Xilinx Zynq FPGA, TI DSP, MCU기반의 프로그래밍 및 회로 설계 전문가 과정

강사 - Innov (이상훈)
gcccompil3r@gmail.com
학생 - 이유성
dbtjd1102@naver.com

복습 6일차

*typedef
자신이 필요한 자료형을 정의해서 쓸 수 있다.
말 그대로 새로운 자료형(type)을 정의(define)하는 것이다.
데이터 타입을 바꿈 int -> INT
int* -> PINT; (의미는 같다 외형만 다르다)

```
#include <stdio.h>
typedef int INT;
typedef int * PINT;

 int main(void)
{
        INT num= 3;
        PINT ptr = &num;
        printf("num = %d\n", *ptr);
        return 0;
}
```

num = 3

```
 #include<stdio.h>

 typedef int INT[5];

 int main (void){

        int i;

        INT arr = { 1,2,3,4,5};

        for ( i = 0; i < 5 ; i++)

                printf("arr[%d] = %d\n" , i , arr[i]);
```

```
                return 0 ;

}
```

int 가 배열처럼 쓰고있다

arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5

malloc()은 무엇을 하는가? 메모리 어드케이션의 약자

memory 구조상 heap에 data를 할당함,
(어떤 상황,시점에서 얼마나 더 들어 올지 모르니까)
data가 계속해서 들어올 경우
얼만큼의 data가 들어오는지 알 수 없음,
들어올 때마다 동적으로 할당할 필요성이 있음

free()function
free() 은 무엇을 하는가

Memory 구조상 heap에 data를 할당 해제함
malloc()의 반대 역할을 수행함(malloc의 반대)

```
#include <stdio.h>

int main(void)
{

        char *str_ptr = (char *)malloc(sizeof(char) *20);

        printf("Input String :");
```

```
        scanf("%s", str_ptr);

        if(str_ptr != NULL)

        printf("string = %s\n",str_ptr);

        free(str_ptr);

        return 0;

        }
```

calloc

(어떤타입을 몇개)

```
 #include<stdio.h>

int main(void){

        int*num_ptr = (int *)calloc(2, sizeof(int));

        printf("Input Integer : ");
        scanf("%d%d", &num_ptr[0], &num_ptr[1]);

        if(num_ptr !=NULL)
                printf("Integer = %d, %d\n", num_ptr [0],num_ptr[1]);

                free(num_ptr);

                return 0;

}
```

구조체는 왜 사용하는가 ?

내가 어떠한 데이터 타입을 만드는것, , 구조체도 주소다 구조체 = 커스텀데이터타입

(내가만들고싶은대로 만들수잇다)

자료를 처리하다보니 하나로 묶어야 편함

문자열과 숫자를 한번에 묶어서 관리하고 싶을때,,등,,

어떤 정보와 어떤다른정보가 연결되어있는지,,,

배열은 타입이 같은애들 여러애들 다룰때 ,,,,,,,,,,

```
#include<stdio.h>

struct pos{
        double x_pos;
        double y_pos;

        };

 int main(void){
double num ;
struct pos position;

        num = 1.2;
        position.x_pos = 3.3;
        position.y_pos = 7.7;

        printf("sizeof(position) = %d\n", sizeof(position));
        printf("%lf\n", position.x_pos);
        printf("%lf\n", position.y_pos);

        return 0;

}
```

구조체 안에 구조체

```
 #include<stdio.h>

typedef struct __id_card{
        char name[30];
        char id[15];
        unsigned int age;
 }id_card;


typedef struct __city{

        id_card card;


        char city [30];

        } city;

 int main(void){

        int i;
        city info = {
                {"Marth Kim" , "800903 -1012589", 34 }, "Seoul"
                };

                printf("city = %s, name = %s , id = %s , age = %d\n",
                info.city, info.card.name, info.card.id,info.card.age);


        return 0;
 }
```

city = Seoul, name = Marth Kim , id = 800903 -1012589 , age = 34

```
#include<stdio.h>

 typedef enum __packet{

ATTACK,
DEFENCE,
 HOLD,
 STOP,
 SKILL,
REBIRTH,
DEATH =44,
KILL,
ASSIST
}packet;

iint main(void){

        packet packet;
        for(packet = ATTACK ; packet <=REBIRTH; packet++)
                printf("enum num = %d\n", packet);
        for(packet = DEATH; packet <=ASSIST;packet++)
                printf("enum num = %d\n", packet);

                return 0;
                }
```

```
enum num = 0
enum num = 1
enum num = 2
enum num = 3
enum num = 4
enum num = 5
enum num = 44
enum num = 45
enum num = 46
```

enum 은 통신장비에 들어가면 많이 사용,,

enum은 #define 대신 편하게,,

리눅스에 흔히 나오는,
void(*signal(int signum, void(*handler)(int)))(nit);
프로토타입이란?

