Introduction to Software Design

C04. Array, Pointer

Yoonsang Lee
Spring 2020

온라인 강의 관련 공지

- 강의 및 실습: 계속 온라인으로 진행
  - 사회적거리두기가 가능한 대형 강의실 확보 문제
  - 보다 긴밀한 실습 지도를 위해 실습시간 외에도 이메일을 통해 조교에게 질문 가능
  - 이메일로 질문 답변 과정 중, 필요한 경우 온라인 화상 회의로 문의도 가능
  - 최근 코로나 사태로 인해, 오프라인으로 조교의 연구실을 방문하여 문의하는 것은 당분간 보류

- 대면 중간고사, 기말고사 계획 중

시험 일정 안

- 중간고사: 6월 4일 (목) 실습시간, IT.BT관 508호
  - 범위: 강의 2 ~ 7
- 기말고사: 6월 23일 (화) 18:30~20:30, 장소 미정
  - 범위: 강의 8 ~ 13

Topics Covered

- 배열(Array)
- 문자열(String)
- 포인터(Pointer)
- 배열과 포인터의 유사점/차이점
- 포인터의 증가/감소 연산
- 함수의 인자로 배열을 전달하기

배열(Array)

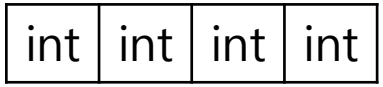
- 동일한 자료형의 변수를 여러 개 모아놓은 것

- ex) 100명의 나이를 저장하는 경우를 생각해보자
  - 만일 이렇게 한다면?
  - int age1, age2, age3, … , age99, age100;
  - 평균을 구하고 싶다면? 최대값을 구하고 싶다면?
  - 매우 불편할 것임!

- **int ages[100]; // 배열**
- **반복문을 통해** 각각의 요소에 접근 가능

배열의 선언 및 배열 요소의 접근

```
int arr[4];

// int 배열을 구성하는 요소의 자료형
// arr 배열의 이름
// [4] 배열의 길이
```



- 배열 ages의 첫 번째 요소: ages[0]
- 배열 ages의 두 번째 요소: ages[1]
- …
- 배열 ages의 **i+1**번째 요소: ages[**i**]

array**[i]** → subscript operator 혹은 indexing operator라고 불림

- 배열 요소의 위치 정보를 나타내는 인덱스(index)는 1이 아닌 **0**부터 시작!

배열에서 [ ]의 사용법

- 배열을 선언할 때

```
int ages[5];
```

  - [ ]안의 숫자는 배열의 길이를 의미

- 배열의 요소에 접근할 때

```
num = ages[2];
ages[1] = 30;
```

  - [ ]안의 숫자(i)는 각 요소가 배열에서 몇 번째(i+1)에 있는 지를 의미

배열을 선언과 동시에 초기화하기

초기화리스트

```
int arr1[5] = {1, 2, 3, 4, 5};
```

1	2	3	4	5

배열 arr1

```
int arr2[] = {1, 2, 3, 4, 5, 6, 7};
```

```
int arr2[7] = {1, 2, 3, 4, 5, 6, 7};
```

컴파일러가 배열의 길이 정보를 알 수 있음

배열의 길이 구하기

- int arr[5];

- sizeof(arr) ?
- → 20 : 전체 바이트 크기

- 배열의 길이를 구하려면: sizeof(arr) / sizeof(int)
- → 5

C & Python Examples

- C

```c
#include <stdio.h>
int main()
{
    int arr[] = {2,4,6,8,10};
    printf("len: %ld\n", sizeof(arr)/sizeof(int));

    int sum = 0;
    for(int i=0; i<sizeof(arr)/sizeof(int); ++i)
        printf("%d\n", arr[i]);

    int min = arr[0];
    int max = arr[0];
    for(int i=0; i<sizeof(arr)/sizeof(int); i++)
    {
        if(arr[i] < min)
            min = arr[i];
        if(arr[i] > max)
            max = arr[i];
    }
    printf("min: %d\n", min);
    printf("max: %d\n", max);
    return 0;
}
```

