<Day0518>

* 포인터 리뷰

상속의 목적 –

1.재사용성

2.서로 다른 타입을 동종의 타입으로 처리하기위해 (upcasting을 통해 부모의 타입으로)

Void \* 는 모든 타입을 다 담을수있다.(어떤 타입인지 모르므로 이것이 단점이 될수도있다)

포인터는 주소를 담기위한 변수.

주소- 데이터를 주기억장치RAM에 적재를 하는데 그 각각의 방에대한 번호(0부터 시작하는 양의정수)

32비트 – 주소선 32가닥(방번호를 정해주기위한 선)

2의 32승=1g x 4 = 4g x 1byte = 4GB 메모리를 4G이상 박을수없다..

64비트는 주소선 32가닥 + 데이터선 32가닥(데이터를 ram에서 cpu로 가져오는선)

자동변수(=지역변수) – 선언만 해주면 스택에 쌓이게 된다. int age;

자동변수라고 하는이유는? 동적메모리할당을할때(malloc)-heap에

Cpu는 물리메모리의 최소단위를 4btye로 한다. 그후 메모리의 위치의 이름을 age로 하겠다.

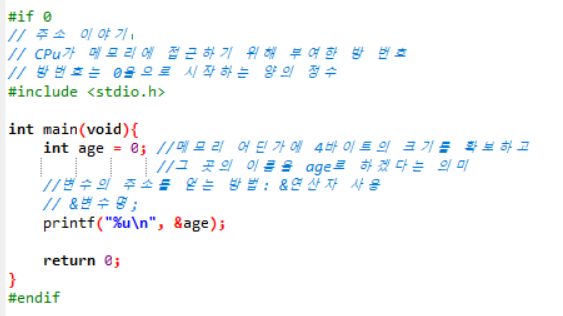
Age의 위치를 알기위해서는 &를 사용.

%p를 쓰는이유 – 16진수로 찍겟다.

%u – 0을 포함한 양의 정수unsigned로.

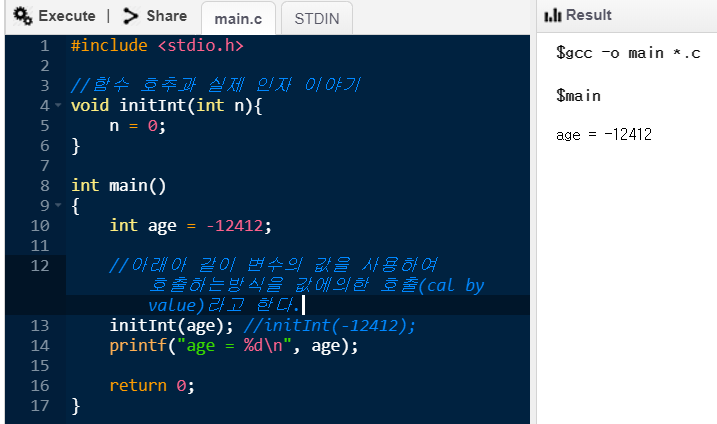
* 둘다 가능.

&연산자는 단항연산자이므로 오른쪽에 하나.즉 age만 존재해야한다.



변수가 그대로 전달되는 것이 아니라 복사가 되어 호출된다.

위와 같은 방식을 ‘값에 의한 호출’



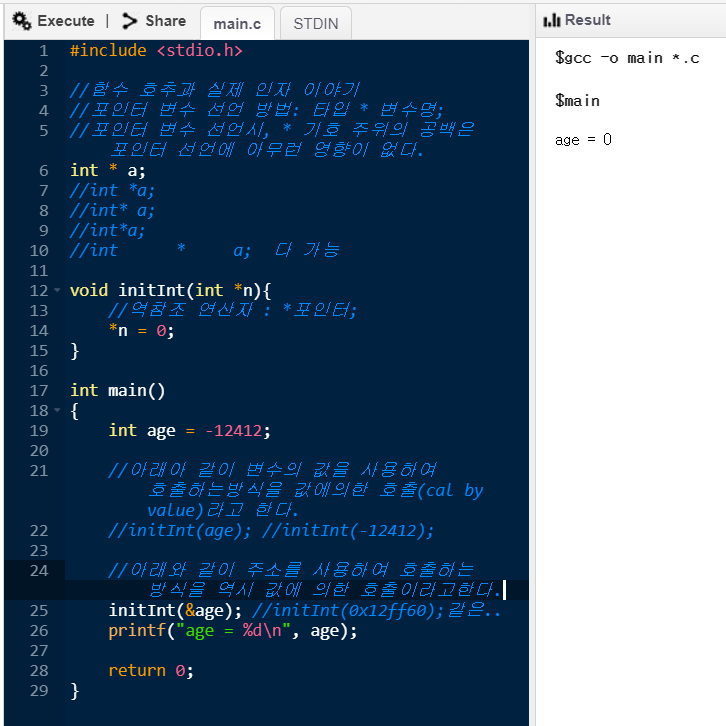
주소는 0을 포함한 양의 정수인데 그것을 int 에 집어 넣는 것은 말이 되지만

위치에 대한 변수를 저장하기위한 방법으로 \* 를 사용해야한다.

Cf. char \* p, q; //앞의 p는 char\*로 해석되지만 q는 char로 해석된다.

c언어에서는 모든 변수는 값에의한호출 밖에 없다. (참조에의한 호출은 없다..)

다만 포인터를 가지고 참조에의한 호출을 흉내낼뿐이다.



1. &age
2. 타입 \*변수명;
3. 역참조: \*ptr;

🡺이것 3가지를 기억하자!

\*\* \*\*\*를 사용하는 이유는 무엇일까? -

대상체의 주소를 담기위해 \*(일중포인터)사용함

Int\* 의 타입의 주소를 담기위해 int \* \*pp 라고 쓴다.

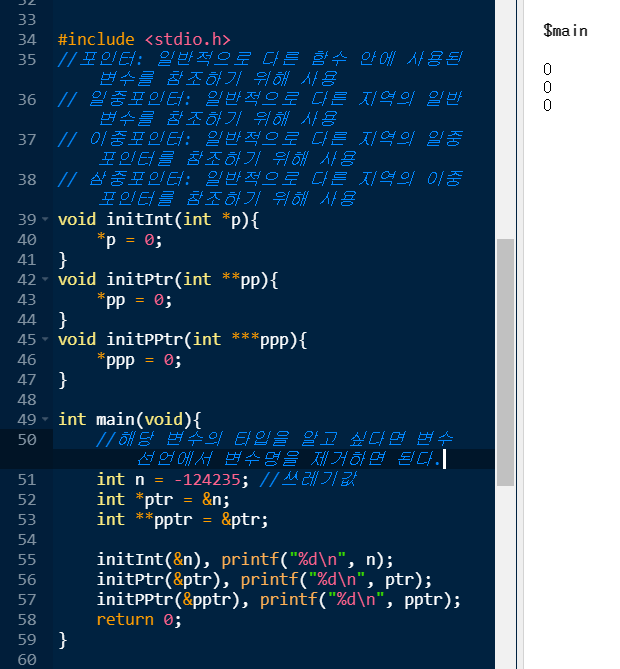
(0x30)

0x30

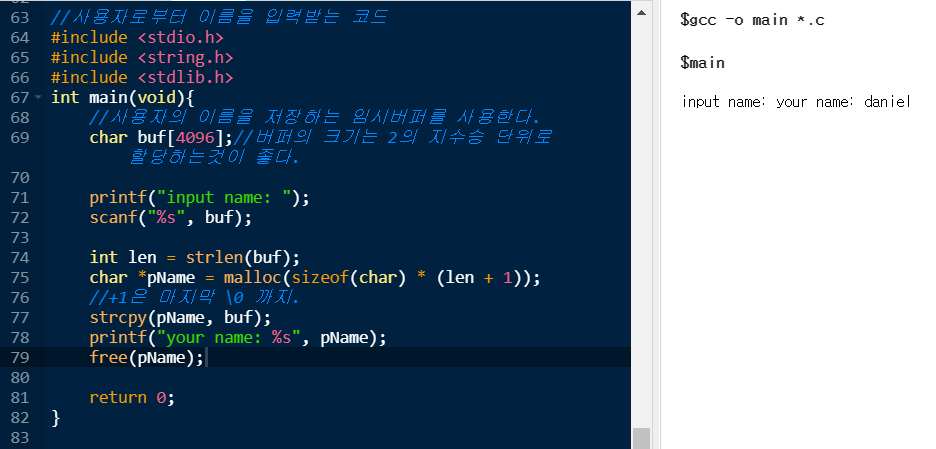
Int \* \*pp ------🡪 int \*p

이중포인터->일중 포인터에서 역참조는 두번이 아니라 1번이다!

