

삼성 청년 SW 아카데미

알고리즘

<알림>

본 강의는 삼성 청년 SW아카데미의 컨텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

Day2-1. 방향 배열

2차원 맵 정보가 주어지고, 문제에서 요구되는 조건을 처리해야 하는 경우가 많다.

특정 방향 값을 구할 때, 방향의 offset을 담은 배열을 활용해서

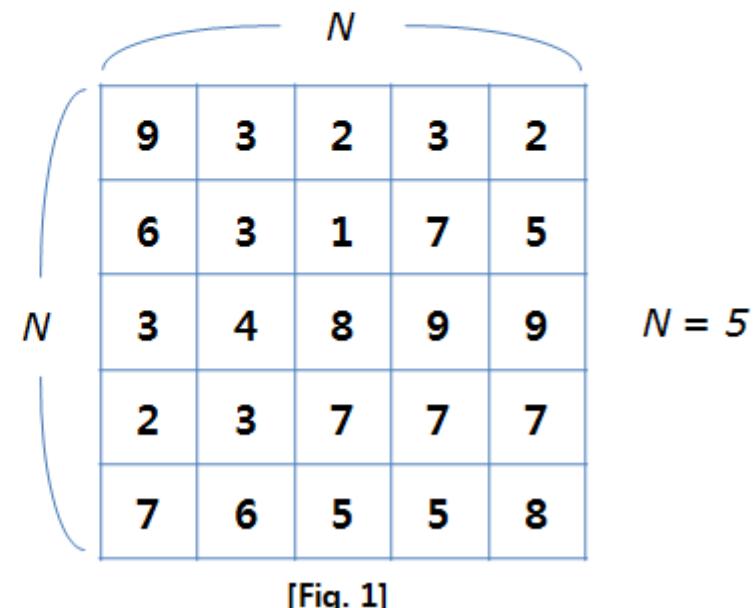
- if를 남발하지 않고,
- 간결하고
- 유지보수 하기 쉽게 작성할 수 있게 한다.

등산로 조성 문제

등산로를 조성하려고 한다.

등산로를 만들기 위한 부지는 $N * N$ 크기를 가지고 있으며, 이곳에 최대한 긴 등산로를 만들 계획이다.

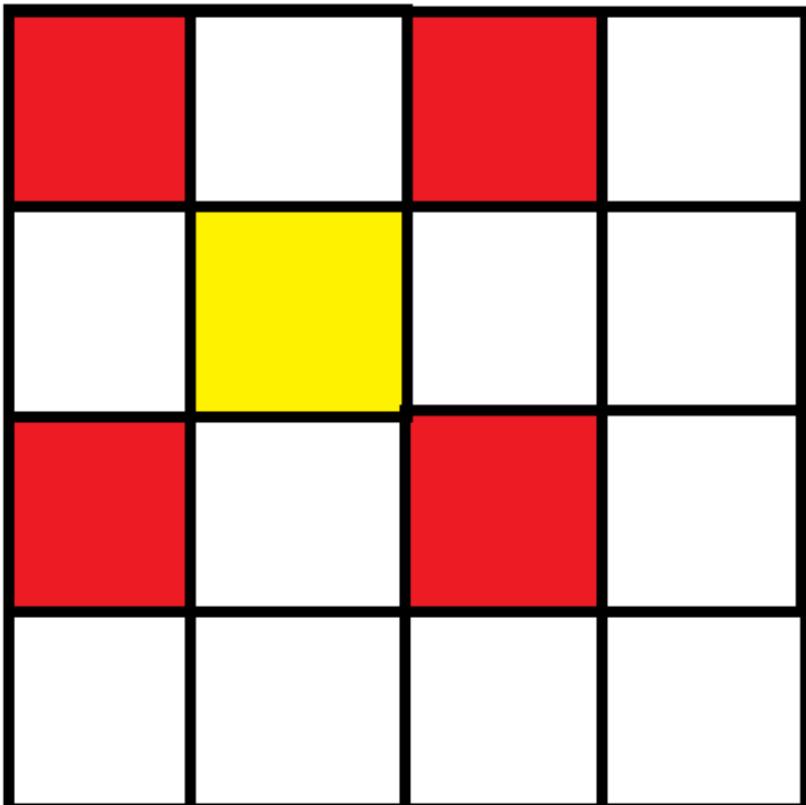
등산로 부지는 아래 [Fig. 1]과 같이 숫자가 표시된 지도로 주어지며, 각 숫자는 지형의 높이를 나타낸다.