- Python

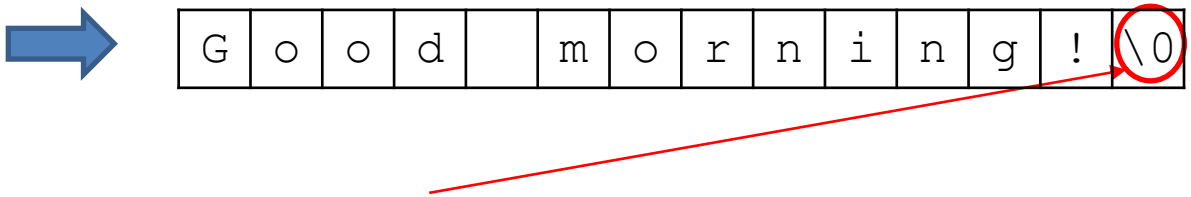
```python
arr = [2,4,6,8,10]
print('len: %ld'%len(arr))

sum = 0
for i in range(len(arr)):
    print('%d'%arr[i])

print('min: %d'%min(arr))
print('max: %d'%max(arr))
```

- Python에서 C의 array와 비슷한 용도로 일반적으로 쓰이는 것은 list이다. (이 두 개가 정확히 같지는 않다)

문자열(String)

- “Hello”
- C언어에서는 문자열도 배열로 표현된다
- -> char형 배열!

```
char str[14]="Good morning!";
```



Null character (₩0): C에서 문자열의 끝을 표시하기 위해 꼭 필요하다!

13글자로 된 문자열을 저장하기 위해서는 길이가 14이상인 배열이 필요하다!

C와 Python의 문자열

- 문자열을 사용하려면 메모리 공간에 저장된 문자열 데이터의 시작 주소와 끝 주소를 알아야 한다.

- C의 문자열
  - 배열이며, 배열은 본래 메모리 공간상의 시작 주소로 접근한다.
  - 문자열 맨 끝 다음에 오는 null 문자로 끝 주소를 알 수 있다.
  - 그래서 C의 문자열을 null-terminated string이라고 부르기도 함.

- Python의 문자열
  - (엄밀히 말하면 python interpreter의 구현에 따라 다르지만, 보통 많이 사용하는 CPython의 경우에는,)
  - `str` 객체이며, 메모리 공간상의 시작 주소와 문자열의 길이를 가지고 있다.
  - 시작 주소와 길이를 통해 끝 주소를 알 수 있다.

배열과 문자열의 차이점

```
char arr1[]={'H', 'i', '!'};

char arr2[]={'H', 'i', '!', '\0'};    // == char arr2 = "Hi!";
```

- arr1은 단순히 char형 배열이다.
- arr2는 문자열이다.

- C는 char형 배열에 null character가 존재해야 문자열로 간주한다.

C & Python Examples

- C

```c
#include <stdio.h>

int main()
{
    char str[] = "Good morning!";

    printf("배열 str의 길이: %d\n",
sizeof(str)/sizeof(char));
    printf("널문자를 문자로 출력: %c\n", str[13]);
    printf("널문자를 숫자로 출력: %d\n", str[13]);

    printf("%s\n", str);

    str[8] = '\0';
    printf("%s\n", str);

    str[6] = '\0';
    printf("%s\n", str);

    str[1] = '\0';
    printf("%s\n", str);

    return 0;
}
```

- Python

```
str = 'Good morning!'
str[0] = '\0'
# TypeError: 'str'
object does not support
item assignment
```

- Python의 string은 변경불가능(immutable) 하므로 이러한 연산이 불가능하다.

scanf()를 이용한 문자열 입력

```
#include <stdio.h>

int main()
{
    char str[50];

    printf("문자열 입력: ");
    scanf("%s", str);
    printf("입력받은 문자열: %s₩n", str);

    return 0;
}
```

배열 이름 str 앞에는 &를 붙이지 않는다!

만일 he is my friend라고 입력하면?

-> scanf는 공백을 기준으로 데이터를 구분하므로 공백을 포함하는 문자열을 읽어 들이지 못한다.

메모리 레이아웃

- **일직선**으로 늘어선 1차원 배열로 생각하면 됨.
  - 0번지 ~ 최대번지
  - 번지수는 1바이트마다 1씩 증가
  - 예:

메모리 주소(번지수)