<다중 포인터의 의미>

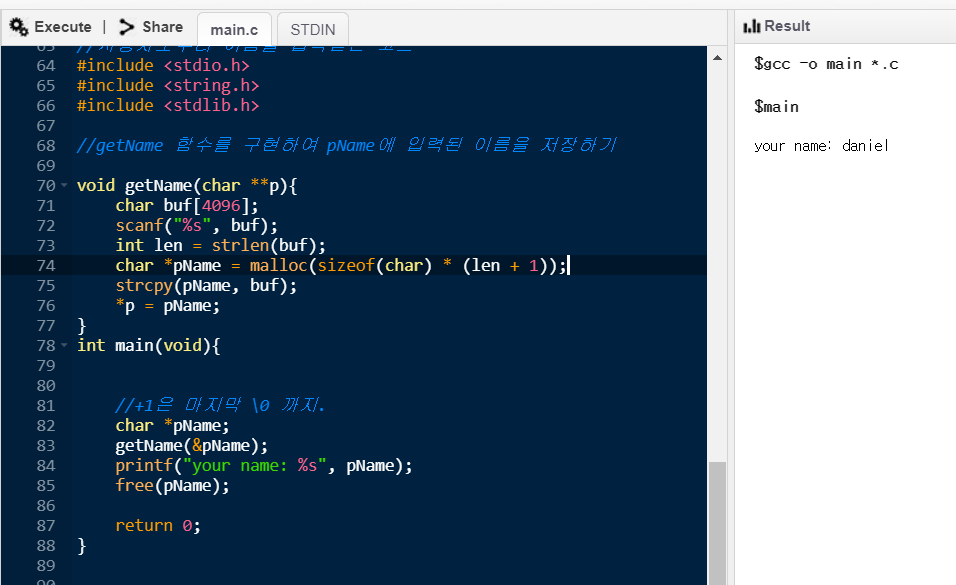


사용자의 이름을 임시로 저장할 공간이 필요(application buffer)

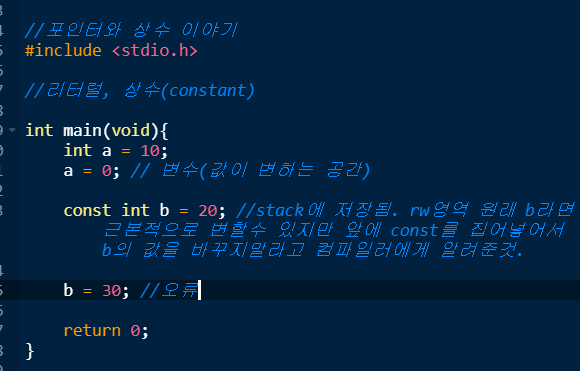


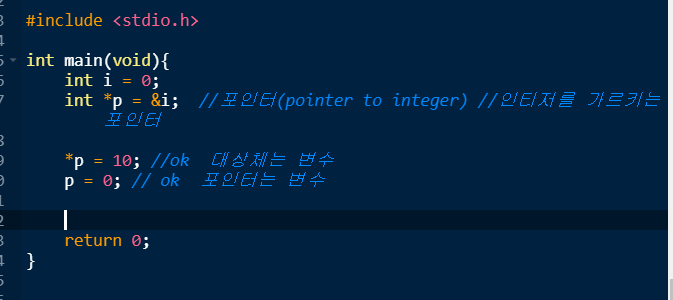
닫지 않고 계속 만들게되면 사용할수 있는 heap메모리가 줄어든다.(memory leak 발생)

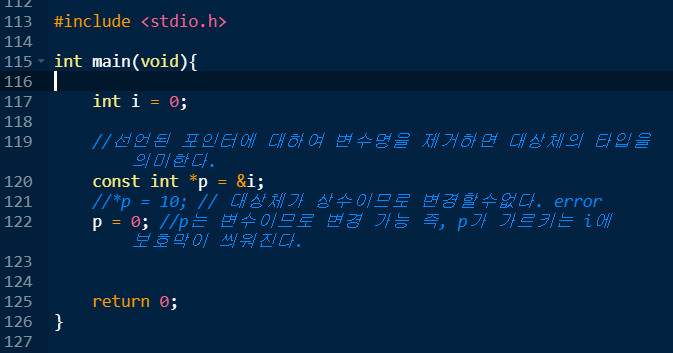
->사용되지 않는 자원은 그때그때 지워줘야한다. free(pName);



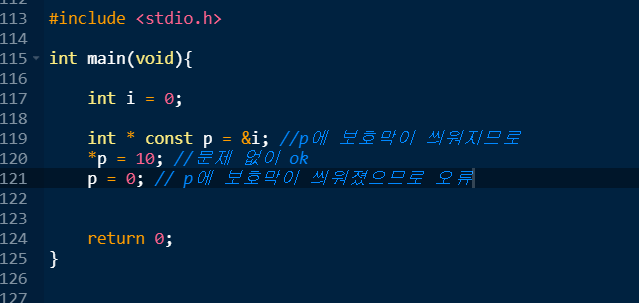
1. data – 전역변수
2. stack – 지역변수
3. heap – 동적할당 등 1.2.3은 rw 가능하다
4. text – ro만 가능(read only!) 20=0 에서 20이 literal이다.



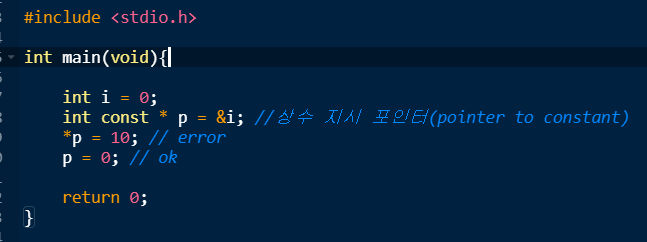




상수 지시 포인터 (pointer to constant)

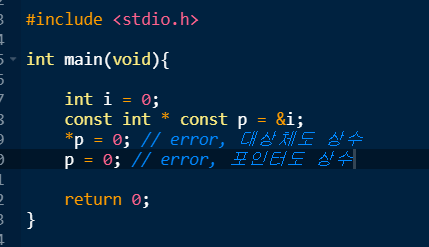


상수포인터(constant pointer to)



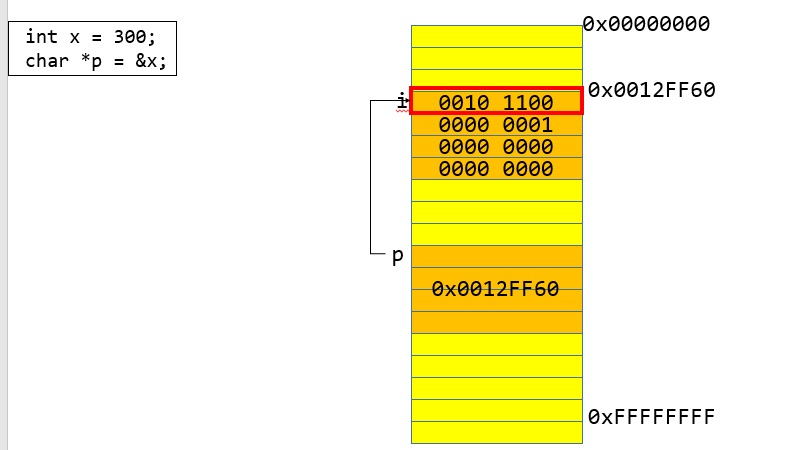
상수지시 포인터2

정리! 포인터 기호를 중심으로 \* 가 왼쪽에 있으면 대상체가 상수화 / 오른쪽에 있으면 포인터가 상수화가 된다.



