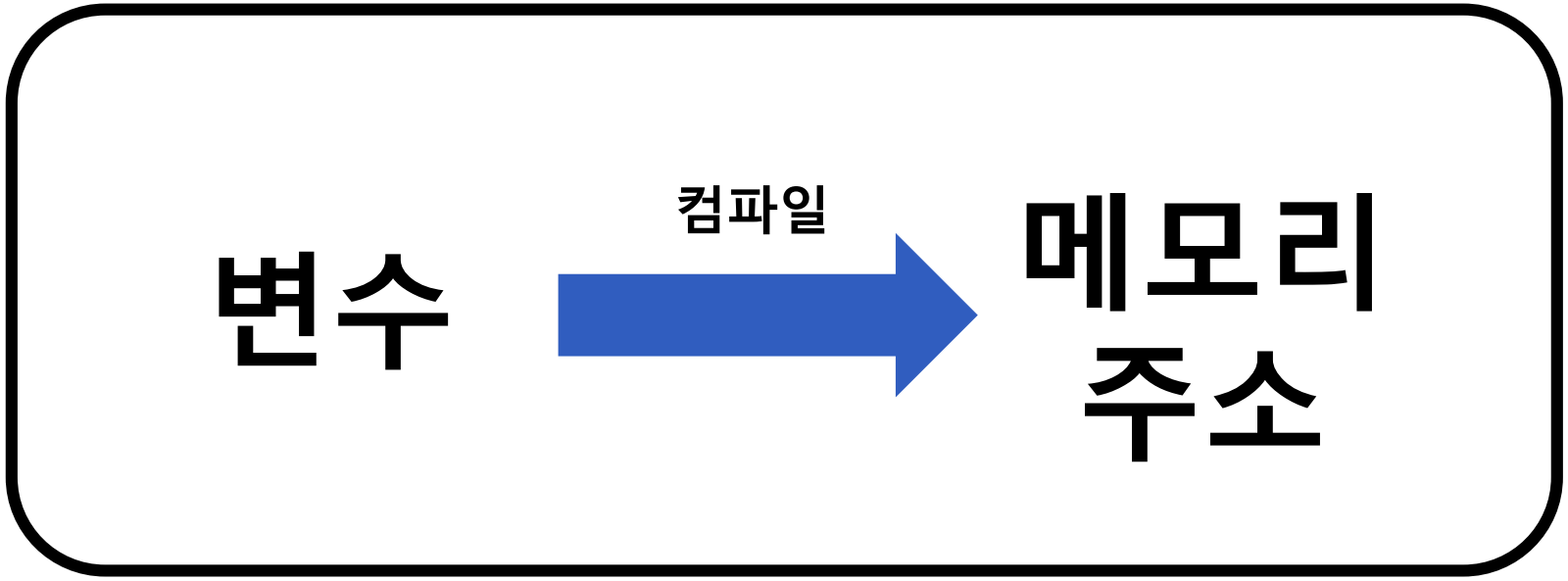
C언어 튜터링

2학기 1주차

포인터

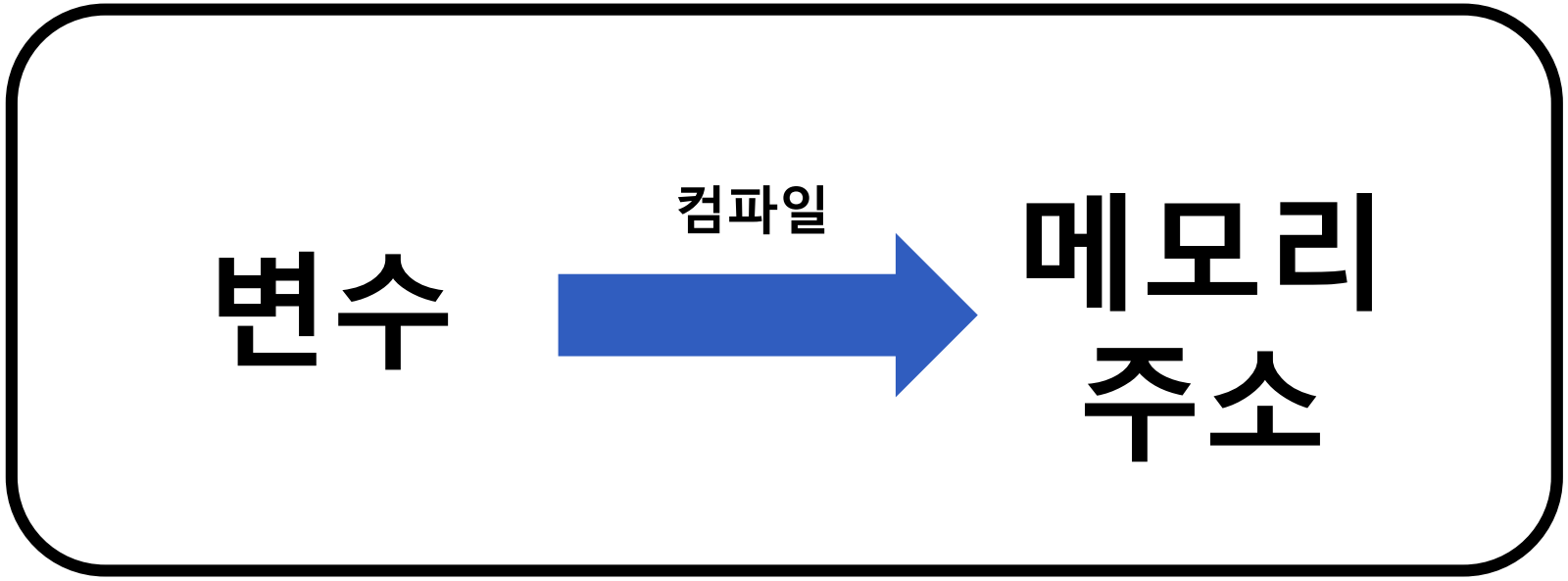
목차

운영체제의 메모리 관리 방식



- 지금까지 프로그래밍을 하면서 변수를 사용하였고,
  변수를 사용하여 메모리에 데이터를 저장하거나 읽었다.
- C언어에서 원시코드를 컴파일 하면 그동안 사용했던 변수들이 모두 메모리 주소로 바뀌어서 적용된다.

운영체제의 메모리 관리 방식



- 결국 컴파일된 실행파일에서는 변수의 이름보다는
  변수가 위치한 메모리의 주소가 더 중요하다는 이야기이다.

- 그리고 지금 한 말을 다시 생각하면 변수 이름을 사용하지 않고
  변수의 주소만 알고 있다면 변수 값을 읽거나 바꿀 수 있다.

직접 주소 방식

```c
#include <stdio.h>

void Test()
{
  short soft = 0x0000;
  soft = tips; // 오류
}

int main()
{
  short tips = 0x0005;
  Test();

  return 0;
}
```

```c
#include <stdio.h>

void Test(short data)
{
  short soft = 0x0000;
  soft = tips;
}

int main()
{
  short tips = 0x0005;
  Test(tips);

  return 0;
}
```

