C Programming – Day 3

2017.09.04.

JunGu Kang
Dept. of Cyber Security

아주대학교 Whois

포인터

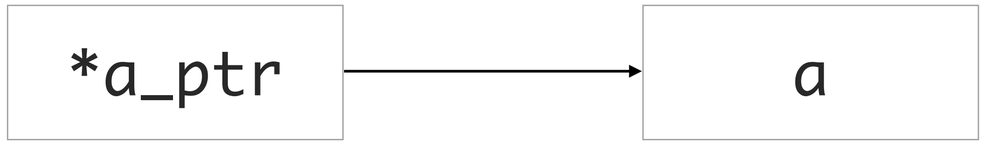
Pointer

변수는 메모리에 할당되고 메모리에는 **주소값**이 있다.

Pointer

포인터 변수는 **가리키고자 하는 메모리의 주소값을 저장**한다.

Pointer



포인터 변수의 선언

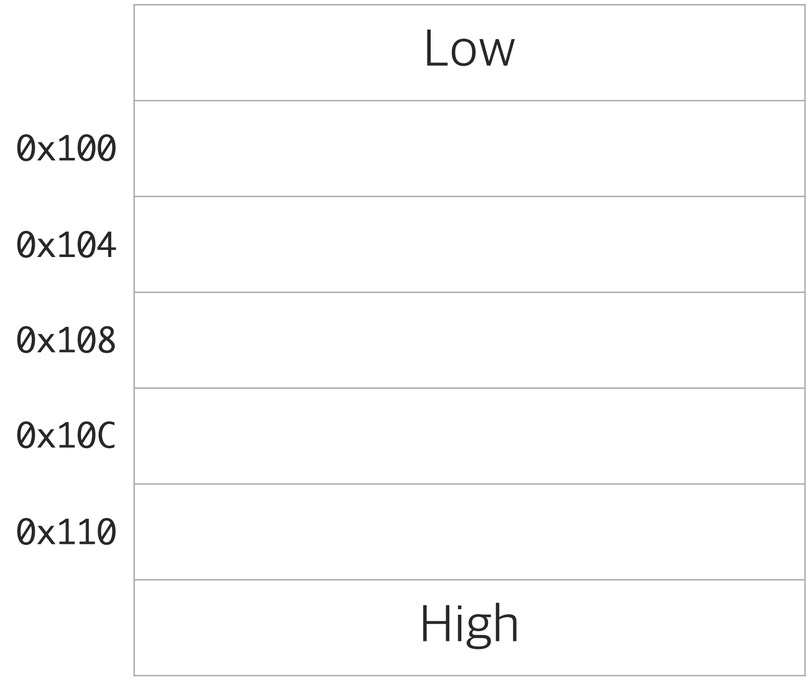
```
type * name;
```

포인터 연산자

```
type * name;
```

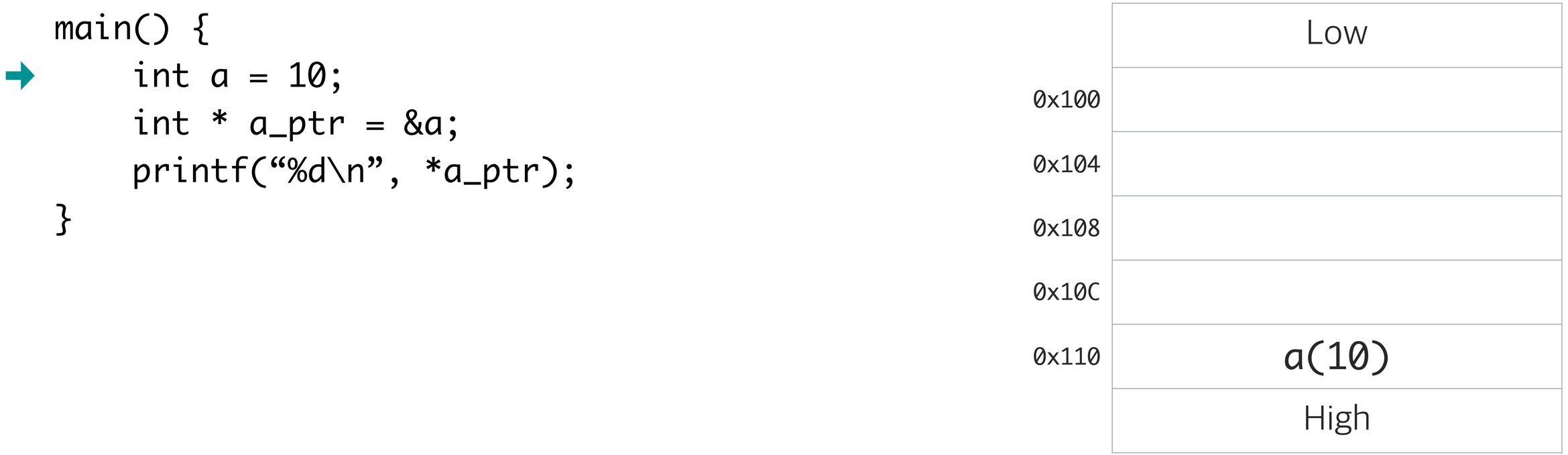
Pointer

```
main() {
    int a = 10;
    int * a_ptr = &a;
    printf("%d\n", *a_ptr);
}
```