상수 지시 상수포인터(constant pointer to constant)

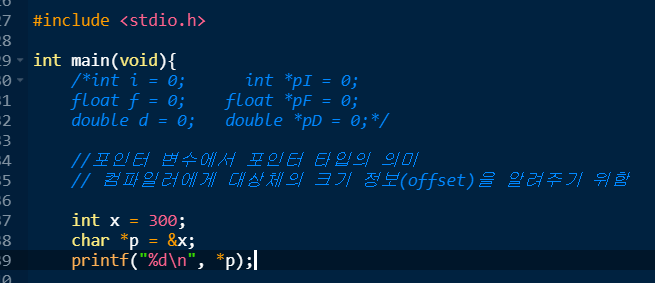
포인터의 타입을 왜 대상체의 타입과 일치시켜야할까? 어차피 4바이트인데?



P가 가르키는 곳으로 이동했을 때 그 데이터를 읽을 크기의 정보가 \* 앞에 붙어있는타입이다.

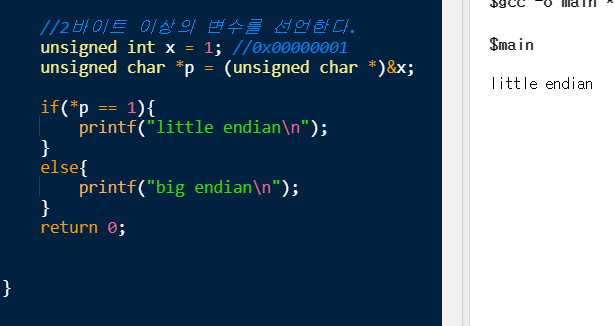
만약 4바이트중 제일 앞 바이트에 0010 1100 이고 두번째 바이트를 0000 0001이라하면 대상체는 int엿지만 포인터가 char이면 앞에 한바이트만 읽게 되므로 올바른 값이 안나온다.

->그래서 반드시 포인터타입은 대상체의 타입과 일치시켜주자.



그렇다면 위예제에서 항상 타입을 일치를 시켜야하는가? 그렇진 않다. 돌아가는 시스템이 빅엔디안인지 리틀엔디안인지 알기위해!

컴퓨터가 돌아가는 시스템이 빅엔디안인지 리틀엔디안인지 비교해줌



1. 배열 array : 동일한 타입의 원소 또는 객체가 연속되어있는 메모리 공간을 의미

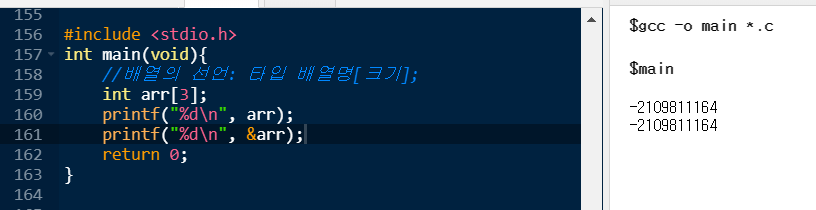
즉, 원소와 원소사이에는 갭(gap)이 없음 -> 선형구조(linear)

1. 배열의 선언 : 타입 배열명[크기] 로 선언. 크기가 n일 때, 마지막 원소의 인덱스는 n-1이다.

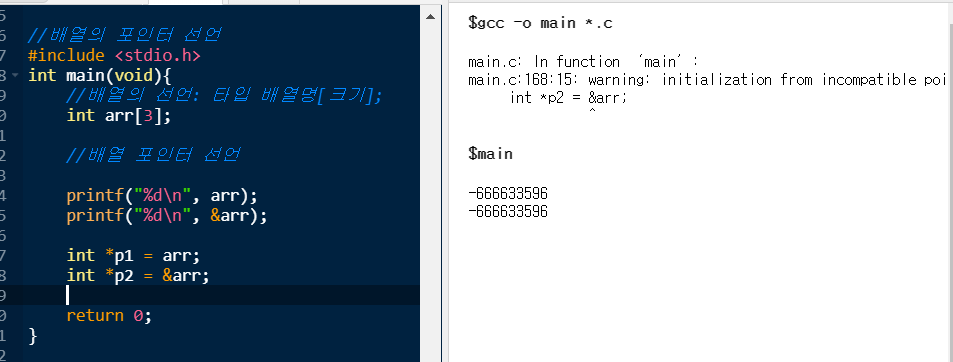
배열의 이름은 시작주소이다.

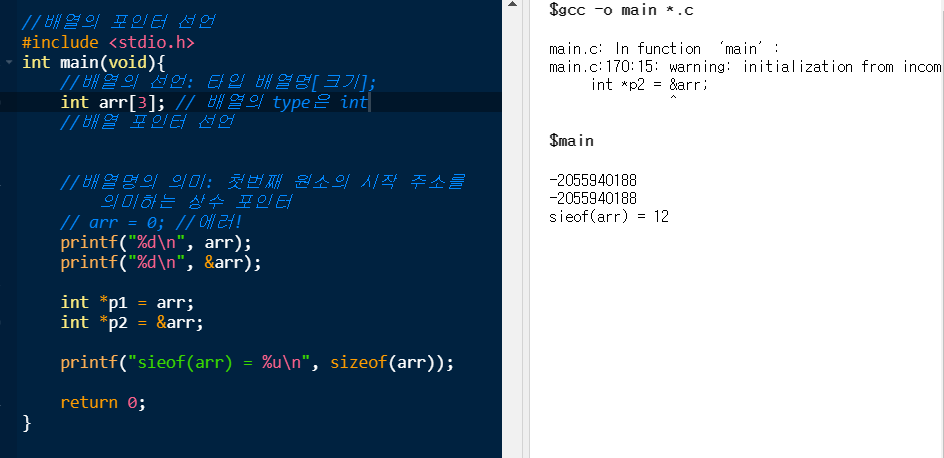
그렇다면 배열의 주소를 구하기 위해서는 &를 써야할까 말아야할까? – 상황에 따라 다르다.

<배열의 포인터선언>



위 둘은 값은 똑같지만 큰 차이가있다.





왜 12바이트가 나올까?

배열의 이름은 단독으로 사용할 경우, 첫번째 원소의 포인터로 해석되지만 예외2가지가 있는데

1. sizeof연산자를 사용할 때
2. & 연산자를 사용할 때

arr : 배열의 시작주소, 타입은 첫번째 원소

&arr : 배열의 시작주소, 타입은 배열 전체

그래서

int arr[3];

int \*p1 = arr; //ok

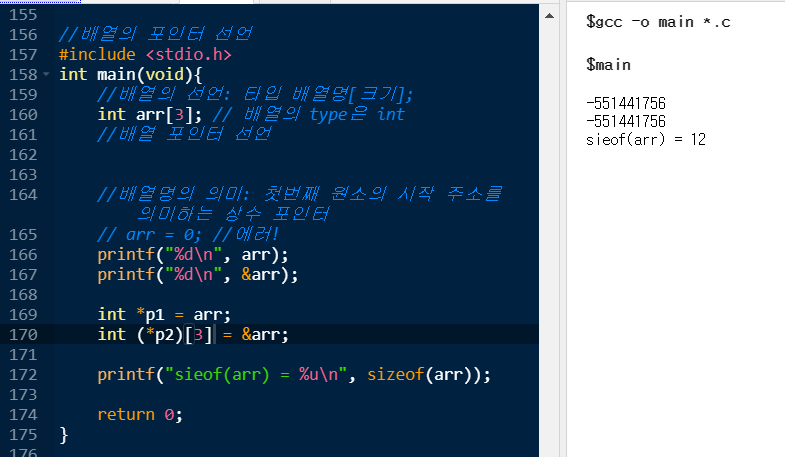
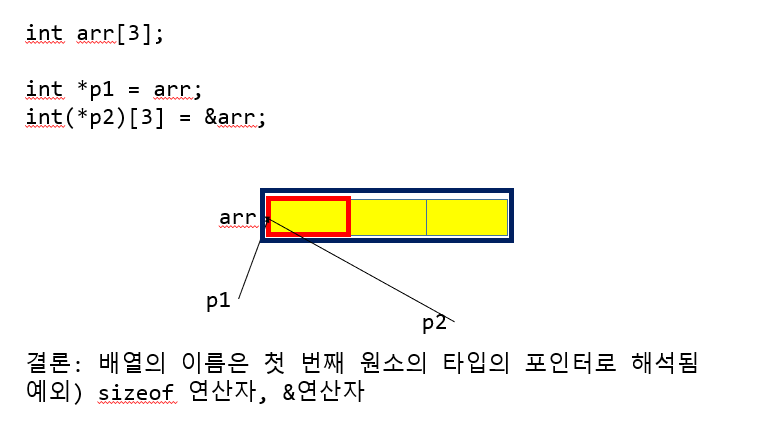
int \*p2 = &arr; //문제있다.. &쓰면 전체타입이므로 rv lv가 일치하지 않는다.