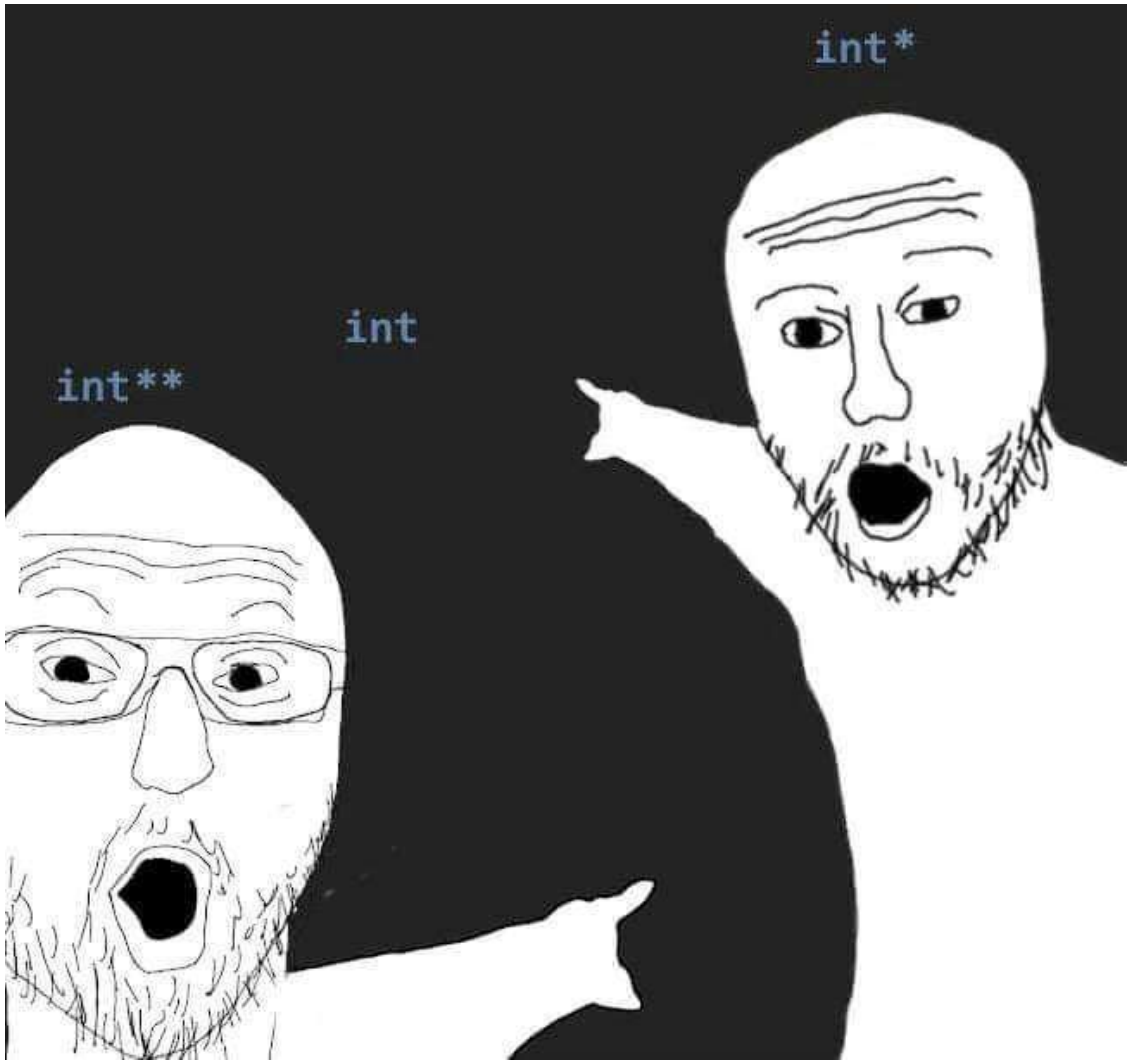
<직접 주소 방식>

- 메모리를 사용할 때 프로그래머가 사용할 **메모리 주소를 직접 적는 방식**이다

<직접 주소 방식의 한계>

- 함수 안에 선언한 변수는 해당 함수에서만 사용할 수 있다.
- 다른 함수에 선언한 변수가 메모리에 존재해도 문법적으로 접근할 수 없다.

간접 주소 방식



<간접 주소 방식>

- 간접 주소 방식의 대표적인 방식은 포인터이다.