Pointer

```
main() {
    int a = 10;
    int * a_ptr = &a;
    printf("%d\n", *a_ptr);
}
```



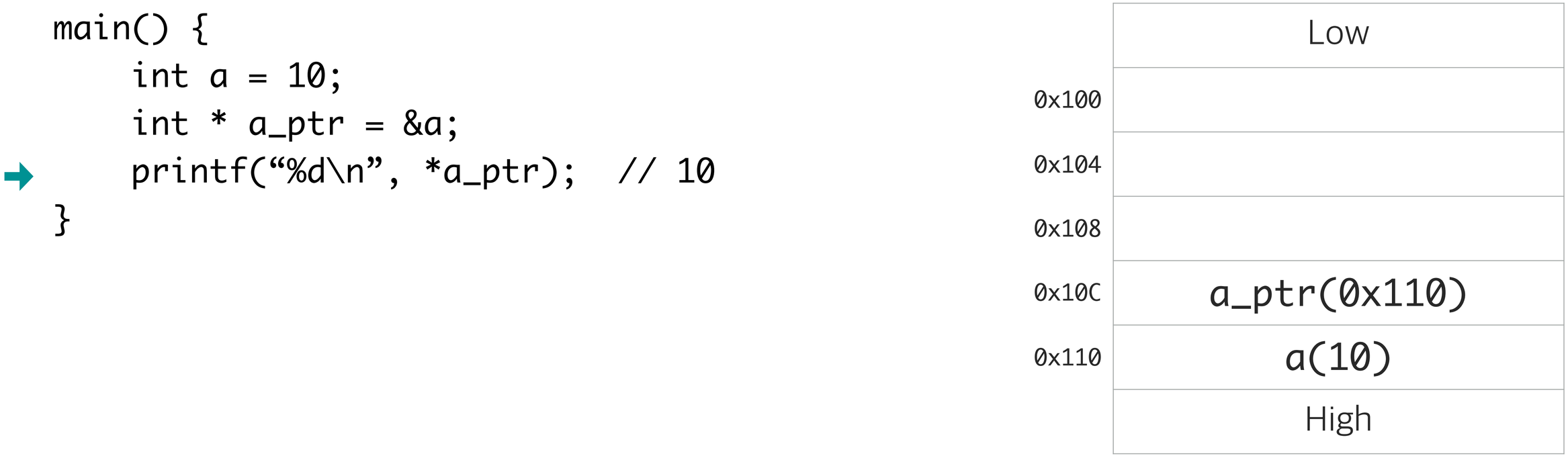
Pointer

```
main() {
    int a = 10;
    int * a_ptr = &a;
    printf("%d\n", *a_ptr);
}
```

