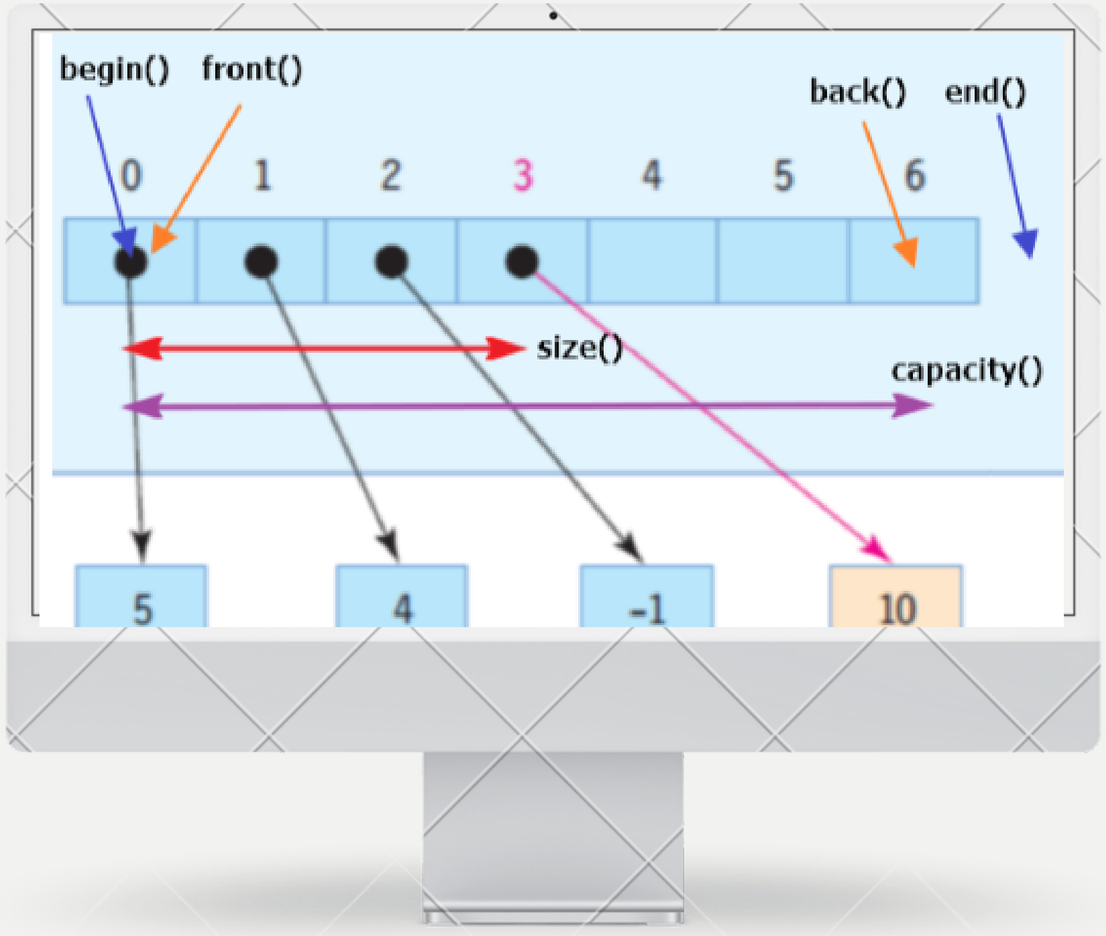
2023-11-01 김석희

vector



vector란?

- 표준 라이브러리에 있는 컨테이너로 사용자가 손쉽게 사용하기 위해 정의된 class
- 동적 배열로 구성
- 각 원소들이 선형적으로 배열
- 벡터 컨테이너의 원소를 참조할 때 반복자를 이용해서 순차적 참조 가능
- 처음 원소로부터의 상대적인 거리를 이용하여 접근 가능

vector란?

- 표준 라이브러리에 있는 컨테이너로 사용자가 손쉽게 사용하기 위해 정의된 class
- 동적 배열로 구성
- 각 원소들이 선형적으로 배열
- 벡터 컨테이너의 원소를 참조할 때 반복자를 이용해서 순차적 참조 가능
- 처음 원소로부터의 상대적인 거리를 이용하여 접근 가능

: 배열인데 길이가 변하는 배열
+ 추가 기능들

vector란?

<장점>

1. 배열과 달리 자동으로 메모리를 할당시켜주어 처음부터 원소의 개수를 지정해둘 필요가 없고, 원소의 삽입/삭제 시 효율적인 메모리 관리가 가능하다.
2. vector의 중간의 원소를 삭제하거나, vector의 크기를 구하는 작업 등을 알아서 해주는 유용한 멤버 함수들이 많다.
3. 배열 기반이므로 랜덤 접근(Random Access)이 가능하다.

vector란?