int[3] \*pArr; -> int \*pArr[3]; -> int (\*pArr)[3];

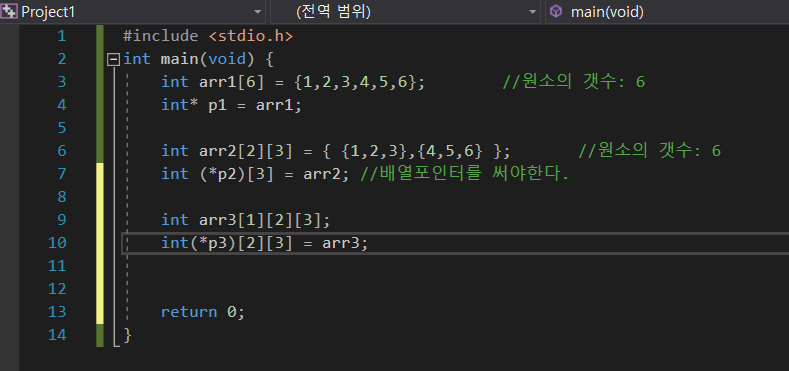
int \*pArr[3] 에서 마지막 [3]을 보면 배열이다..(포인터배열) ~~= &arr 가 되려면 배열이 되면 안되고 포인터가 되어야하므로 괄호를 넣어서 포인터로 만들어준다. 즉 int(\*pArr)[3]; 배열포인터로 만들어준다.

배열포인터: (pointer to array) : 배열 전체를 가리키는 포인터

Int(\*pArr)[3] = &arr; // ok!!



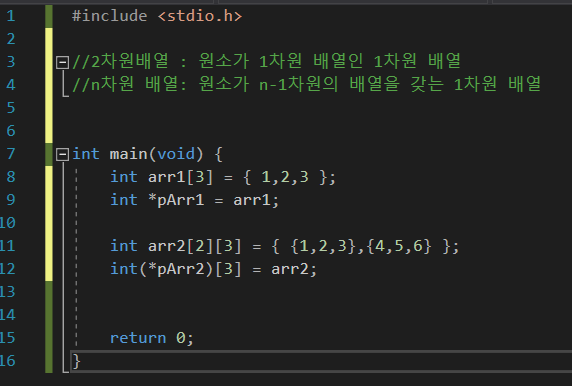
🡺결론!! 배열의 이름은 첫번째 원소의 타입의 포인터로 해석됨 예외) sizeof 연산자, &연산자

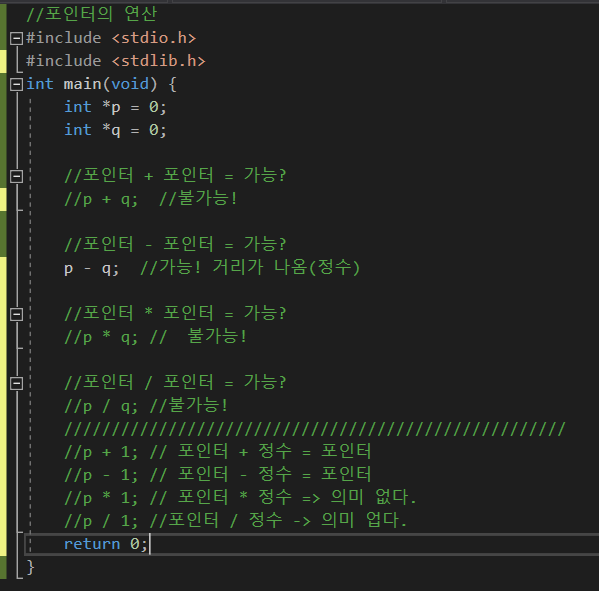


다중포인터가 아니라 배열포인터를 써야한다.(다차원배열시에)

Day0519

리뷰





<포인터의 덧셈> - 무의미!

포인터 + 포인터를 하게되면 쓸데없는 즉 무의미한 값이 나온다.

<포인터의 뺄셈> - 의미!

포인터- 포인터를 하게되면 그 사이의 거리가 나오기 때문에 의미가 있다.

그 변위의 거리는 어디있는가의(위치의) 의미가 아니라 양의 개념이다.(어느정도의 거리가 되는지)

두 위치사이의 ‘거리’인 ‘정수’가 나온다. (굉장히 많이 사용함!)

<포인터의 곱셈> - 무의미!

곱셈은 결국 덧셈의 연속이므로 의미가 없다..

<포인터의 나눗셈> - 무의미!

의미가 없다.

<포인터 + 정수>

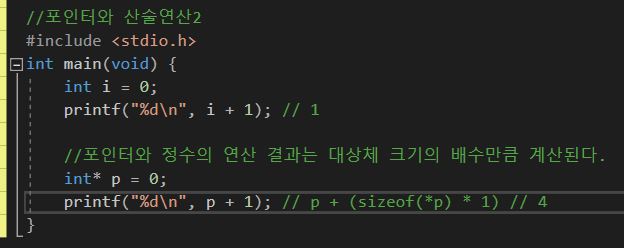
가능하다! 그냥 위치만 더해지는 포인터가 된다.

<포인터 – 정수>

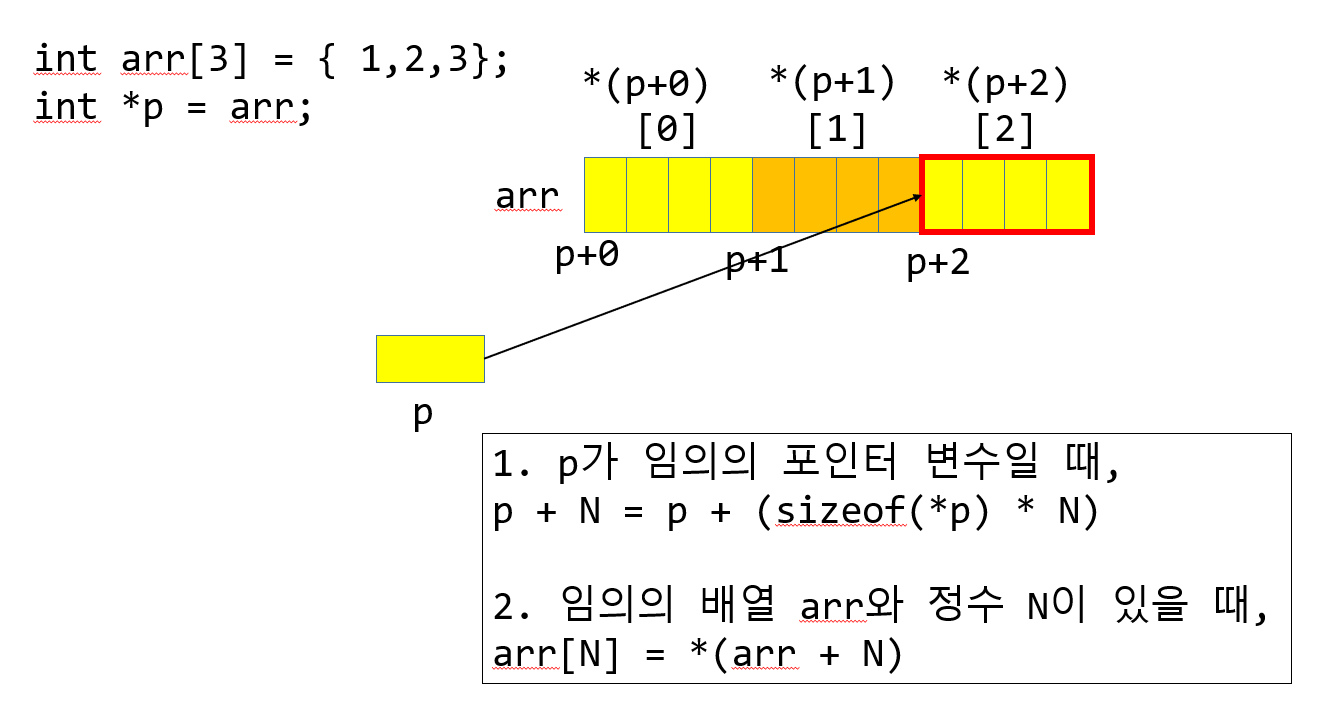
가능하다! 그냥 위치만 빼지는 포인터가 된다.