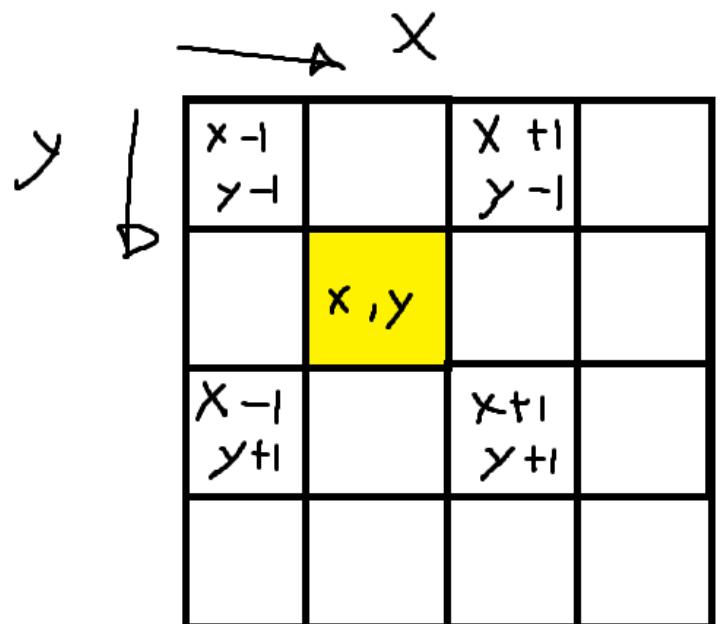
4x4 배열이 있다.

(1,1) 좌표 기준으로 대각선 방향에 1을 입력하자.

```
int arr[4][4] = { 0 };
```



$y=1, x=1$ 기준 대각선에 1 찍기



```
int y=1, x=1;
```

```
arr[y - 1][x - 1] = 1; //좌측 상단
```

```
arr[y - 1][x + 1] = 1; //우측 상단
```

```
arr[y + 1][x - 1] = 1; //좌측 하단
```

```
arr[y + 1][x + 1] = 1; //우측 하단
```

<https://gist.github.com/hoconoco/8c05f023bbde347871343ac6da69d8e0>

3X3 맵이 있다.

x,y 의 값을 입력 받고

상/하/좌/우의 합을 구하자

1	2	3
4	5	6
7	8	9

상/하/좌/우도 이제 쉽다

```
int x, y;  
cin >> y >> x;  
  
int sum = 0;  
  
sum = arr[y - 1][x] + arr[y + 1][x] + arr[y][x - 1] + arr[y][x + 1];  
cout << sum;
```

<https://gist.github.com/hoconoco/b3513e80c88e4365afdf09b516ffffe20>

좌표 1,2 을 입력하면

6을 기준으로 $3+5+9 = 17$ 이어야 하지만,

24가 나온다.

이유는 바로 7 값이 추가되었기 때문

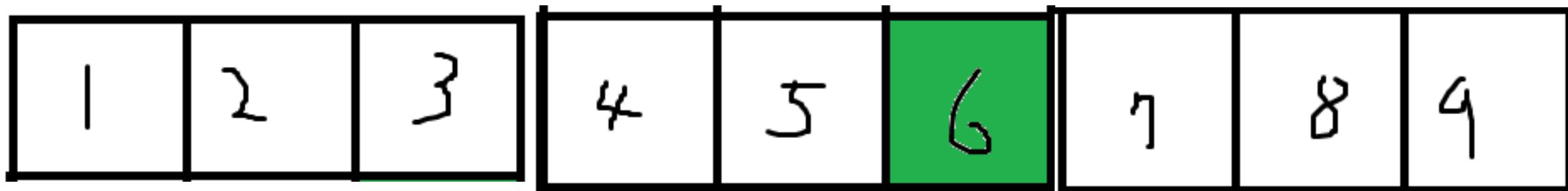
$$3+5+9+7=24$$

1	2	3
4	5	6
7	8	9

컴퓨터는 2차원 배열을 2차원으로 인식하지 않는다.

2차원으로 그리는 것은 사람을 위한 것이며,

컴퓨터는 내부적으로 1차원으로 인식한다.



arr[1][3] 을 출력할 수 있다
7이 나온다.

이런 사태를 방지하기 위해
범위를 지정해야 한다



```
int arr[3][3] = {  
    {1,2,3},  
    {4,5,6},  
    {7,8,9}  
};  
  
cout << arr[1][3];  
return 0;
```