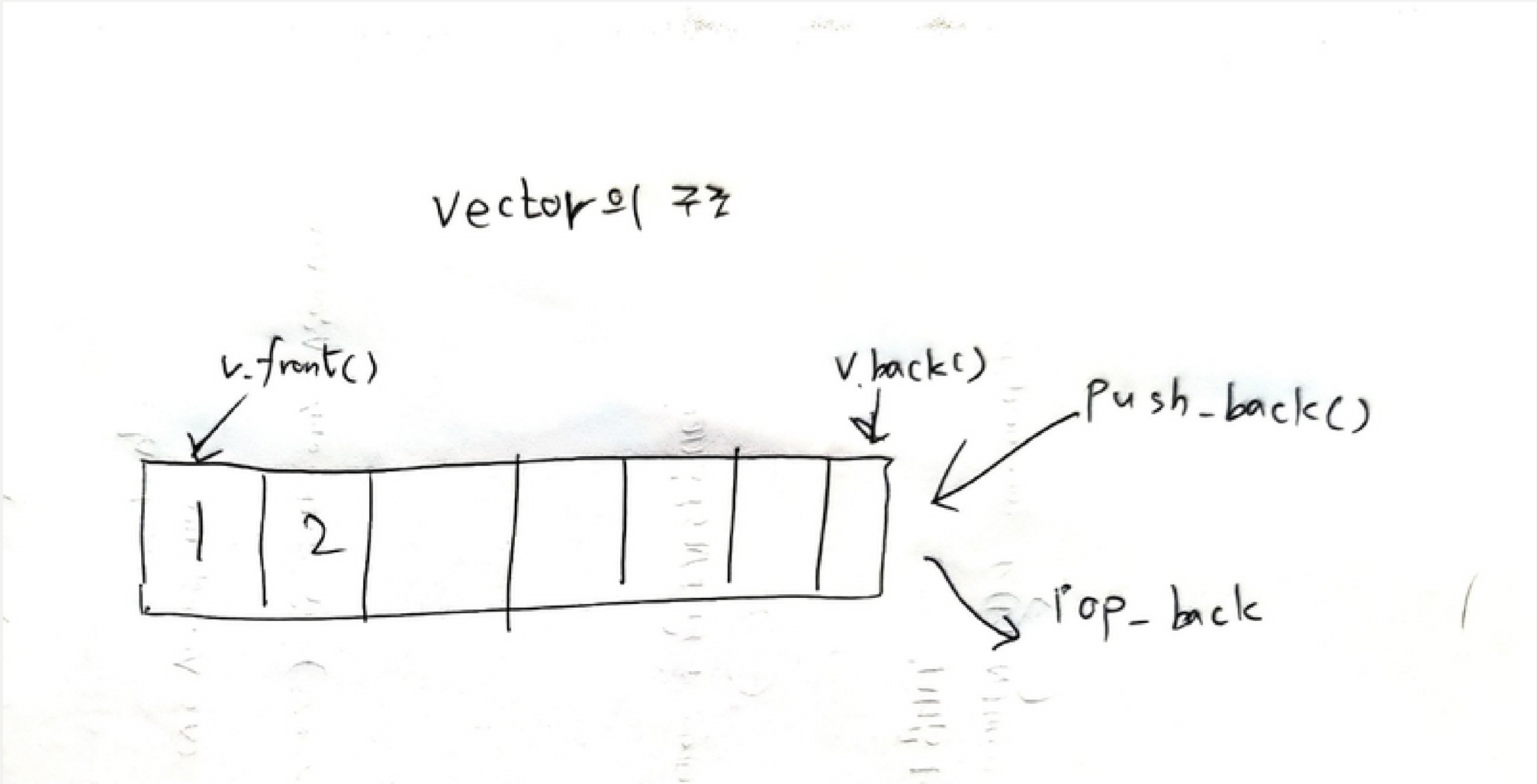
<장점>

1. 배열과 달리 자동으로 메모리를 할당시켜주어 처음부터 원소의 개수를 지정해둘 필요가 없고, 원소의 삽입/삭제 시 효율적인 메모리 관리가 가능하다.
2. vector의 중간의 원소를 삭제하거나, vector의 크기를 구하는 작업 등을 알아서 해주는 유용한 멤버 함수들이 많다.
3. 배열 기반이므로 랜덤 접근(Random Access)이 가능하다.

<단점>

1. 배열 기반의 container이므로 원소의 삽입, 삭제가 자주 수행되면 시간적인 측면에서 비효율적이다.

vector란?



우린 비슷한 것을 사용해왔다.

