C교육 4주차

201702075 조수환

ARGOS

INDEX

ARGOS

C교육 4주차

3주차 과제 풀이

- 과제 1
- 과제 2
- 과제 3
- 과제 4

ARGOS

과제 1

조건문과 아스키 코드의 값을 활용하여, 알파벳 소문자를 대문자로 바꾼다.

```c
#include <stdio.h>

int main(void) {
    char str[100];
    scanf("%s", str);

    for(int i=0 ; i<100 ; i++) {
        if(str[i] >= 97 && str[i] <= 122) str[i] -= 32;
    }

    printf("%s\n", str);
    return 0;
}
```

65	0x41	A	97	0x61	a
66	0x42	B	98	0x62	b
67	0x43	C	99	0x63	c
68	0x44	D	100	0x64	d
69	0x45	E	101	0x65	e
70	0x46	F	102	0x66	f
71	0x47	G	103	0x67	g

입력받은 문자의 값이 97 이상 122 이하라면, 그 값에서 32를 뺀다.

(소문자는 97 ~ 122의 아스키 코드 값, 대문자는 소문자와 32만큼 값이 차이남

과제 1

ARGOS

과제

대문자 변환(필수)

입력받은 소문자를 대문자로 바꿔 출력하시오.

(* 문자열을 입력할 때 공백(space)을 입력하시면 안됩니다.)

```
sh2358@edu-argos:~/C-education/homework/week3$ ./homework1
hello! my name is
HELLO!
```

scanf() 함수는 문자열을 입력받을 때 공백(space)을 입력의 끝으로 인식합니다.

그래서 앞의 hello! 까지만 대문자로 변환되어 출력됩니다.

(공백을 포함해서 문자열을 입력받는 방법은 구글에게!)

과제 2

피보나치 수열의 점화식을 정의하면 f(n) = f(n – 1) + f(n – 2), f(0) = 0, f(1) = 1

위의 점화식을 바탕으로 재귀함수 fibonacci(int n)를 정의한다.

```
long long fibonacci(int n) {
    if(n <= 1) return n;
    else return fibonacci(n - 2) + fibonacci(n - 1);
}
```

if(n <= 1) return n; -> f(0) = 0, f(1) = 1

return fibonacci(n - 2) + fibonacci(n - 1) -> f(n) = f(n - 1) + f(n - 2)

과제 2

입력받은 n의 값이 고작 5씩 늘어날 뿐인데 실행 시간이 급격히 증가한다.

```
sh2358@edu-argos:~/C-education/homework/week3$ ./homework2
n = 40
f(n) = 102334155
Run Time: 0.564649
sh2358@edu-argos:~/C-education/homework/week3$ ./homework2
n = 45
f(n) = 1134903170
Run Time: 6.234045
sh2358@edu-argos:~/C-education/homework/week3$ ./homework2
n = 50
f(n) = 12586269025
Run Time: 69.010695
```

n = 55 부터는 아예 10분이 넘어가 버린다.

```
sh2358@edu-argos:~/C-education/homework/week3$ ./homework2
n = 55
f(n) = 139583862445
Run Time: 767.437532
```

이 정도면 사람이 계산하는게 더 빠를 지경이다. 왜 이럴까?

과제 2

ARGOS

컴퓨터의 실행 시간은 연산 횟수에 비례한다. (값의 크기는 큰 관계가 없다.)

```
scanf("%d", &n);
clock_t start = clock();
for(int i=0 ; i<n ; i++) sum += 1;
clock_t end = clock();
```