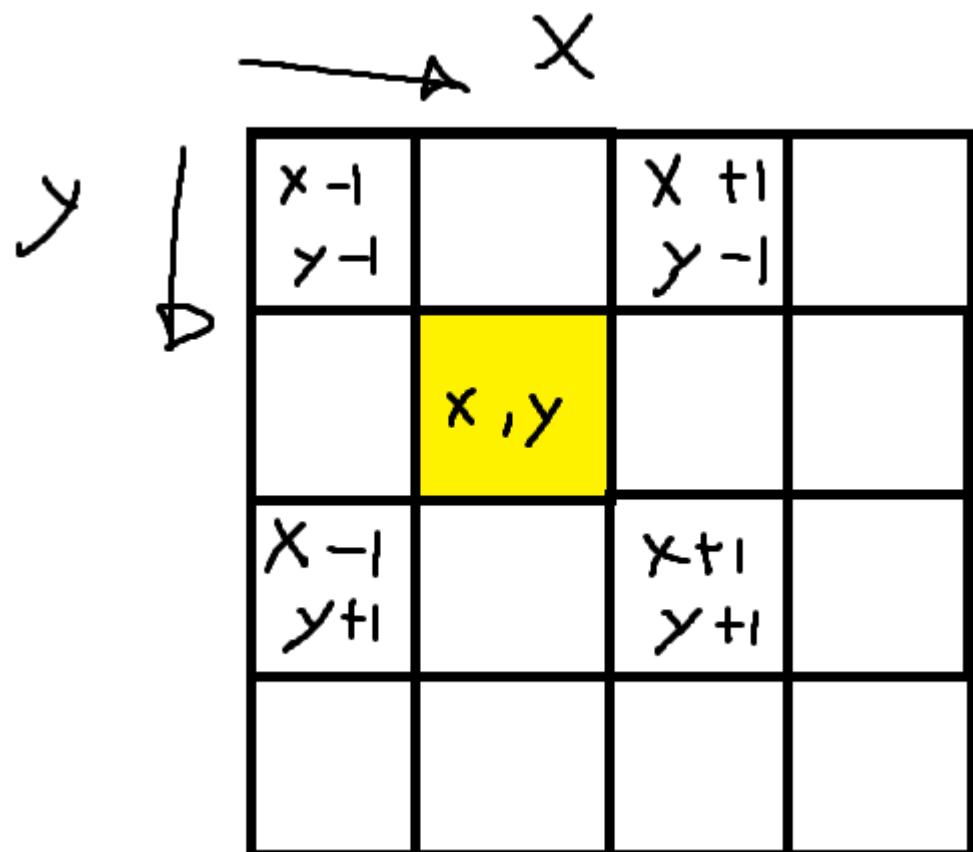
값을 더하기 전에 범위를 체크한다.

- 디버깅해서 조사한다.
- 코드가 유지보수하기 힘들다
- 이제 방향 배열을 도입하자

<https://gist.github.com/hoconoco/1dba0ae657ec9815d572b887347960c2>

```
int sum = 0;  
if( y-1>=0 )  
    sum += arr[y - 1][x];  
if( y+1<3 )  
    sum += arr[y + 1][x];  
if( x-1>=0 )  
    sum += arr[y][x - 1];  
if( x+1<3 )  
    sum += arr[y][x + 1];  
cout << sum;
```

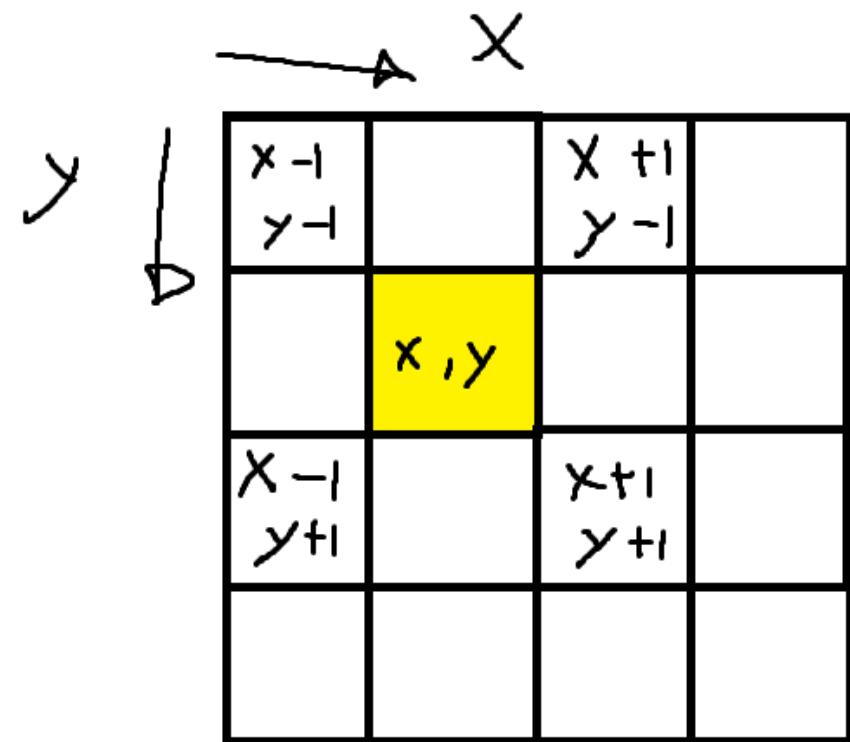
2차원 배열 또는 격자(grid) 기반 문제에서
특정 위치로 이동할 때 사용되는 도구이다.



x, y 뒤에 붙는 $+1, -1$ 을 순서대로 배열로 구현한다.