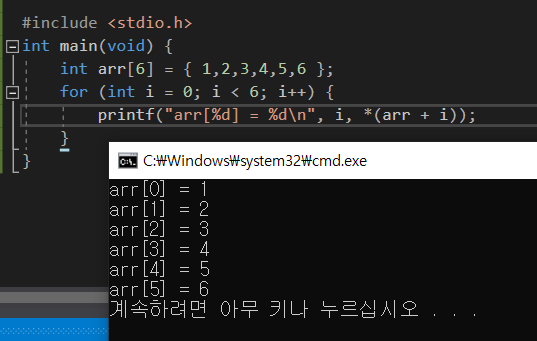
🡺결론: 포인터는 제한적인 산술 연산이 가능하다.

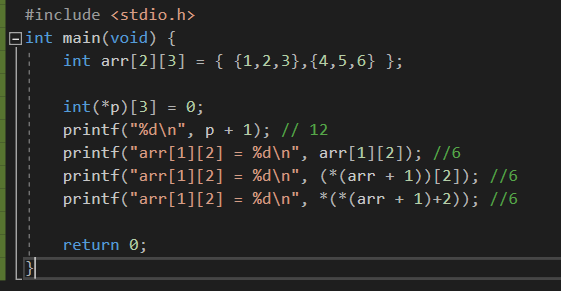
1. 포인터 – 포인터
2. 포인터 + 정수
3. 포인터 – 정수 만 가능하다

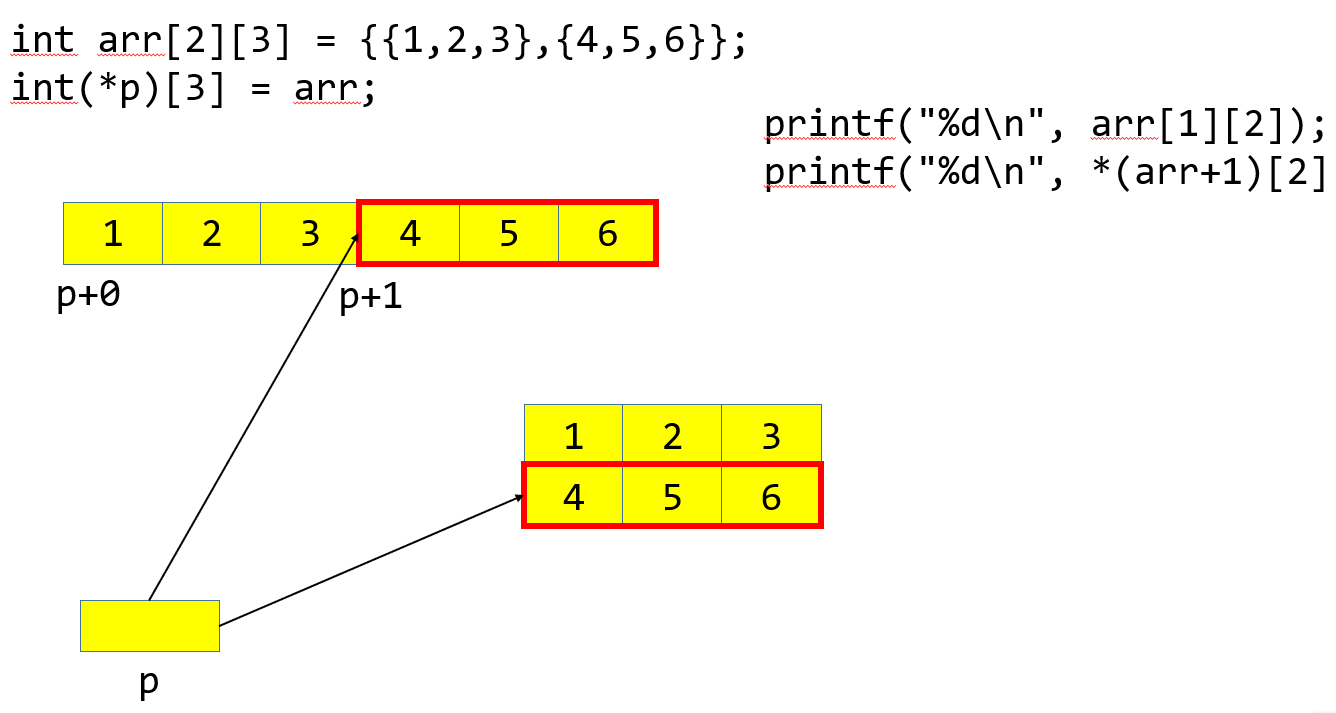


배열 때문에 4가 나온다..

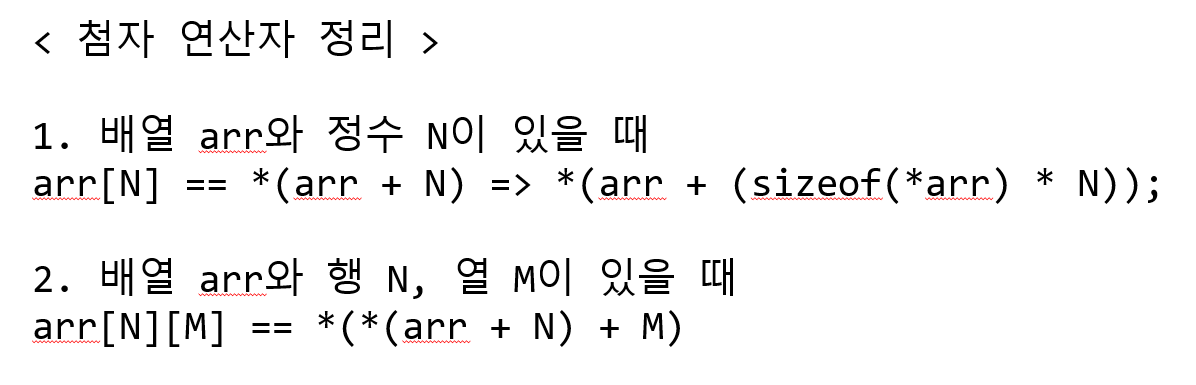
\*(p + 0) == p[0]