<포인터의 용도>

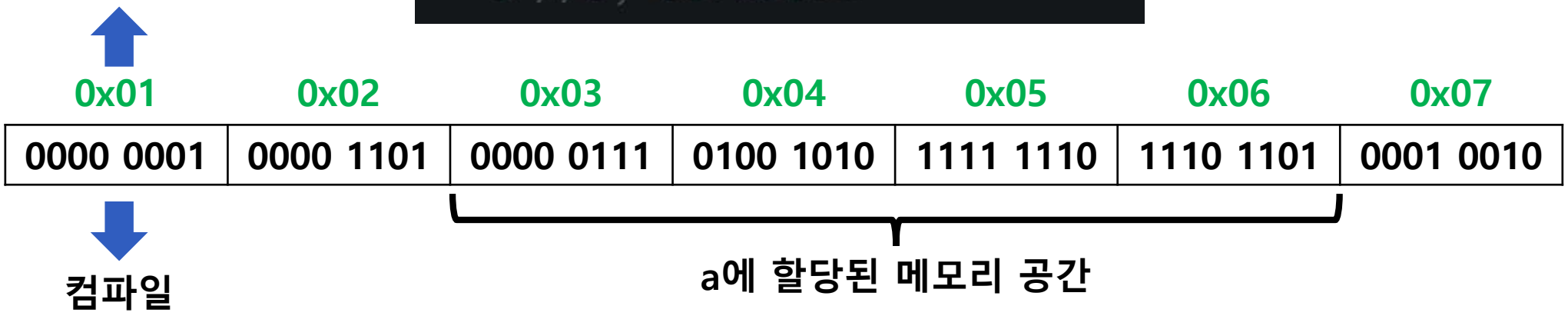
1. 포인터를 선언 할 때
2. 해당 주소에 값 접근

→ 포인터(변수)가 가리키는 변수에 접근하는 것

포인터 개요 - 주소연산자(&)

- **메모리는 일렬로 연속되어 있는** 크기가 1byte인 **배열이다.**
- 변수는 선언될 때, 메모리에 그 변수를 위한 공간이 할당된다.
- **주소연산자(&)** : 변수에 할당된 메모리 공간의 시작 주소를 구한다.

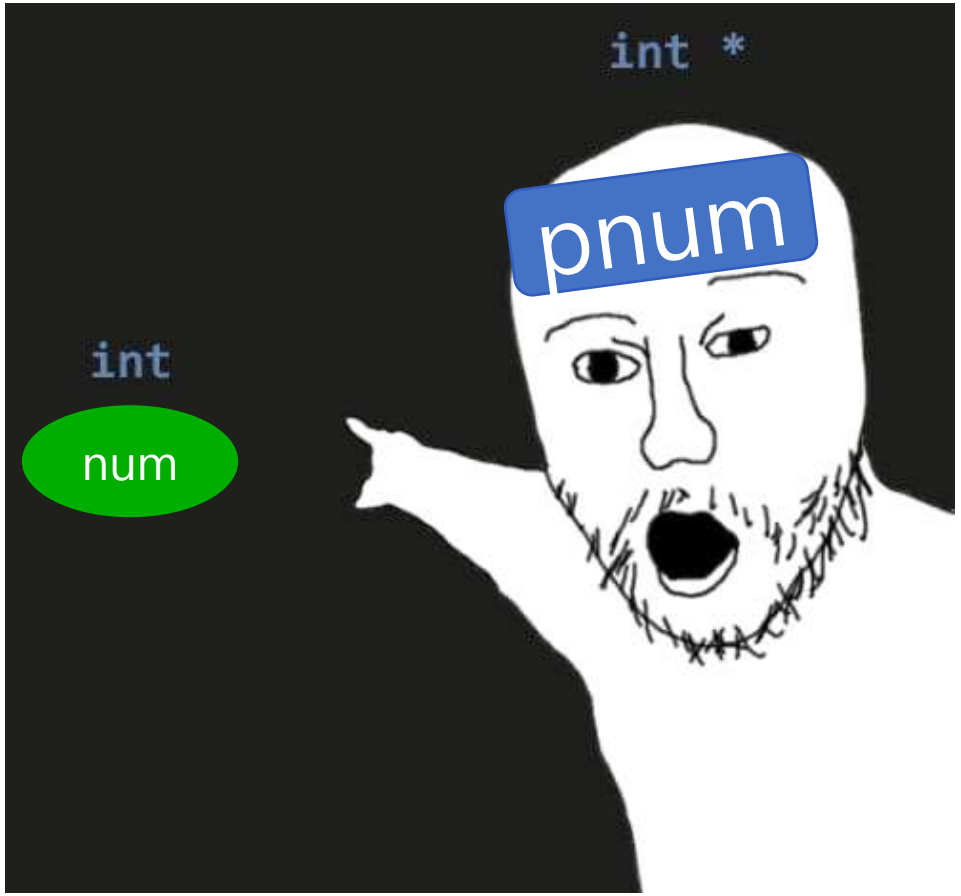
```
1 int a = 0;
2 printf("%d, %p", a, &a);
3
4 //결과
5 //0, 007ADC9C
```



