7주차 : STL

강사: 김련성

목차

0. STL이란?

STL(Standard Template Library)

- STL은 C++을 위한 라이브러리입니다. 알고리즘, 컨테이너, 함수자 그리고 반복자라고 불리는 네 가지의 구성 요소를 제공합니다.
- 이번 강의는 STL의 컨테이너를 중점적으로 다룹니다.
- STL의 핵심은 어떤 기능이 주요 기능이 될 지에 따라 (삽입, 삭제, 검색) 적합한 자료구조를 판단해서 사용하는 것입니다.

1. cin, cout

C++의 표준 입출력

1. cin, cout

지금까지 C언어의 <stdio.h>헤더파일에 있는 scanf와 printf를 통해 입출력문을 사용했습니다.

```c
#include <stdio.h>
int main()
{
    int A, B;
    scanf("%d %d", &A, &B);
    printf("%d", A + B);
    return 0;
}

```

C++ 에서도 그렇게 할 수 있지만, C++의 표준 입출력은 **"cin, cout"**을 사용합니다.

```cpp
#include <iostream>
using namespace std;
int main() {
    int A, B;
    cin >> A >> B;
    cout << A + B << "\n";
    return 0;
}
```

지금까지 scanf와 printf를 사용하기 위해 <stdio.h>를 include 했던 것 처럼, C++에서 cin, cout을 사용하려면 <iostream> 헤더파일을 include 하면 됩니다.

(iostream = input/output stream)

1. cin, cout

```cpp
#include <iostream>
using namespace std;
int main() {
    int A, B;
    cin >> A >> B;
    cout << A + B << "\n";
    return 0;
}
```

include 아래의 "using namespace std;"는
std(standard)라는 네임스페이스를 사용한다는 뜻입니다.
using namespace std;를 써주지 않는다면 STL함수를
쓸 때마다 앞에 "std::"를 붙여줘야 합니다.
STL을 사용할 때 습관처럼 include 밑에 쓰면 됩니다.
자세한 것은 구글에 검색해보세요.

scanf()는 cin이, printf()는 cout이 대신합니다.

cin과 cout은 scnaf(), printf()와 달리, 자료형을 지정해주지 않아도(ex. %d, %s, %f) 입력받는, 출력하는 변수의 자료형에 맞춰서 알아서 잘 작동합니다.

우리는 꺽쇠의 방향만 잘 신경써서 적어주면 됩니다.

cin은 ">>", cout은 "<<"입니다.

1. cin, cout

cin, cout 속도를 빠르게하기

```cpp
#include <iostream>
using namespace std;
int main() {

    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

    int A;
    cin >> A;
    cout << A << "\n";
    return 0;
}
```

이렇게 적용해주면, cin과 cout의 실행속도가 scanf(), printf() 보다 빠르게 해줍니다.
대신, cin, cout과 stdio 함수와 함께 쓰면 문제가 생길 수 있으니 같이써선 안됩니다.

15552 - 빠른 A+B

앞에서 배운 cin, cout을 사용해서 풀어보세요
이전 슬라이드의 ios_base::sync_with_stdio(false);를 적용해야 합니다.