입력받은 n만큼 1을 더하는 연산 수행

```
n = 20000000
Run Time : 0.035280
```

2천만번

```
n = 200000000
Run Time : 0.308916
```

2억번

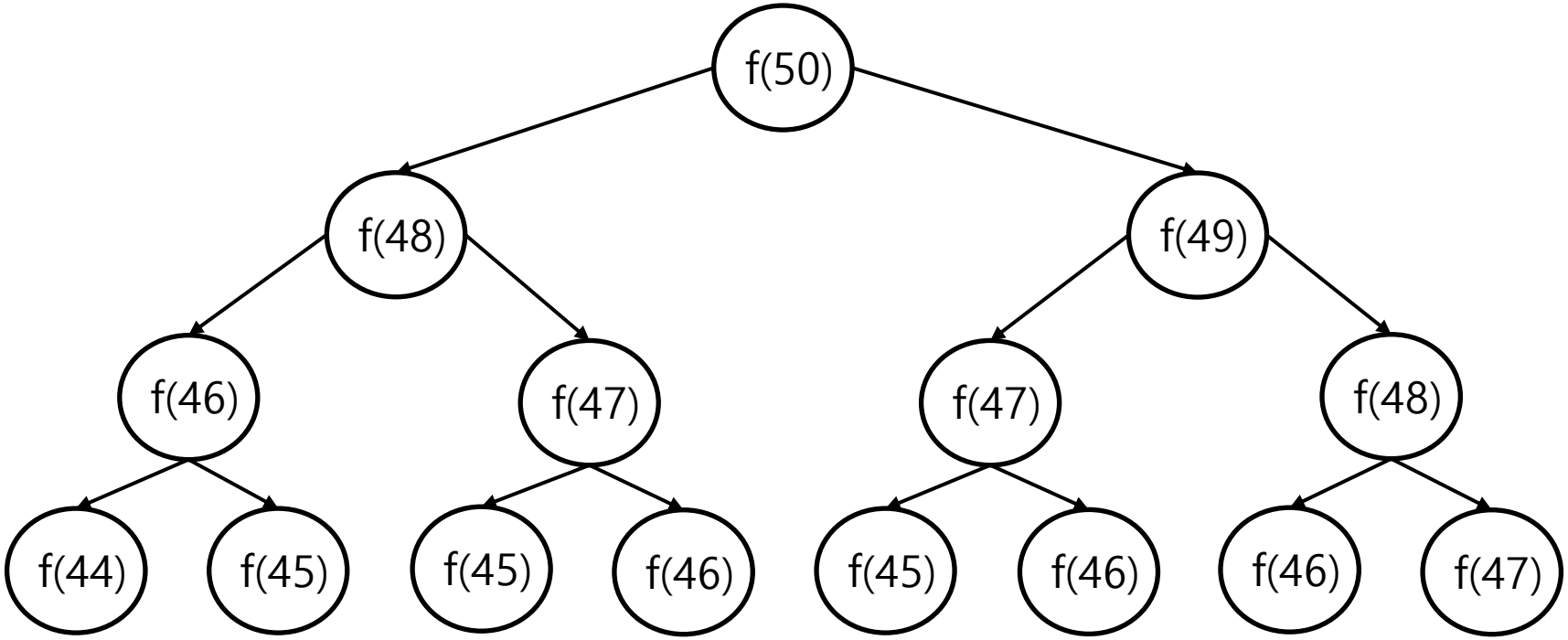
```
n = 2000000000
Run Time : 3.057456
```

20억번

* 연산 횟수에 비례해서 실행 시간이 약 10배씩 늘어나는 것을 볼 수 있다.

과제 2

```
return fibonacci(n - 2) + fibonacci(n - 1);
```



과제 2

ARGOS

Exponential time(지수 시간)

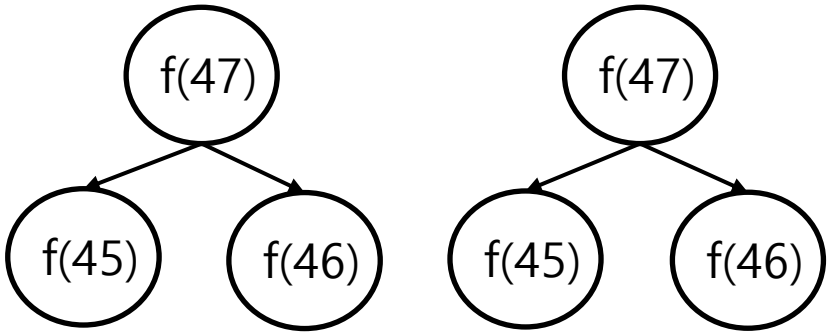
함수를 한번 호출하는 것을 연산 1번이라고 생각해보자.

그렇다면 fibonacci(50)은 무려 약 2^{50}번의 연산을 해야한다.

이는 약 1000조가 넘는 값이고, n의 값이 1 커질때 마다 두배씩 늘어날 것이다.

이러한 구조를 갖는 프로그램은 실행 시간이 짧을 수가 없다.

