**1포인터 크기 개념**

**Size Of Pointer포인터의 크기**

** 32비트 시스템의 포인터 크기: 4바이트**

** 64비트 시스템의 포인터 크기: 8바이트**

**2 포인터 크기 원리**

**포인터는 메모리 위치를 가리키기 위한 주소를 담는 변수다. 만약 32비트 시스템이라면 CPU의 메모리 접근을 위한 주소가 32비트 만큼 가능하기 때문에 포인터도 32비트(4바이트)가 된다. 반면, 64비트 시스템이라면 64비트(8바이트) 만큼 가능하기 때문에 포인터도 64비트(8바이트)가 된다.**

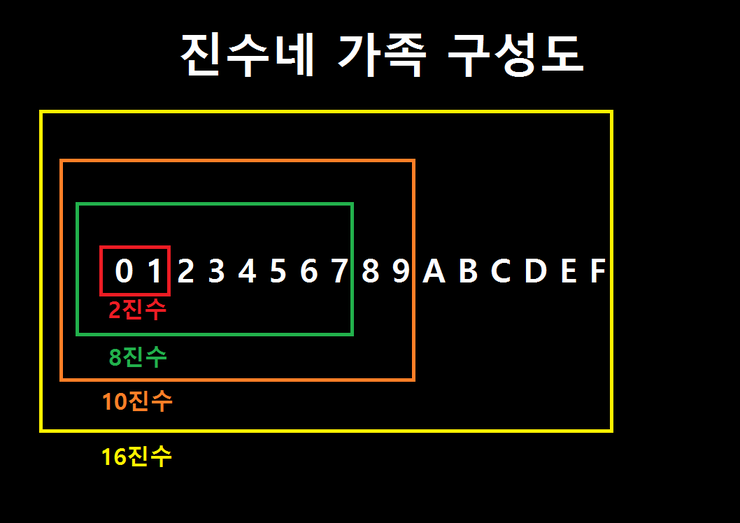
**3 예시(64비트 시스템)**

**#include<stdio.h>voidmain(){char\*pc;int\*pi;printf("%ld",sizeof(pc));// 8printf("%ld",sizeof(pi));// 8}**

**→64비트 시스템이기 때문에 char와 int 타입의 포인터 사이즈가 각각 8바이트(64비트)임**

**\* 포인터의 크기는 ?   
   
8 비트 시스템의 경우 1 byte   
16 비트는 2 byte   
32 비트는 4 byte   
64 비트는 8 byte   
   
- 왜 그럴까 ?   
   
컴퓨터의 산술 연산이 ALU 에 의존적이기 때문이다.   
ALU 의 연산은 범용 레지스터에 종속적이고   
컴퓨터가 64 비트라는 의미는 이들이 64 비트로 구성되었음을 의미한다.   
   
변수의 정의는 메모리에 정보를 저장하는 공간이였다.   
포인터의 정의는 메모리에 주소를 저장하는 공간이다.   
그렇다면 64 비트로 표현할 수 있는 최대값 또한 저장할 수 있어야한다.   
포인터의 크기가 작다면 이 주소를 표현할 방법이 없기 때문에   
최대치인 64 비트(8 byte) 가 포인터의 크기가 된 것이다.**

**1진범의 변환 개념**



**그림에서 보이는거와 같이 2진법,10진법,16진법에 대해 알아보겠습니다.**

**2진법:0,1만 사용 가능**

**10진법:0~9까지만 표현이 가능하다.**

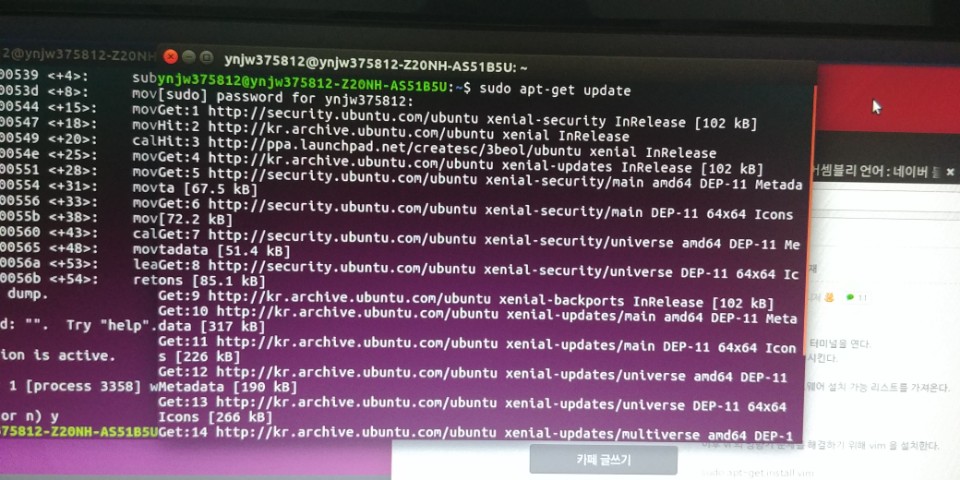
**16진법:0~9까지는 숫자로 표현이 되지만 10부터는 알파벳 a부터 시작해 표현이 되고 최대 15(f)까지만 표현이 가능하다.**

