민 선 생 코 딩 학 원 훈 련 반

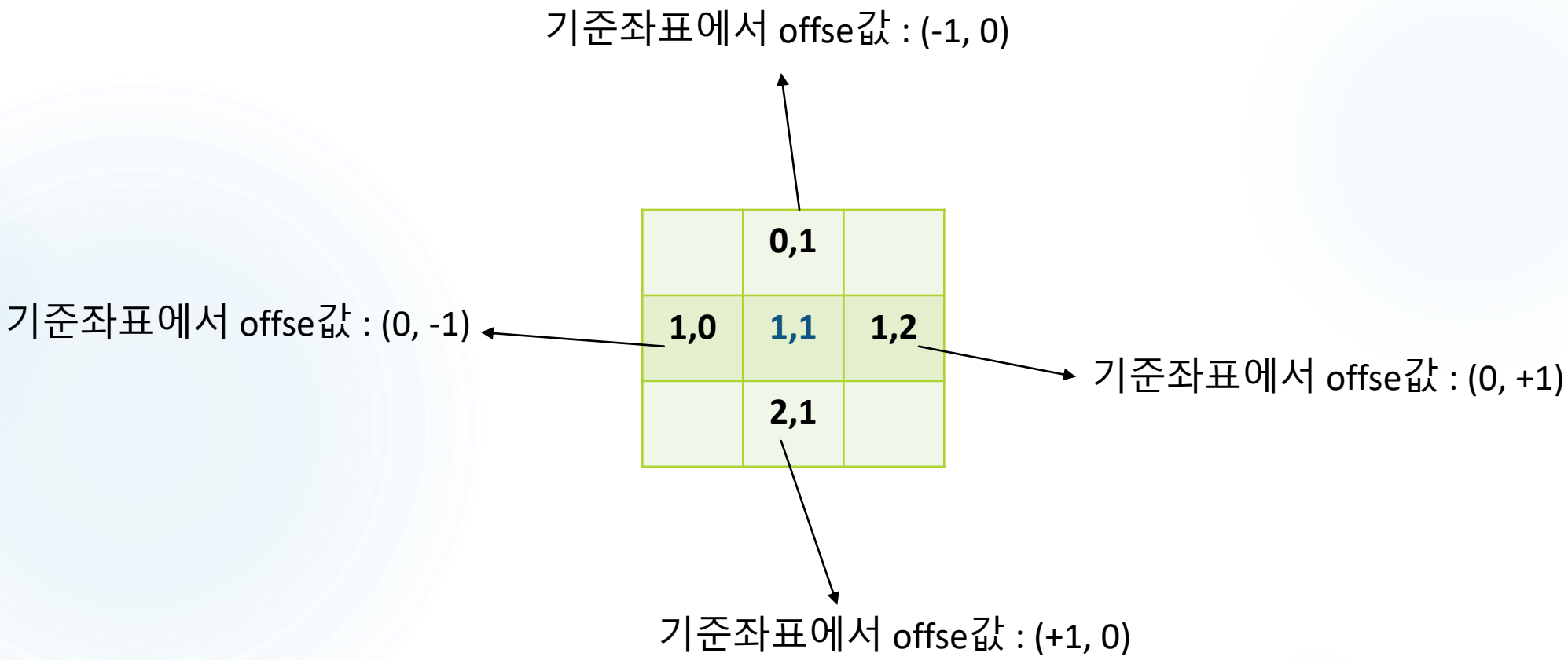
수업노트 LV-19

배우는 내용

1. Direct 코딩 기법
2. 2중 포인터
3. 구조체 배열
4. 2중 for문 패턴 찾기

네 방향 Offset

- Base 좌표가 **(1, 1)**일 때, 상하좌우칸의 Offset값은?



위 아래 왼쪽 오른쪽 합 구하기

- dy, dx 변수에 기준 좌표를 입력 받고,
  상하좌우의 합을 구하는 방법

→ **sum = d[y - 1][x] + d[y + 1][x] + d[y][x -1] + d[y][x + 1]**

만약 1, 1을 입력받았다면?

1	3	7	2
2	2	6	1
1	4	5	1
1	1	2	1

2 + 3 + 4 + 6 = 15

**단, 이 방식은 dy dx가 (0,0) 일 때
메모리 침범되는 버그 발생**

네 방향 합을 for문으로 구하기1

- Base 좌표가 y, x일 때 네 방향의 Offset은

  (y - 1, x + 0) -> 위
  (y + 1, x + 0) -> 아래
  (y + 0, x - 1) -> 왼쪽
  (y + 0, x + 1) -> 오른쪽