언어, 방식에 따른 입출력속도 비교

https://www.acmicpc.net/blog/view/56

https://www.acmicpc.net/blog/view/57

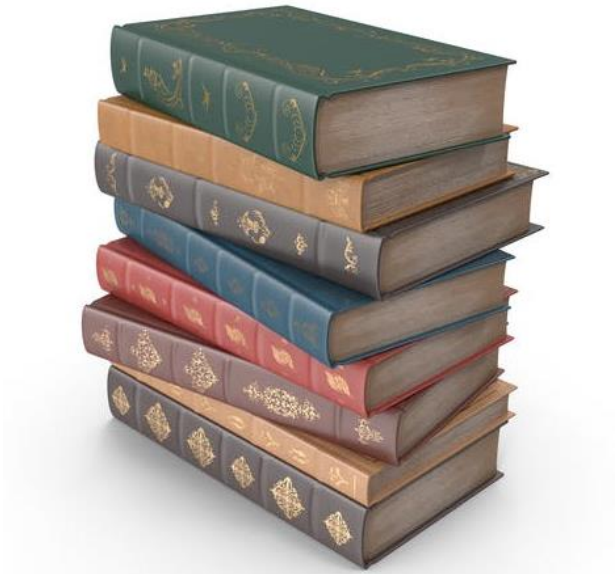
IT/BT관 3층의 queue카페

2. queue, stack

queue와 stack

queue는 줄 혹은 줄을 서서 기다리는 것을 의미하고,
stack은 무언가를 쌓아 올린다는 것을 의미합니다.
queue와 stack을 사용하려면
각각 <queue>와 <stack>헤더파일을 include 해야합니다.

queue
FIFO(First In First Out, 선입선출) 자료구조
queue<자료형>[변수명] 으로 선언합니다.
ex) queue<int> q;

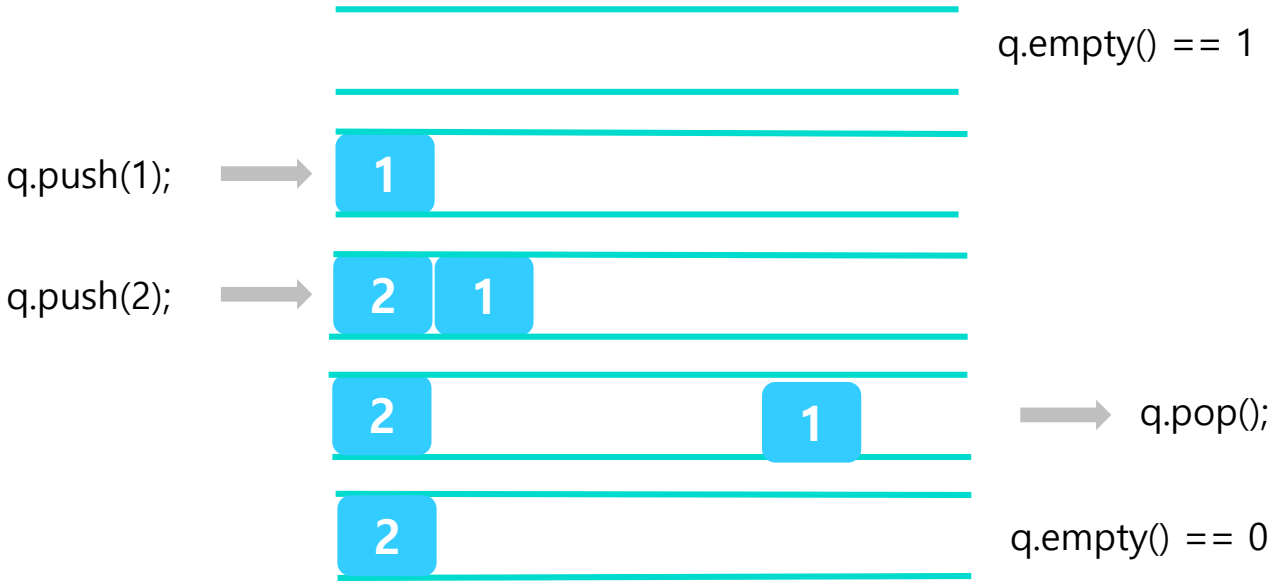


stack
LIFO(Last In First Out, 후입선출) 자료구조
stack<자료형>[변수명] 으로 선언합니다.
ex) stack<int> s;