	Low
0x100	
0x104	
0x108	
0x10C	a_ptr(0x110)
0x110	a(10)
	High

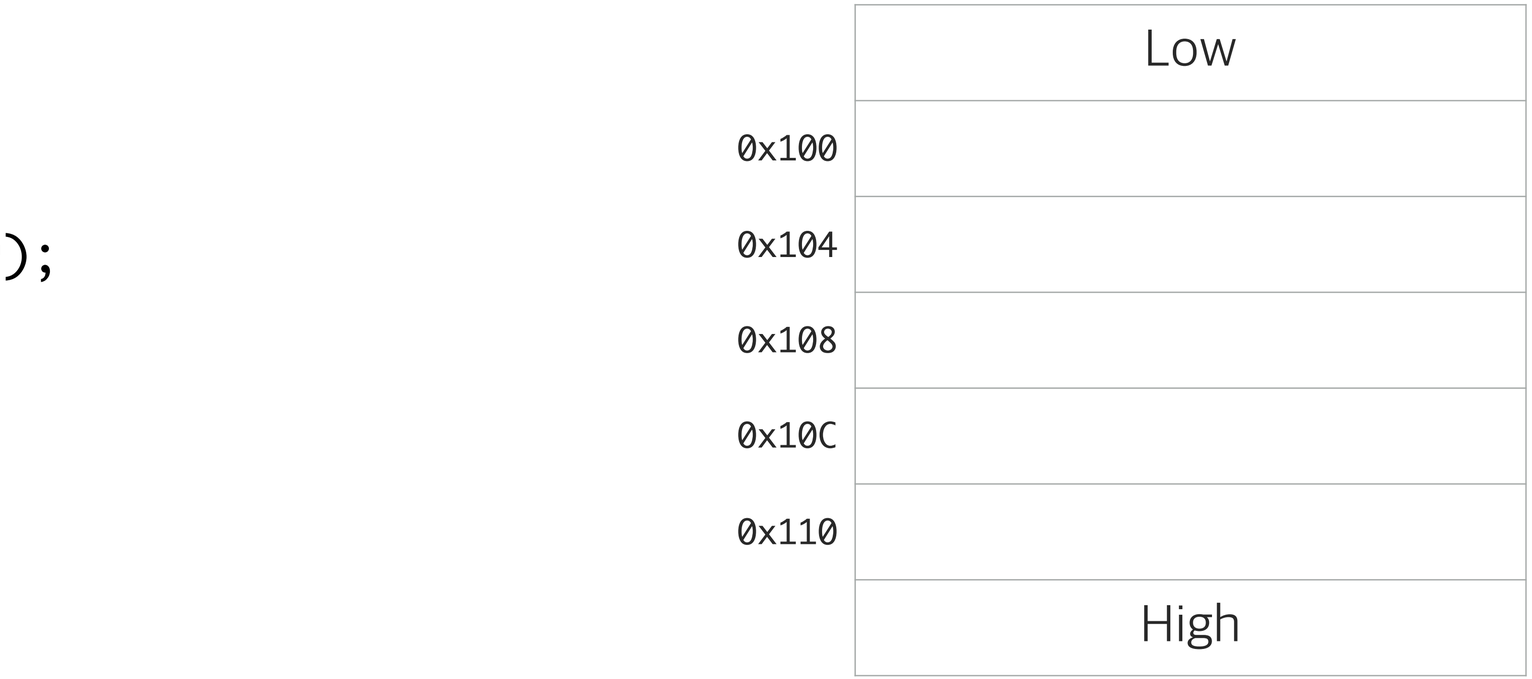
Pointer

```
main() {
    int a = 10;
    int * a_ptr = &a;
    printf("%d\n", *a_ptr);  // 10
}
```



Pointer

```
main() {
    int a = 10;
    int * a_ptr = &a;
    printf("%d\n", *a_ptr);
}
```



포인터 변수의 초기화

포인터 변수는 **반드시 초기화를 해야 한다.**
그러나 절대 **상수로 초기화해서는 안된다.**

포인터 변수의 초기화

```
int * ptr;
// 초기화하지 않았으므로 쓰레기값이 들어있을 것이고,
// 그 메모리 주소를 가리키고 있을 것이다.
```

포인터 변수의 초기화

```
int * ptr = 0x100;  // 0x100에 뭐가 있는지는 아무도 모른다.
```

포인터 변수의 초기화

포인터 변수의 대상이 될 변수가 아직 없다면,
NULL로 초기화하여 아무것도 가리키지 않도록 한다.

포인터 변수의 초기화

```
char * char_ptr = NULL;
int * int_ptr = NULL;
long * long_ptr = NULL;
float * float_ptr = NULL;
double * double_ptr = NULL;
```

포인터의 자료형

포인터 변수는 메모리의 주소를 저장하기 때문에
크기가 type과 관계없이 같다.

포인터의 자료형

포인터의 type은 포인터의 크기가 아니라

포인터가 가리키는 변수를 어떻게 읽고 쓸지 결정한다.

