
# C Programming – 4

임베디드스쿨2기

Lv2과정

2021. 04. 06

박태인

# 1. 모든 것은 포인터다

## 1) 포인터 통일

해당 내용을 통해서 결국 C언어의 모든 것이
포인터라는 것을 확인 할 수 있을 것이다.
이중 포인터, 삼중 포인터, 배열, 다중 배열,
포인터 배열, 함수 포인터를 별개로 볼 필요가 없다.
다만 이것을 진행하기 위해서는 몇개지 개념이 필요 하다.

**1 . 메모리 계층 구조**
**2. 스택(Stack)은 아래로 자란다.**
**3. GP Register에 대한 명확한 개념과 각각의 용도**

## 2) 디버깅 명령어

Info registers : 실제 HW 레지스터 정보를 확인 할 수 있고, **여기서는 펌웨어 제어와 관련된 레지스터 정보는 보여 주지 않는다.**
우리가 이 내용을 진행하면서 주의를 둘 부분은 아래와 같다.

rsp, rbp, rip, rax, rcx 정도에 해당한다.

- rsp : 현재 스택의 최상위
- rbp : 현재 스택의 기준점
- rip : 다음에 실행 할 instruction의 주소 값을 가르킴
- rax : 무조건적으로 함수의 리턴값이 저장되며 연산용으로도 활용 가능
- rcx : 보편적으로 for 루프의 카운트에 활용이 되며 연산용으로도 활용 가능

- si : 어셈블리 명령어 기준으로 한 줄씩 실행한다.
- p/x : 16진수로 특정 결과를 출력한다.
- x : 메모리의 내용을 살펴본다.

# 1. 디버깅 과정(test_func.c) - (1)

```c
#include <stdio.h>

int my_func(int num)
{
        return num >> 1;
}

int main(void)
{
        int num = 3, res;

        res = my_func(num);

        printf("res = %d\n", res);

        return 0;
}
```

```
res = 1
```

```
04$ gcc -g -o test_func test_func.c
04$ gdb test_func
```

```
[Inferior 1 (proces
(gdb) b main
Breakpoint 1 at 0x5
(gdb) r
Starting program: /

Breakpoint 1, main
(gdb) disas
```

0000 0010 = 3
>> 0000 0001 = 1

오른쪽 쉬프트 연산하여 위와 같이 **num 3**의 값이

**Res = 1 의 값으로**

출력 된다.

지금부터 이 소스 코드를 이용해 gdb 를 분석한다!

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
   0x0000555555555167 <+12>:    movl   $0x3,-0x8(%rbp)
   0x000055555555516e <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555171 <+22>:    mov    %eax,%edi
   0x0000555555555173 <+24>:    callq  0x555555555149 <my_func>
   0x0000555555555178 <+29>:    mov    %eax,-0x4(%rbp)
   0x000055555555517b <+32>:    mov    -0x4(%rbp),%eax
   0x000055555555517e <+35>:    mov    %eax,%esi
   0x0000555555555180 <+37>:    lea    0xe7d(%rip),%rdi        # 0x555555556004
   0x0000555555555187 <+44>:    mov    $0x0,%eax
   0x000055555555518c <+49>:    callq  0x555555555050 <printf@plt>
   0x0000555555555191 <+54>:    mov    $0x0,%eax
   0x0000555555555196 <+59>:    leaveq
   0x0000555555555197 <+60>:    retq
```