queue

queue의 멤버함수

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




```cpp
#include <iostream>
#include <queue>
using namespace std;
queue<int> q;
int main() {
    cout << "q.size() : " << q.size() <<"\n";
    cout << "q.empty() : " << q.empty() << "\n\n";
    cout << "q.push(1)" << "\n";
    q.push(1);
    cout << "q.push(2)" << "\n\n";
    q.push(2);
    cout << "q.front() : "<< q.front() << "\n";
    cout << "q.back() : " << q.back() << "\n";
    cout << "q.size() : " << q.size() << "\n";
    cout << "q.empty() : " << q.empty() << "\n\n";
    cout << "q.pop()" << "\n";
    q.pop();
    cout << "\n";
    cout << "q.front() : " << q.front() << "\n";
    cout << "q.back() : " << q.back() << "\n";
    cout << "q.size() : " << q.size() << "\n";
    cout << "q.empty() : " << q.empty() << "\n";
    return 0;
}
```

```
q.size() : 0
q.empty() : 1

q.push(1)
q.push(2)

q.front() : 1
q.back() : 2
q.size() : 2
q.empty() : 0

q.pop()

q.front() : 2
q.back() : 2
q.size() : 1
q.empty() : 0
```

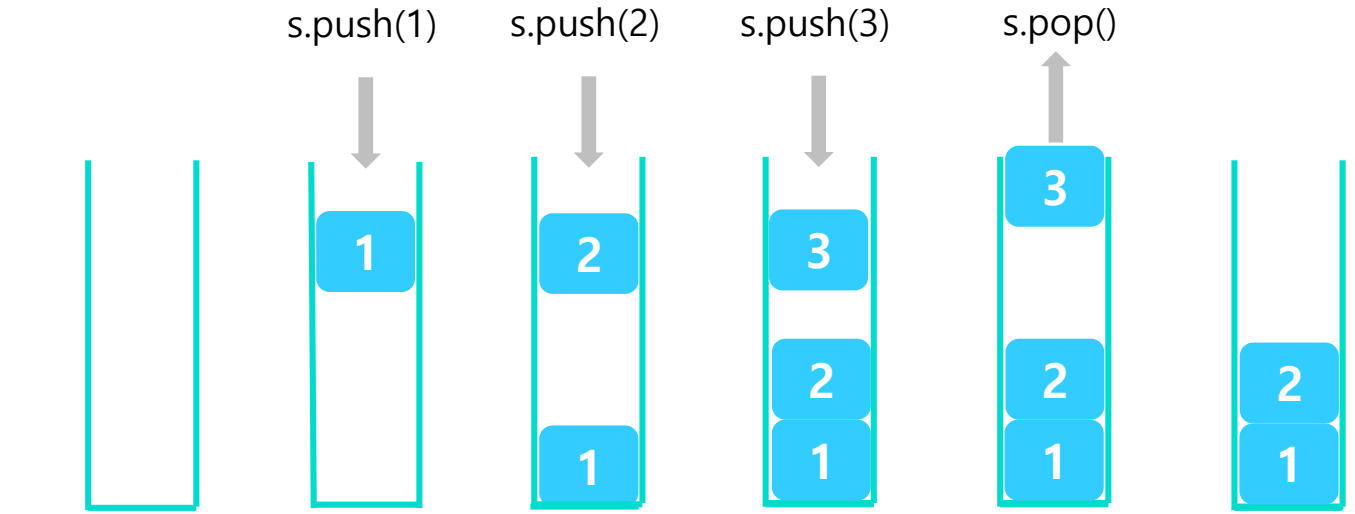
queue 예시

← queue<int> q;