포인터 연산자

```
main() {
     int a = 10;
     int * ptr = &a;
     printf(“addr of a : %p”, &a);  // a의 주소값
     printf(“data of %p : %d”, ptr, *ptr);  // ptr이 가리키는 곳에 저장된 값
}
```

포인터 연산자

*는 피연산자가 가리키는 곳으로 가서 데이터를 가져온다.

포인터 연산자

&는 피연산자가 위치한 주소를 알려준다.

포인터 연산자

*와 &는 서로 반대 관계이다.

포인터 연산자

```
*(&a) == a
```

메모리 주소 연산

```
main() {
    char * char_ptr = 0x100;
    // 이렇게 초기화하면 안된다.

    printf("%p\n", ++char_ptr);  // 0x101
    printf("%p\n", ++char_ptr);  // 0x102
    printf("%p\n", ++char_ptr);  // 0x103
}
```

메모리 주소 연산

```
main() {
    int * int_ptr = 0x100;
    // 이렇게 초기화하면 안된다.

    printf("%p\n", ++int_ptr);  // 0x104
    printf("%p\n", ++int_ptr);  // 0x108
    printf("%p\n", ++int_ptr);  // 0x10c
}
```

메모리 주소 연산

```
main() {
    double * double_ptr = 0x100;
    // 이렇게 초기화하면 안된다.

    printf("%p\n", ++double_ptr);  // 0x108
    printf("%p\n", ++double_ptr);  // 0x110
    printf("%p\n", ++double_ptr);  // 0x118
}
```

배열

Array

변수들의 집합

배열의 선언

```
type name[length];
```

배열의 선언

```
int student_no[100];
char string[1000];
```

배열의 선언

```
int length = 100;
char string[length];  // 이렇게 선언하면 안된다.
```

배열의 선언

배열의 길이는 **상수로만** 선언할 수 있다.

* ANSI C 기준. 최신 표준에서는 변수로도 선언할 수 있다.

배열의 선언과 초기화

```
int array1[5] = {1, 2, 3, 4, 5};
int array2[5] = {1, 2, 3};  // 이렇게 생략하면 나머지는 0으로 초기화된다.
int array3[5] = {0};  // 그래서 이렇게 쓰면 모두 0으로 초기화된다.
int array4[] = {1, 2, 3, 4, 5, 6, 7, 8};  // 길이는 자동으로 8이 된다.
```

배열 원소에 접근

```
main() {
    int arr[5];

    arr[0] = 1, arr[1] = 2, arr[2] = 3, arr[3] = 4, arr[4] = 5;

    printf("%d, %d, %d, %d, %d\n", arr[0], arr[1], arr[2], arr[3], arr[4]);
}
```

배열 원소에 접근

```
main() {
    int arr[5];

    arr[0] = 1, arr[1] = 2, arr[2] = 3, arr[3] = 4, arr[4] = 5;

    for(int i = 0; i < 5; i++) printf("%d, ", arr[i]);
    printf("\n");
}
```

Array

배열의 index는 **0부터** 시작한다.

배열의 원소에 접근

```
main() {
    int arr[5];

    printf("%d", arr[-1]);  // 문제없이 접근 가능하다.
    printf("%d", arr[100]);  // 이것도 마찬가지.
}
```

배열의 원소에 접근

실제 배열에 존재하지 않는 index에도 접근할 수 있다.

배열의 원소에 접근

접근이 허가되지 않은 메모리 공간에 접근할 수 있다.

배열의 크기와 길이

```
main() {
    int arr[5];

    printf(“size: %d\n”, sizeof arr);  // 크기
    printf(“length: %d\n”, sizeof arr / sizeof (int));  // 길이
}
```

배열과 문자열

문자열은 문자들의 Sequence이다.
즉, char형 **변수들을 이어붙인 배열**이다.

배열과 문자열

```
char string1[10] = “Hello!”;  // 문자열은 이렇게 초기화할 수 있다.
char string2[] = “Hello, World!”;  // 문자열의 경우에도 자동으로 14가 된다.
```

배열과 문자열

문자열은 Null로 끝난다.

‘H’	‘e’	‘l’	‘l’	‘o’	\0

배열과 문자열

실제로는 이렇게 저장되어 있을 수 있다.