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

int main() {
    string s, t;
    s = "hello, ";
    t = "world!";
    s += t;
    cout << s;
    return 0;
}
```

```
hello, world!
```

마치 char형 vector

vector 사용법

```
#include <vector>
```

vector 사용법

```
#include <vector>

using namespace std;

  vector<int> v;
```

```
vector< pair<int,int> > v;
```

```
vector<[Data type]> [Name]
```

vector 생성자

vector<[type]> v	[type]형의 빈 vector를 생성
vector<[type]> v(n)	0으로 초기화 된 n개의 원소를 가지는 [type]형의 vector를 생성
vector<[type]> v(n, m)	m으로 초기화 된 n개의 원소를 가지는 [type]형의 vector를 생성
vector<[type]> v2(v1)	v1을 복사하여 v2 vector를 생성
vector<vector<[type]>> v	[type]형의 2차원 vector 생성
vector<[type]> v = {a1, a2, a3, ... }	{a1, a2, a3, ...} 으로 초기화 된 [type]형의 vector 생성

vector 생성자

```cpp
#include <iostream>
#include <vector>

using namespace std;



int main() {
    vector<int> v1;          // 빈 vector
    vector<int> v2(5);       // {0, 0, 0, 0, 0}
    vector<int> v3(5, 1);    //{1, 1, 1, 1, 1}
    vector<int> v4(v3);      // v4 = v3
    vector<vector<int>> v5; // 2차원 벡터
    vector<int> v6 = {1, 2, 3, 4, 5};
```

```cpp
    cout << "v1 : ";
    for (int i = 0; i < v1.size(); i++)
        cout << v1[i] << ' ';
    cout << "\nv2 : ";
    for (int i = 0; i < v2.size(); i++)
        cout << v2[i] << ' ';
    cout << "\nv3 : ";
    for (int i = 0; i < v3.size(); i++)
        cout << v3[i] << ' ';
    cout << "\nv4 : ";
    for (int i = 0; i < v4.size(); i++)
        cout << v4[i] << ' ';
    cout << "\nv6 : ";
    for (int i = 0; i < v6.size(); i++)
        cout << v6[i] << ' ';
}
```

vector 생성자

결과)

C:₩WINDOWS₩system32₩cmd.exe

```
v1 :
v2 : 0 0 0 0 0
v3 : 1 1 1 1 1
v4 : 1 1 1 1 1
v6 : 1 2 3 4 5 계속하려면 아무 키나 누르십시오 . . .
```

```cpp
int main() {
    vector<int> v1;          // 빈 vector
    vector<int> v2(5);       // {0, 0, 0, 0, 0}
    vector<int> v3(5, 1);    //{1, 1, 1, 1, 1}
    vector<int> v4(v3);      // v4 = v3
    vector<vector<int>> v5; // 2차원 벡터
    vector<int> v6 = {1, 2, 3, 4, 5};
```

```cpp
    cout << "v1 : ";
    for (int i = 0; i < v1.size(); i++)
        cout << v1[i] << ' ';
    cout << "\nv2 : ";
    for (int i = 0; i < v2.size(); i++)
        cout << v2[i] << ' ';
    cout << "\nv3 : ";
    for (int i = 0; i < v3.size(); i++)
        cout << v3[i] << ' ';
    cout << "\nv4 : ";
    for (int i = 0; i < v4.size(); i++)
        cout << v4[i] << ' ';
    cout << "\nv6 : ";
    for (int i = 0; i < v6.size(); i++)
        cout << v6[i] << ' ';
}
```

vector 사용법(1)

v.assign(n, m)	m으로 n개의 원소 할당
v.at(index)	index번째 원소를 반환한다. 유효한 index인지 체크하기 때문에 안전하다.
v[index]	index번째 원소를 반환한다. 배열과 같은 방식이며 유효성을 체크하지 않는다.
v.front()	첫 번째 원소를 반환한다.
v.back()	마지막 원소를 반환한다.
v.clear()	모든 원소를 제거한다. 메모리는 그대로 남아있게 된다. (size는 줄어들고 capacity는 유지)
v.begin()	첫 번째 원소를 가리키는 반복자(iterator)를 반환한다.
v.end()	마지막 원소 다음을 가리키는 반복자(iterator)를 반환한다.
v.push_back(m)	마지막 원소 뒤에 원소 m을 삽입한다.
v.pop_back()	마지막 원소를 제거한다.

vector 사용법