Stack

Stack의 멤버함수

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



s.empty() == 1

s.top() == 2
s.empty() == 0

```cpp
#include <iostream>
#include <stack>
using namespace std;
stack<int> s;
int main() {
    cout << "s.size() : " << s.size() <<"\n";
    cout << "s.empty() : " << s.empty() << "\n\n";
    cout << "s.push(1)" << "\n";
    s.push(1);
    cout << "s.push(2)" << "\n";
    s.push(2);
    cout << "s.push(3)" << "\n\n";
    s.push(3);
    cout << "s.size() : " << s.size() << "\n";
    cout << "s.empty() : " << s.empty() << "\n\n";
    cout << "s.top() : " << s.top() << "\n";
    cout << "s.pop()" << "\n";
    s.pop();
    cout << "\n";
    cout << "s.size() : " << s.size() << "\n";
    cout << "s.empty() : " << s.empty() << "\n";
    cout << "s.top() : " << s.top() << "\n";
    return 0;
}
```

```
s.size() : 0
s.empty() : 1

s.push(1)
s.push(2)
s.push(3)

s.size() : 3
s.empty() : 0

s.top() : 3
s.pop()

s.size() : 2
s.empty() : 0
s.top() : 2
```

stack 예시

← stack<int> s;

10845 - 큐 & 10828 - 스택

위 두 문제는 <string>헤더를 include 하여
string으로 명령을 입력받고 명령에 따라 동작을 수행해야 합니다.
앞에서 배운 queue와 stack의 함수를 써서 풀어보세요.

2841 – 외계인의 기타 연주

3. vector

3. vector

vector의 멤버함수

멤버 함수	기능
v.size()	v의 사이즈(물리적인 저장 용량이 아닌 원소의 개수)를 리턴
v.resize(new_size)	v를 new_size로 사이즈를 바꿔줌
v.empty()	v의 사이즈가 0인지 아닌지를 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 val을 추가
v.pop_back()	v의 마지막 원소를 삭제
v.clear()	v의 모든 원소를 삭제

iterator에 대한 설명은 바로 뒤에 나오는 수 정렬하기 문제를 보면서 하겠습니다.

vector는 사이즈가 자동으로 할당되는 배열입니다.
맨 뒤쪽에서 삽입과 삭제가 가능합니다.

vector를 사용하려면 <vector>헤더파일을 include 해야합니다.

vector<자료형>[변수명]으로 선언합니다.
ex) vector<int> v;

3. vector

2750 – 수 정렬하기

문제

N개의 수가 주어졌을 때, 이를 오름차순으로 정렬하는 프로그램을 작성하시오.

입력

첫째 줄에 수의 개수 N(1 ≤ N ≤ 1,000)이 주어진다. 둘째 줄부터 N개의 줄에는 숫자가 주어진다. 이 수는 절댓값이 1,000보다 작거나 같은 정수이다. 수는 중복되지 않는다.

출력

첫째 줄부터 N개의 줄에 오름차순으로 정렬한 결과를 한 줄에 하나씩 출력한다.

vector로 입력을 받아 문제를 풀어봅니다.

3. vector

2750 – 수 정렬하기

vector로 입력을 받고
<algorithm>의 내장함수 sort()로 정렬을 하고
iterator를 사용하여 출력합니다.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<int>v;
int main()
{
    int N, A;
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> A;
        v.push_back(A);
    }

    sort(v.begin(), v.end()); //sort() <algorithm>에 내장된 함수, v를 오름차순으로 정렬

    vector<int>::iterator iter; // 반복자

    for (iter = v.begin(); iter != v.end(); iter++) {
        cout << *iter << "\n";
    }