리턴, 함수명,인자에 대한 기술서
그렇다면 위 함수에 대한 프로토타입은 뭘까?

이전에 배웠던 int (*p)[2] ; -> int (*)[2]p

리턴 : void (*) (int)　　<ㅡ함수포인터
함수명 signal
인자 : int signum 과 void(*handler)(int)

void(*p)(void) :
void를 리턴하고 void를 인자로 취하는 함수의 주소값을 저장할 수 있는 변수 p

7일차 복습

* 함수 포인터를 도대체 왜 쓰는가 ?

1. 비동기 처리
2. HW 개발 관점에서 인터럽트
3. 시스템 콜(유일한 SW 인터럽트임)

여기서 인터럽트들(SW, HW)은
사실상 모두 비동기 동작에 해당한다.
결국 1 번(비동기 처리)가 핵심이라는 의미다.

그렇다면 비동기 처리라는 것은 무엇일까 ?

기본적으로 동기 처리라는 것은
송신하는쪽과 수신하는쪽이 쌍방 합의하에만 달성된다.
(휴대폰 전화 통화 등등)
반면 비동기 처리는 이메일, 카톡등의 메신저에 해당한다.
그래서 그냥 일단 던져 놓으면 상대방이 바쁠때는 못보겠지만
그다지 바쁘지 않은 상황이라면 메시지를 보고 답변을 줄 것이다.
이와 같이 언제 어떤 이벤트가 발생할지
알 수 없는 것들을 다루는 녀석이 바로 함수 포인터다.

사람이 이런데서는 임기응변을 잘 해야 하듯이
컴퓨터 관점에서 임기응변을
잘 하도록 만들어주는 것이 바로 함수 포인터다.
(결론: 비동기 처리 - 함수 포인터)

meommove()는 언제 사용하는가 ?