‘H’	‘e’	‘l’	‘l’	‘o’	\0	27	34	75	25	47	89	38	42	86	24	47

배열과 문자열

문자열의 끝에 Null이 없다면?

‘H’	‘e’	‘l’	‘l’	‘o’	37	27	34	75	25	47	89	38	42	86	24	47

배열과 문자열

```
main() {
    int string[100];

    scanf("%s", string);  // & 연산자가 붙지 않는다.
}
```

배열과 문자열

```
main() {
    int string[100];

    printf(“%s”, string);
}
```

다차원 배열

다차원 배열

변수를 꼭 한 줄로만 이어붙여야 하는가?

다차원 배열의 선언과 초기화

```
int 2d_array[2][2] = { {1, 2}, {3, 4} };
int 3d_array[2][2][2] = { { {1, 2}, {3, 4}, },
                          { {5, 6}, {7, 8}, } };
int 4d_array[2][2][2][2] = { { { {1, 2}, {3, 4} },
                               { {5, 6}, {7, 8} } },
                             { { {9, 10}, {11, 12} },
                               { {13, 14}, {15, 16} } };
                            // 이런짓은 되도록 하지 말자.
```

다차원 배열

배열의 차원에는 제한이 없지만 4차원 이상은 쓰지 말자.

다차원 배열의 원소에 접근

```
main() {
    int 2d_array[2][2] = { {1, 2}, {3, 4} };

    printf("%d, %d\n", 2d_array[0][0], 2d_array[0][1]);
    printf("%d, %d\n", 2d_array[1][0], 2d_array[1][1]);
}
```

다차원 배열의 원소에 접근

```
main() {
    int 2d_array[10][10];

    for(int i = 0; i < 10; i++) {
        for(int j = 0; j < 10; j++) printf("%d ", 2d_array[i][j]);
        printf("\n");
    }
}
```

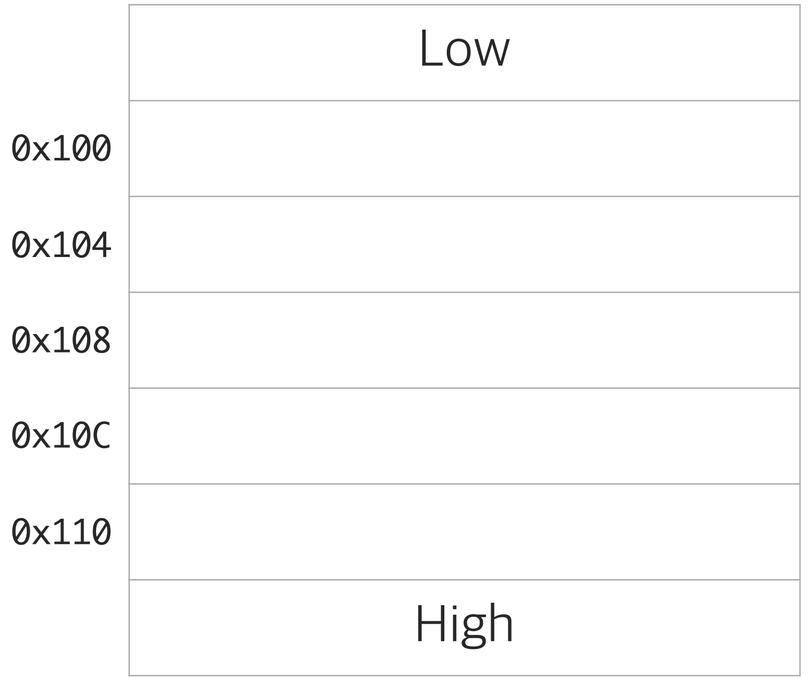
배열과 포인터

배열의 이름은 포인터

배열의 이름은 **배열의 주소를 가리키는 포인터**

배열과 포인터

```
main() {
    int arr[4] = {0, 1, 2, 3};
    print("%p\n", arr);
    print("%p\n", &arr[0]);
    print("%p\n", &arr[1]);
    print("%p\n", &arr[2]);
    print("%p\n", &arr[3]);
}
```



배열과 포인터

```
main() {
    int arr[4] = {0, 1, 2, 3};
    print("%p\n", arr);
    print("%p\n", &arr[0]);
    print("%p\n", &arr[1]);
    print("%p\n", &arr[2]);
    print("%p\n", &arr[3]);
}
```