    return 0;
}
```

선언

원소 추가

iterator(반복자)
반복자는 포인터와 비슷합니다. 컨테이너에 저장된 원소들의 위치를 나타냅니다.
[컨테이너]::iterator [변수명]으로 선언할 수 있습니다.
ex) vector<int>::iterator iter;

vector 함수로 begin(), end()로 반복자를 받아 ++, --로 증감을 통해 컨테이너 내의 원소들을 순회 할 수 있습니다.

1158 – 요세푸스 문제

vector로 입력을 받고

iterator와 vector 함수를 적절히 사용하여 풀 수 있습니다.

4. map, pair

map

map의 멤버함수

멤버 함수	기능
m.size()	m의 노드 개수를 리턴
m.empty()	m의 사이즈가 0인지 아닌지를 확인
m.begin()	m의 첫 번째 원소를 가리키는 iterator 리턴
m.end()	m의 마지막 원소를 가리키는 iterator 리턴
m[k] = v 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 (key가 k인 노드가 존재하지 않는 경우, m의 마지막 원소를 가리키는 iterator 리턴)
m.count(k)	m에서 key가 k인 노드의 개수를 리턴

map<int, string> m

Keys	Values
2019123123	한라산
2019123456	백두산
2020123123	소백산
2020123456	핫산

map 컨테이너, key와 value

map 컨테이너는 key를 기준으로 정렬되어 key와 value가 쌍으로 저장됩니다. 연관 있는 두 값을 묶어서 관리하고, key를 통한 빠른 검색이 필요할 때 사용합니다.
이때, 각각의 key는 중복될 수 없습니다.

map을 사용하려면 <map>을 include 해야합니다.
map<key의 자료형, value의 자료형> [변수명] 으로 선언합니다.
ex) map<int, string> m;

pair

pair 컨테이너는 이름이 'first', 'second'인 두 개의 변수를 저장할 수 있는 구조체입니다.
pair는 다른 컨테이너에 비해 간단한 구조이기 때문에 멤버함수가 적습니다.

pair<string, string> p1;

first	second
삼겹살	소주

pair<string, string> p2;

first	second
치킨	맥주

pair 컨테이너, first와 second

pair를 사용하려면 <utility>를 include 해야합니다.
pair<자료형, 자료형>[변수명] 으로 선언합니다.
ex) pair<int, int> p;

```cpp
#include <iostream>
#include <utility>
#include <string>
using namespace std;
pair<string, string> p1; //pair 선언
pair<string, string> p2;
string wheat;

int main()
{
    //pair 생성
    string s1 = "삼겹살";
    string s2 = "소주";
    pair<string, string> p1 = make_pair(s1, s2);
    pair<string, string> p2 = make_pair("치킨", "맥주");
    //pair의 멤버 변수에 접근
    wheat = p1.second + p2.second;

    cout << wheat << "\n";