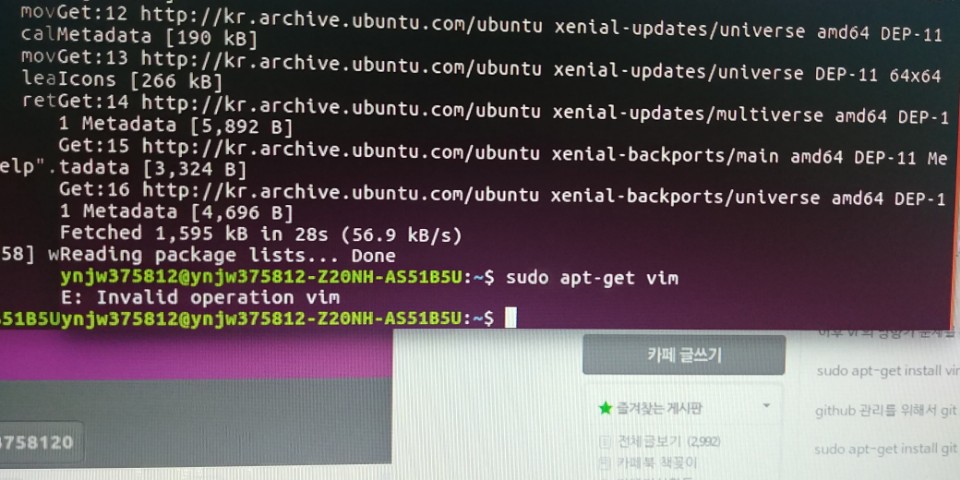
**1. <2진법과 16진법 간의 변환>  
2진수를 4비트 단위로 분할하고,각각의 4비트 들을 10진수로 변환한다. 이것을 다시 16진수로 변환한다.  
예)  
 (0011 0010 1111 1000)-->(3 2 15 8)--->(32F8)  
  
2.<16진법과 2진법 간의 변환>  
16진수에서 10진수로 변환하고 다시 2진수로 변화한다.  
예)  
 (C 4 D 2)---->(12 4 13 2)---->(1100 0100 1101 0010)**