과제 3



과제 2의 함수 호출 과정을 보면 아주 비효율 적인 부분이 있다.

f(47)의 값을 구하기 위해 f(45)와 f(46)을 두번이나 호출한다.

만약 f(47)의 값을 연산을 통해 알고 있다면, 저런 과정을 필요가 없다.

과제 3

한 마디로 초기값을 이용하여 반복문을 사용하면,

과제 2의 재귀함수처럼 불필요한 연산을 하지 않을 수 있다.

```
long long fibo[100];
int n;
printf("n = ");
scanf("%d", &n);

fibo[0] = 0;
fibo[1] = 1;

clock_t start = clock();
for(int i=2 ; i<=n ; i++) {
    fibo[i] = fibo[i-2] + fibo[i-1];
}
clock_t end = clock();
```

```
n = 50
f(n) = 12586269025
RunTime : 0.000005
```

f(n – 2), f(n – 1)의 값이
배열에 저장되어 있어
가져오기만 하면 된다.

위의 방식대로면 연산을 입력받은 n만큼만 하면되서 시간이 얼마 안걸린다.

과제 4

과제 4의 핵심 원리는 다음과 같다.

n자리의 코드는 1~9의 숫자 뒤에 n-1자리의 코드를 붙인 것과 같다.

즉 1로 시작하는 n자리 코드 갯수 = 1 뒤에 올 수 있는 n-1자리 코드 갯수

과제 4

```
scanf("%d", &n);
int arr[n][10];
for(int i=0 ; i<=9 ; i++) arr[0][i] = 1;

for(int i=1 ; i<n ; i++) {

    arr[i][0] = arr[i-1][1] + arr[i-1][2];
    arr[i][9] = arr[i-1][8] + arr[i-1][7];
    for(int j=1 ; j<=8 ; j++) {
        arr[i][j] = arr[i-1][j-1] + arr[i-1][j+1];
    }

    arr[i][1] += arr[i-1][3];
    arr[i][8] += arr[i-1][6];
    for(int j=2 ; j<=7 ; j++) {
        arr[i][j] = arr[i][j] +  arr[i-1][j-2] + arr[i-1][j+2];
    }
}

for(int i=1 ; i<=9 ; i++) sum += arr[n-1][i];
```

핵심 알고리즘

도전 해보신 분들 중 설명이 더 필요한 분이 있으면 교육 이후에 질문해주세요!

C교육 4주차

포인터

- 포인터의 정의 및 사용
- 역참조
- Call By Value
- 실습

ARGOS

포인터의 정의 및 사용

포인터(Pointer)

- 메모리 주소를 가리키는 역할을 해서 point + er
- 값이 아닌 특정 메모리 주소를 저장하는 변수를 포인터 변수라 함
- 사용법 : [자료형]* [변수명] = [주소값], ex) int* ptr = &a
- 메모리 주소의 형식 지정자는 "%p"이고, 16진수로 표현된다.

```c
int a = 4;
int* ptr1 = &a;
int * ptr2 = &a;
int *ptr3 = &a;
printf("%p %p %p\n", ptr1, ptr2, ptr3);
```

```
sh2358@edu-argos:~/C-education/practice$ ./pointer
0x7ffce74c12cc 0x7ffce74c12cc 0x7ffce74c12cc
```

선언 형식이 달라도 다 같은 뜻이다.

포인터의 정의 및 사용

변수 a, b, c, d의 주소를 저장한 포인터 변수를 선언하고

형식에 맞게 출력하시오

```c
#include <stdio.h>

int main(void) {
    int a = 1;
    int b = 2;
    int c = 3;
    int* ptr_a = &a;
    /*
        write your code
                        */
    return 0;
}
```

위의 캡처 처럼 괴상한 숫자 + 영어가

총 4개 출력 될 겁니다.

* 주소가 16진수로 표기되서 영어가 있는겁니다.

(a = 10, b = 11 ...) 주소 값에 따라 없을수도 있어요.

포인터의 정의 및 사용

각자 숫자는 달라도 아마 아래 결과화면 처럼 4씩 차이가 날 겁니다.

```
sh2358@edu-argos:~/C-education/practice$ ./pointer
0x7fffec00a25c 0x7fffec00a260 0x7fffec00a264
```

int a　　int b　　int c

* 맨 뒤의 c는 12입니다. 4를 더하면 16이라 25c에서 260이 됩니다.