포인터 선언과 사용

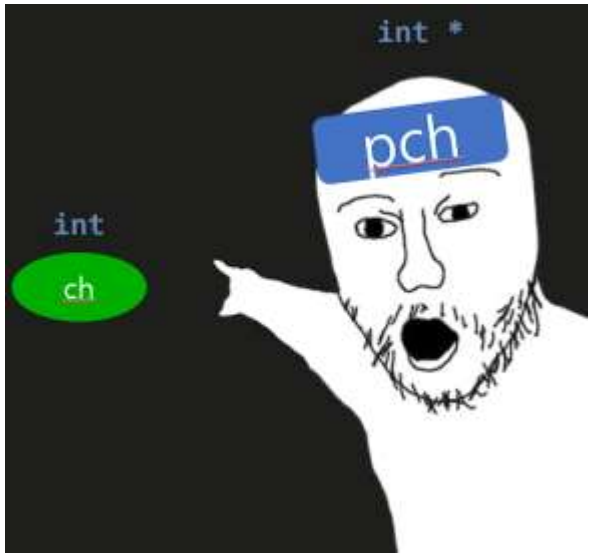
- **포인터 (변수) 선언**
  구문 : 변수 명 앞에 ***(참조연산자)**만 덧붙이면 된다.

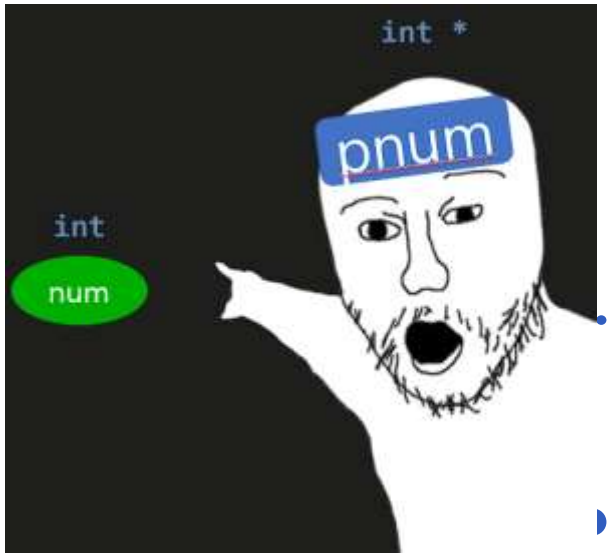
예)
```
char   *pch;
int    *pnum;
```

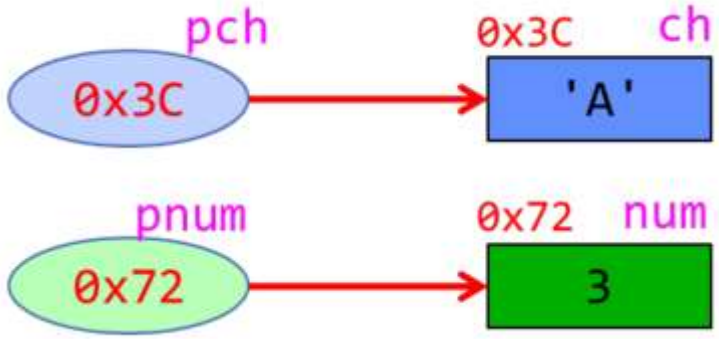


포인터 선언과 사용 – 예시1

```c
#include <stdio.h>

int main()
{
  char ch = 'A', *pch;
  int num = 3, *pnum;

  pch = &ch; // pch에 변수 ch의 주소 대입
  pnum = &num; // pnum에 변수 num의 주소 대입

  printf("%p %c\n", pch, ch);
  printf("%p %d\n", pnum, ch);
}

/*
결과
0x7ffcdfb80ea3 A
0x7ffcdfb80ea4 65
*/
```







포인터 선언과 사용 – 예시2

```c
#include <stdio.h>

int main()
{
  char ch = 'A', *pch = & ch;
  int num = 3, *pnum = &num;

  *pch = 'B';
  *pnum += 5;

  printf("%c %d\n", ch, num);
}

/*
결과
B 8
*/
```

포인터 주의사항 - 초기화

- **포인터 주의사항 1 (초기화)**
  - 선언 후 연결 없이 바로 사용하면?

```
int *pnum;    // pnum에는 쓰레기 값
*pnum = 9;  // 에러
```

```
int *pnum, num;
pnum = &num; // 반드시 어떤 변수에 연결 후 사용
*pnum = 9;
```

- **널(NULL) 포인터**
  - **주소 값 0**을 나타내는 특별한 기호로 **아무것도 가리키지 않을 의미**
  - NULL의 값은 0이므로, 조건문에서 사용하면 거짓에 해당
  - 예기치 못한 오류 방지를 위해 포인터 변수를 NULL로 초기화

```
int *pnum = NULL;
```

0과 NULL의 차이 설명 영상

https://www.youtube.com/shorts/_dEqwTTaHMs

포인터 주의사항
– 주소연산자와 참조연산자

- **포인터 주의사항 2**
  - **&(주소연산자)**는 포인터를 포함한 모든 변수에 사용가능