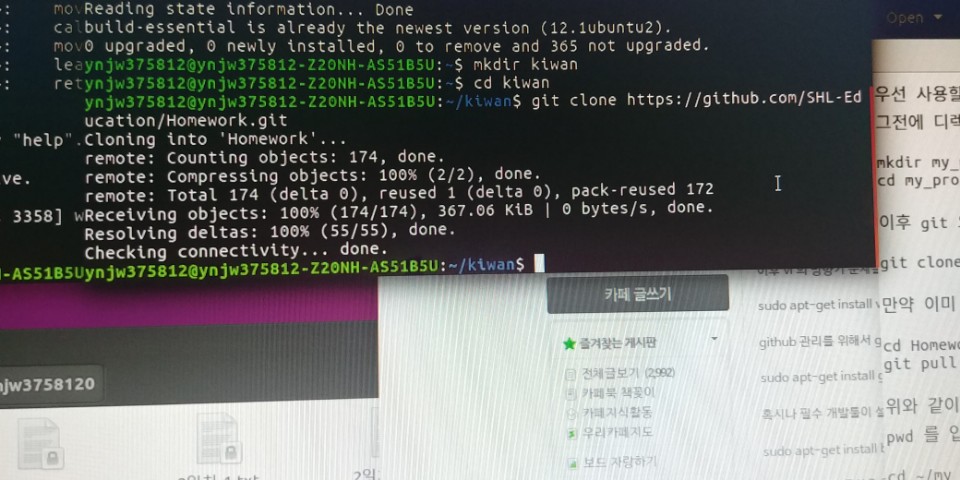
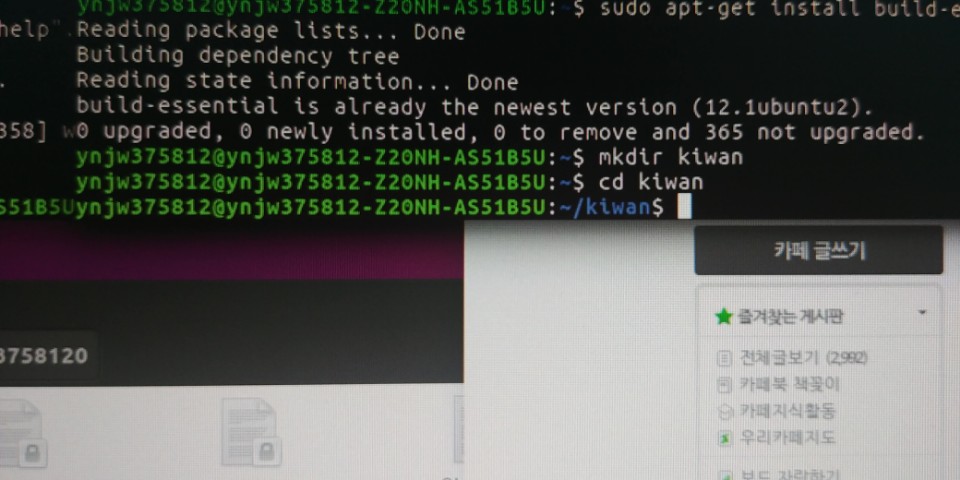
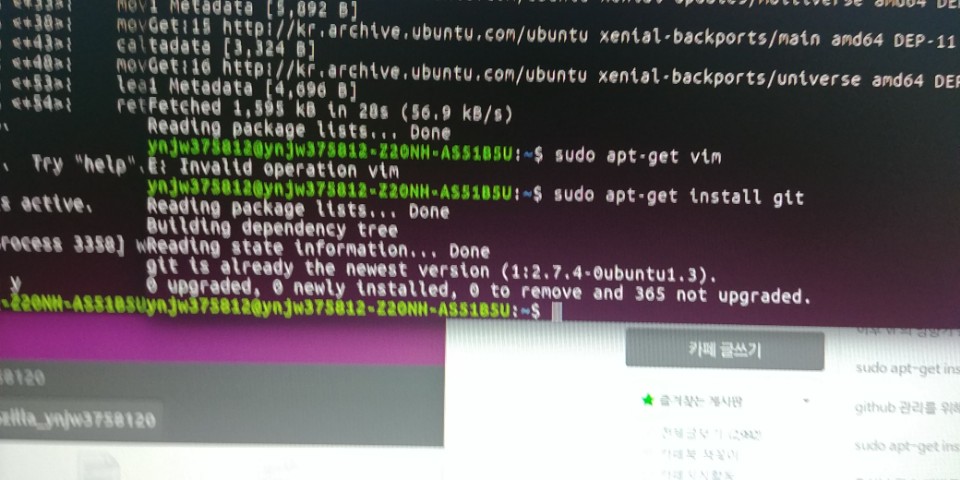
**즉 16진수에서 2진법이나 2진법에서 16진법의 변환은 무조건 10진수의 방법을 거치고 간다는것이 핵심이다. 이 과정에서**