자료형	표현	크기
int	정수	4byte

주소 값이 4씩 차이나는 이유는 자료형 int의 크기가 4byte라서 그런겁니다.

포인터의 정의 및 사용

변수들은 자료형에 맞는 크기를 가진다. 포인터 변수도 크기가 있을까?

```c
char a = 'a';
int b = 4;
double c = 5;
char* ptr_a = &a;
int* ptr_b = &b;
double* ptr_c = &c;
printf("size of char : %ld\n", sizeof(a));
printf("size of int : %ld\n", sizeof(b));
printf("size of double : %ld\n", sizeof(c));
printf("size of char* : %ld\n", sizeof(ptr_a));
printf("size of int* : %ld\n", sizeof(ptr_b));
printf("size of double* : %ld\n", sizeof(ptr_c));
```

sizeof()는 인자로 받은 변수의 크기를 알려준다.

각각 char, int, double 변수와 그들을 가리키는 포인터 변수를 선언한다.

각각의 크기는 어떻게 나올까?

포인터의 정의 및 사용

자료형	표현	크기
int	정수	4byte
float	실수(소수)	4byte
double	실수(소수)	8byte
char	문자	1byte

```
sh2358@edu-argos:~/C-education/practice$ ./bit
size of char : 1
size of int : 4
size of double : 8
size of char* : 8
size of int* : 8
size of double* : 8
```

각 변수들은 자료형에 맞는 크기가 출력되었으나

그들을 가리키는 포인터 변수는 모두 같은 크기가 출력된다.

포인터의 정의 및 사용

자료형의 크기는 해당 자료를 표현하기 위해 필요한 범위와 같다.

int를 예로 들어보자. int 자료형의 크기는 4byte(=32bit)이다.

(int)

00000000 00000000 00000000 00000000

1bit

bit 한 개는 0과 1, 2개를 표현 한다. 32개의 bit는 2^{32} 개의 숫자를 표현 할 수 있다.

이는 총 4,294,967,296개 이며, 그렇기 때문에 자료형 int의 표현범위가

-2,147,483,648 ~ 2,147,483,647가 되는 것이다.

포인터의 정의 및 사용

int가 정수를 표현하는 변수라면, 포인터 변수는 주소 값을 표현한다.

그러므로 주소 값을 모두 표현 할 수 있을 만큼의 범위를 가져야 한다.

그리고 메모리 주소 값의 경우의 수는 운영체제에 따라 갈린다.

디바이스 이름	DESKTOP-U929A0Q
프로세서	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
설치된 RAM	16.0GB(15.9GB 사용 가능)
장치 ID	00EC35F6-BF1B-4684-B189-1A1FFED5D033
제품 ID	00325-81724-06961-AAOEM
시스템 종류	64비트 운영 체제, x64 기반 프로세서

64bit 운영체제에서는 64bit로 표현 가능 한 만큼의 메모리를 쓸 수 있다.

포인터의 정의 및 사용

따라서 아래와 같은 결과가 나온 이유는 해당 프로그램이 64비트 OS에서

돌아갔기 때문이고, 포인터 변수는 2^{64}만큼의 메모리 주소값을

표현해야 하기 때문에 가리키는 자료형에 관계없이 크기가 8(byte)이 된다.

```
sh2358@edu-argos:~/C-education/practice$ ./bit
size of char : 1
size of int : 4
size of double : 8
size of char* : 8
size of int* : 8
size of double* : 8
```

```
sh2358@edu-argos:~/C-education/practice$ gcc -o bit bit.c -m32
sh2358@edu-argos:~/C-education/practice$ ./bit
size of char : 1
size of int : 4
size of double : 8
size of char* : 4
size of int* : 4
size of double* : 4
```

32비트 환경에서 컴파일

2^{32}개의 경우 표현 -> 4byte

역참조

역참조

- 포인터가 가리키는 주소에 저장된 값을 참조 하는것
- 메모리 주소 앞에 "*"을 붙여 쓰면, 해당 주소의 값에 접근한다.
- 사용법 : *[주소값 or 포인터 변수]

```c
int a = 4;
int* ptr = &a;
printf("ptr = %p *ptr = %d\n", ptr, *ptr);
*ptr = 40;
printf("*ptr = %d\n", *ptr);
```

```
sh2358@edu-argos:~/C-education/practice$ ./pointer
ptr = 0x7ffe70ac248c *ptr = 4
*ptr = 40
```

근데 이러면 변수 a랑 *ptr은 결국 같은 거 아닌가?