  - ***(참조연산자)**는 포인터 변수에서만 가능

```
int num = 9, *pnum = &num;
printf("%p %p %d₩n", &pnum, pnum, *pnum);
printf("%p %d %d₩n", &num, num, *num); // 컴파일 오류
```

포인터 주의사항 – 자료형 일치

- **포인터 주의사항 3 (대입)**
  - **포인터의 자료형과 연결**된 **변수의 자료형**은 **일치**해야 한다.
  - 서로 다른 자료형의 포인터 간 대입
    - 문법적으로는 허용은 하지만 컴파일 경고가 발행
    - **프로그램 오류의 원인**이 된다.

포인터 주의사항 – 포인터의 크기

```c
#include <stdio.h>

int main()
{
    char *pch;
    int *pnum;
    double *pdnum;

    printf("%ld\n", sizeof(pch));
    printf("%ld\n", sizeof(pnum));
    printf("%ld\n", sizeof(pdnum));
}

/*
결과
8
8
8
*/
```

- 포인터의 크기는 자료형과 관계없이 **8바이트**이다.

temp를 이용한 SWAP 함수

```c
#include <stdio.h>

int main(void)
{
  int x = 1;
  int y = 2;

  int tmp = x;
  x = y;
  y = tmp;

  printf("x\t : %d\n", x);
  printf("y\t : %d\n", y);
  printf("tmp\t : %d\n", tmp);

  return 0;
}
```

매개변수 전달 방법

- Call by Value
- Call by Reference
  - Call by Address
  - Call by Reference

포인터를 이용한 SWAP 함수

```c
#include <stdio.h>

void swap(int *a, int *b)
{
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(void)
{
  int x = 1;
  int y = 2;

  swap(&x, &y);
  printf("x : \t%d\n", x);
  printf("y : \t%d\n", y);
}
```

배열의 시작이 0인 이유 (1)

~ 배열의 첨자가 0부터 시작하는 이유 ~

1. 배열을 그림으로 나타내면

```
int arr[5];
```
↳ int는 4byte 이다.