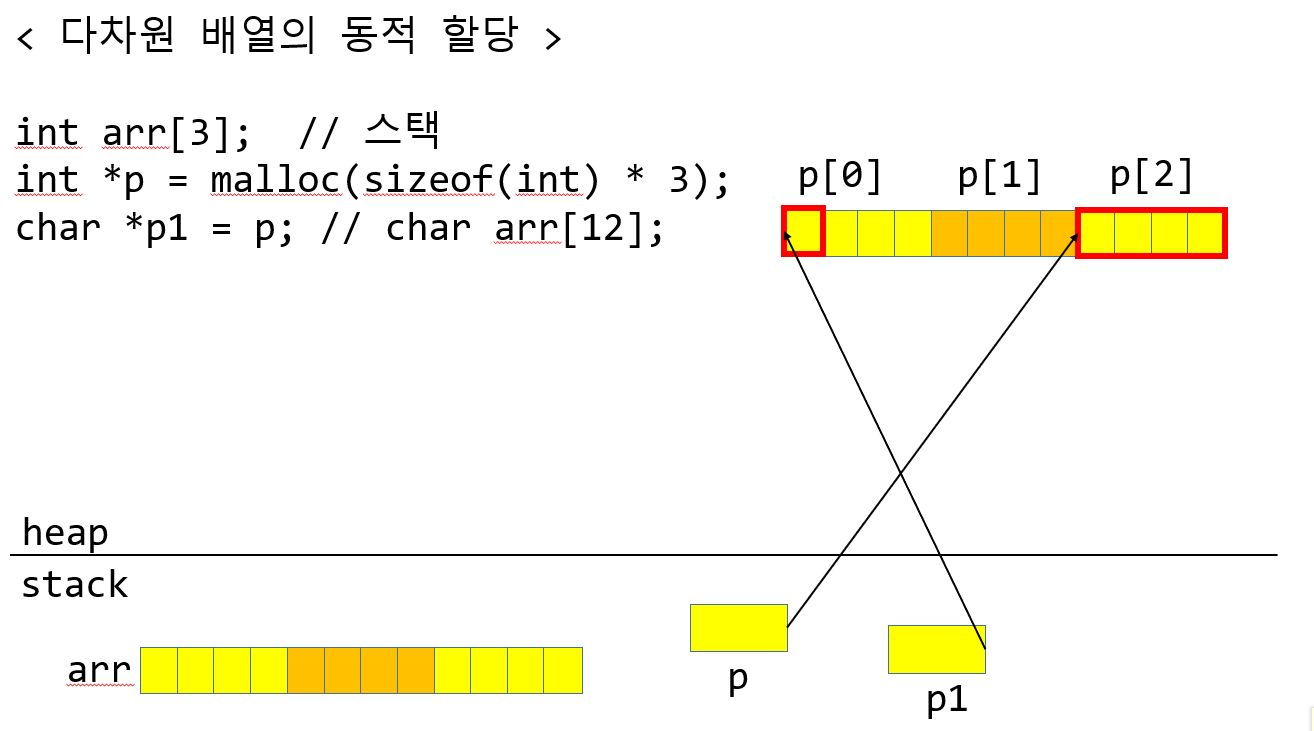




Printf 2번째 줄 처럼 짜는게 좋다. 그러나 나머지도 이해할순 있어야한다.

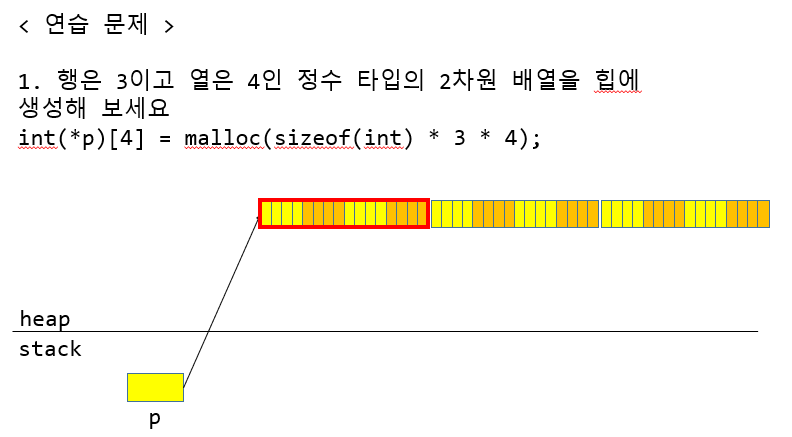


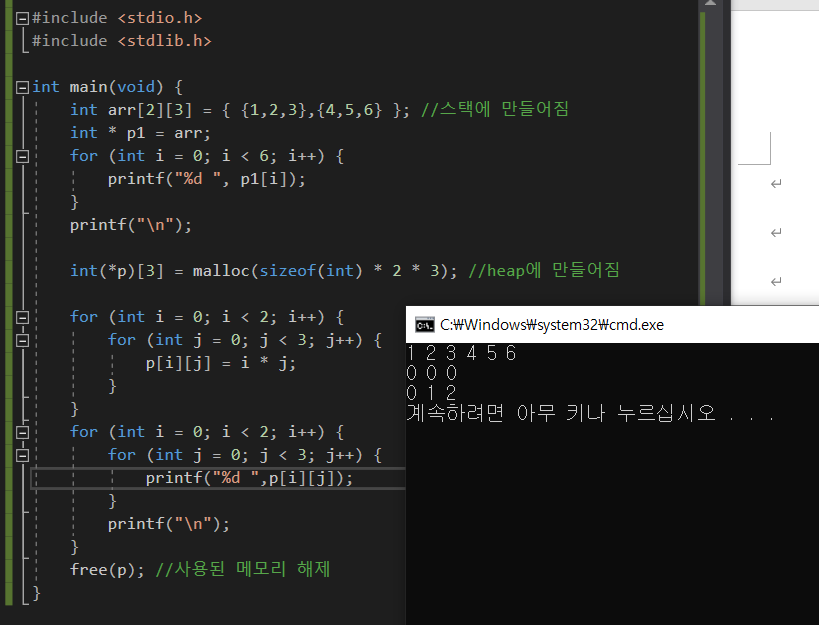
<다차원 배열의 동적할당>

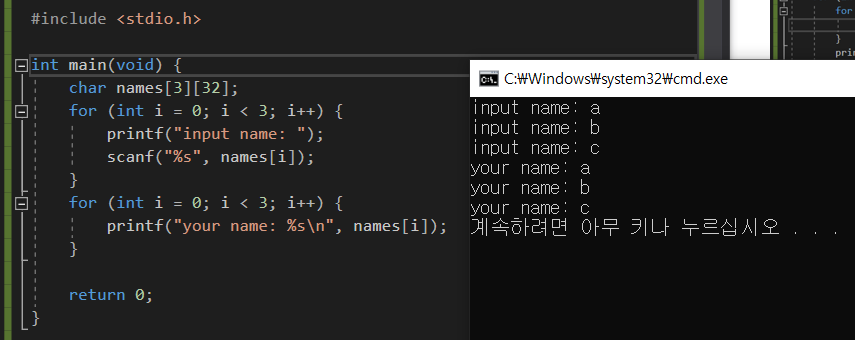


연습문제

-행은 3이고 열은 4인 정수타입의 2차원배열을 힙에 생성해보세요



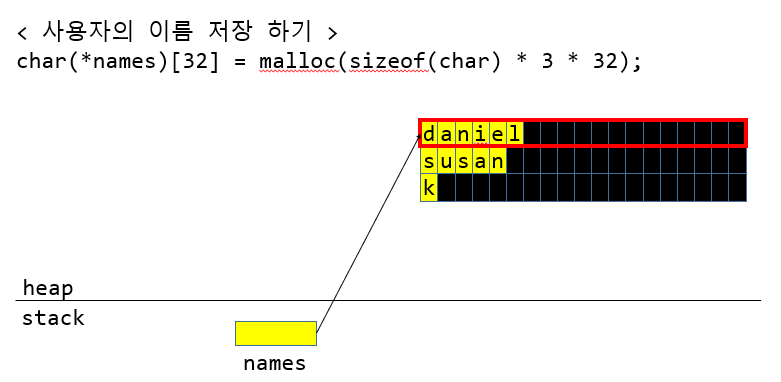


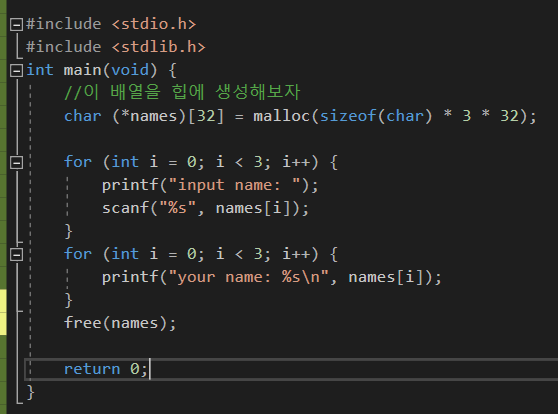


위 코드의 단점

위 코드는 대용량의 자료구조를 스택에 생성한다는 단점이 있다.