- direct라는 배열에 offset을 저장

direct

-1	**0**
1	**0**
0	**-1**
0	**1**

direct라는 offset 배열을 이용하여 for문으로 합 구할 수 있음
→ **Direct 코딩 기법**

네 방향 합 for문으로 구하기2

- for문을 네 번 반복시켜 네 방향의 합을 구할 수 있음
- dy, dx는 base + offset을 한 좌표를 뜻 함

```
for (t = 0; t < 4; t++)
{
    dy = y + direct[t][0];
    dx = x + direct[t][1];

    sum += data[dy][dx];
}
```

direct

-1	0
1	0
0	-1
0	1

1	3	7	2
2	2	6	1
1	4	5	1
1	1	2	1

- 이렇게 코딩하는 방식을 Direct라고 함

이 방식도 dy, dx가 (0,0) 일 때
메모리 침범되는 버그 발생

dy dx값이 배열 범위 넘어가는 버그 방지

- 이렇게 if문을 걸어서
  계산된 dy, dx좌표가 배열 범위 안에 있을 때만 계산

```
for (t = 0; t < 4; t++)
{
    dy = y + direct[t][0];
    dx = x + direct[t][1];

    if (dy >= 0 && dy < 4 && dx >= 0 && dx < 4)
    {
        sum += arr[dy][dx];
    }
}
```

1	3	7	2
2	2	6	1
1	4	5	1
1	1	2	1

버그가 발생하지 않음

Direct 예시 – 상하좌우의 합이 최소 인 곳 구하기

- getMin 함수를 사용하여 구한다.
- min값이 갱신될때 마다
  minDy, minDx 업데이트

```
void getMin(int y, int x)
{
    int t;
    int sum = 0;
    int dx, dy;

    for (t = 0; t < 4; t++)
    {
        dy = y + direct[t][0];
        dx = x + direct[t][1];

        if (dy >= 0 && dy < 4 && dx >= 0 && dx < 4)
        {
            sum += arr[dy][dx];
        }
    }

    if (min > sum)
    {
        min = sum;
        minDy = y;
        minDx = x;
    }
}
```

코딩 설계의 필요성

- for문 훈련문제를 통해 **설계하는 습관을 만듬**
  (설계 : 어떻게 내가 코딩할 것인지 미리 **계획을 적어두고**, **시뮬레이션** 돌려보는것)

- 익숙하지 않은 문제를 보자마자 설계없이 코딩을 하게 된다면?
  - **외운 소스코드를 아무 생각없이 적게 되어,**
    아무리 수정해도 완성되지 않는 프로그램이 만들어짐
  - **잘못된 방향으로 코딩**
  - **생각지도 못한 케이스 출현 가능성 큼**

아주 익숙한 코딩이 아니라면, 설계는 반드시 할 것

1중 포인터 복습

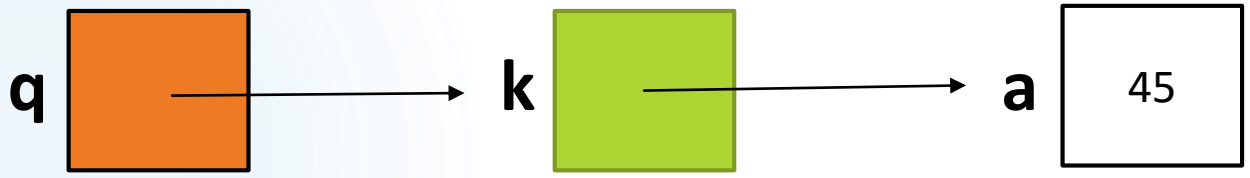
- **1중 포인터** : 변수를 가르키는 포인터


```c
int main()
{
    int a = 45;

    int* k;

    k = &a;

    *k = 100;
}
```

포인터 k가 a의 주소를 저장하고 있을 때
k가 a를 가르킨다 라고 표현한다

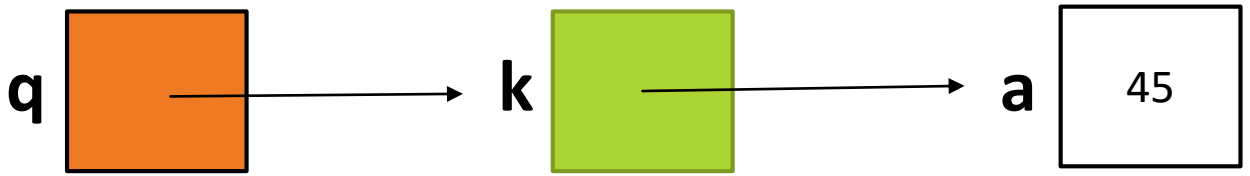
2중 포인터

- **1중 포인터** : 변수를 가르키는 포인터
- **2중 포인터** : 1중 포인터를 가르키는 포인터




```c
int main()
{
    int a = 45;
    int* k = &a;
    int** q;

    q = &k;
}
```

포인터 능력 사용하기



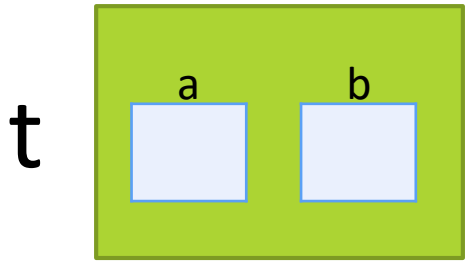
- 위 그림과 같이 가르키는 상태일 때

```
*k = 100;        // a에 100이 들어감
**q = 200 ;      // a에 200이 들어감
```