Call By Value

Call By Value -> C언어에서의 함수 인자는 모두 값으로써 넘겨진다.

이게 무슨 말 인지는 아래의 예시를 보자.

```c
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

(인자 a와 b의 값을 서로 바꾸는 함수)

```c
int main(void) {
    int a = 4, b = 40;
    swap(a, b);
    printf("a = %d\nb = %d\n",a, b);
    return 0;
}
```

swap 함수를 호출하고 인자로 a(4), b(40)을 넣었으므로

예상되는 출력결과는 a = 40, b = 4 이다.

Call By Value

하지만 a와 b의 값은 바뀌지 않았다

```
sh2358@edu-argos:~/C-education/practice$ ./swap
a = 4
b = 40
```

다시 봐도 코드에 문제가 있는 것 같지는 않다. 왜 이럴까?

```
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Call By Value

swap함수를 호출하면 다음과 같은 과정이 일어난다.

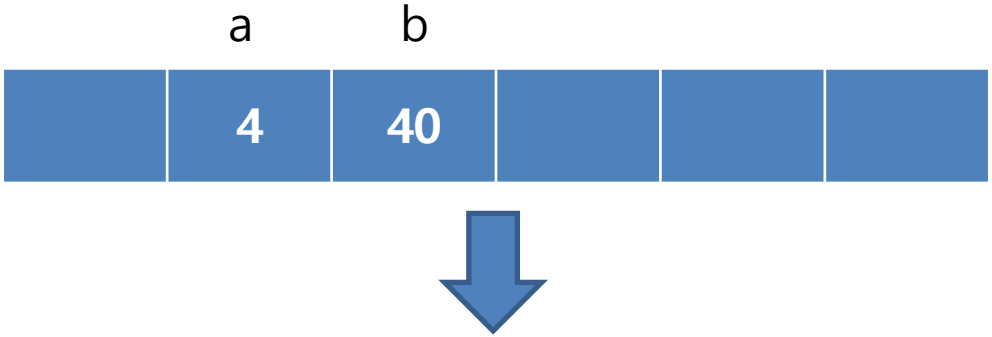
```
swap(a, b);
```

먼저 인자로 a, b를 넣었으니 a와 b의 값을 알아야 한다.

메모리에 저장된 a와 b의 값을 찾았다. 이제 이걸 함수에 갖다 쓰면 되는데..

Call By Value

여기서 a와 b를 직접 쓰는 것이 아니라 a와 b의 "값"만 빌려 쓴다.

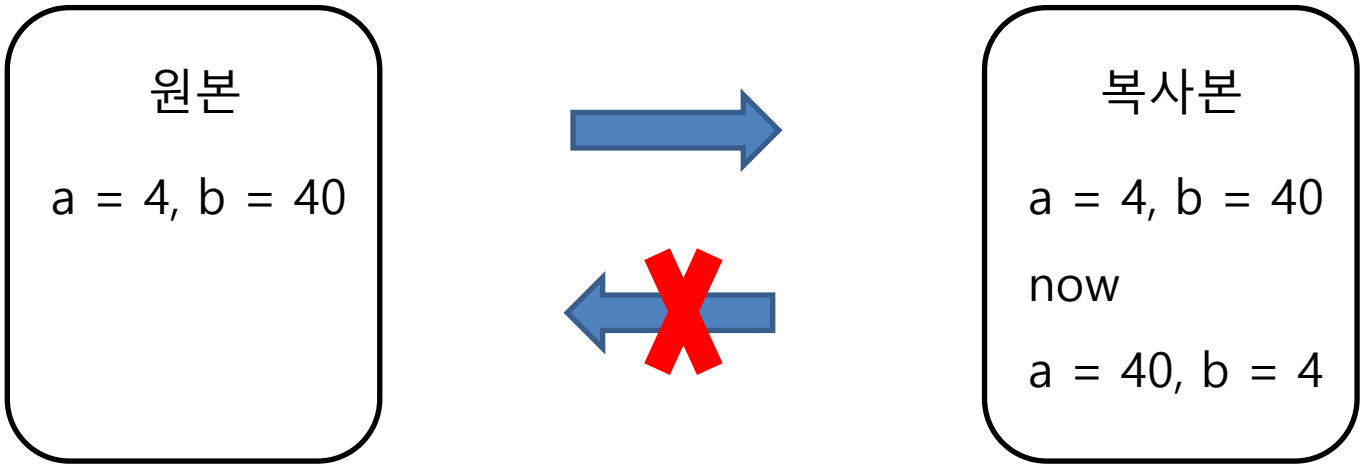



```
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

변수 a, b가 아니라 a, b가 가지는 값으로만 함수를 실행한다.

Call By Value

그래서 실제 변수의 값엔 아무일도 일어나지 않는다.



복사본에 아무리 낙서를 해봐야 원본이 수정되지 않는 것 처럼

Call By Value

때문에 함수로 실제 변수의 값을 바꾸려면 변수의 주소가 필요하다.

사실 우리는 이러한 예시를 이미 본 적이 있다.