```cpp
void print_(vector<int> v) {
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << ' ';
    cout << '\n';
}

int main() {
    vector<int> v1; // 빈 vector

    v1.push_back(1);
    v1.push_back(2);
    print_(v1); // {1,2};

    v1[1] = 9999;
    print_(v1); // {1,9999};

    for (int i = 0; i < 10; i++)
        v1.push_back(i + 1);
    print_(v1); // {1,9999,1,2,3,4,5,6,7,8,9}

    v1.pop_back();
    v1.pop_back();
    print_(v1); // {1,9999,1,2,3,4,5,6,7}

    v1.clear();
    print_(v1); // { }
}
```

vector 사용법

```cpp
void print_(vector<int> v) {
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << ' ';
    cout << '\n';
}

int main() {
    vector<int> v1; // 빈 vector

    v1.push_back(1);
    v1.push_back(2);
    print_(v1); // {1,2};

    v1[1] = 9999;
    print_(v1); // {1,9999};

    for (int i = 0; i < 10; i++)
        v1.push_back(i + 1);
    print_(v1); // {1,9999,1,2,3,4,5,6,7,8,9}

    v1.pop_back();
    v1.pop_back();
    print_(v1); // {1,9999,1,2,3,4,5,6,7}

    v1.clear();
    print_(v1); // { }
}
```

```
1 2
1 9999
1 9999 1 2 3 4 5 6 7 8 9 10
1 9999 1 2 3 4 5 6 7 8
```

vector 사용법(2)

v.rbegin()	거꾸로 시작해서 첫 번째 원소를 가리키는 반복자(iterator)를 반환한다.
v.rend()	거꾸로 시작해서 마지막 원소를 가리키는 반복자(iterator)를 반환한다.
v.reserve(n)	n개의 원소를 저장할 공간을 예약한다.
v.resize(n) ☆	크기를 n개로 변경한다. 커진 경우에는 빈 곳을 0으로 초기화한다.
v.resize(n, m) ☆	크기를 n개로 변경한다. 커진 경우에는 빈 곳을 m으로 초기화한다.
v.size() ☆	원소의 개수를 반환한다.
v.capacity()	할당된 공간의 크기를 반환한다. (size()와 다름)
v2.swap(v1)	v1과 v2를 swap한다.
v.insert(iter, m)	iter가 가리키는 위치에 m의 값을 삽입한다. 그리고 해당 위치를 가리키는 반복자(iterator)를 반환
v.insert(iter, k, m)	iter가 가리키는 위치부터 k개의 m 값을 삽입한다. 다음 원소들은 뒤로 밀린다.
v.erase(iter) ☆	iter 반복자가 가리키는 원소를 제거한다. capacity는 그대로 유지된다.
v.erase(start, end) ☆	start 반복자부터 end 반복자까지 원소를 제거한다.
v.empty() ☆	vector가 비어있으면 true를 반환한다.

vector 사용법(2)

```
vector<int> v1; // 빈 vector

v1[5] = 200;   //error!
```

vector 사용법(2)

```
vector<int> v1; // 빈 vector

v1.resize(20);

v1[5] = 200;  // success!
```

vector 사용법(2)

```cpp
int main() {
    vector<int> v1; // 빈 vector

    int N;
    cin >> N;

    for (int i = 0; i < N; i++) {
        cin >> v1[i]; // error!
    }
}
```

```
10
1

 *  터미널 프로세스 "C:\Windows\System32\cmd.exe /d /c cmd /C C:\Users\JH\Documents\K_Su
kH\algorithm\algo_practice\practic"이(가) 종료되었습니다(종료 코드: 3221225477).
```

vector 사용법(2)

```cpp
int main() {
    vector<int> v1; // 빈 vector

    int N;
    cin >> N;

    for (int i = 0; i < N; i++) {
        int a;
        cin >> a;
        v1.push_back(a);
    }
}
```

vector 사용법(2)

```cpp
int main() {
    vector<int> v1; // 빈 vector

    int N;
    cin >> N;

    v1.resize(N);
    for (int i = 0; i < N; i++) {
        cin >> v1[i];
    }

    for (int i = 0; i < v1.size(); i++)
        cout << v1[i] << " ";
}
```

vector 사용법(2)

```cpp
int main() {
    vector<int> v1; // 빈 vector

    int N;
    cin >> N;

    v1.resize(N);
    for (int i = 0; i < N; i++) {
        cin >> v1[i];
    }

    for (int i = 0; i < v1.size(); i++)
        cout << v1[i] << " ";
}
```

```
5
1 2 3 4 5
```

```
1 2 3 4 5
```

vector 사용법(2)

```cpp
vector<int> v1 = {1,2,3,4};// 빈 vector
//입력받은 숫자만큼 pop_back()
int N;
cin >> N;
for(int i = 0 ; i < N; i++){
    v1.pop_back();
}
```

vector 사용법(2)

```cpp
vector<int> v1 = {1,2,3,4};// 빈 vector
//입력받은 숫자만큼 pop_back()
int N;
cin >> N;
for(int i = 0 ; i < N; i++){
    v1.pop_back();
}
```


