);         //v : 3
v.push_back(5);         //v : 3 5
v.push_back(6);         //v : 3 5 6

s.insert(v.begin(), v.end()); //s : 1 2 3 4 5 6

s.erase(4);            //s : 1 2 3 5 6
s.erase(s.begin());    //s : 2 3 5 6
```

```
//지울 원소를 입력받음
int toErase;
scanf("%d", &toErase);

it = s.find(toErase);

//지울 원소가 존재하는 원소일 때만 지움
if(it != s.end())
    s.erase(it);
```