	Low
0x100	
0x104	0
0x108	1
0x10C	2
0x110	3
	High

배열과 포인터

```
main() {
    int arr[4] = {0, 1, 2, 3};
    print("%p\n", arr);  // 0x104
    print("%p\n", &arr[0]);
    print("%p\n", &arr[1]);
    print("%p\n", &arr[2]);
    print("%p\n", &arr[3]);
}
```

	Low
0x100	
0x104	0
0x108	1
0x10C	2
0x110	3
	High

배열과 포인터

```
main() {
    int arr[4] = {0, 1, 2, 3};
    print("%p\n", arr);
    print("%p\n", &arr[0]);  // 0x104
    print("%p\n", &arr[1]);
    print("%p\n", &arr[2]);
    print("%p\n", &arr[3]);
}
```

	Low
0x100	
0x104	0
0x108	1
0x10C	2
0x110	3
	High

배열과 포인터

```
main() {
    int arr[4] = {0, 1, 2, 3};
    print("%p\n", arr);
    print("%p\n", &arr[0]);
    print("%p\n", &arr[1]);  // 0x108
    print("%p\n", &arr[2]);
    print("%p\n", &arr[3]);
}
```

	Low
0x100	
0x104	0
0x108	1
0x10C	2
0x110	3
	High

배열과 포인터

```
main() {
    int arr[4] = {0, 1, 2, 3};
    print("%p\n", arr);
    print("%p\n", &arr[0]);
    print("%p\n", &arr[1]);
    print("%p\n", &arr[2]);  // 0x10C
    print("%p\n", &arr[3]);
}
```

	Low
0x100	
0x104	0
0x108	1
0x10C	2
0x110	3
	High

배열과 포인터

```
main() {
    int arr[4] = {0, 1, 2, 3};
    print("%p\n", arr);
    print("%p\n", &arr[0]);
    print("%p\n", &arr[1]);
    print("%p\n", &arr[2]);
    print("%p\n", &arr[3]);  // 0x110
}
```

	Low
0x100	
0x104	0
0x108	1
0x10C	2
0x110	3
	High

배열과 포인터

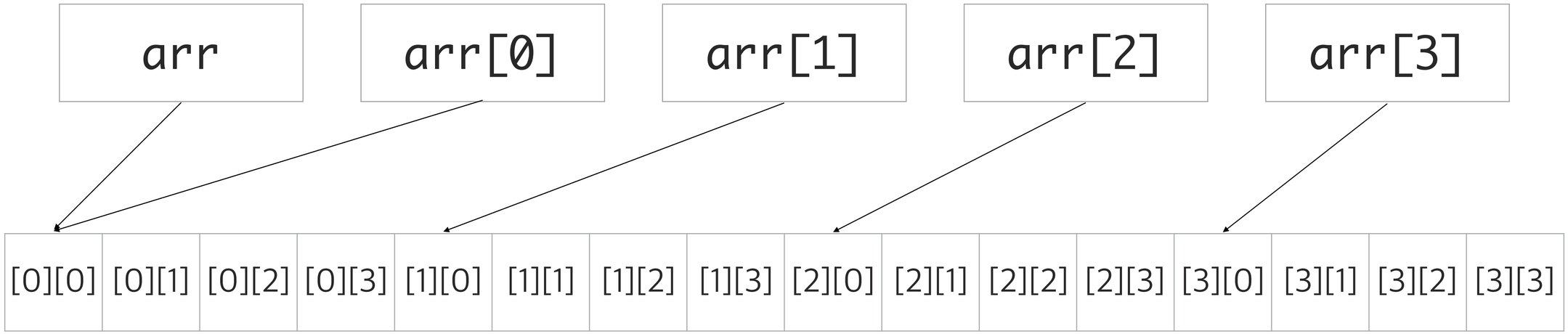
```
main() {
     int a = 0;
     int * ptr = &a;
     print(“%p\n”, ptr);
     print(“%p\n”, &ptr[0]);
     print(“%p\n”, &ptr[1]);
     print(“%p\n”, &ptr[2]);
     print(“%p\n”, &ptr[3]);
     // 배열의 이름이 아닌 포인터도 이런 표현이 가능하다.
}
```