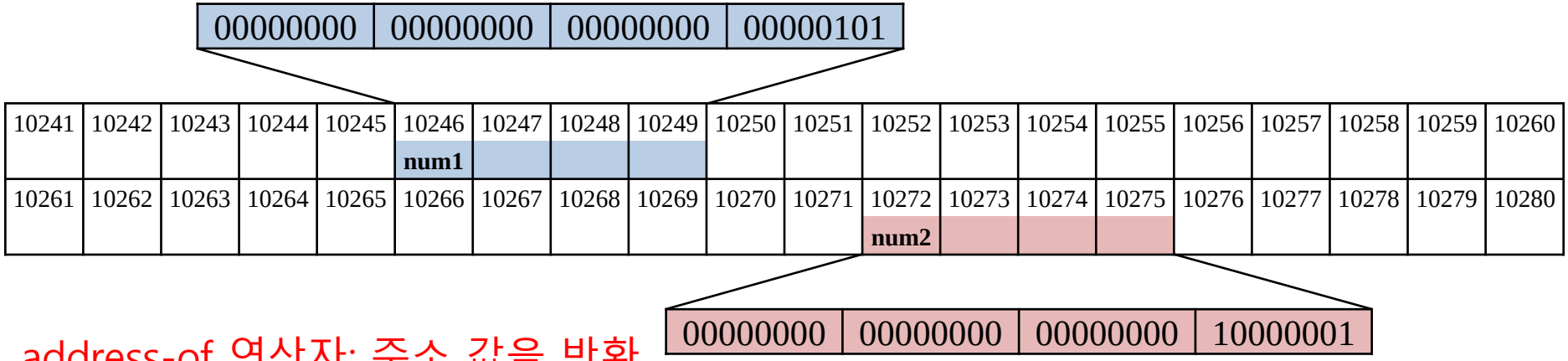
해당 번지에 저장된 내용

10241	10242	10243	10244	10245	10246	10247	10248	10249	10250	10251	10252	10253	10254	10255	10256	10257	10258	10259	10260
10261	10262	10263	10264	10265	10266	10267	10268	10269	10270	10271	10272	10273	10274	10275	10276	10277	10278	10279	10280

(위 그림의 주소 숫자는 하나의 예에 불과함...)

int형 변수가 저장된 모습

```
int num1 = 5;
int num2 = 129;
```



address-of 연산자: 주소 값을 반환

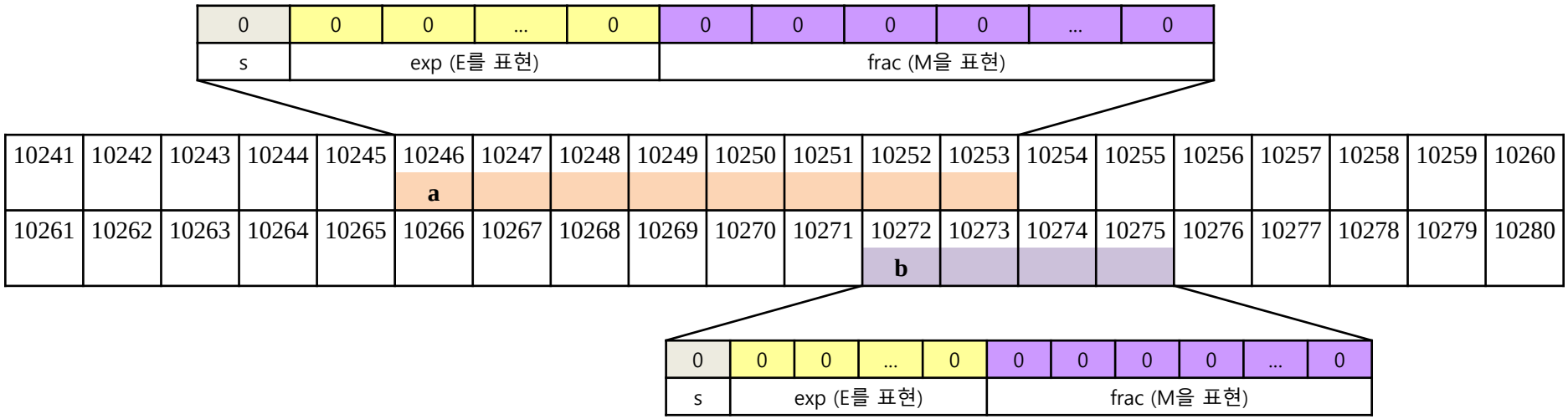
&num1 == ? -> 10246
&num2 == ? -> 10272

(참고)
ARM 계열 – Big-endian : 위에 표시된 순서

Intel x86 계열 – Little-endian : 바이트 단위로 반대 순서
예) 5 -> 00000101 00000000 00000000 00000000