사용하려면 string.h 헤더파일 사용

memory move의 함성어
메모리의 값을 복사할 때 사용함
momove(목적지 원본 길이)로 사용함,

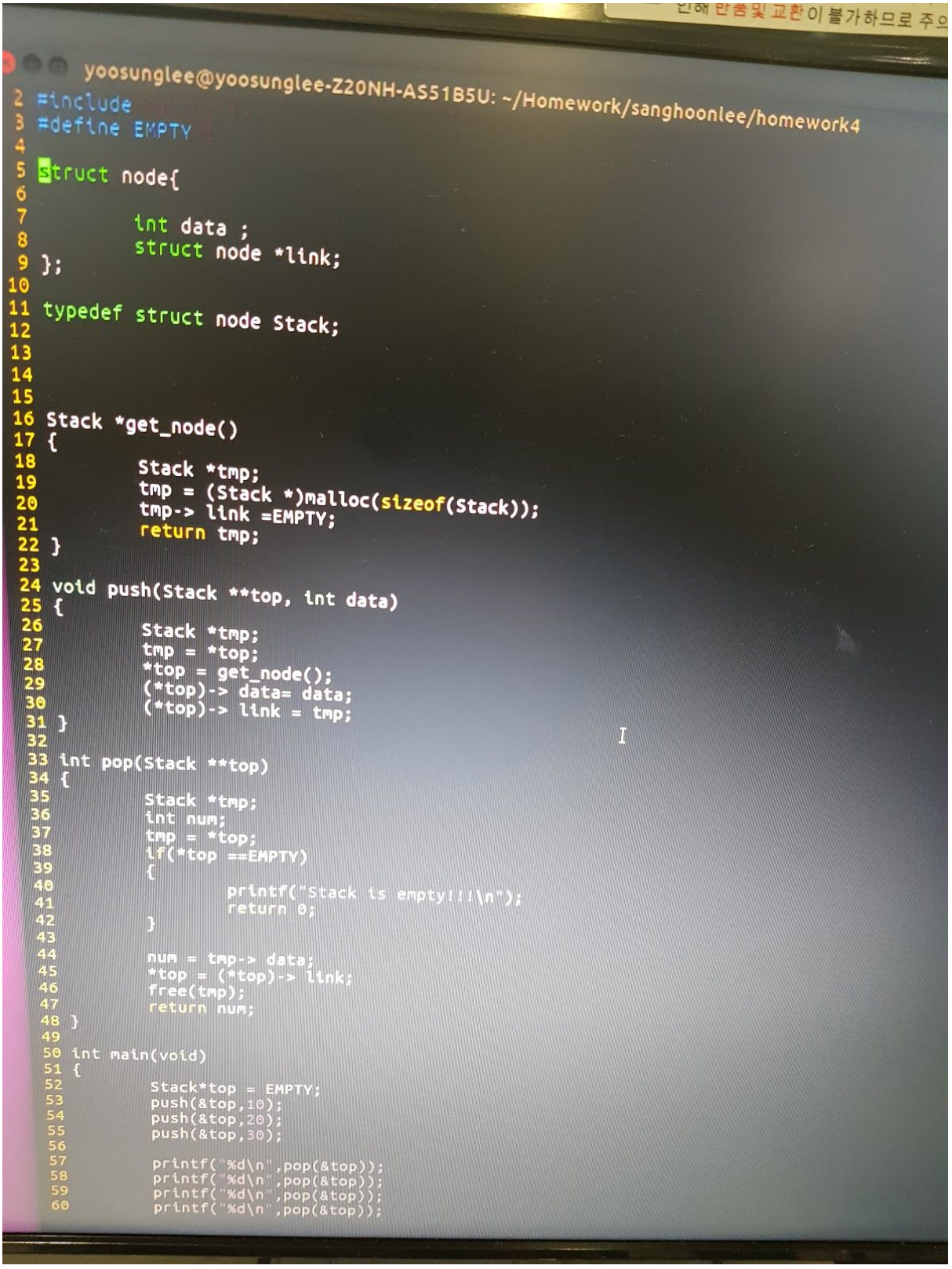
```
 #include<string.h>

 int main (void){

        int i =0;
        int src[5] = {1,2,3,4,5};
        int dst[5];

        memmove(dst,src,sizeof(src));

        for(i=0 ; i<5; i++)

        printf("dst[%d] =  %d\n", i , dst[i]);

                return 0;

                }
```

```
dst[0] =  1
dst[1] =  2
dst[2] =  3
dst[3] =  4
dst[4] =  5
```

