구조체와 DATA ALIGNMENT

2학년 전유경

CONTENTS

1. 구조체란?

구조체

C언어의 기본 타입을 가지고 새롭게 정의할 수 있는
사용자 정의 자료형

→ 여러 자료형을 가진 변수들을 하나로 묶어
자료형으로 사용할 수 있도록 정의하는 것

사용 이유

복합된 데이터를 효율적으로 처리할 수 있다.

구조체 정의하기

```
struct student {
  int id;
  char major[15];
  char name[10];
  int avg_grade;
};
```

```
int main() {
  struct student James;

  James.id = 12345678;
  James.major[0] = "정";
  James.major[1] = "보";
  James.major[2] = "보";
  James.major[3] = "호";
  James.major[4] = "학";
  James.major[5] = "과"; //strcpy(James.major, "정보보호학과"); 를 이용하여도 된다.
  strcpy(James.name, "James");
  James.avg_grade = 3;
}
```

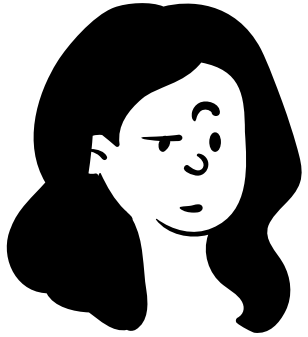
2. 구조체 포인터

```c
#include <stdio.h>

struct Student {
  int id;
  int age;
};

int main() {
  struct Student James;
  struct Student* p; //p는 구조체가 아닌 구조체를 가리키는 포인터

  p = &James;

  (*p).id = 12345678;
  (*p).age = 20;

  printf("James의 id : %d\n", James.id);
  printf("James의 age : %d", James.age);
}
```

구조체 포인터란?

구조체를 가리키는 포인터

2. 의문점 - 구조체 메모리 저장 방식

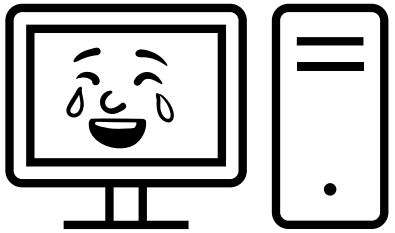


나

구조체 포인터가 갖고 있는 값(*p)과
선언한 구조체 변수의 주소는 서로 같겠지?

```
James 구조체의 주소 : 008FFDAC
James.id의 주소 : 008FFDAC
James.age의 주소 : 008FFDB0
*p의 값 : 00BC614E
```

땡ㅋㅋ!!