double, float형 변수가 저장된 모습

```
double a = 3.14;
float b = 1.1;
```

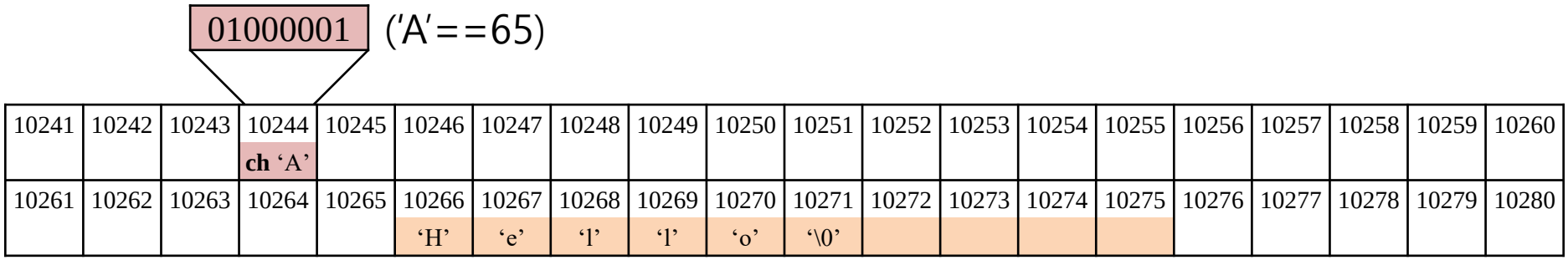


&a == ? -> 10246

&b == ? -> 10272

char형 변수, 문자열이 저장된 모습

```
char ch = 'A';
char str[10] = "Hello";
```



&ch == ? -> 10244

str == ? -> 10266

포인터(Pointer)

- 변수의 메모리 주소를 저장하는 변수

```
double a = 3.14;
float b = 1.1;
```

10241	10242	10243	10244	10245	10246	10247	10248	10249	10250	10251	10252	10253	10254	10255	10256	10257	10258	10259	10260
					a														
10261	10262	10263	10264	10265	10266	10267	10268	10269	10270	10271	10272	10273	10274	10275	10276	10277	10278	10279	10280
											b								

&a == ? -> 10246

&b == ? -> 10272

```
int a_address = &a;
// 변수 a_address에 10246 저장??
```

포인터(Pointer)

10241	10242	10243	10244	10245	10246	10247	10248	10249	10250	10251	10252	10253	10254	10255	10256	10257	10258	10259	10260
					a														
10261	10262	10263	10264	10265	10266	10267	10268	10269	10270	10271	10272	10273	10274	10275	10276	10277	10278	10279	10280
											b								

- 정확한 방법:

```
double* a_address = &a;
```

- 다양한 자료형의 변수의 주소를 저장하기 위한 **포인터 형**이 따로 존재
  - **int***
  - **double***
  - **…**

포인터 : 변수의 메모리 주소를 저장하는 변수

- int* : int형 포인터 – int형 변수의 주소를 저장
- int* pnum1; // int형 포인터 변수 pnum1

- double* : double형 포인터 – double형 변수의 주소를 저장
- double* pnum2; // double형 포인터 변수 pnum2

- char*, float*, …

포인터 변수가 메모리에 저장된 모습

```c
#include <stdio.h>

int main()
{
    char ch1 = 'a';
    char* pch1 = &ch1;

    printf("value of ch1: %d₩n", ch1);
    printf("address of ch1: %p₩n", &ch1);
    printf("value of pch1: %p₩n", pch1);
    printf("address of pch1: %p₩n", &pch1);

    return 0;
}
```

```
value of ch1: 97
address of ch1: 1636819
value of pch1: 1636819
address of pch1: 1636804
```

