# 펭귄 스터디 내용09

작성자 : 류성수
작성일자 : 24.07.02

url : https://www.youtube.com/watch?v=po80VB5prIY&list=PLz—ENLG_8TMdMJIwyqDIpcEOysvNoonf&index=10

## ● 포인터

```c
#include <stdio.h>

#define MAX 10000

void main()
{
        int array[MAX];

        for(int i = 0; i<MAX; i++)
        {
                array[i] = 2*i;
                printf("array[%d] = %d\n",i,array[i]);
        }

        int a = array;
        printf("a = %d\n",a);
        return;
}
```

실행파일에서 각 변수의 주소값은 컴파일 단계에서 결정되는 것이 아닌, 런타임에서 결정되는 것이기에, 항상 실행할 때마다, 변수의 주소값이 달라진다는 것을 확인하였다.

| 첫 번째 실행 | 두 번째 실행 |
| --- | --- |
| array[9992] = 19984<br>array[9993] = 19986<br>array[9994] = 19988<br>array[9995] = 19990<br>array[9996] = 19992<br>array[9997] = 19994<br>array[9998] = 19996<br>array[9999] = 19998<br>a = -1686896640 | array[9992] = 19984<br>array[9993] = 19986<br>array[9994] = 19988<br>array[9995] = 19990<br>array[9996] = 19992<br>array[9997] = 19994<br>array[9998] = 19996<br>array[9999] = 19998<br>a = -84722512 |

## ● 다중포인터

```c
#include <stdio.h>

void main()
{
        int a;
        int *pa;
        int **ppa;

        a = 3;
        printf("a = %x\n",a);
        printf("&a = %x\n",&a);

        pa = &a;
        printf("pa = %x\n",pa);
        printf("*pa = %x\n",*pa);
        printf("&pa = %x\n",&pa);

        ppa = &pa;
        printf("&ppa = %x\n", &ppa);
        printf("ppa = %x\n", ppa);
        printf("*ppa = %x\n",*ppa);
        printf("**ppa = %x\n",**ppa);

        // printf("***ppa = %x\n",***ppa)
        // a의 3도 주소로 생각하면서 접근하려 하면, 리눅스가 프로세스를 죽임
}
```

이중포인터를 실행해보면서, *연산자에 따라서 CPU가 몇 번의 횟수로 주소를 참조하는지를 확인하였다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy09# ./pointer.out
a = 3
&a = f84d4874
pa = f84d4874
*pa = 3
&pa = f84d4878
&ppa = f84d4880
ppa = f84d4878
*ppa = f84d4874
**ppa = 3
```

## ● 양날의 검

#### 1. 코에 걸면 코걸이, 귀에 걸면 귀걸이

기본적으로 메모리의 값은 의미가 정해지지 않은 비트열이다.
이런 부분덕분에, 각 메모리의 값은 때에 따라서, HW를 제어하는 핀이 될 수도 있고, 상태를 관리하는 flag가 될 수도 있다.
또한, 그 자체가 메모리 주소값이냐, 코드의 명령어냐, 변수냐 등등 해석할 수 있는 방법이 여러 가지이다.

그렇기에, 해당 비트열을 해석할 수 있게 사용자가 직접 명령어를 통해서, 혹은 실행파일의 메모리 영역에 따라서 이 비트뭉텅이를 CPU에게 어떻게 해석할건지를 알게 한다.

#### 2. 자유에는 책임이 따른다.

여기서 중요한게, 방금 위의 코드처럼 ***ppa 로 잘못 명령을 내리게 되면, (a = 3 이 변수값을 주소값으로 읽게함) 오류가 발생하게 된다는 것이다.
이 부분은 OS 자체가 보호하거나 그 자체가 저장된 곳일 수도 있기 때문에, 이렇게 잘못접근되는 명령은 OS의 철퇴를 맞고 프로그램이 강제종료 된다.

이러한 부분이 C/C++ 의 디버깅에서 매우 어려운 요소 중에 하나이다.
해당 주소 값은 런타임에서 결정되기에, 주소연산을 잘못하거나, 방금처럼 옳지못한 주소접근을 하게되면, 중단점을 걸고 한줄한줄 찾아야 하는 번거로움이 있다.