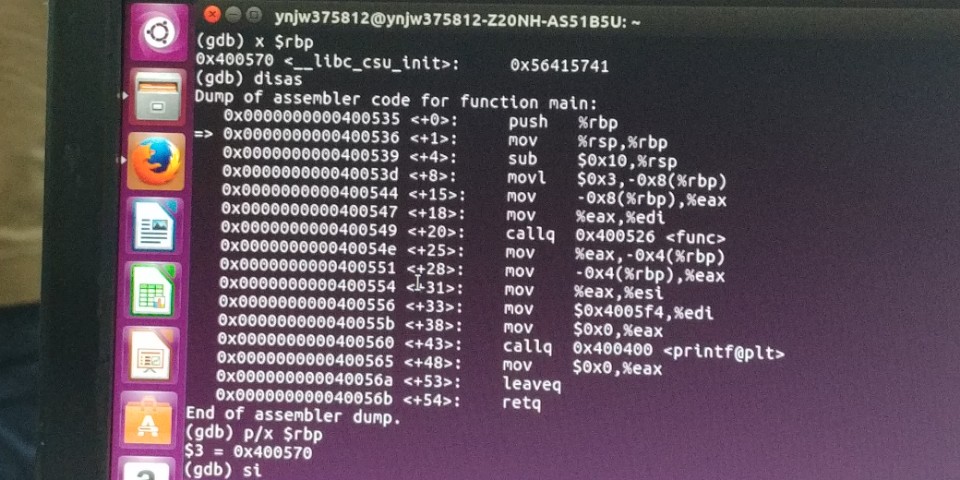
**64 32 16 8 4 2 1 의 순서를 아는것이 핵심이다.**

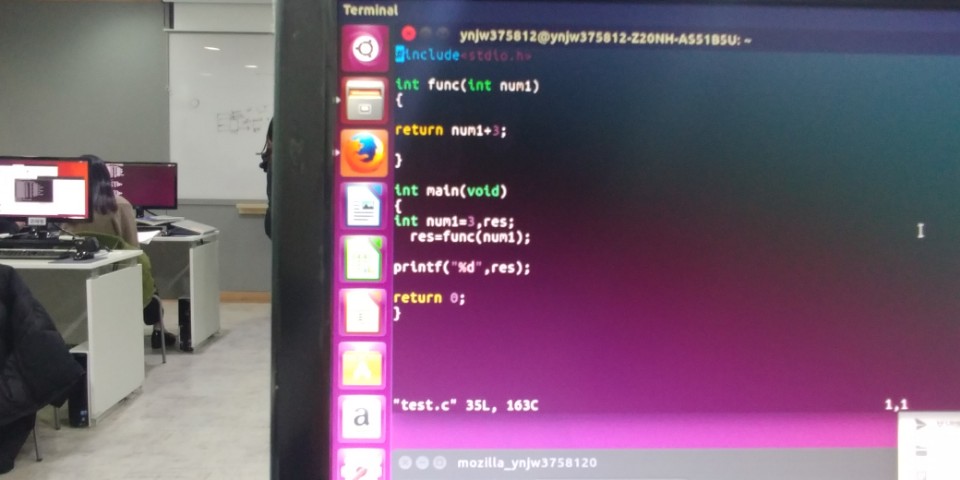
**어제 오늘의 실습현황**



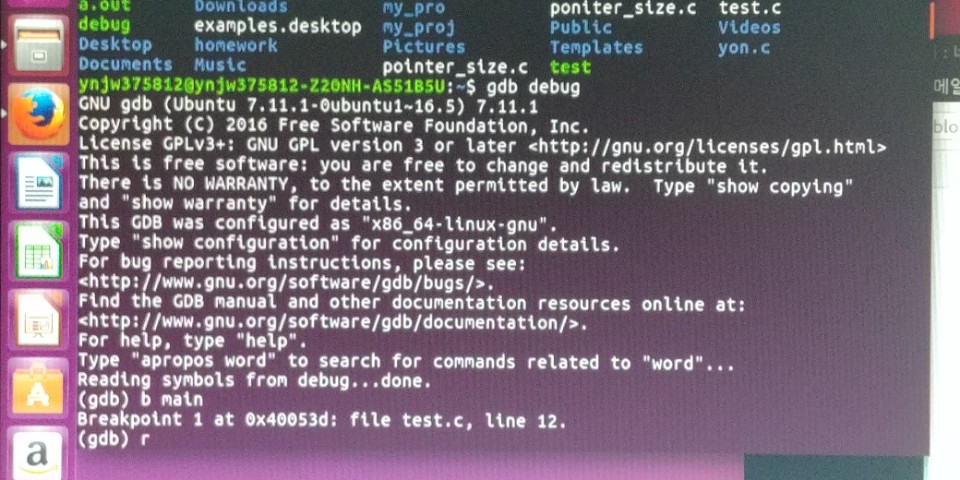




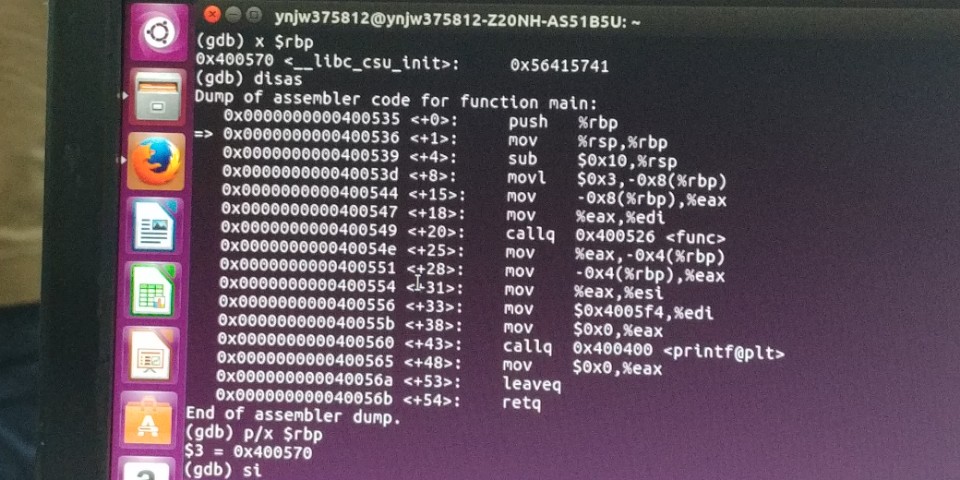
 <Break point>



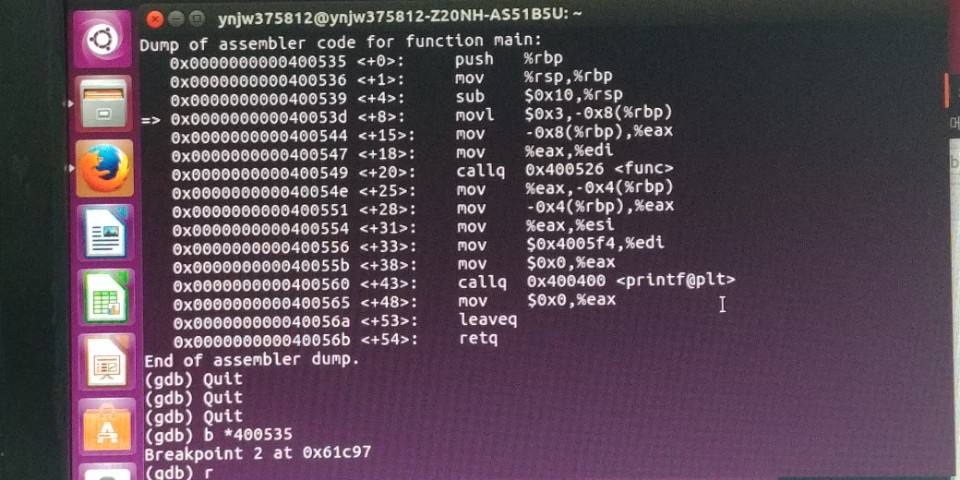
<c언어 #Include<stdio,h>



**<gdb debug>**



**<push 동작>**



**<sub(빼기 연산자) 동작>**

**위에 그림들을 보면 c언어 표면에서 볼수없던 기계어(assembly )를 나타낸것이다. 일단 디버깅을하고 break point 을 걸어주는것은 main에서 끝난다는것을 의미하며 처음에 push %rbp를 통해 rbp를 메모리안에 아래쪽으로 넣고 mov명령어를 이용해 %rsp(최상위)를 %rbp(기준점)에 넣어주면 rsp는 바이트 만큼 주소가 내려오고 sub(빼기)명어를 이용해 0x10의 