



c – HW₃

임베디드스쿨1기

Lv1과정

2020. 08. 06

강경수

1. 배열

- 1) 여러 개의 값을 동시에 핸들링할때 사용.
- 2) `int array_a[3];` 옆과 같이 배열 선언시 `int`형 변수 세개를 한 번에 선언.
- 3) `int array_a[i];` >> compile error 배열은 변수로 크기를 선언할 수 없다.
하기 위해선 동적할당 필요.
- 4) `int arr[3] = {1};` equal `int arr[3] = {1,0,0};`
- 5) `int`형 배열의 각각의 포인터 값은 `int`형 변수가 줄줄이 나열 한 것과 같다.
즉 포인터로 효과적으로 배열을 컨트롤 할 수 있다.
- 6) 배열의 이름 즉 `array_a`는 `&array[0]`과 같다. 하지만 포인터변수는 아니다.
값만 가질 뿐이다.

2. 포인터

1) RAM 메모리와 포인터크기의 관계

기준 : 32bit cpu

포인터 변수 크기 : 4byte(32bit)

데이터 기본 크기 : 1byte(8bit) ← ASCII코드가 1byte이기 때문에?

정리1: 32bit cpu란, 레지스터가 한번에 처리하는 데이터 단위가 32bit이기 때문에. 32개의 신호선을 한번의 CLOCK에 처리한다고 생각 할 수 있음.

정리2: 32bit가 연산가능한 수의 범위. 2^{32} = 약 42억 표현가능한 경우의 수 42억가지.

정리3: 컴퓨터가 42억개 처리가능한 수만 가지고 연산하면.. 공간이 부족함 따라서 이 42억개 각각을 하나의 주소로 생각하고 그 안에 기본처리단위로 1byte씩 할당함

→ 4GB메모리 용량.

즉 방 개수(주소)가 1000개(42억개) 이고 그 방의 평수가 10평(1byte)이라고 생각할 수 있음

따라서 총면적은 $1000 \times 10\text{평} = 10000\text{평}$ (4GB)

2. 포인터

2) Call by reference vs Call by Value?

혼란을 줄 수 있다. 포인터변수도 결국에는 Value일 뿐이며 이 메모리 주소도 매핑된 값(Value) 일 뿐이다.

포인터 변수의 값도 변수 값처럼 쌓인다→포인터 변수 4byte의 크기 할당)

3) 그렇다면 왜 포인터의 Data type 정의가 필요한가?

int* a; char*b 등등 → 해당 주소에서 int형(4byte) 및 char(1byte)만큼 값을 읽으라는 의미.

4) const int* p 와 int* const p의 차이점

const int* p 의 경우 *pa = b 불가능하다. (즉 값변경 불가)

int* const p 의 경우 pa = &b 불가능하다. (즉 주소값 변경 불가)