- 좌측상단 : 0
- 우측상단 : 1
- 좌측하단 : 2
- 우측하단 : 3

```
int dx[] = { -1, 1, -1, 1 };
int dy[] = { -1, -1, 1, 1 };
```



대각선 방향 이외에도 다양한 방향을 표현할 수 있다

- 상하좌우 탐색
- 8방향 탐색
- 체스의 나이트 이동
- 등등

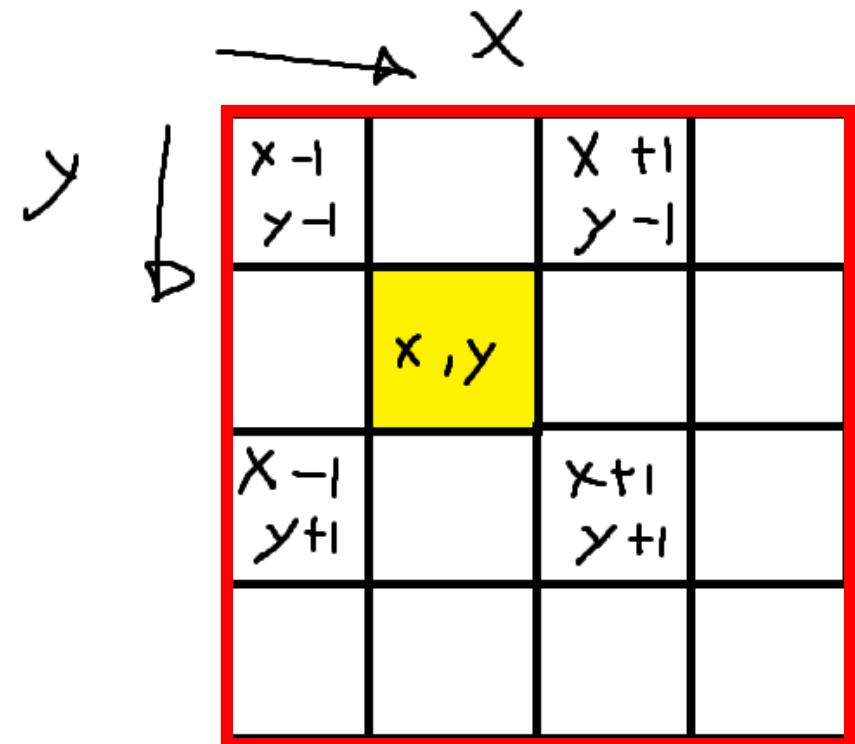
```
//방향배열  
int dx[] = { -1, 1, -1, 1 };  
int dy[] = { -1, -1, 1, 1 };  
  
for (int i = 0; i < 4; i++) {  
    int nx = x + dx[i];  
    int ny = y + dy[i];  
    arr[ny][nx] = 1;  
}
```

방향 배열을 사용하면 경계를 벗어나는 경우가 있다.

- 경계를 벗어나지 않는 코드도 구현해야 한다

기본적으로 맵의 경계를 벗어나면 안된다.

- or 문제에 따라 특정 경계를 구현해야 할 수 있다



코드를 기능 별로 구현했기 때문에,
디버깅도 매우 편리하다

- 레벨업을 위해선

최적화 or 유지보수를 위한 코드 작성이 필수다

<https://gist.github.com/hoconoco/c7c602471bbe18665a83e0b741c199b0>

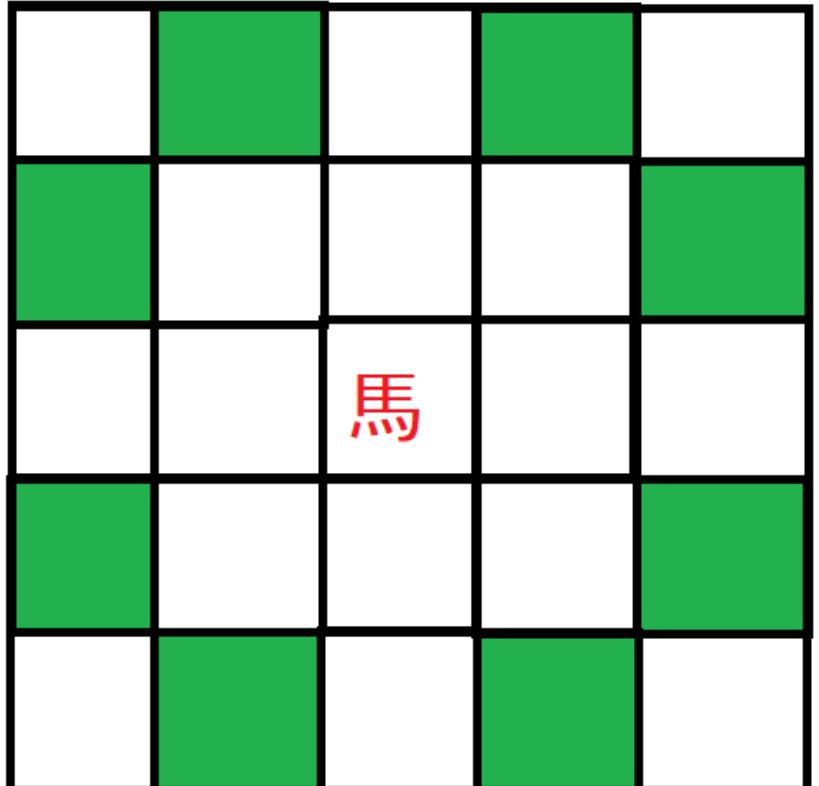
```
// 방향 배열
int dx[] = { -1, 1, -1, 1 };
int dy[] = { -1, -1, 1, 1 };

// 방향기반 탐색
for (int i = 0; i < 4; i++) {
    int nx = x + dx[i];
    int ny = y + dy[i];

    // 경계 벗어나면 무시
    if (nx < 0 || ny < 0 || nx >= 4 || ny >= 4) {
        continue;
    }
    // 기록
    arr[ny][nx] = 1;
}
```