```
printf("a = %d\n", a);
```
→ 얘는 그냥 변수를 쓰는데

```
scanf("%d", &a);
```
→ 얘는 왜 변수의 주소를 쓸까?

A. C에서는 어떤 값을 불러올 때 원본이 아닌 복사본을 가져온다.

(2021 C 교육 2주차 – 슬라이드 21 (입출력 함수))

Call By Value

위의 설명대로 올바른 동작을 하는 swap 함수를 작성해보면

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

변수의 주소를 인자로 받는다.

주소를 역참조 하여 값을 바꾼다.

```
int main(void) {
    int a = 4, b = 40;
    swap(&a, &b);
    printf("a = %d\nb = %d\n",a, b);
    return 0;
}
```

&을 붙여 a와 b의 주소를 넣는다.

```
sh2358@edu-argos:~/C-education/practice$ ./swap
a = 40
b = 4
```

값이 성공적으로 바뀌었다!

실습

정수 a, b, c의 값을 입력받고, 세 정수 중 가장 큰 정수를

변수 a의 값과 바꾸려고 한다. 해당 함수를 작성하시오 (단 $a \neq b \neq c$)

```c
#include <stdio.h>

void swap_max(int* a, int* b, int* c) {
    /*
        write your code
                        */
}

int main(void) {
    int a, b, c;
    scanf("%d %d %d", &a, &b, &c);
    swap_max(&a, &b, &c);
    printf("%d %d %d\n",a, b, c);
    return 0;
}
```

```
sh2358@edu-argos:~/C-education/practice$ ./practice1
1 3 5
5 3 1
```

```
sh2358@edu-argos:~/C-education/practice$ ./practice1
1 5 3
5 1 3
```

```
sh2358@edu-argos:~/C-education/practice$ ./practice1
5 1 3
5 1 3
```

C교육 4주차

포인터의 활용

- 포인터 연산
- 배열 포인터
- 실습

ARGOS

포인터 연산

변수를 연산하듯이 포인터 변수 또한 연산이 가능하다.

```c
int a = 4;
int* ptr = &a;
printf("%p %p %p\n", ptr, ptr + 1, ptr + 2);
return 0;
```

```
sh2358@edu-argos:~/C-education/practice$ ./pointer
0x7ffd2b55529c 0x7ffd2b5552a0 0x7ffd2b5552a4
```

ptr　　ptr + 1　　ptr + 2

int형 변수의 크기가 4byte라서 ptr + 1의 값이 ptr과 4만큼 차이남.

배열 포인터

다음은 저번주 C 교육 배열에 관한 내용 중 인덱스에 대한 내용이다.

- 각각의 element의 위치를 가리키는 숫자를 index라 한다.
- 배열의 인덱스는 0부터 시작한다.

배열의 원소들이 각각의 메모리에 저장되어 있고,

인덱스가 원소들의 위치를 가리키는 역할을 한다면,

포인터 또한 같은 역할을 할 수 있지 않을까?

배열 포인터

ARGOS

1부터 5까지의 정수를 저장한 배열 arr[5]을 선언하고

배열 arr을 가리키는 포인터 변수 ptr을 선언한다.