```
6
1 2 3 4 -90900416 58903 16193304 16187584 1229414213 1346985812 1095262019 116262612
7804861 1885958236 1937136741 1701080931 1953064749 925907245 875705905 758396473 18
8707 0 -1214908404 134276637 2 949 1 1042 -1545952319 32 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 404226048 404232216 404232216 404232216 404232216 404232216 404232216 134744072 0
2136 673720360 673720360 673720360 673720360 673720360 673720360 134744072 202115080
116108 202116108 202116108 202116108 202116108 202116108 202116108 202116108 202116108
02116108 202116108 202116108 202116108 202116108 202116108 202116108 202116108 202116108
```

vector 사용법(2)

```
for (int i = 0; i < N; i++) {
    if (v1.empty() == true)
        break;
    v1.pop_back();
}
```

```
6
* 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

vector 사용이유

편리해서

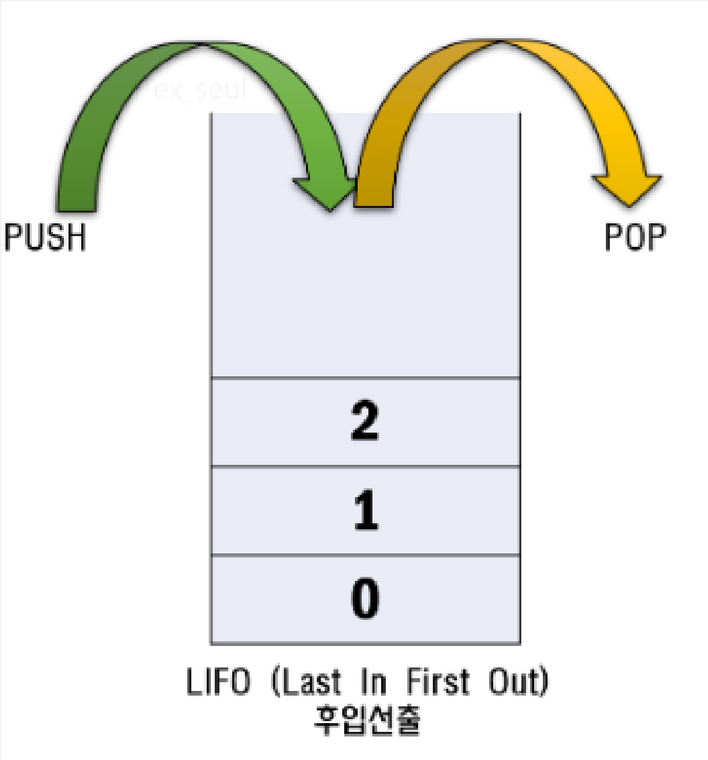
1. 시작부터 배열을 값으로 초기화 시킬 때,
2. push_back(), pop_back()을 쓰고싶을 때
3. sort함수사용을 편리하게 하고싶을때
4. range based for

vector 응용 (1): 생성과 초기화

```cpp
#include <vector>

using namespace std;

void print_(vector<int> v) {
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << ' ';
    cout << '\n';
}
int main() {
    vector<int> v1;
    vector<int> v2;

    v1.resize(5); // {0,0,0,0,0}
    print_(v1);

    v2.resize(5, 1); // {1,1,1,1,1}
    print_(v2);
}
```

```
0 0 0 0 0
1 1 1 1 1
```

vector 응용 (2): push_back



스택 (STACK)

push_back()
제일 뒤에 원소 추가

pop_back()
제일 뒤의 원소 제거

vector 응용 (2): push_back

1
3
5
4
0
0
7
0
0
6

- [1]
- [1,3]
- [1,3,5]
- [1,3,5,4]
- [1,3,5] (0을 불렀기 때문에 최근의 수를 지운다)
- [1,3] (0을 불렀기 때문에 그 다음 최근의 수를 지운다)
- [1,3,7]
- [1,3] (0을 불렀기 때문에 최근의 수를 지운다)
- [1] (0을 불렀기 때문에 그 다음 최근의 수를 지운다)
- [1,6]

```
if (user_in == 0) {
    arr.pop_back();
} else {
    arr.push_back(user_in);
}
```

vector 응용 (3): sort

sort 함수는 arr로 충분하지 않나요?

2차원 배열일때

```
int arr[3][2] = { {2,5} ,{4,3}, {7,1} };
sort(arr, arr + 3);
```

위와 같이 2차원 배열을 단순히 sort를 하면 아래와 같은 오류가 발생한다.

```
C2075        '_Val': 배열 초기화에는 중괄호로 묶인 이니셜라이저 목록이 필요합니다.
C3863        배열 형식 'int [2]'은(는) 할당할 수 없습니다.
```

오류 C2075 '_Val': 배열 초기화에는 중괄호로 묶인 이니셜 라이저 목록이 필요합니다

오류 C3863 배열 형식 'int [2]'은(는) 할당할 수 없습니다.

vector 응용 (3): sort

sort 함수는 arr로 충분하지 않나요?

9개의 서로 다른 자연수가 주어질 때, 이들 중 최댓값을 찾고 그 최댓값이 몇 번째 수인지를 구하는 프로그램을 작성하시오.