    return 0;
}
```

```
소주맥주

C:₩Users₩
```

11557 – Yangjojang of The Year

7785 – 회사에 있는 사람

2493 – 탑

2493 - 탑

문제

KOI 통신연구소는 레이저를 이용한 새로운 비밀 통신 시스템 개발을 위한 실험을 하고 있다. 실험을 위하여 일직선 위에 N개의 높이가 서로 다른 탑을 수평 직선의 왼쪽부터 오른쪽 방향으로 차례로 세우고, 각 탑의 꼭대기에 레이저 송신기를 설치하였다. 모든 탑의 레이저 송신기는 레이저 신호를 지표면과 평행하게 수평 직선의 왼쪽 방향으로 발사하고, 탑의 기둥 모두에는 레이저 신호를 수신하는 장치가 설치되어 있다. 하나의 탑에서 발사된 레이저 신호는 가장 먼저 만나는 단 하나의 탑에서만 수신이 가능하다.

예를 들어 높이가 6, 9, 5, 7, 4인 다섯 개의 탑이 수평 직선에 일렬로 서 있고, 모든 탑에서는 주어진 탑 순서의 반대 방향(왼쪽 방향)으로 동시에 레이저 신호를 발사한다고 하자. 그러면, 높이가 4인 다섯 번째 탑에서 발사한 레이저 신호는 높이가 7인 네 번째 탑이 수신을 하고, 높이가 7인 네 번째 탑의 신호는 높이가 9인 두 번째 탑이, 높이가 5인 세 번째 탑의 신호도 높이가 9인 두 번째 탑이 수신을 한다. 높이가 9인 두 번째 탑과 높이가 6인 첫 번째 탑이 보낸 레이저 신호는 어떤 탑에서도 수신을 하지 못한다.

탑들의 개수 N과 탑들의 높이가 주어질 때, 각각의 탑에서 발사한 레이저 신호를 어느 탑에서 수신하는지를 알아내는 프로그램을 작성하라.

입력

첫째 줄에 탑의 수를 나타내는 정수 N이 주어진다. N은 1 이상 500,000 이하이다. 둘째 줄에는 N개의 탑들의 높이가 직선상에 놓인 순서대로 하나의 빈칸을 사이에 두고 주어진다. 탑들의 높이는 1 이상 100,000,000 이하의 정수이다.

출력

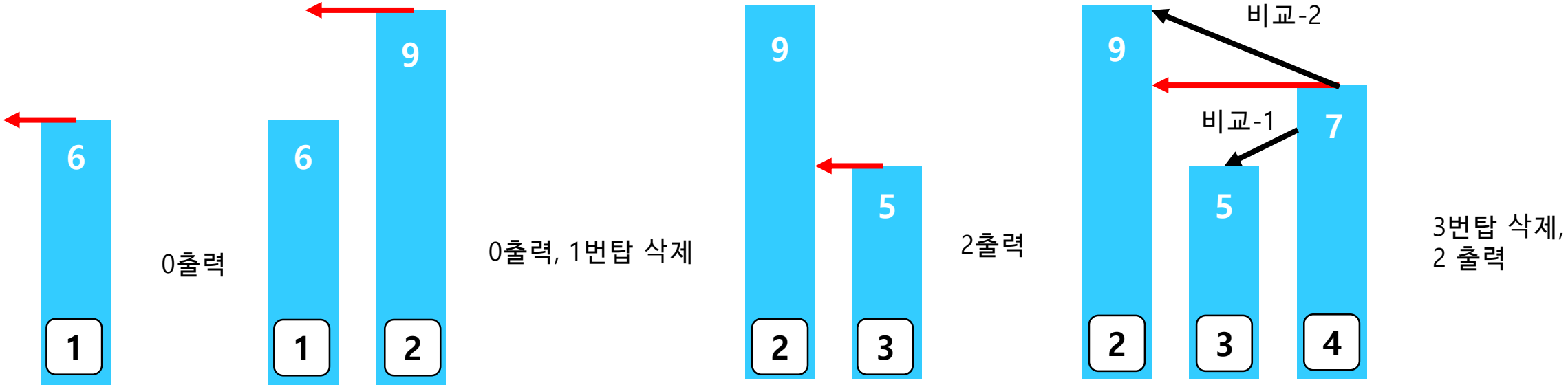
첫째 줄에 주어진 탑들의 순서대로 각각의 탑들에서 발사한 레이저 신호를 수신한 탑들의 번호를 하나의 빈칸을 사이에 두고 출력한다. 만약 레이저 신호를 수신하는 탑이 존재하지 않으면 0을 출력한다.

"높이가 서로 다른 탑이 있다.
왼쪽으로 레이저를 쏠 때,
i번째 탑이 쏜 레이저는 몇 번째
탑이 맞을까?"

입력을 보면,
탑의 수는 1<=N<=500000
탑의 높이는 1<=H<=100000000 입니다.
앞에서 배운 cin, cout을 쓰면서
ios_base::sync_with_stdio(false)
cin.tie(NULL);
cout.tie(NULL);
을 사용하지 않으면 시간초과가 납니다.

2493 - 탑

- 레이저를 왼쪽으로 쏘기 때문에 왼쪽으로, 가까운 탑부터 비교합니다.
- 현재 탑보다 낮은 탑은 이후에 필요가 없으므로 삭제하고 다음 값을 비교합니다.
- 현재탑보다 더 높은 탑이 나오거나 현재 탑보다 높은 탑이 없을 때까지 반복합니다.
- 현재탑 보다 더 높은 탑이 나올경우 그 탑의 인덱스를 출력하고, 더 높은 탑이 없을 경우 0을 출력합니다.
- pair를 사용해 인덱스와 높이를 저장하고, stack을 통해 탑의 정보를 저장하고, 비교, 삭제 합니다.



2493 – 탑 해답

<인덱스, 높이> pair와 pair를 저장할 stack을 만듭니다.

stack이 비어있으면 0을 출력하고, pair를 stack에 push해줍니다.

stack이 비어있지 않으면
top과 입력값을 비교하고,

top이 작으면 더 이상 필요 없으므로 pop하여 top을 삭제합니다.

top이 입력값보다 크면 top을 stack에서 의 index를 출력해주고 반복문을 빠져나오면 됩니다.