장기의 마(馬)를 아는가?!

장기의 마 가 이동할 수 있도록 방향 배열을 작성하자!



이제 한 가지 더 응용할 수 있으면 된다

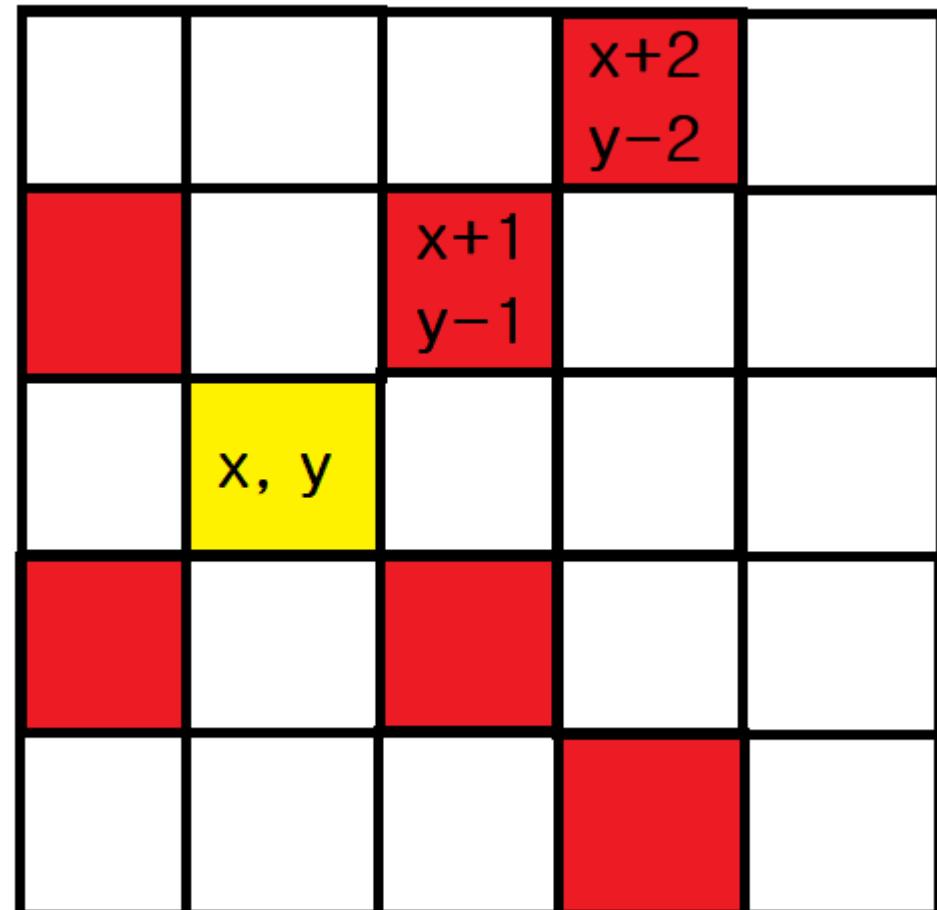
```
// 마의 이동 방향 배열  
int dx[] = { -2, -2, -1, -1, 1, 1, 2, 2 };  
int dy[] = { -1, 1, -2, 2, -2, 2, -1, 1 };
```

<https://gist.github.com/hoconoco/0b7ba025ef4d8a518b1bbd8094418ede>

기준 좌표에서 끝까지 모든 대각선 경로에 1을 기록하려 한다면?!

- while 도 좋지만, 방향을 기준으로 얼마나 더 갈지를
코드에 대입할 수 있으면 된다

```
int nx = x + dx[i];  
int ny = y + dy[i];
```



거리에 대한 변수를 추가하여
특정 방향으로 원하는 거리만큼
탐색이 가능하다

```
//방향기반 탐색
for (int i = 0; i < 4; i++) {
    //거리
    for (int d = 1; d < 5; d++) {
        int nx = x + dx[i] * d;
        int ny = y + dy[i] * d;

        // 경계 벗어나면 무시
        if (nx < 0 || ny < 0 || nx >= 5 || ny >= 5) {
            continue;
        }
        //기록
        arr[ny][nx] = 1;
    }
}
```

<https://gist.github.com/hoconoco/67f17f5ae22d2d6b2f6f6ee90d19274b>

Day2-2. vector

N의 크기만큼 배열을 생성하고
데이터를 입,출력하는 코드를 작성하려 한다

어떤 문제가 발생할 수 있을까?