=> 이것을 메모리 상에 나타내면 밑의 그림과 같다.


```
int arr[5] = {98, 5, 21, 3, 14};
```
↳ int는 4byte 이다.

=> 이것을 메모리 상에 나타내면 밑의 그림과 같다.

2. 배열을 계산식으로 나타내면

1000 + 4 x 0 = 1000

1000 + 4 x 1 = 1004

1000 + 4 x 2 = 1008

1000 + 4 x 3 = 1012

1000 + 4 x 4 = 1016

배열에서 값(첨자 값)을 구하는 방식은

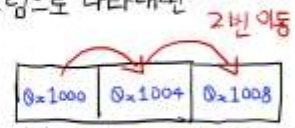
칫 시작주소로 부터 타입 사이즈 만큼 더해가며 찾는 것이다.

- 칫 번째 98 이라는 값은 자기 자신이니깐 0번 이동. 그냥 1000.
- 두 번째 5 는 1000 에서 4바이트 만큼 1번 이동한다. 1004 이다.

↳ 그림으로 나타내면

- 세 번째 21는 1000 에서 4바이트 만큼 2번 이동한다. 1008 이다.

↳ 그림으로 나타내면



(3~5 번 째는 생략)

이해를 해보니 당연하지 아니한가?

이러한 이유로 배열은 1이 아닌 0부터 시작한다.

배열 표기법과 포인터 표기법의 관계

배열	포인터

char data[5];
data[1] = 5;
*(data + 1) = 5;
	char data;
	char *p = &data;
	*p = 3;
	p[0] = 3;

배열의 이름은 그 배열의 시작 주소이다

```c
#include <stdio.h>

int main()
{
    int han1[3] = {10, 15};
    int han2[3];

    for(int i = 0; i < 3; i++)
        printf("han1[%d] = %d 주소 = %p\n", i, han1[i], &han1[i]);
    printf("han1=%p\n", han1); // &han1[0]

    printf("---------------\n");

    for(int i = 0; i < 3; i++)
        printf("han2[%d] = %d 주소 = %p\n", i, han2[i], &han2[i]);
    printf("han2=%p\n", han2); // &han2[0]
}
/*
결과
han1[0] = 10 주소 = 0x7fffc8497a30
han1[1] = 15 주소 = 0x7fffc8497a34
han1[2] = 0 주소 = 0x7fffc8497a38
han1=0x7fffc8497a30
---------------
han2[0] = 0 주소 = 0x7fffc8497a3c
han2[1] = 0 주소 = 0x7fffc8497a40
han2[2] = 0 주소 = 0x7fffc8497a44
han2=0x7fffc8497a3c
*/
```

배열의 이름은 그 배열의 시작 주소이다

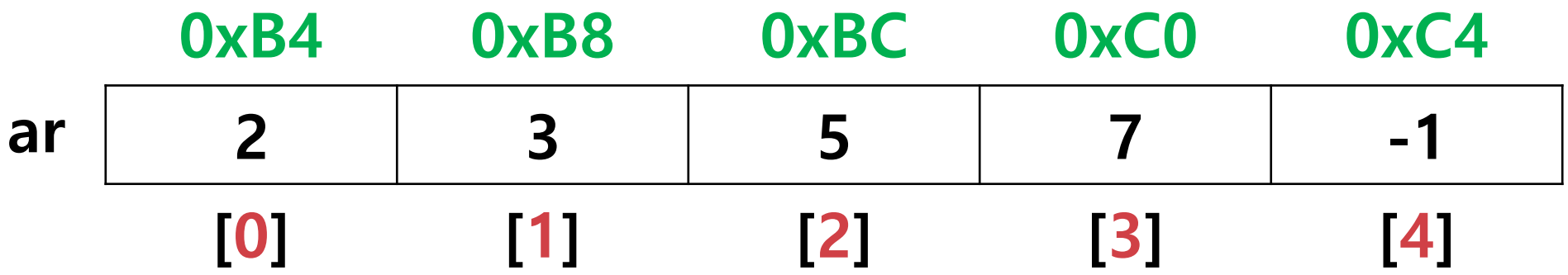
```
char *p = &data[0];
char *p = &*(data + 0);
char *p = &*data;
char *p = data;
```