```cpp
#include <iostream>
#include <stack>
#include <utility>
using namespace std;
stack<pair<int, int>> s;
int N, H; //탑의 개수, 입력받는 탑의 높이

int main() {

    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);

    cin >> N;
    for (int i = 1; i <= N; i++)
    {
        cin >> H;
        while (!s.empty()) //스택이 비어있지 않다면
        {
            if (s.top().second > H) //s.top이 K보다 크다면 => 신호 수신
            {
                cout << s.top().first << "\n"; //s.top의 인덱스 출력
                break;//while문 나온다 => if문 아래의 s.pop()은 실행되지 않아요
            }
            s.pop(); //K보다 낮은 탑 스택에서 지운다.
        }
        if (s.empty())
            cout << 0 << "\n"; //s.empty()이면 0 출력(신호를 받을 탑이 없음)
        s.push(make_pair(i, H)); //현재 탑을 스택에 push
    }

    return 0;
}
```

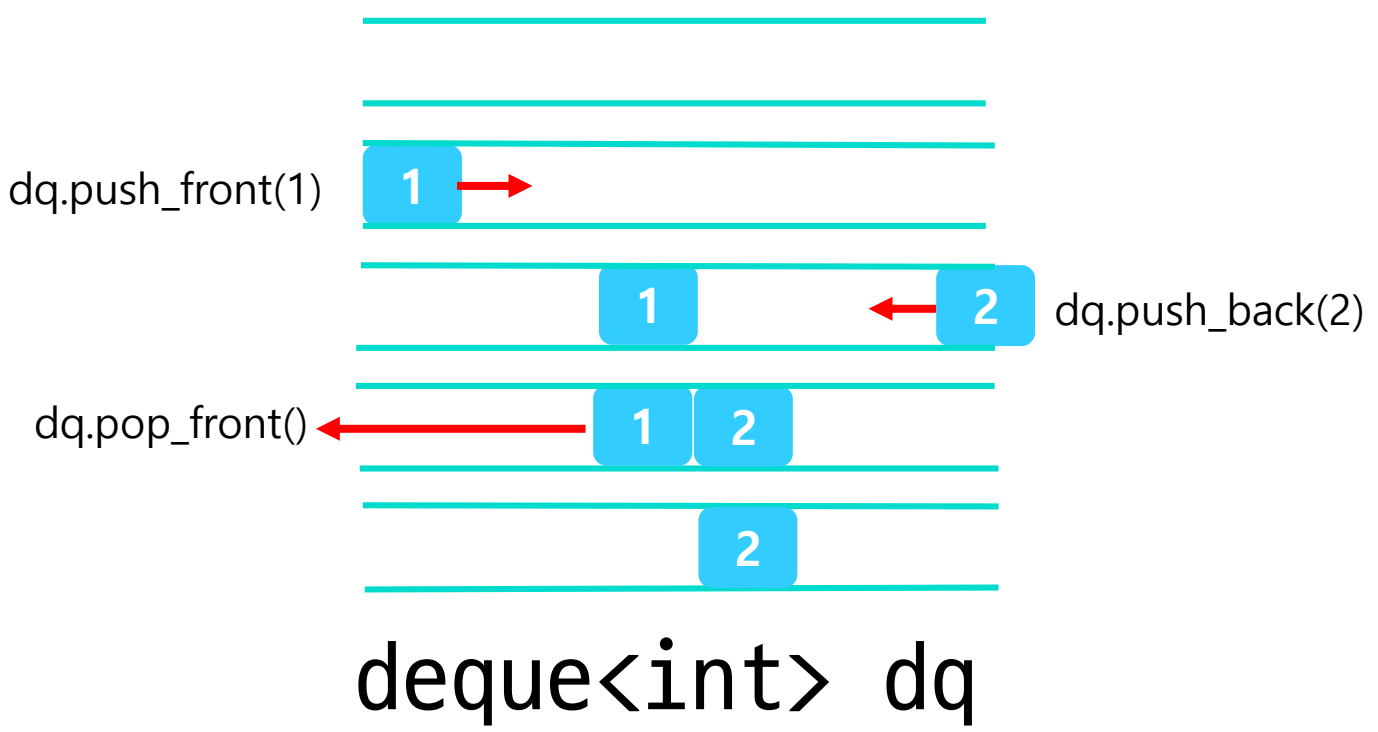