이제 이 데이터를 힙에 생성해보자



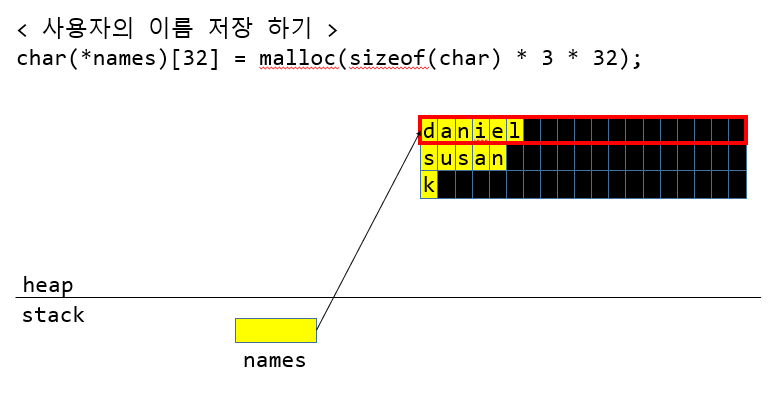


이렇게 하더라도 단점이 존재한다.

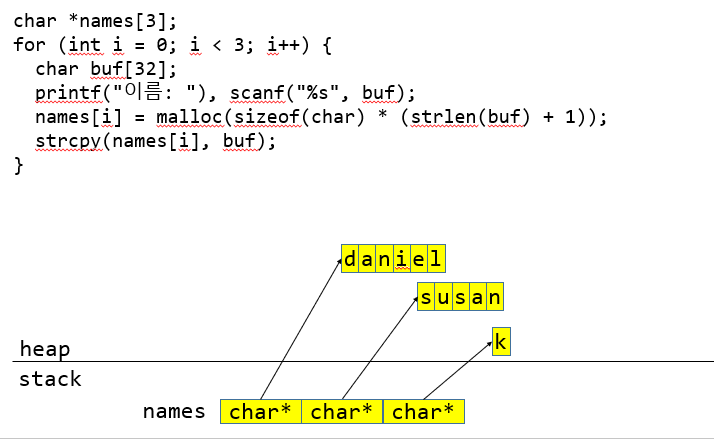
* 열이 32라서 만약 이름이 작다면 죽게되는 메모리들이 많다.
* 즉 힙 메모리에 대하여 낭비가 발생한다. 이를 해결하기 위해 문자열 포인터를 사용한다.

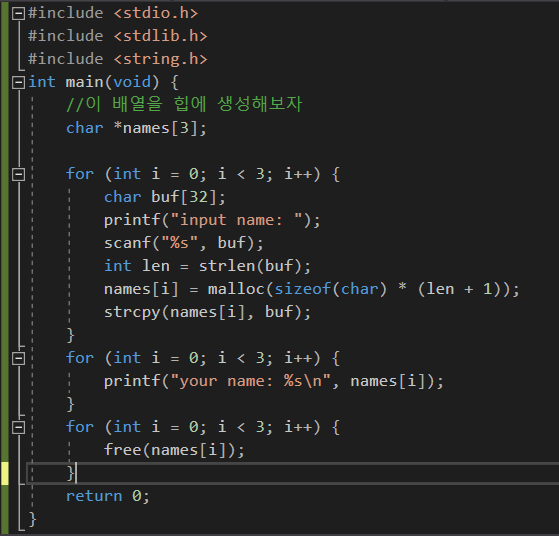
그래서 각각의 이름들을 개별적으로 할당하자!

위코드



아래코드



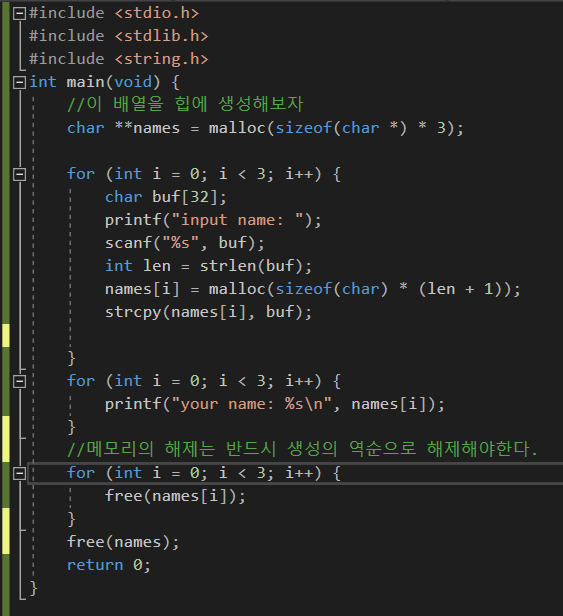
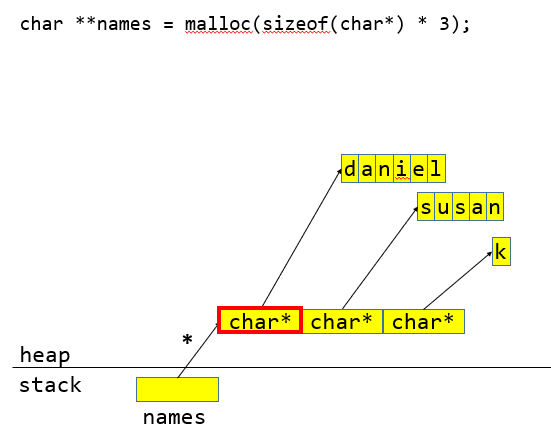


그러나 또 단점이 있다..

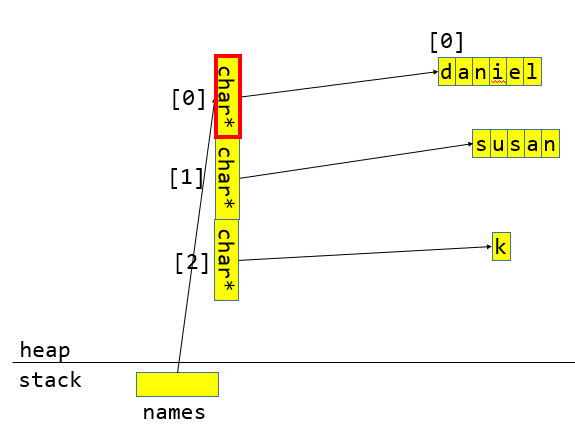
사람이 많아지면 또 스택에 대용량의 배열이 생기게 된다.

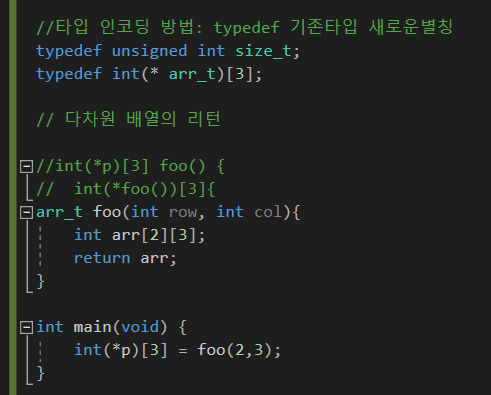
그래서 스택에 있는 배열(포인터 배열)을 힙으로 올려버리면된다.