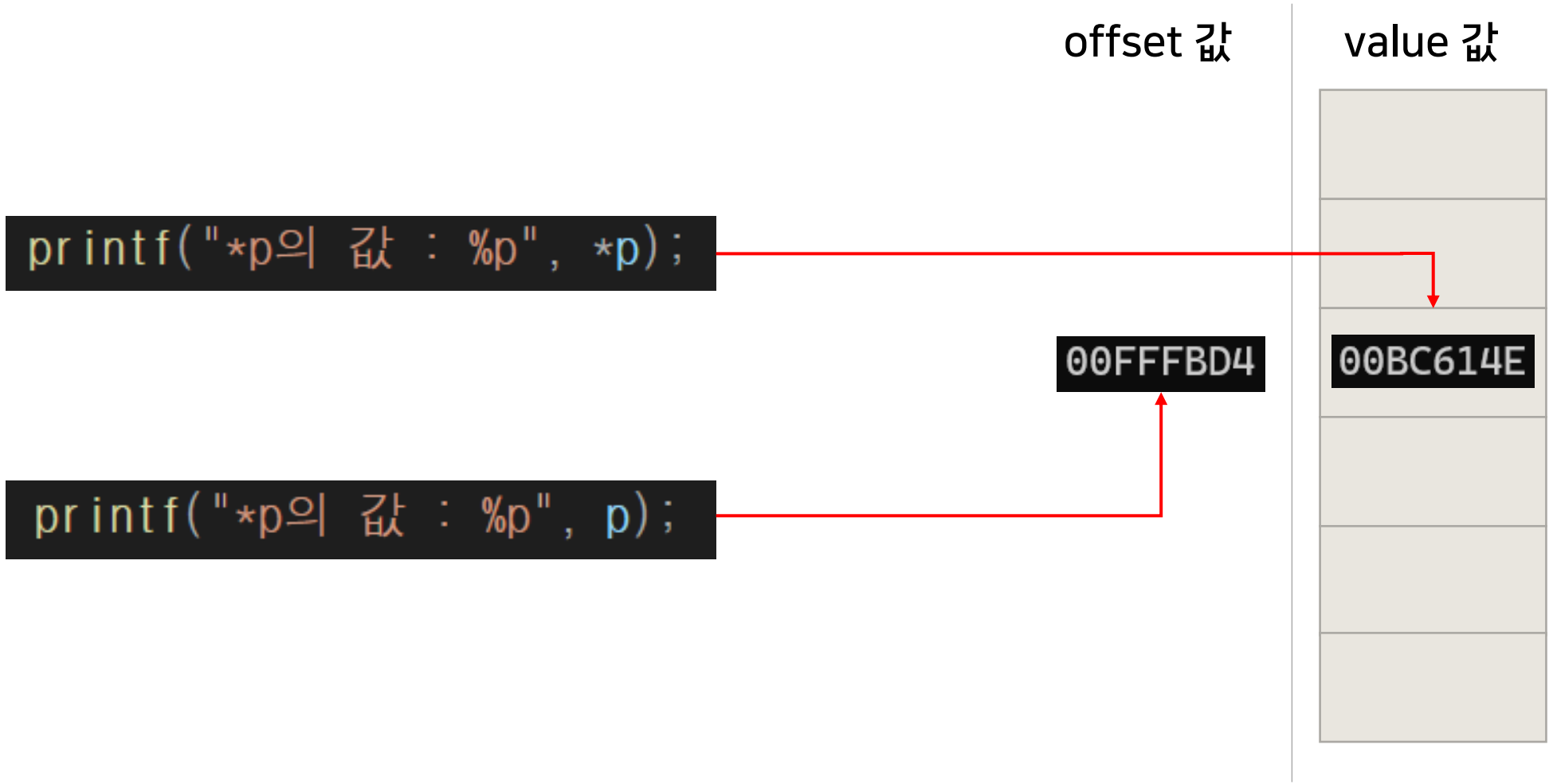


```c
#include <stdio.h>

struct Student {
    int id;
    int age;
};

int main() {
    struct Student James;
    struct Student* p; //p는 구조체가 아닌 구조체를 가리키는 포인터

    p = &James;

    (*p).id = 12345678;
    (*p).age = 20;

    printf("James 구조체 주소값 : %p₩n", &James);
    printf("James의 id 주소값 : %p₩n", &James.id);
    printf("James의 age 주소값 : %p₩n", &James.age);
    printf("*p의 값 : %p", *p);
}
```

```
James 구조체 주소값 : 00DFFA08
James의 id 주소값 : 00DFFA08
James의 age 주소값 : 00DFFA0C
*p의 값 : 00BC614E
```

```
James 구조체 주소값 : 00FFFBD4
James의 id 주소값 : 00FFFBD4
James의 age 주소값 : 00FFFBD8
*p의 값 : 00FFFBD4
```

```
printf("*p의 값 : %p", p);
```


offset 값
value 값
printf("*p의 값 : %p", *p);
printf("*p의 값 : %p", p);
00FFFBD4
00BC614E

2. Data Structure Alignment & Byte Padding

```
3   struct Student {
4       int id;
5       int age;
6       char gender;
7   };
8
9   int main() {
10      struct Student James;
```

```
James.id의 크기 : 4
James.age의 크기 : 4
James.gender의 크기 : 1

James 구조체의 크기 : 12byte
```

4 + 4 + 1 = 12 ?

데이터 정렬(Data Alignment)과 **바이트 패딩** 때문!

Data Structure Alignment 란?