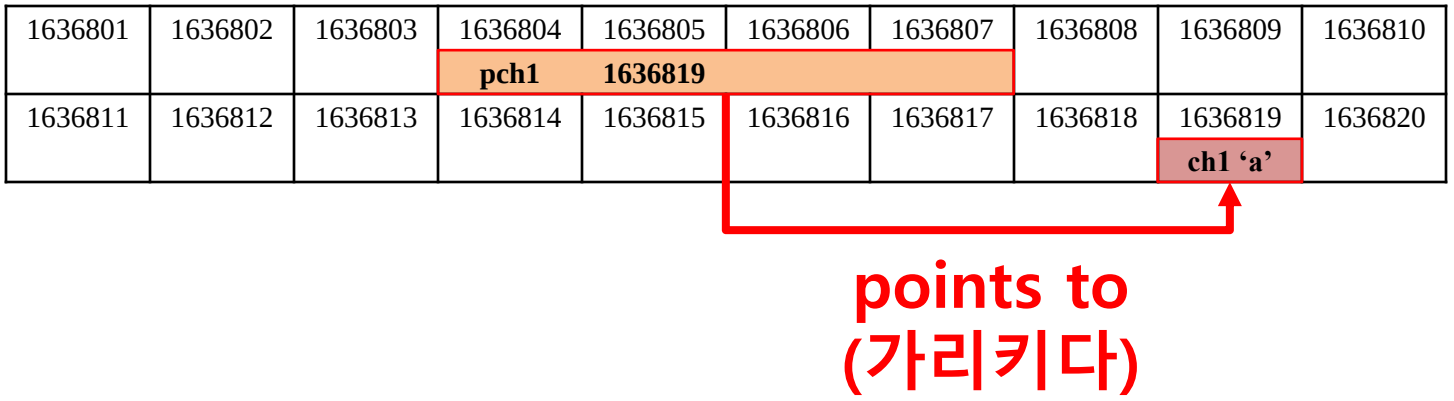
실제로 할당되는 메모리 주소는 실행할 때마다 달라진다.

%p로 메모리 주소를 출력하면 16진수로 출력된다.

하지만 본 강의 슬라이드에서는 편의상 메모리 주소를 10진수로 표시하기로 한다.

포인터 변수가 메모리에 저장된 모습

```
value of ch1: 97
address of ch1: 1636819
value of pch1: 1636819
address of pch1: 1636804
```



- 그래서 변수의 메모리 주소를 저장하는 변수를 **pointer** 라고 한다.

포인터에서 *의 사용법

- 포인터 변수를 선언할 때

```
int* pnum;
```

- 포인터가 가리키는 변수를 참조할 때 (간접참조연산자)

```
num2 = *pnum;
*pnum = 20;
```

참고1) *의 띄어쓰기는 상관없다

```
int* pnum;      // (1)
int * pnum;     // (2)
int *pnum;      // (3)
```

int*를 하나의 자료형으로 인식하기 쉬운 방법인 int와 *를 붙여쓰는 (1)번을 추천.

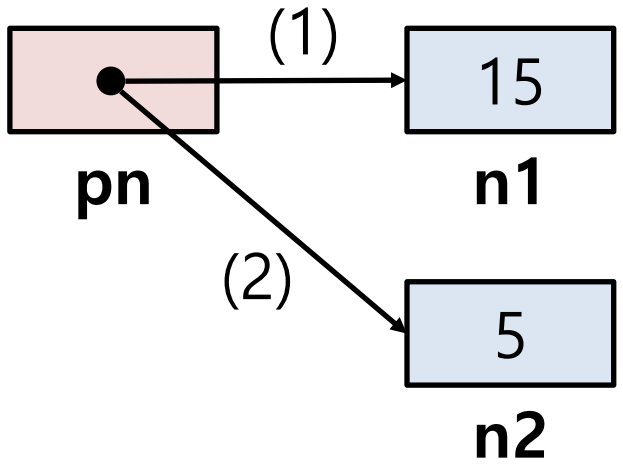
참고2) 아래와 같은 선언도 가능

```
int num, * pnum;
```

하지만 int*를 붙여 쓰지 않기 때문에 추천하지 않음.

포인터가 가리키는 변수가 바뀔 수도 있다

```c
#include <stdio.h>

int main()
{
    int n1 = 10, n2 = 10;
    int* pn;

    pn = &n1;
    (*pn) += 5;      // n1 += 5;

    pn = &n2;
    (*pn) -= 5;      // n2  -= 5;

    printf("n1: %d, n2: %d\n", n1, n2);
    return 0;
}
```



Quiz #1

- Go to https://www.slido.com/
- Join #isd-hyu
- Click “Poll”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as “attendance”.

다양한 포인터 형이 존재하는 이유

- 어차피 어떤 변수가 저장된 메모리 공간의 시작 주소만 알면 되지 않나요? - (X)
- int*형 변수로 *연산을 통해 메모리 접근 시 4바이트 메모리 공간에 정수의 형식으로 데이터를 읽고 쓴다.
- float*형 변수로 *연산을 통해 메모리 접근 시 4바이트 메모리 공간에 실수의 형식으로 데이터를 읽고 쓴다.

```c
#include <stdio.h>
int main()
{
    float num = 3.14;
    int* pnum = &num;
    *pnum = 10;
    printf("%f\n", num);
    return 0;
}
```