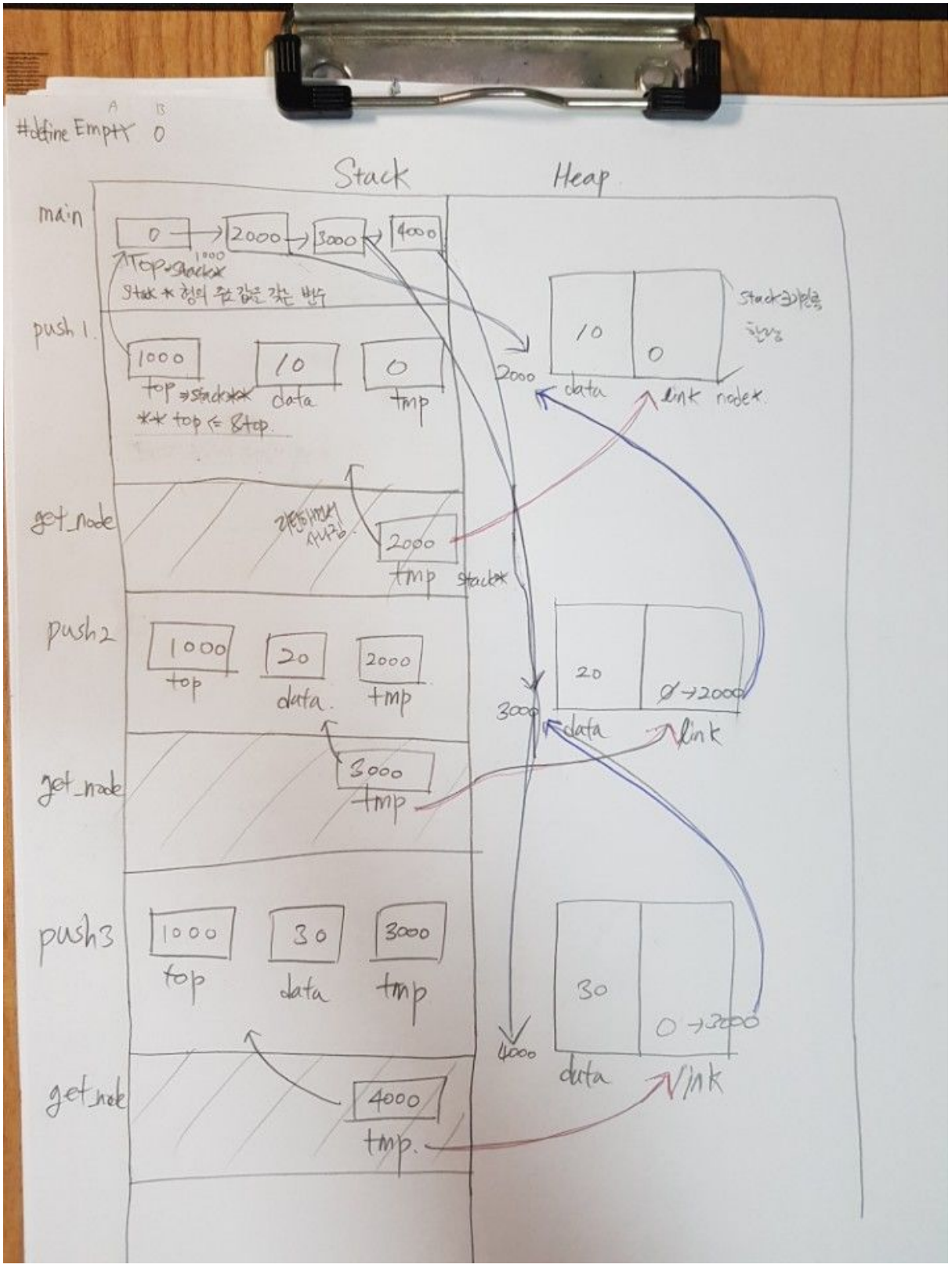
memcopy

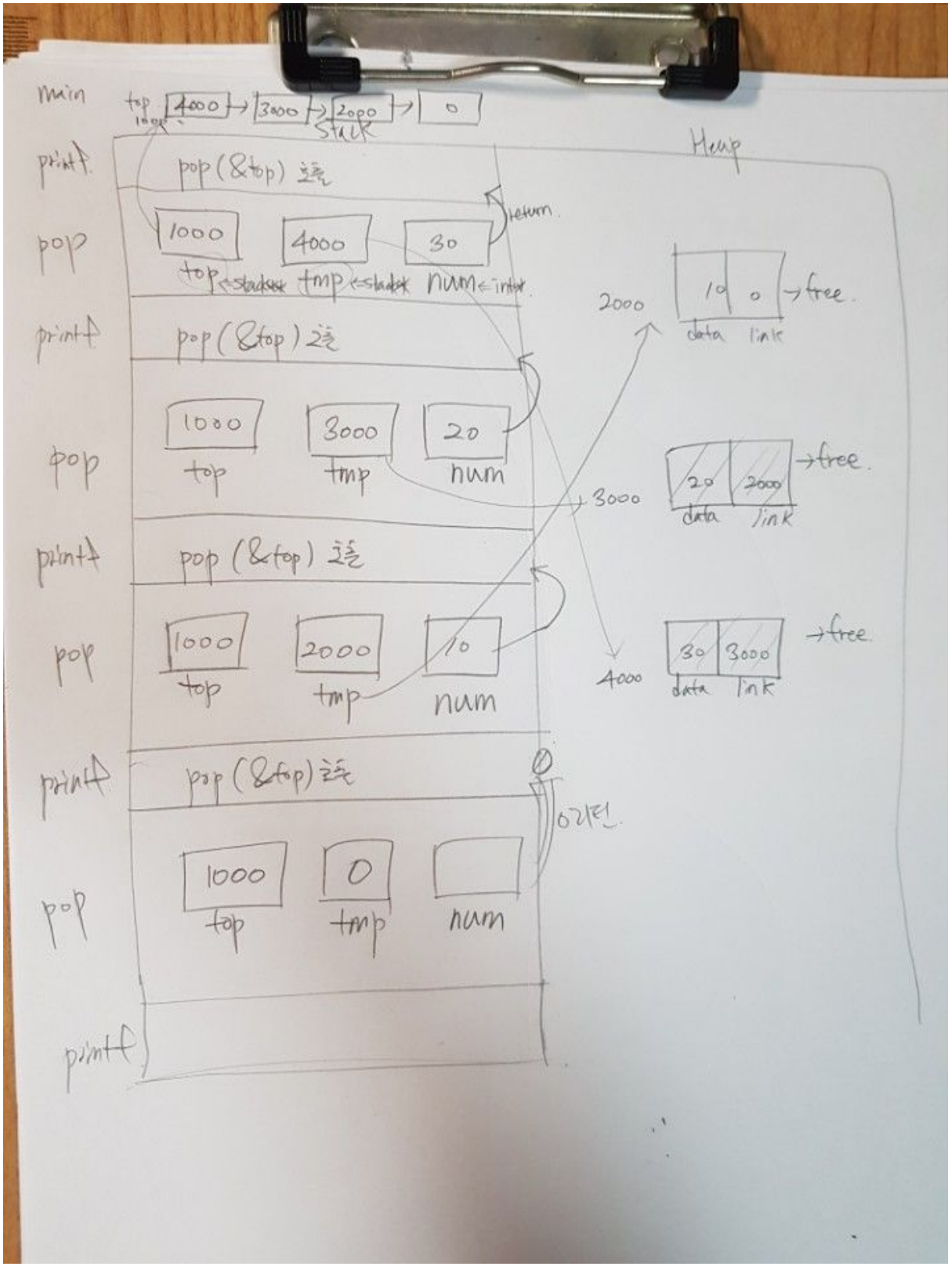
성능은 이게 더 좋은데 컴파일에서 문제가 발생할 수 있음,

stack 구현하기



```
yoosunglee@yoosunglee-Z20NH-AS51B5U: ~/Homework/sanghoonlee/homework4
#include
#define EMPTY

struct node{

        int data ;
        struct node *link;
};

typedef struct node Stack;



Stack *get_node()
{
        Stack *tmp;
        tmp = (Stack *)malloc(sizeof(Stack));
        tmp-> link =EMPTY;
        return tmp;
}

void push(Stack **top, int data)
{
        Stack *tmp;
        tmp = *top;
        *top = get_node();
        (*top)-> data= data;
        (*top)-> link = tmp;
}

int pop(Stack **top)
{
        Stack *tmp;
        int num;
        tmp = *top;
        if(*top ==EMPTY)
        {
                printf("Stack is empty!!!\n");
                return 0;
        }

        num = tmp-> data;
        *top = (*top)-> link;
        free(tmp);
        return num;
}