배열과 포인터

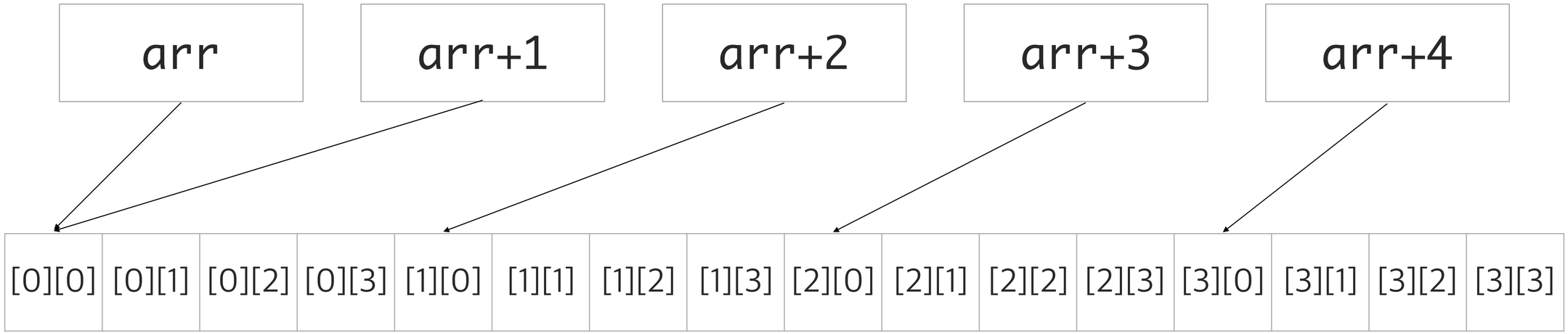
*(ptr + i)와 ptr[i]는 같다.

다차원배열과 포인터

2차원배열과 포인터



2차원배열과 포인터



2차원 배열과 포인터

```
main() {
     char arr[4][4];
     printf(“%d”, sizeof arr);  // 16
     printf(“%d”, sizeof arr[0]);  // 4
     printf(“%d”, sizeof arr[1]);  // 4
     printf(“%d”, sizeof arr[2]);  // 4
     printf(“%d”, sizeof arr[3]);  // 4
}
```

다차원 배열과 포인터

`arr`와 `arr[0]`는 가리키는 주소는 같지만 의미는 다르다.

2차원 배열과 포인터

```
main() {
     char arr[4][4];
     char (*ptr)[4] = arr;  // 4칸(4바이트)씩 건너뛰는 포인터
}
```

2차원 배열과 포인터

```
main() {
    int arr[4][4];
    int (*ptr)[4];  // 4칸(16바이트)씩 건너뛰는 포인터
}
```

2차원 배열과 포인터

```
int * ptr[i] != int (*ptr)[i]
```

2차원 배열과 포인터

2차원 배열에서도 `*(ptr + i)`와 `ptr[i]`는 같다.

2차원 배열의 인자 전달

```
do_something(int (*arr)[4]);

main() {
     int arr[4][4];
     do_something(arr);
}
```

다차원 배열과 포인터

3차원 이상은 2차원의 확장이다.

문자열 포인터

배열의 이름은 포인터

문자열도 메모리에 저장되므로 포인터로 가리킬 수 있다.

문자열 포인터

```
main() {
     char str[] = “Hello, World!”;
     str = “I want to change.”;  // 이렇게 할 수 없다.
}
```

문자열 포인터

```
main() {
     char * str = “Hello, World!”;
     str = “I want to change.”;  // 문자열 포인터를 이용하면 가능하다.
}
```

문자열 포인터

```
main() {
     char str[] = “Hello, World!”;
     str[0] = ‘A’;  // 배열에 저장되었기 때문에 가능하다.
}
```

문자열 포인터

```
main() {
     char * str = “Hello, World!”;
     str[0] = ‘A’;  // 상수이기 때문에 불가능하다.
}
```

다음 수업 준비

복습 및 과제

- 오늘 수업 내용 복습
- 과제 반드시 제출
  - 질문은 얼마든지 가능하니 반드시 제출
- 학습시 리눅스 이용

다음 수업 예습

- 포인터 중 오늘 다루지 않은 나머지 모든 것
- 구조체, 공용체, 열거형
- 사용자 정의 자료형

C Programming – Day 3

2017.09.04.

JunGu Kang
Dept. of Cyber Security

아주대학교 Whois