잘못된 포인터의 사용

```
int* ptr;
*ptr = 200;

// ptr이 쓰레기 값으로 초기화되어 200이 저장되는 위치를 알 수가 없는 위험한 코드
```

```
int* ptr = 10;
*ptr = 200;

// 10번지가 현재 어떻게 사용되고 있는지도 모르는 상태에서 10번지에 데이터를 저장하려고 하는 위험한 코드
```

널(Null) 포인터

- 포인터가 가리키는 변수가 나중에 결정되는 경우에는 널 포인터 (null pointer)로 초기화 하는 것이 안전하다.

```
int* ptr = NULL;
```

  - NULL은 0으로 define되어 있다. 이는 0번지들 가리키는 것이 아니라, 아무 것도 가리키지 않는 것으로 해석이 된다.

- 보통 아래와 같이 의미 있는 주소값이 포인터에 대입되어 있는지 체크한 후 필요한 작업을 하도록 하는 식으로 코드 작성

```
if (ptr2 == NULL)
    ptr2 = &num1;

*ptr2 = 10;
```

32bit & 64bit 아키텍처

- **32bit CPU**
- **32bit OS**

- 메모리 주소를 32bit 로 표현

- 시스템이 인식할 수 있는 최대 메모리
- → 2^{32} 바이트 = 4GB (이론상)

- 포인터의 크기는?
- → 32bit = 4 바이트

- **64bit CPU**
- **64bit OS**

- 메모리 주소를 64bit 로 표현

- 시스템이 인식할 수 있는 최대 메모리
- → 2^{64} 바이트 = 16TB (이론상)

- 포인터의 크기는?
- → 64bit = 8 바이트

배열의 이름

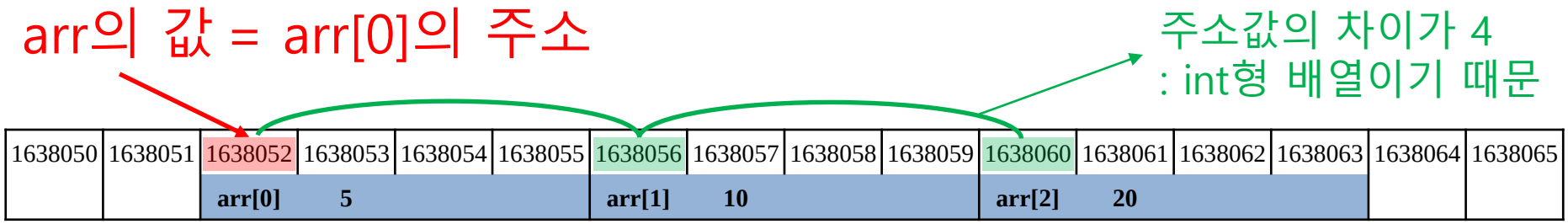
```
#include <stdio.h>

int main()
{
    int arr[3] = {5, 10, 20};
    printf("배열 이름 자체의 값: %p₩n", arr);
    printf("첫 번째 요소의 주소: %p₩n", &arr[0]);
    printf("두 번째 요소의 주소: %p₩n", &arr[1]);
    printf("세 번째 요소의 주소: %p₩n", &arr[2]);

    return 0;
}
```

```
배열 이름 자체의 값: 1638052
첫 번째 요소의 주소: 1638052
두 번째 요소의 주소: 1638056
세 번째 요소의 주소: 1638060
```

배열이 메모리에 저장된 모습

```
int arr[3] = {5, 10, 20};
```

```
배열 이름 자체의 값: 1638052
첫 번째 요소의 주소: 1638052
두 번째 요소의 주소: 1638056
세 번째 요소의 주소: 1638060
```



- 배열의 이름은 배열의 시작 주소 값 (첫 번째 요소의 주소값)을 의미
- 다시 말하면, arr == &arr[0]

배열과 포인터의 유사점

- **둘 다 (어떤) 주소를 나타냄**

- 배열, 포인터 모두 ***연산**을 사용 가능
- 배열, 포인터 모두 **[ ]연산**을 사용 가능

```
int arr[] = {5, 10, 15};
int* parr = arr;

// 5 5 5 5 를 출력
printf("%d %d %d %d₩n", arr[0], *arr, parr[0], *parr);
```

배열과 포인터의 차이점

- **배열은 포인터가 아니다!**

- 배열의 이름에 다른 값을 대입할 수 없다.

```
int arr[3] = {5, 10, 20};
int num = 30;
arr = &num; // X
```

컴파일 에러!

error C2106: '=' : 왼쪽 피연산자는 l-value이어야 합니다.