주소만큼 rsp를 내려주는데 여기서 0x10은 16진수 이므로 -16비트에서 -8비트 총 24비트의 주소값이 내려가고 movl을통해 rbp의 주소를 8바이트만큼 내려보내고 0x3의 정수를 넣는다 0x3은 한마디로 저장장치라고 할 수 있으며 cpu에 있는것이다.즉 메모리와는 별개하는것이다.그리고 rbp의 주소를 내려보내면서 다른주소에 0x3을 넣어주고 callq(jump 개념)명령어를 이용해 main에있던 num1=3의 값을 int func(int num1)에 점프시켜 보내는것이다.여기서 알아야 될것은 main num1과 int func의 num1다른 놈이라는것이라는것 알아야한다. 전역변수와 지역변수 차이라고 할 수 있다.**

**그 이후 연산자에서 num1+3에 해당하는 0x4(%rbp)의 명령어로 return이 발생하고 그걸다시 main함수로 불러오는 어셈블리의 코드이다.이부분은 가볍게 넘어가는것이 괜찬을거라 생각한다.**

**$0x0,%eax의 의미는 func1의 있던 num1의 값을 0으로 만드는것이고 그 다음 callq를 이용해 printf로 표현을 한다.그리고 $0x0,%eax는 return 0의 해당하는 어셈블리 명령어 이다.**