컴파일러가 컴퓨터 메모리에 데이터를 저장할 때, **데이터를 정렬하는 방식**을 나타낸다.

Data Structure Padding 이란?

컴파일러가 **Alignment 규칙에 맞추어 데이터를 할당하기 위한 수단**이다.

Alignment 규칙을 위해 컴파일러는 구조체를 컴파일하면서 **메모리 주소 사이에 바이트를 추가**하여

메모리의 데이터를 정렬하는 것이다.

컴파일러가 패딩 데이터를 추가하는 규칙

1. 구조체의 크기는 구조체를 구성하는 멤버의 자료형 중, 가장 크기가 큰 자료형의 배수가 된다.
2. 구조체를 구성하는 자료형 중, 가장 크기가 큰 자료형으로 데이터를 정렬한다.

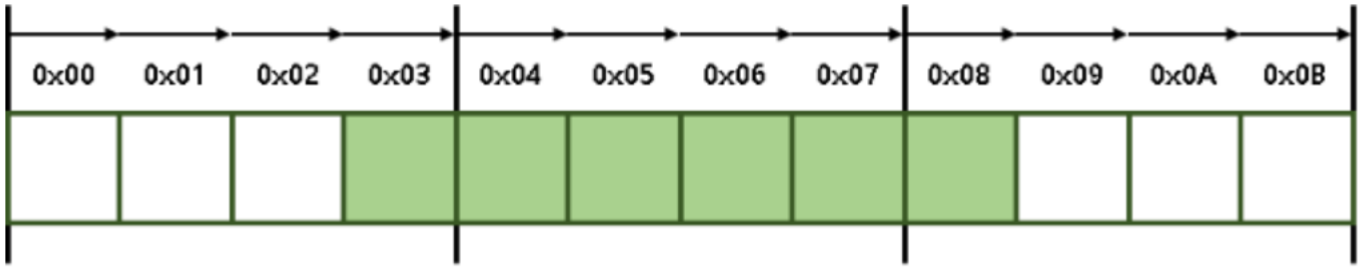
```
3   struct Student {
4       int id;
5       int age;
6       char gender;
7   };
8
9   int main() {
10      struct Student James;
```

4byte (int)	4byte (int)	1byte (char)	3byte (padding)

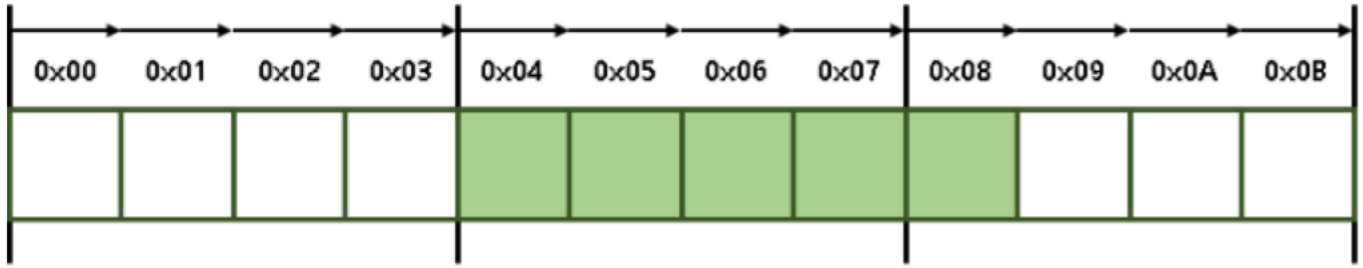
총 12byte

바이트 패딩을 하는 이유

이렇게 바이트 패딩을 해주는 이유는 멤버 변수를 메모리에서 CPU로 읽어 들일 때,
한 번에 읽을 수 있도록 컴파일러가 레지스터의 블록에 맞추어 바이트 패딩을 해 최적화를 시키기 위함이다.
이를 통해 CPU의 연산 횟수를 줄여 성능을 향상시킬 수 있다.