- 배열 크기 알 수 없다.
- 메모리 문제 등등

int N;

C++에서는 동적으로 크기를 할당할 수 있는 vector를 사용할 수 있다

- `#include <vector>`
- `vector <데이터 타입> 이름;`

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

int main() {
    //vector<데이터타입> 이름;
    vector<int> a;
```

.push_back() : vector에 값 추가하기

.size() : vector 사이즈

```
//vector에 데이터 추가  
a.push_back(10);  
a.push_back(20);  
a.push_back(30);  
a.push_back(40);  
  
for (int i = 0; i < a.size(); i++) {  
    cout << a[i] << ' ';  
}
```

<https://gist.github.com/hoconoco/869eed35c7d73e5b17167a6f36f532fc>

F11 : 코드 한 줄씩 실행 (조사식 확인)

F10 : 함수 코드 한 줄로 실행

- 프로시저 단위 실행

F9 : 브레이크 포인트

F5 : 브레이크 포인트까지 실행

조사식 추가

- 디버그 → 창 → 조사식

The screenshot shows a debugger interface with a code editor and a watch window.

Code Editor: The file is named "FileName.cpp". It contains the following code:

```
#include<vector>
using namespace std;
int main() {
    vector<int> a;
    //vector에 데이터 추가
    a.push_back(10);
    a.push_back(20);
    a.push_back(30);
    a.push_back(40);
    for (int i = 0; i < a.size(); cout << a[i] << ' ')
```

A red dot at line 13 indicates a breakpoint.

Watch Window: The "조사식 1" (Watch 1) window displays the variable "a" and its elements. The variable "a" is of type std::vector<int, std::allocator<int>> and has a size of 4. Its elements are [0]: 10, [1]: 20, [2]: 30, and [3]: 40.

이름	값	형식
a	{ size=4 }	std::vector<int, std::allocator<int>>
[capacity]	4	unsigned _int64
[allocator]	allocator	std::_Compressed...
[0]	10	int
[1]	20	int
[2]	30	int
[3]	40	int
[Raw 뷰]	{_Mypair=allocator }	std::vector<int, std::allocator<int>>

굉장히 많은 API가 존재한다.

기본만 익혀본다.

.pop_back() : vector의 데이터를 뒤에서부터 삭제한다.

- 디버깅해서 확인한다.

```
//데이터 뒤에서부터 삭제  
a.pop_back();  
  
for (int i = 0; i < a.size(); i++) {  
    cout << a[i] << ' ';  
}
```

<https://gist.github.com/hoconoco/e60d7487a87a854b7915e0171f7c43dd>

.resize() : vector 크기 할당하기

.clear() : vector 데이터 전부 삭제

- 디버깅해서 확인한다.

```
a.resize(10);  
//10 출력  
printf("%d\n", a.size());  
  
a.clear();  
printf("%d\n", a.size());
```

<https://gist.github.com/hoconoco/33df39a88cf3d102a0fd4239de14439>

다양한 이유가 있지만,
Coding Test를 위해서도 꼭 필요하다.

SWEA 기준
N개의 테스트 케이스가 존재하고,
테스트 케이스마다 초기화할 필요가 있는 데, 그 때 자주 사용하게 된다.

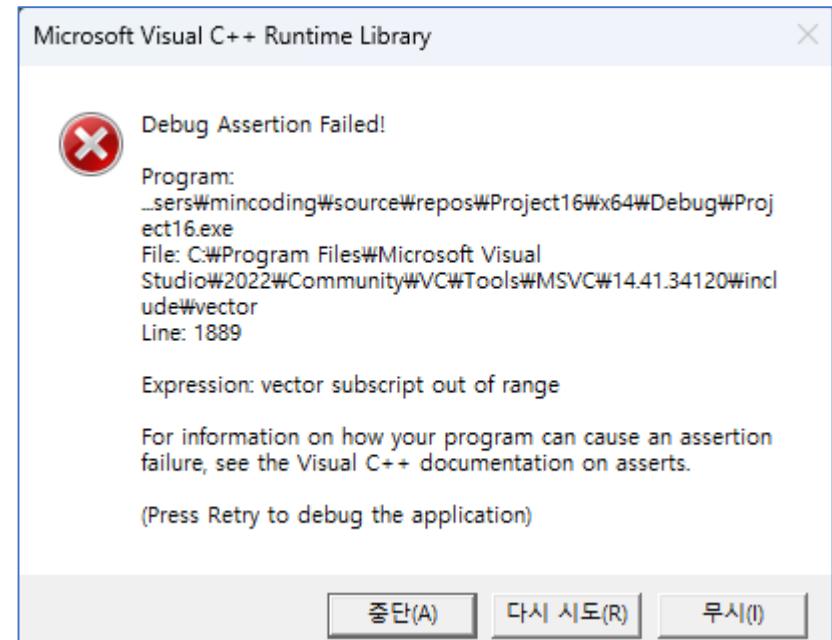
vector는 동적 배열이다 vector의 크기는 동적으로 변한다.

따라서 다음과 같은 코드를 사용 할 수 없다.

- 코드를 작성할 땐 아무런 문제가 없다.
- 다만, 실행하면 문제가 생긴다.
- 크기를 정해야 index 접근이 가능하다.