```
int arr[5] = {1, 2, 3, 4, 5};
int* ptr = arr;
for(int i=0 ; i<5 ; i++) {
    printf("arr[%d] = %d    ", i, arr[i]);
    printf("*(ptr + %d) = %d\n" , i, *(ptr + i));
}
```

반복문을 이용하여 arr[0] 부터 arr[4]까지의 값을 출력하고,

포인터 연산을 통해 *ptr 부터 *(ptr + 4)까지의 값을 출력한다.

과연 같은 값이 출력될까?

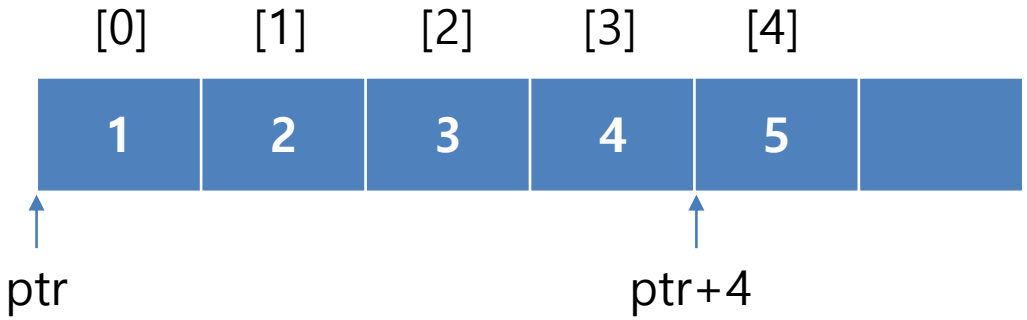
배열 포인터

ARGOS

인덱스가 원소를 가리키듯이, 포인터도 같은 역할을 해준다.

```
sh2358@edu-argos:~/C-education/practice$ ./pointer
arr[0] = 1   *(ptr + 0) = 1
arr[1] = 2   *(ptr + 1) = 2
arr[2] = 3   *(ptr + 2) = 3
arr[3] = 4   *(ptr + 3) = 4
arr[4] = 5   *(ptr + 4) = 5
```

메모리를 통해 좀 더 자세히 보면 아래의 그림과 같다.



배열 포인터


```
int* ptr = arr;
```

포인터 변수에는 주소값이 들어가야 한다. 근데 왜 arr엔 &를 안 붙였나?

배열의 이름은 해당 배열의 시작 주소를 의미한다. 즉, arr == &arr[0]

```
char str[9];
scanf("%s", str);
```

scanf로 문자열을 입력받을 때, &을 붙이지 않는 것도 같은 이유다.

실습

포인터를 써서 문자열을 다뤄보자!

문자열 str을 출력 했을 때 Pointer 까지만 출력되게 하시오.

```c
#include <stdio.h>

int main(void) {
    char str[20] = "Pointer Practice";
    char* ptr = str;
    /*
        write your code use variable ptr
                                        */
    printf("%s\n", str);
    return 0;
}
```

Hint : Null Byte

```
sh2358@edu-argos:~/C-education/practice$ ./practice2
Pointer
```

* 포인터 변수 ptr만 사용하여 str을 변경해보세요.

C교육 4주차

과제

- 문자열 복사(필수)
- 정렬(도전)

ARGOS

과제

문자열 복사(필수)

문자열을 복사하는 함수를 작성하시오. (하다 안되면 구글링 하셔두 됩니다)

```c
#include <stdio.h>

void copy([redacted]) {
    /*
        write your code
                        */
}

int main(void) {
    char str1[100] = "Copy This String";
    char str2[100];
    copy([redacted]);
    printf("%s\n", str2);
    return 0;
}
```

그래도 안되면 질문해주세요

과제

정렬(도전)

ARGOS

배열의 원소를 입력받고, 오름차순으로 정렬하는 함수를 작성하시오

```c
#include <stdio.h>

void sort(int* arr, int length) {
    /*
        write your code
                        */
}

int main(void) {
    int arr[5];
    for(int i=0 ; i<5 ; i++) {
        scanf("%d", arr + i);
    }

    sort(arr, 5);

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

```
sh2358@edu-argos:~/C-education/homework/week4$ ./homework2
5 4 3 2 1
1 2 3 4 5
```

이미 자료구조 등의 학과 수업으로

배열 정렬을 구현 해보신분들은

굳이 안하셔도 될 것 같아요

하다가 안되면 구글링 or 질문

Q & A

Thank You for Listening