데이터 정렬을 하지 않는 경우
-> 메모리에 3번 접근



데이터 정렬을 한 경우
-> 메모리에 2번 접근

아래의 경우에는 구조체의 총 크기가 얼마일까?

```c
#include <stdio.h>

struct Struct1 {
  char a;
  char b;
  int c;
  double d;
};

int main() {
  struct Struct1 ex;
```

8(double) x 4 = 32byte?

아래의 경우에는 구조체의 총 크기가 얼마일까?

```c
#include <stdio.h>

struct Struct1 {
  char a;
  char b;
  int c;
  double d;
};

int main() {
  struct Struct1 ex;
```

```c
printf("구조체의 크기 : %d\n\n", sizeof(ex));
```

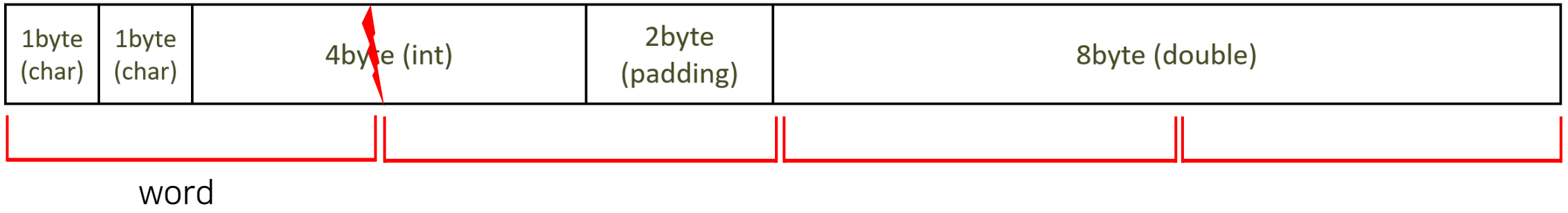
```
구조체의 크기 : 16
```

```c
#include <stdio.h>

struct Struct1 {
  char a;
  char b;
  int c;
  double d;
};

int main() {
  struct Struct1 ex;
```

```
a의 크기 : 1
b의 크기 : 1
c의 크기 : 4
d의 크기 : 8

a의 주소 : 00D3F820
b의 주소 : 00D3F821
c의 주소 : 00D3F824
d의 주소 : 00D3F828
```

1byte (char)	1byte (char)	2byte (padding)	4byte (int)	8byte (double)

왜 이렇게 패딩이 되지 않았을까?



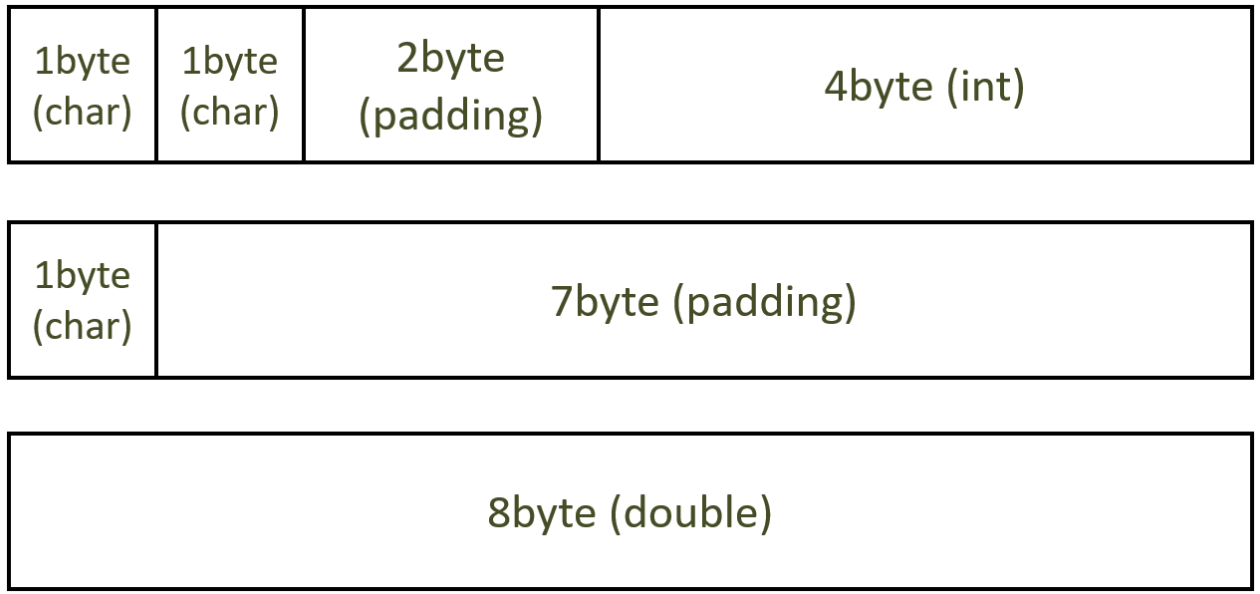
프로세서의 메모리 하위 시스템이 메모리에 접근하는 것을 프로세서의 WORD 크기에 대해서 분할하고 정렬하는 것으로 블록을 나누고 CPU가 바로 연산하도록 하기 위함이다.