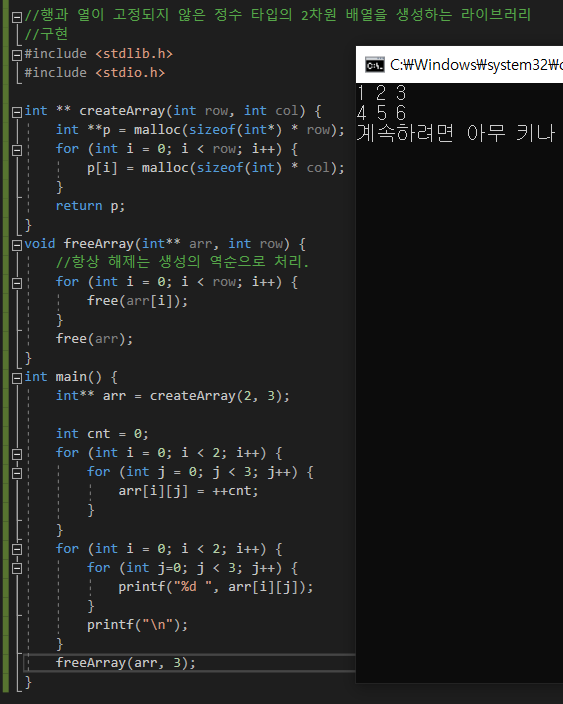
->모든 자료구조는 모두다 힙에 만들어야한다!



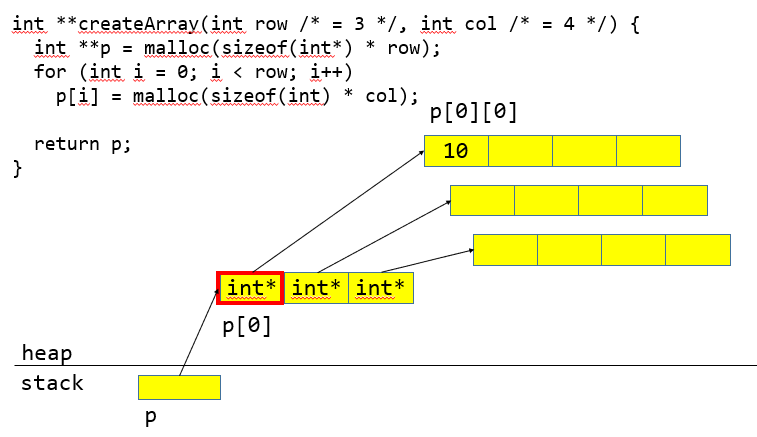
행과 열이 고정되어있지 않으며, 원래 배열은 갭이 없어야하는데 이건 갭이 존재한다. 그래서 names는 첨자연산자를 2번써서 2차원배열로 착각할수 있지만 그렇지 않다.(톱니형배열)



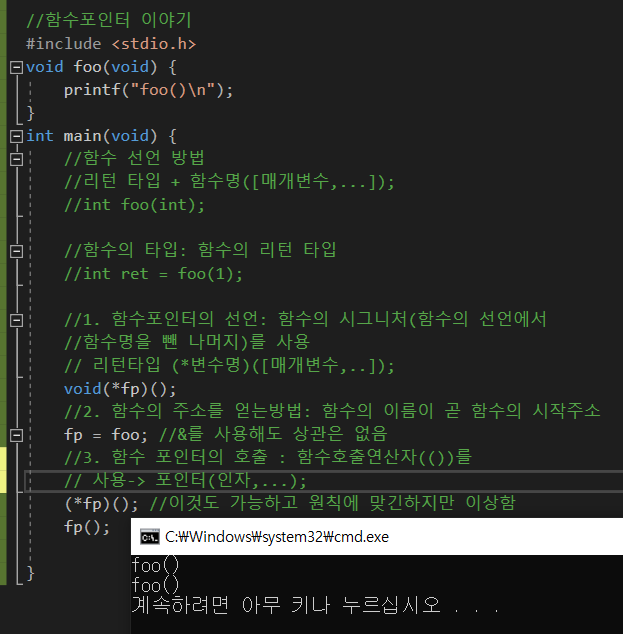


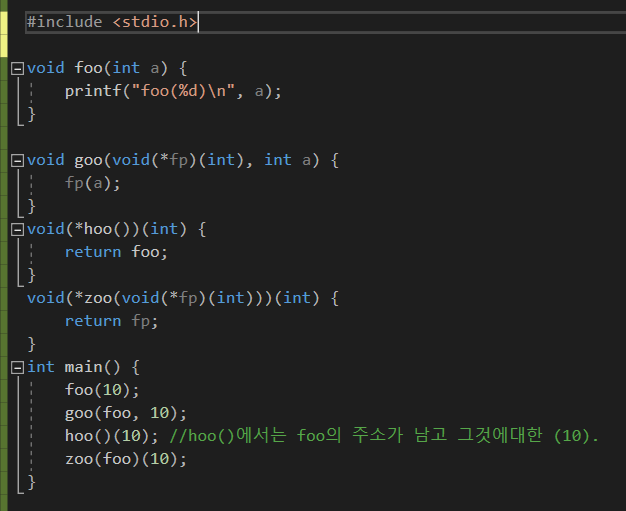


행 열 고정되지않은 정수타입의 2차원 배열을 생성하는 라이브러리 구현

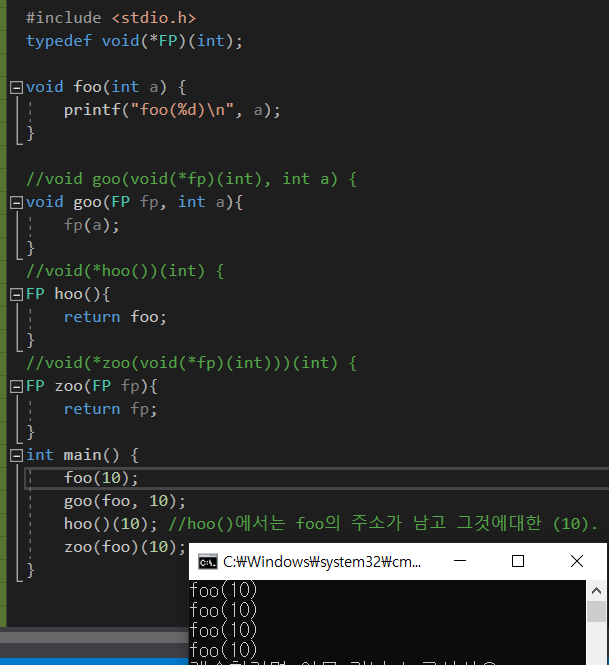


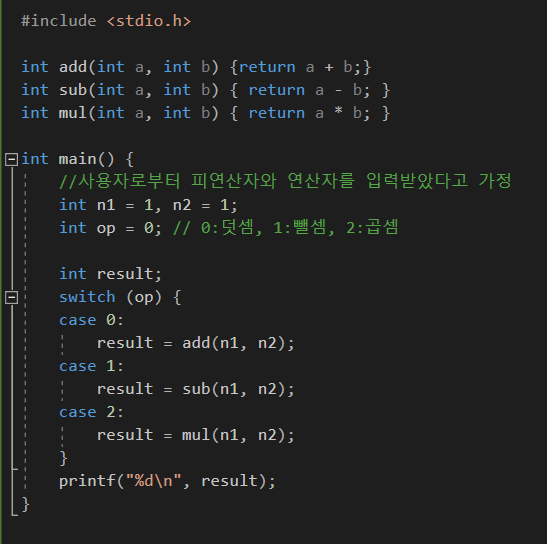
<함수포인터>



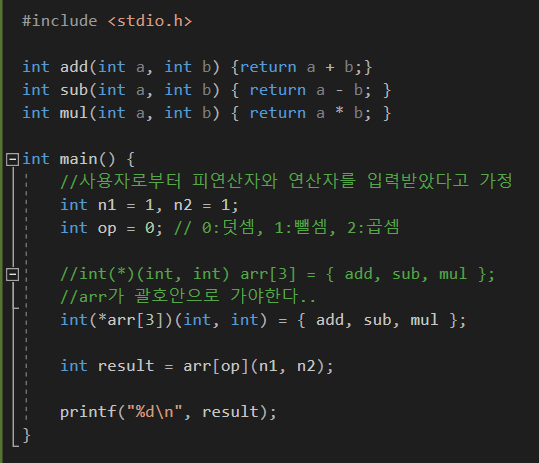


위방식을 알아야하긴 하지만 복잡하므로 타입 인코딩을 사용하여 아래처럼 쓴다.





위 코드는 기능에 비해 코드가 장황하다는 단점이 있다. 그래서 아래코드로



다형성 이라고한다.