5. deque, priority queue

deque(double ended queue)

앞에서 배운 queue는
앞으로는 삭제, 뒤로는 삽입을 수행했습니다.
deque(double ended queue)는
앞과 뒤에서 삽입과 삭제를 할 수 있습니다.

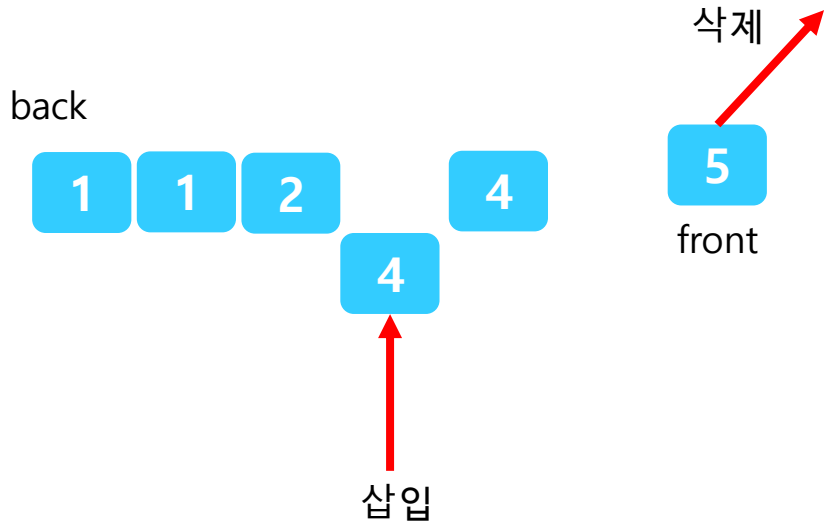
deque를 사용하려면 <deque> 헤더파일을
include 해야 합니다.

deque<자료형>[변수명] 으로 선언합니다.
ex) deque<int> dq;



priority queue 우선순위 큐

priority queue는 앞에서 배운 queue와 비슷하지만, 각 원소들이 우선순위를 갖고 있습니다. priority queue에서 높은 우선순위를 가진 원소는 낮은 우선순위를 가진 원소보다 먼저 처리됩니다.



priority_queue를 사용하려면 <queue> 헤더파일을 include 해야 합니다.
queue<자료형>[변수명] 으로 선언합니다.
ex) queue<int> q;

queue
FIFO(First In First Out, 선입선출) 자료구조
queue<자료형>[변수명] 으로 선언합니다.
ex) queue<int> q;

끝

수고하셨습니다.

과제 목록

- 15552 - 빠른 A+B
- 10845 - 큐
- 10828 – 스택
- 2841 – 외계인의 기타 연주
- 2750 – 수 정렬하기
- 1158 – 요세푸스 문제
- 11557 – Yangjojang of The Year
- 7785 – 회사에 있는 사람
- 2493 – 탑