구조체 변수 만들기 복습

- 구조체 복습

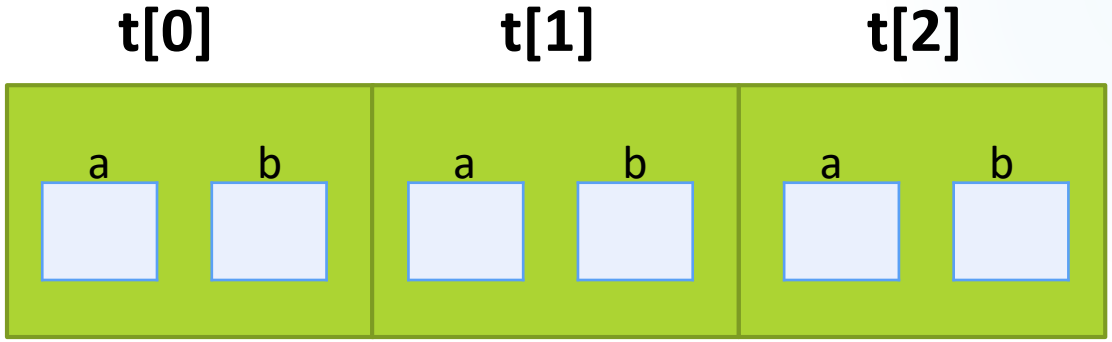
```
struct ABC
{
    int a;
    char b;
};

int main()
{
    ABC t;


    return 0;
}
```



구조체 배열 선언 방법

- 구조체 배열 선언 방법

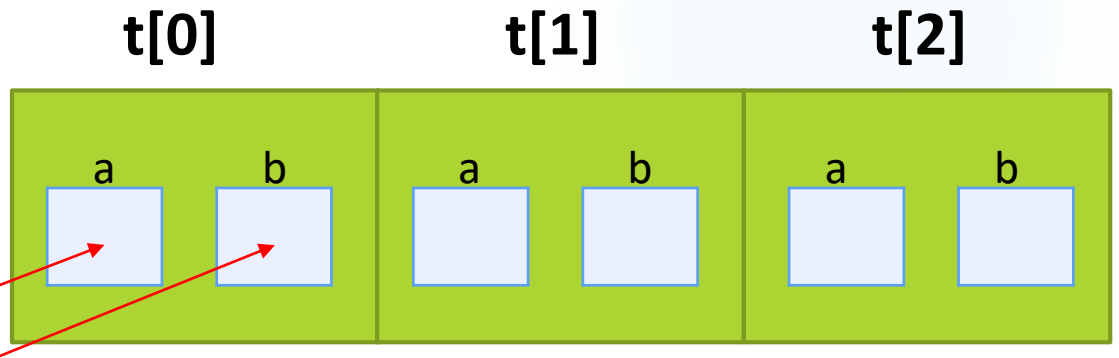
```
struct ABC
{
    int a;
    char b;
};

int main()
{
    ABC t[3];


    return 0;
}
```



구조체 배열 입력 방법

- 구조체 배열 입력 방법

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

int main()
{
    ABC t[3];
    int x;

    for (x = 0; x < 3; x++)
    {
        cin >> t[x].a >> t[x].b;
    }

    return 0;
}
```



isPattern 함수를 이용하여 2차 배열의 패턴 찾기 – main 함수

map

1	3	5	1	5
3	7	9	5	8
4	5	3	9	7
2	7	6	1	9
1	6	2	3	8

pattern

3	5
7	9

isPattern함수에 좌표를 넘겼을 때
Pattern과 똑같은지 아닌지 알려주는 함수라고 할 때
main함수는 이렇게 만들 수 있음

```
int main()
{
    int x, y;
    int flag = 1;

    for (y = 0; y <= 3; y++)
    {
        for (x = 0; x <= 3; x++)
        {
            int ret = isPattern(y, x);

            if (ret == 1)
            {
                flag = 1;
                break;
            }
        }

        if (flag == 1) break;
    }

    if (flag == 1) cout << "발견";
    else cout << "미발견";
}
```

isPattern 함수를 이용하여 2차 배열의 패턴 찾기 – isPattern 함수

map

1	3	5	1	5
3	7	9	5	8
4	5	3	9	7
2	7	6	1	9
1	6	2	3	8

pattern

3	5
7	9

isPattern함수에 좌표를 넘겼을 때
Pattern과 똑같은지 아닌지 알려주는 함수

```
int isPattern(int dy, int dx)
{
    int x, y;
    for (y = 0; y < 2; y++)
    {
        for (x = 0; x < 2; x++)
        {
            if (pattern[y][x] != map[dy + y][dx + x])
            {
                return 0;
            }
        }
    }

    return 1;
}
```