2. 포인터

5) Pointer of Pointer

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
int a;
```

```
int *ap;
```

```
int **app;
```

```
a = 10;
```

```
ap = &a;
```

```
app = &ap;
```

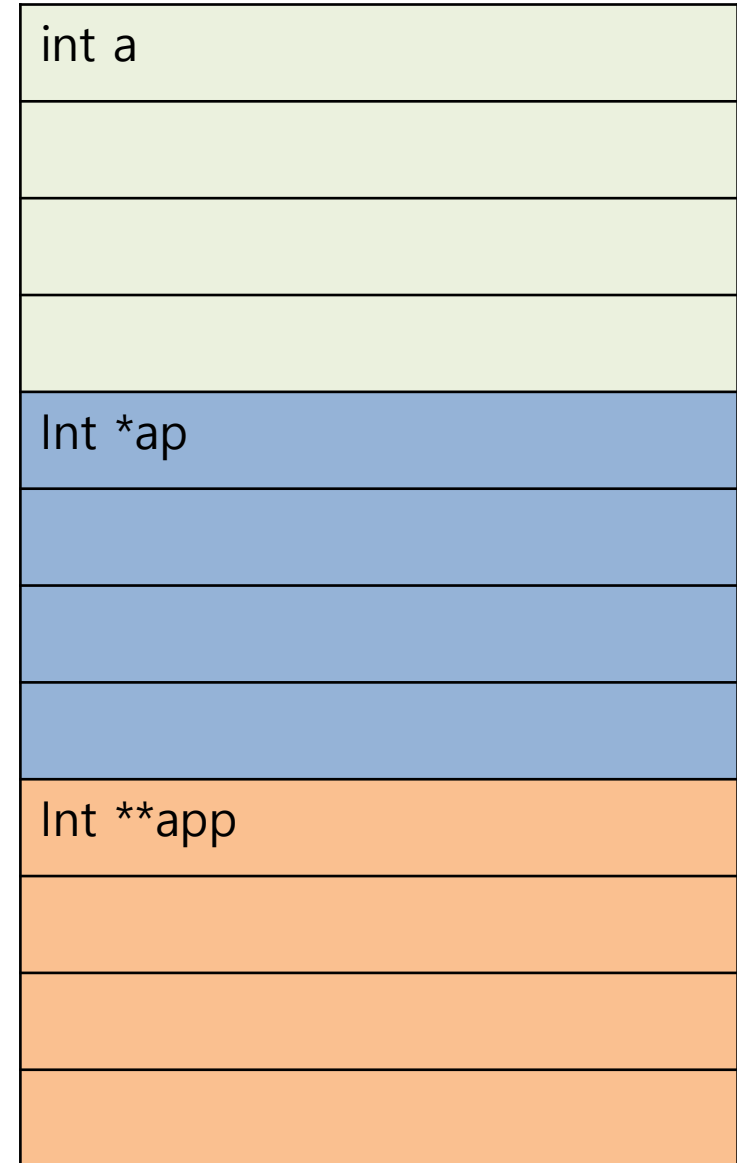
```
printf("%d\n", a); //10출력
```

```
printf("%d\n", *ap); //10출력
```

```
printf("%d\n", *(*app));
```

```
}
```

포인터의 포인터라는 개념이 어렵게 느껴질 수 있지만 결국에 포인터변수의 주소값도 값, 즉 Value이기 때문에 단순히 메모리에 쌓여나가는 구조이며. 0xF110BAC1 이런 주소도 값일 뿐이다. →크기는 4byte




3. HW1

해당 main 문을 보고 Memory를 완성하세요

```
int main(void)
{
    char c = 'a';
    int n = 7;
    double d = 3.14;
}
```

지역변수는 stack영역에서 낮은값에서 높은값 으로
쌓아져 나간다. 그리고 메모리주소값 1증가당
1byte가 할당되어 증가하므로

char c → 1byte 증가
int n → 4byte 증가
double d → 8byte 증가



0x1000	char c = 'a' ;
0x1001	int n =7;
0x1002	
0x1003	
0x1004	
0x1005	double d = 3.14;
0x1006	
0x1007	
0x1008	
0x1009	
0x1010	
0x1011	
0x1012	

3. HW2

해당 main 문을 보고 Memory를 완성하세요

```
int main(void)
{
    int a[2][3] = {{0,1,2},{3,4,5}};
}
```

위 와 같이 2차원 배열 a를 선언 및 정의하면.
배열 하나하나의 크기는 4byte 즉 메모리 영역에서 4증가
따라서 옆 과같이 할당 된다.

따라서 2차원 배열에 접근하기 위해서는 포인터 선언시
하나씩 증가한다.

0x1000	{0}
0x1004	{1}
0x1008	{2}
0x1012	{3}
0x1016	{4}
0x1020	{5}

3. HW3

아래에 해당하는 값을 넣으시오

```
int main(void)
{
    int a[2][3] = {{0,1,2},{3,4,5}};
}
```

- 1) &a = &(a[0][0]) 과 같으며 메모리 영역은 0x1000
- 2) &a+1 = &(a[0][1])과 같으며 메모리 영역은 0x1004 (4byte증가)
- 3) a+1 = &(a[1][0])과 같으며 메모리 영역은 0x1016 (16byte증가)
- 4) 5) 6) a라는 변수가 int a[2][3] 의 시작주소를 의미하긴 하지만
이는 값일뿐 포인터 변수가 아니기 때문에 예시는
사용 할 수 없는 값임

1) &a 0x1000
2) &a+1 0x1004
3) a+1 0x1016