```
vector<int> a;

for (int i = 0; i < 10; i++) {
    cin >> a[i];
}
```



실행하면 배열처럼 입력이 가능하다

- 이 문제는 `vector`로 2차원 데이터를 표현할 때에도 유효하다

- `vector` 내 `vector`
- `vector` 내 `string`
- `vector` 내 구조체
- 등등

```
vector<int> a;  
a.resize(3);  
  
for (int i = 0; i < 3; i++) {  
    cin >> a[i];  
}
```

<https://gist.github.com/hoconoco/84294d9b272fbdc6ce49201257c10afb>

코드를 작성하고 빌드한다.

- 디버깅해서 확인한다.

//vector 생성하면서 크기 초기화 하는 방법

//크기 10

```
vector<int> a(10);
```

//vector 초기 크기 설정하면서 default 값 설정하기

//크기 10, 요소 전부 3으로 세팅

```
vector<int> b(10, 3);
```

<https://gist.github.com/hoconoco/6bcd46bc6b406743cf20542e186c2540>

a,b에 숫자 2개 입력 받아,
벡터를 $a*b$ 크기만큼 $a+b$ 값으로 초기화 하자

조사식으로 확인한다

```
int a, b;  
cin >> a >> b;  
  
vector<int> v(a * b, a + b);
```

<https://gist.github.com/hoconoco/c03e43b7352fdfdc3b5525e8d69e2f5e>

이름	값	형식
v	{ size=30 }	std::ve...
[cap...]	30	unsig...
[allo...]	allocator	std::...
[0]	13	int
[1]	13	int
[2]	13	int
[3]	13	int
[4]	13	int
[5]	13	int
[6]	13	int
[7]	13	int
[8]	13	int
[9]	13	int
[10]	13	int
[11]	13	int
[12]	13	int
[13]	13	int
[14]	13	int
[15]	13	int
[16]	13	int
[17]	13	int

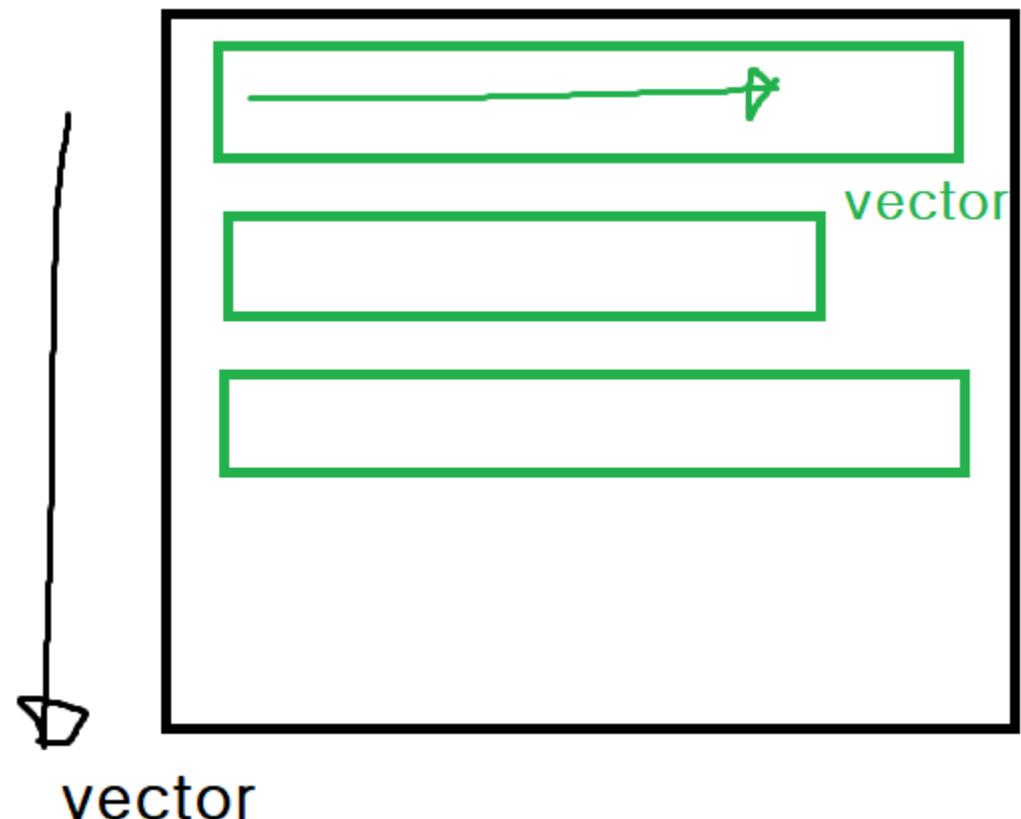
vector는 크기가 동적이기 때문에
개발자들의 많은 사랑을 받는다.

- vector 안에 vector
- vector 안에 구조체
- vector 배열 등등

우리는 그 중 코딩 테스트에 흔히 사용되는 형태를 다뤄 보자.

2차원 배열과 다르게
2차원 vector는 행, 열이 가변적이다