- **&(*data)의 의미**는 **data가 가리키는 대상(*data)의 주소를 얻겠다(&)는 뜻**이다.
- **data가 가리키는 대상의 주소라는 의미는** 결국 **data 변수가 저장된 메모리의 주소와 같다.**
- 그래서 **&(*data)는 data**라고도 적을 수 있다.

배열의 시작이 0인 이유 (2)

```
int ar[5] = {2, 3, 5, 7, -1};
printf("%p %d %d\n", ar, ar[0], *ar);
```

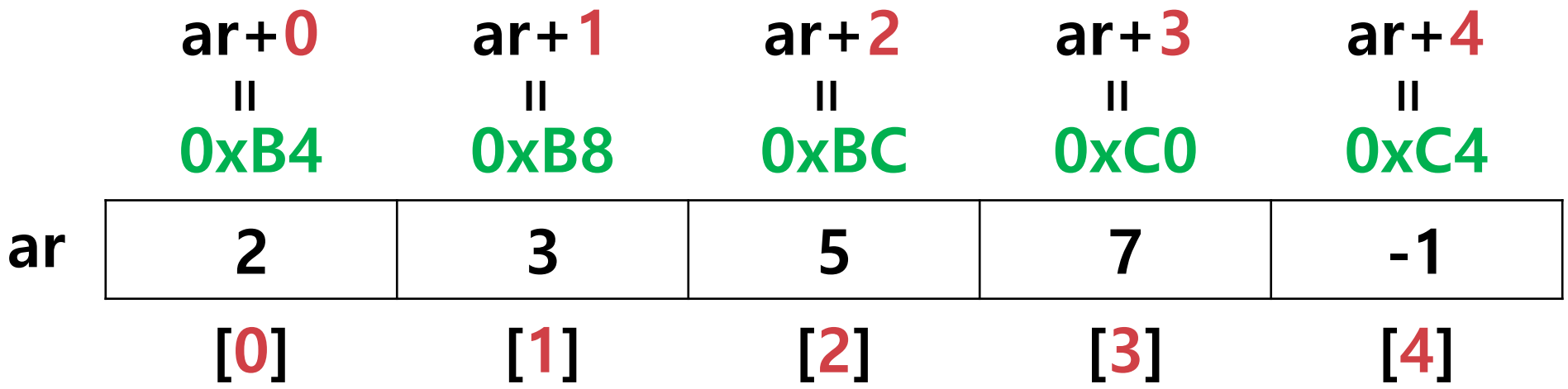
```
// 실행 결과
0x1E40B4 2 2
```



배열의 시작이 0인 이유 (2)

```c
int ar[5] = {2, 3, 5, 7, -1};
printf("%p %d %d\n", ar+1, ar[1], *(ar+1));
```

```
// 실행 결과
0x01EE4B8 3 3
```



배열과 포인터의 관계 정리

- **배열과 포인터의 관계 정리**
  - 배열과 포인터는 동일한 형태로 사용 가능하다.

```
int ar[5], *p=ar;
```

	ar+0	ar+1	ar+2	ar+3	ar+4
	=	=	=	=	=
	p+0	p+1	p+2	p+3	p+4
	=	=	=	=	=
	0xB4	0xB8	0xBC	0xC0	0xC4
ar	2	3	5	7	-1
	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]
	=	=	=	=	=
	p[0]	p[1]	p[2]	p[3]	p[4]
	=	=	=	=	=
	*(ar+0)	*(ar+1)	*(ar+2)	*(ar+3)	*(ar+4)
	=	=	=	=	=
	*(p+0)	*(p+1)	*(p+2)	*(p+3)	*(p+4)

END