- [배열 이름의 값]과 [배열 이름의 주소값]이 같다.

```
int arr[3] = {5, 10, 20};
printf("arr: %p₩n", arr);
printf("&arr: %p₩n", &arr);
```

```
arr: 1636840
&arr: 1636840
```

두 값이 같다 (포인터라면 다른 주소값이 나올 것)

배열과 포인터의 차이점

- sizeof 연산자 결과가 다르다.

```
int arr[3] = {5, 10, 20};
int* parr = arr;
int size1 = sizeof(arr);
int size2 = sizeof(parr);
```

size1==12 : 배열 전체의 크기
size2==4 : 포인터 변수의 크기

C Example

```c
#include <stdio.h>
int main()
{
    int arr[3] = {5, 10, 20};
    int num = 30;
    //arr = &num;   // compile error
    int* parr = arr;

    printf("%d %d %d %d\n", arr[0], *arr, parr[0], *parr);

    printf("arr: %p\n", arr);
    printf("&arr: %p\n", &arr);

    printf("parr: %p\n", parr);
    printf("&parr: %p\n", &parr);

    printf("size of arr: %d\n", sizeof(arr));
    printf("size of parr: %d\n", sizeof(parr));

    return 0;
}
```

- Python에는 pointer의 개념이 존재하지 않는다.

포인터에 대한 증가/감소 연산

```
pi: 1636948, pi+1: 1636952, pi+2: 1636956
pd: 1636932, pd+1: 1636940, pd+2: 1636948
```

```c
#include <stdio.h>

int main()
{
    int i = 1;
    double d = 1.2;
    int* pi = &i;
    double* pd = &d;

    printf("pi: %p, pi+1: %p, pi+2: %p₩n", pi, pi+1, pi+2);
    printf("pd: %p, pd+1: %p, pd+2: %p₩n", pd, pd+1, pd+2);

    return 0;
}
```

포인터에 대한 증가/감소 연산

```
int i = 1;
double d = 1.2;
int* pi = &i;
double* pd = &d;
```

```
pi: 1636948, pi+1: 1636952, pi+2: 1636956
pd: 1636932, pd+1: 1636940, pd+2: 1636948
```

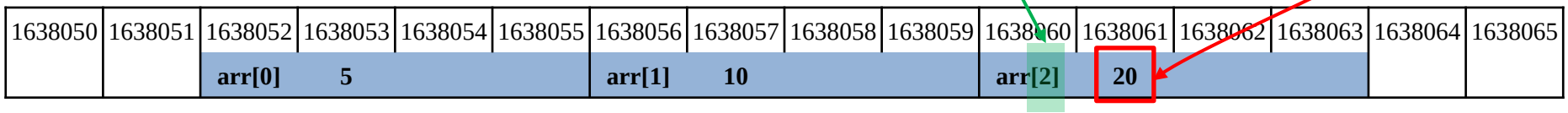
- int형 포인터에 1을 더하면, 실제 값은 4 증가
- double형 포인터에 1을 더하면, 실제 값은 8 증가
- …
- (특정 자료형) 포인터에 1을 더하면, 실제 값은 sizeof(자료형)만큼 증가

- 감소도 마찬가지

배열의 [ ] 연산의 의미

- **arr[i] : 배열에서 index i에 위치하는 요소의 값 (i+1번째 값)**

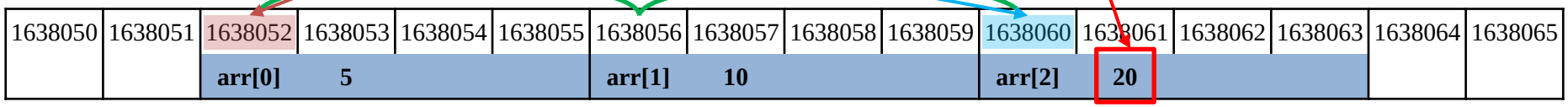
- 예) `int arr[3] = {5, 10, 20};`
- arr[2]: int형배열 arr에서 index 2에 위치하는 요소의 값



포인터의 증가/감소 연산의 의미

- ***(arr+i) : 배열의 시작 주소로부터 i 만큼 증가된 주소에 저장되어 있는 값**

- 예) `int arr[3] = {5, 10, 20};`
- *(arr+2): int형 배열 arr의 시작 주소로부터 2 만큼 증가된 주소에 저장되어 있는 값