예를 들어, 서로 다른 9개의 자연수

3, 29, 38, 12, 57, 74, 40, 85, 61

이 주어지면, 이들 중 최댓값은 85이고, 이 값은 8번째 수이다.

vector<int, int> arr;
<value, index> 저장

vector 응용 (3): sort

```
sort(arr.begin(), arr.end());
cout << arr[8].first << "\n" << arr[8].second;
```

```
bool compare(pair<int,int> x, pair<int, int> y){
    return x.first > y.first;
}
 sort(arr.begin(), arr.end(), compare);
 cout << arr[0].first <<"\n" << arr[0].second;
```

vector 응용 (4): .range based for

range-based for

vector 응용 (4): .range based for

range-based for

```
for(auto i : arr){
    cout << i << " ";
}
```

vector 응용 (4): .range based for

range-based for

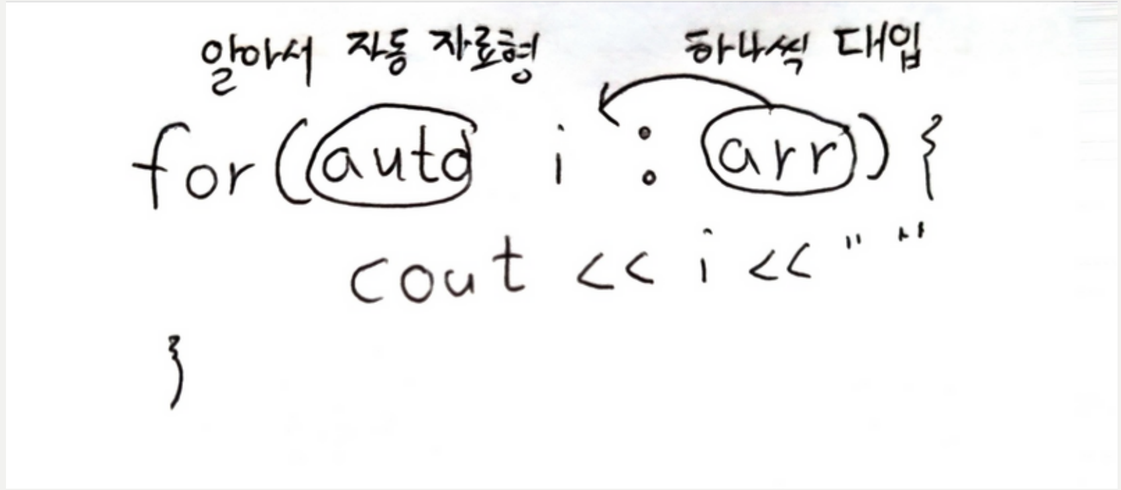
```cpp
for(auto i : arr){
    cout << i << " ";
}
```

```python
#python
for i in range(10) :
    print(i)
```

vector 응용 (4): .range based for

range-based for

```
for(auto i : arr){
    cout << i << " ";
}
```



vector 응용 (4): .range based for

```cpp
vector<int> arr = {1,2,3,4,5};

for(auto i : arr){
    cout << i << " ";
}
```

```
1 2 3 4 5
```

vector 응용 (4): .range based for

array로 해도 괜찮은거 아닌가요?

vector 응용 (4): .range based for

```cpp
int arr[5] = {1, 2, 3, 4, 5};

for (auto i : arr) {
    cout << i << " ";
}
```

```
1 2 3 4 5
```

vector 응용 (4): .range based for

```
int arr[10] = {1, 2, 3, 4, 5};


for (auto i : arr) {
    cout << i << " ";
}
```

```
1 2 3 4 5 0 0 0 0 0
```

vector 응용 (4): .range based for

```cpp
int arr[10] = {1, 2, 3, 4, 5};

for (auto i : arr) {
    cout << i << " ";
}
```

```
1 2 3 4 5 0 0 0 0 0
```

이게 왜 문제가 되나요??

vector 응용 (4): .range based for

(1 ≤ N ≤ 1,000,000,

```cpp
int main() {
    int arr[1000000];

    int N;
    cin >> N;
    for(int i = 0 ; i <N ; i++){
        cin >> arr[i];
    }

    for (auto i : arr) {
        cout << i << " ";
    }
}
```

vector 응용 (4): .range based for

```cpp
int main() {
    vector<int> arr;

    int N;
    cin >> N;

    for(int i = 0 ; i <N ; i++){
        int a;
        cin >> a;
        arr.push_back(a);
    }

    for (auto i : arr) {
        cout << i << " ";
    }
}
```

vector 응용 (4-2): 고급 기술

```
for(int i = 0 ; i <N ; i++){
    int a;
    cin >> a;
    arr.push_back(a);
}
```

```
arr.resize(N);

for(auto &i : arr){
    cin >> i;
}
```

vector 응용 (4-2): 고급 기술

```cpp
int main() {
    vector<int> arr;

    int N;
    cin >> N;

    for(int i = 0 ; i <N ; i++){
        int a;
        cin >> a;
        arr.push_back(a);
    }

    for (auto i : arr) {
        cout << i << " ";
    }
}
```

```cpp
int main() {
    vector<int> arr;

    int N;
    cin >> N;

    arr.resize(N);
    for(auto &i : arr){
        cin >> i;
    }

    for (auto i : arr) {
        cout << i << " ";
    }
}
```