int main(void)
{
        Stack*top = EMPTY;
        push(&top,10);
        push(&top,20);
        push(&top,30);

        printf("%d\n",pop(&top));
        printf("%d\n",pop(&top));
        printf("%d\n",pop(&top));
        printf("%d\n",pop(&top));
```


#define Empty 0
Stack
Heap
main
push 1
get_node
push 2
get_node
push 3
get_node
top
data
tmp
link
node*
Stack*
2000
3000
4000
1000
10
20
30
0


main
top
1000
4000
3000
2000
0
Stack
Heap
printf
pop(&top) 호출
return
pop
1000
4000
30
top ← stack
tmp ← stack
num ← int
printf
pop(&top) 2호출
pop
1000
3000
20
top
tmp
num
printf
pop(&top) 3호출
pop
1000
2000
10
top
tmp
num
printf
pop(&top) 호출
0리턴
pop
1000
0
top
tmp
num
printf
2000
10
0
free
data
link
3000
20
2000
free
data
link
4000
30
3000
free
data
link

이중 포인터의 이점

(트리 계열의 자료 구조를 재귀 호출 없이 구현할 수 있음)

머릿속으로 그린 그림을 코드로 구현할 수 있는
능력을 키우는 것이 자료 구조를 학습하는 이유다.
이것을 잘해야 SW 를 씹어먹을 수 있다.
이게 안되면 백날 해봐야 매일 뻔한 반복 작업이나 하게 된다.
(알고리즘 작성 능력이 월등히 상승하게 됨)