포인터의 증가/감소 연산과 배열의 [ ] 연산의 관계

- 배열에서 index i에 위치하는 요소의 값 (i+1번째 값)
- 배열의 시작 주소로부터 i 만큼 증가된 주소에 저장되어 있는 값

arr[i] == *(arr+i)

- (arr이 배열의 이름이어도, 포인터 변수여도 성립)

C Example

```c
#include <stdio.h>

int main()
{
    int arr[] = {5, 10, 15, 20};
    int* parr = arr;
    int i;

    for(i=0; i<4; i++)
    {
        // 모두 같은 값을 나타낸다.
        printf("%d %d %d %d\n", arr[i], *(arr+i), parr[i],
*(parr+i));
    }

    return 0;
}
```

Quiz #2

- Go to https://www.slido.com/
- Join #isd-hyu
- Click “Poll”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as “attendance”.

Quiz #3

- Go to https://www.slido.com/
- Join #isd-hyu
- Click “Poll”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as “attendance”.

배열의 내용을 보기 좋게 출력해주는 함수?

```
int main()
{
    int arr[] = {5, 10, 15};
    printArray(arr);

    return 0;
}
```

```
void printArray(    ?    )
{
    int i;
    for(i=0; i<3; i++)
        printf("%d ", arr[i]);
    printf("₩n");
}
```

배열의 내용을 보기 좋게 출력해주는 함수?

포인터 매개변수로 배열의 시작 주소 값 전달

```
int main()
{
    int arr[] = {5, 10, 15};
    printArray(arr);

    return 0;
}
```

```
void printArray(int* arr)
{
    int i;
    for(i=0; i<3; i++)
        printf("%d ", arr[i]);
    printf("₩n");
}
```

왜 이렇게 하는 것이 가능?

```
int arr[] = {5, 10, 15};
int* parr = arr;
printf("%d %d %d %d₩n", arr[1], *(arr+1), parr[1],
*(parr+1));
```

이렇게 배열을 포인터처럼, 포인터를 배열처럼 쓸 수 있으니까...

다양한 길이의 배열을 인자로 받고 싶다면?

- 아래 코드에서 무엇이 문제일까?

```
int main()
{
    int arr[] = {5, 10, 15,1};
    printArray(arr);

    return 0;
}
```

```
void printArray(int* arr)
{
    int i;
    for(i=0; i<3; i++)
        printf("%d ", arr[i]);
    printf("₩n");
}
```

  - void printArray(int* arr)로 전달받는 arr은 배열의 시작 주소를 가리키는 포인터이므로 배열의 길이 정보를 가지고 있지 않다.

  - 그렇다면 어떻게 해야 할까? -> 배열의 길이도 인자로 전달!

함수의 인자로 배열을 전달하는 방법

- 포인터 매개변수로 **배열의 시작 주소 값** 전달
- **배열의 길이**도 함께 인자로 전달

```
int main()
{
    int arr[] = {5, 10, 15,1};
    printArray(arr, 4);

    return 0;
}
```

```
void printArray(int* arr, int len)
{
    int i;
    for(i=0; i<len; i++)
        printf("%d ", arr[i]);
    printf("₩n");
}
```

C & Python Examples

- C

```c
#include <stdio.h>

void printArray(int* arr, int len)
{
    printf("Array ");
    for(int i=0; i<len; i++)
        printf("[%d]:%d, ", i, arr[i]);
    printf("\n");
}

int main()
{
    int arr[] = {5, 10, 15, 20, 25, 30};
    printArray(arr, sizeof(arr)/sizeof(int));

    return 0;
}
```

- Python

```python
def printArray(arr):
    print('Array ', end='')
    for i in range(len(arr)):
        print('[%d]:%d, '%(i, arr[i]), end='')
    print()

arr = [5, 10, 15, 20, 25, 30]
printArray(arr)
```

배열을 인자로 전달할 때 또 다른 표기법

```
void printArray(int* arr, int len)
```

- 이렇게 하는 대신

```
void printArray(int arr[], int len)
```

- 이렇게 할 수도 있다.

- 완전히 동일한 의미. arr은 배열이 아닌, 포인터이다.

- 단지 “배열의 느낌”을 좀 더 주는 표기법이라고 할 수 있다.

Next Time

- Labs in this week:
  - Lab1: 과제 9-1
  - Lab2: 과제 9-2

- Next lecture:
  - 10-C05. String Functions, Text Games in C, Const