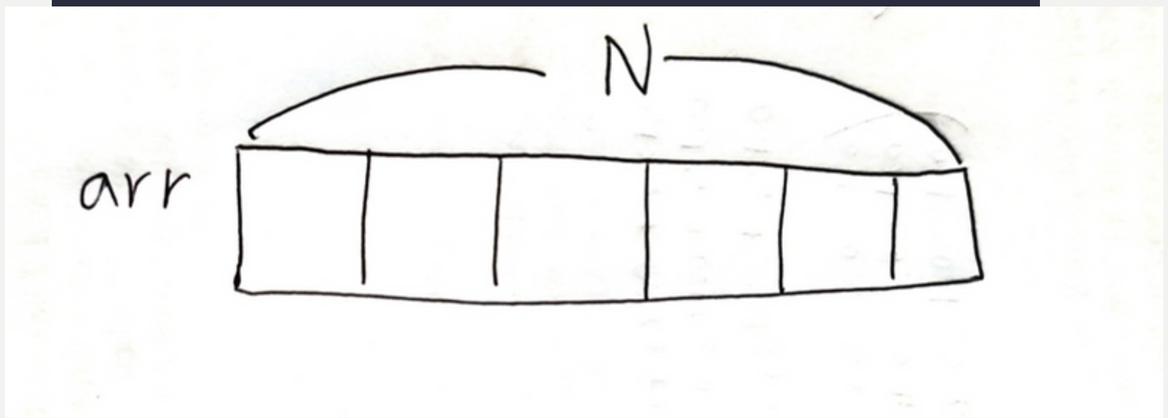
vector 응용 (4-2): 고급 기술

```cpp
int main() {
    vector<int> arr;

    int N;
    cin >> N;

    for(int i = 0 ; i <N ; i++){
        int a;
        cin >> a;
        arr.push_back(a);
    }

    for (auto i : arr) {
        cout << i << " ";
    }
}
```

```cpp
int main() {
    vector<int> arr;

    int N;
    cin >> N;

    arr.resize(N);
    for(auto &i : arr){
        cin >> i;
    }

    for (auto i : arr) {
        cout << i << " ";
    }
}
```

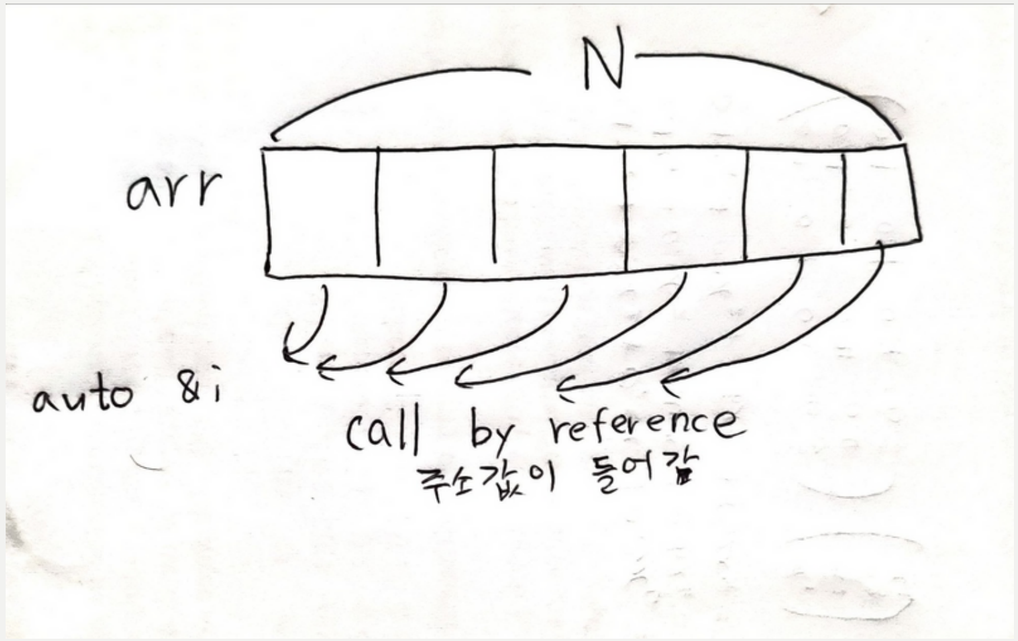
vector 응용 (4-2): 고급 기술

```
arr.resize(N);
```