```
//2차원 vector 만들기  
vector< vector<int> > a;
```



코드를 작성하고 빌드한다.

- vector 형태로 넣으면 된다.
- 디버깅해서 확인한다.

```
//2차원 vector 만들기
vector< vector<int> > a;

//2x3 데이터 입력 받기
//입력 받는 데이터 개수가 늘어나는 만큼 n x 3이 된다.
for (int i = 0; i < 3; i++) {
    int x, y;
    cin >> x >> y;
    a.push_back( { x, y } );
}
```

<https://gist.github.com/hoconoco/7cf0641ce96a4d3dad2866dc353f423a>

구조체는 여러 타입의 변수를 한번에 담을 수 있기 때문에 반드시 여러 사용법을 익혀 한다.

vector

```
struct Node {  
    int a;  
    char b;  
};
```

```
struct Node {  
    int a;  
    char b;  
};
```

```
struct Node {  
    int a;  
    char b;  
};
```

```
struct Node {  
    int a;  
    char b;  
};
```

동일하다

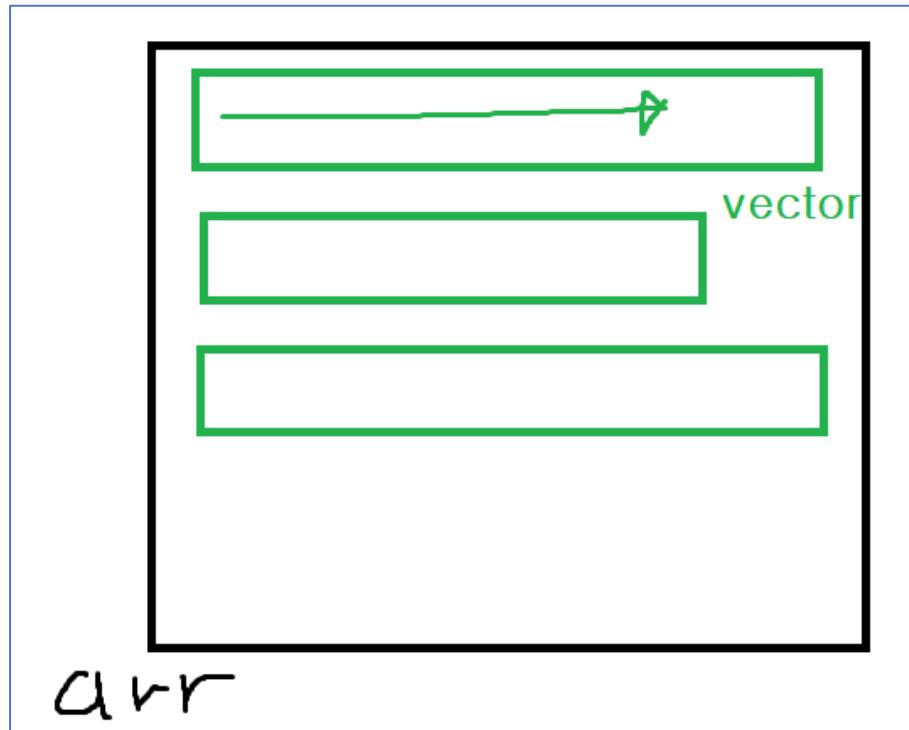
```
vector<struct Node> v;

for (int i = 0; i < 5; i++) {
    int na;
    char nb;
    cin >> na >> nb;
    //vector 내 vector 와 동일
    v.push_back({ na, nb });
}
```

<https://gist.github.com/hoconoco/379547dc29d04983d859012b4005bcb4>

캡데기가 배열이다. (vector 가 여러 개 존재하는 배열)

- 흔종 그 자체이지만, 코딩 테스트에서 자주 사용하는 형태이다. (그래프 - 인접 리스트 구현 시 사용)
- 코드만 간단히 살펴본다.



//vector로 된 배열
`vector<int> a[5];`

<https://gist.github.com/hoconoco/9702be38af0c2d7466e83c8236d59077>

2차원 데이터를 표현하는 방법은 굉장히 다양하고,
본인에게 편한 방법을 찾아서 익혀야 한다.

내일
방송에서
만나요!

삼성 청년 SW 아카데미