위에서는 32비트의 플랫폼을 이용해 컴파일했기 때문에 프로세서의 WORD 크기가 4바이트가 된다.

따라서 4바이트 단위로 분할 및 정렬이 되는 것이다.

```c
#include <stdio.h>

struct Struct1 {
  char a;
  char b;
  int c;
  char d;
  double e;
};

int main() {
  struct Struct1 ex;
```

```
구조체의 크기 : 24

a의 크기 : 1
b의 크기 : 1
c의 크기 : 4
d의 크기 : 1
e의 크기 : 8

a의 주소 : 10484580
b의 주소 : 10484581
c의 주소 : 10484584
d의 주소 : 10484588
e의 주소 : 10484596
```



WORD(= 4byte)로 분할하는데,
왜 c의 옆에 3byte 패딩을 하지 않았을까?

→ 가장 큰 자료형의 사이즈에 맞추어야 하기 때문에 전체 크기는 8의 배수가 되어야 한다.

만약 7byte의 패딩이 아닌 3byte 패딩을 해주었다면 구조체의 크기가 20byte가 되어 8의 배수가 아니게 된다.

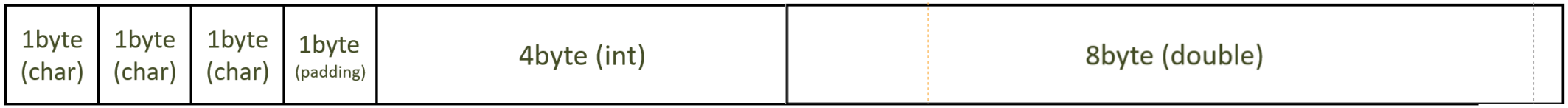
총 9byte의 패딩 사용 → 데이터 누수 ↑

데이터 누수 줄이기

1. 변수 선언 순서를 고려한다.

변수를 선언할 때, 가장 작은 자료형부터 선언하면 메모리의 낭비를 줄일 수 있다.

```
struct Struct1 {
  char a;
  char b;
  char d;
  int c;
  double e;
};

int main() {
  struct Struct1 ex;
```



데이터 누수 줄이기

2. #pragma pack(n) 이용하기

#pragma pack(n)은 pack하는 기준을 워드가 아닌 n으로 수정하라는 뜻이다.

```c
#include <stdio.h>

#pragma pack(1)
struct Struct1 {
  char a;
  char b;
  int c;
  char d;
  double e;
};

int main() {
  struct Struct1 ex;
```

```
구조체의 크기 : 15

a의 크기 : 1
b의 크기 : 1
c의 크기 : 4
d의 크기 : 1
e의 크기 : 8

a의 주소 : 20183344
b의 주소 : 20183345
c의 주소 : 20183346
d의 주소 : 20183350
e의 주소 : 20183351
```

1 + 1 + 4 + 1 + 8 = 15

따라서 패딩이 하나도 되지 않은 것을 확인할 수 있다.

감사합니다