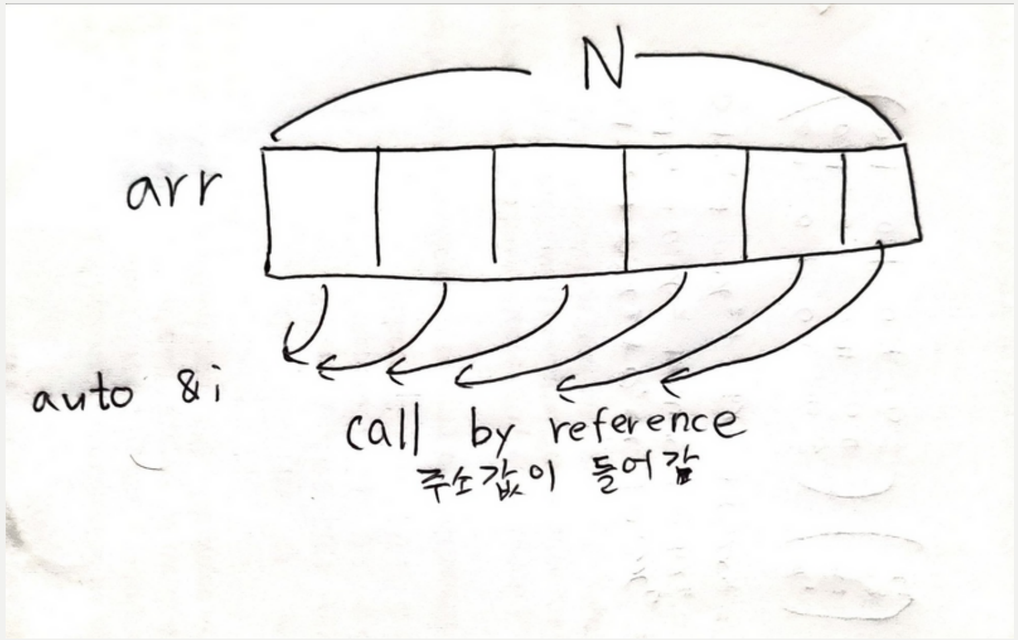
vector 응용 (4-2): 고급 기술

```
arr.resize(N);

for(auto &i : arr){
    cin >> i;
}
```



vector 응용 (4-2): 고급 기술

```
arr.resize(N);

for(auto &i : arr){
    cin >> i;
}
```



i 값을 바꾸면 arr내부의 값도 바뀜

vector 응용 (4-2): 고급 기술

```cpp
int main() {
    vector<int> arr;

    cout << "Input (N) : ";
    int N;
    cin >> N;

    arr.resize(N);
    cout << "Input arr : ";
    for (auto &i : arr) {
        cin >> i;
    }

    cout << "arr : ";
    for (auto i : arr) {
        cout << i << " ";
    }
}
```

```
Input (N) : 5
Input arr : 1 2 3 4 5
arr : 1 2 3 4 5
```

vector 응용 (4-2): 사용 실제 예시

첫째 줄에 테스트 케이스의 개수 t (1 ≤ t ≤ 100)이 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있다. 각 테스트 케이스는 수의 개수 n (1 < n ≤ 100)가 주어지고, 다음에는 n개의 수가 주어진다. 입력으로 주어지는 수는 1,000,000을 넘지 않는다.

출력

각 테스트 케이스마다 가능한 모든 쌍의 GCD의 합을 출력한다.

예제 입력 1 복사

```
3
4 10 20 30 40
3 7 5 12
3 125 15 25
```

vector 응용 (4-2): 사용 실제 예시

첫째 줄에 테스트 케이스의 개수 t (1 ≤ t ≤ 100)이 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있다. 각 테스트 케이스는 수의 개수 n (1 < n ≤ 100)가 주어지고, 다음에는 n개의 수가 주어진다. 입력으로 주어지는 수는 1,000,000을 넘지 않는다.

출력

각 테스트 케이스마다 가능한 모

예제 입력 1 복사

```
3
4 10 20 30 40
3 7 5 12
3 125 15 25
```

```cpp
arr.clear();
arr.resize(N);
for (auto &x : arr) {
    cin >> x;
}
```

vector 사용이유

편리해서

1. 시작부터 배열을 값으로 초기화 시킬 때,
2. push_back(), pop_back()을 쓰고싶을 때
3. sort함수사용을 편리하게 하고싶을때
4. range based for

5. 중복 값 제거 (set자료형과 유사한 효과)

.

.

.

.

문제

sort

2750, 11728

pair & sort

10818, 10814,2822, 2562

push_back

10773,4949

다 푼 사람은 3986