TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – GJ (박현우)
uc820@naver.com

1. 복습 – (5일차)

- 함수 포인터!!

✓ 함수 프로토타입이란?

리턴, 함수명, 인자에 대한 기술서이다.

그렇다면, **void (* signal(int signum, void (* handler)(int)))(int);**
위 함수에 대한 프로토타입은 뭘까?

이전에 공부했던 int (*p)[2]; -> int (*)[2] p 배열포인터 // int *p[2] 포인터 배열과는 다름

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

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

1. 복습 – (6일차)

- 복잡한 함수 포인터!!

1) int (* aaa(void))[2] // 실제 문법
→ int (*)[2] aaa(void) // 인간이 보기 편하게 만듬

풀이: int형 배열 2개짜리 묶음주소를 반환하고 void를 인자로 취하는 함수 aaa

2) int (*(* bbb(void))(void))[2]; // 실제 문법
→ int (*)[2](*)(void) bbb(void) // 인간이 보기 편하게 만듬

풀이: int형 배열 2개짜리 묶음주소를 반환하고 인자로 void를 취하는 함수 포인터를 리턴하고
void를 인자로 취하는 함수 bbb

3) int (*(*(*p[][2])(void))(void))[2] // 실제 문법
→ int (*)[2] (*)(void) (*)(void) p[][2]
→ int (*)[2] (*)(void) bbb(void) p[][2]

풀이: int형 배열2개 짜리 묶음주소를 반환하고 void를 인자로 취하는 함수의 주소 값을 리턴하고
인자로 void를 취하는 함수의 주소 값을 반환하는 배열변수 p

1. 복습 – (6일차)

- 그렇다면!! 대체 이렇게 복잡한 함수 포인터를 왜 쓸까?

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

여기서 인터럽트(SW, HW)은 사실상 모두 비동기 동작에 해당한다.
→ 결국 1번(비동기 처리)가 핵심이라는 뜻이다.

그렇다면, 비동기 처리는 뭘까?
→ 기본적으로 동기 처리라는 것은 송신과 수신의 쌍방 합의하에 달성된다
(ex. 휴대폰 전화 통화.)
반면, 비동기 처리는 이메일 또는 카카오톡 메신저 등에 해당한다.
그래서 그냥 일단 송신해두면 상대방이 바쁘면 못 보지만
바쁘지 않은 상황이라면 메시지를 보고 답변을 준다.
이처럼 언제 어떤 이벤트가 발생할지를 알 수 없는 것들은 다루는 것이 비동기이며 함수 포인터를 사용하는 이유이다.

✓ 즉, 사람도 임기응변을 잘 해야 하듯이, 컴퓨터 관점에서도 임기응변을 잘하도록 만들어주는 것이 바로 함수 포인터이다. (결론: 비동기 처리 – 함수 포인터)

2. 함수 포인터 문제

float (* (* test(void (*p)(void)))(float (*)(int, int)))(int, int)
위와 같은 프로토타입의 함수가 구동되도록 프로그래밍 하시오.

hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```c
#include<stdio.h>

//void (*p)(void)
void aaa(void){

    printf("aaa called\n");

}
float bbb(int n1, int n2){

    return n1 + n2;
}

//float (*)(int,int) (*)(float (*)(int,int))
//float (*)(int, int) ccc(float(*p)(int,int))
float (* test2(float(*p)(int,int)))(int,int){

    float res;

    res = p(3,6);

    printf("test2 res = %f\n",res);

    return bbb;
}

//float (*(* test(void (*p)(void)) )(float (*)(int,int)) )(int,int)
//float (*)(int,int) (*)(float (*)(int,int)) test(void (*p)(void))
float (*(* test(void (*p)(void)))(float (*)(int,int)))(int,int){

    p(); // aaa called;


    return test2;
}

int main(int agrc, char **argv){

    float res;

    res = test(aaa)(bbb)(4,6);

    printf("main res = %f\n",res);
    return 0;
}
```

"func_pointer.c" 45L, 712C 1,1 모두

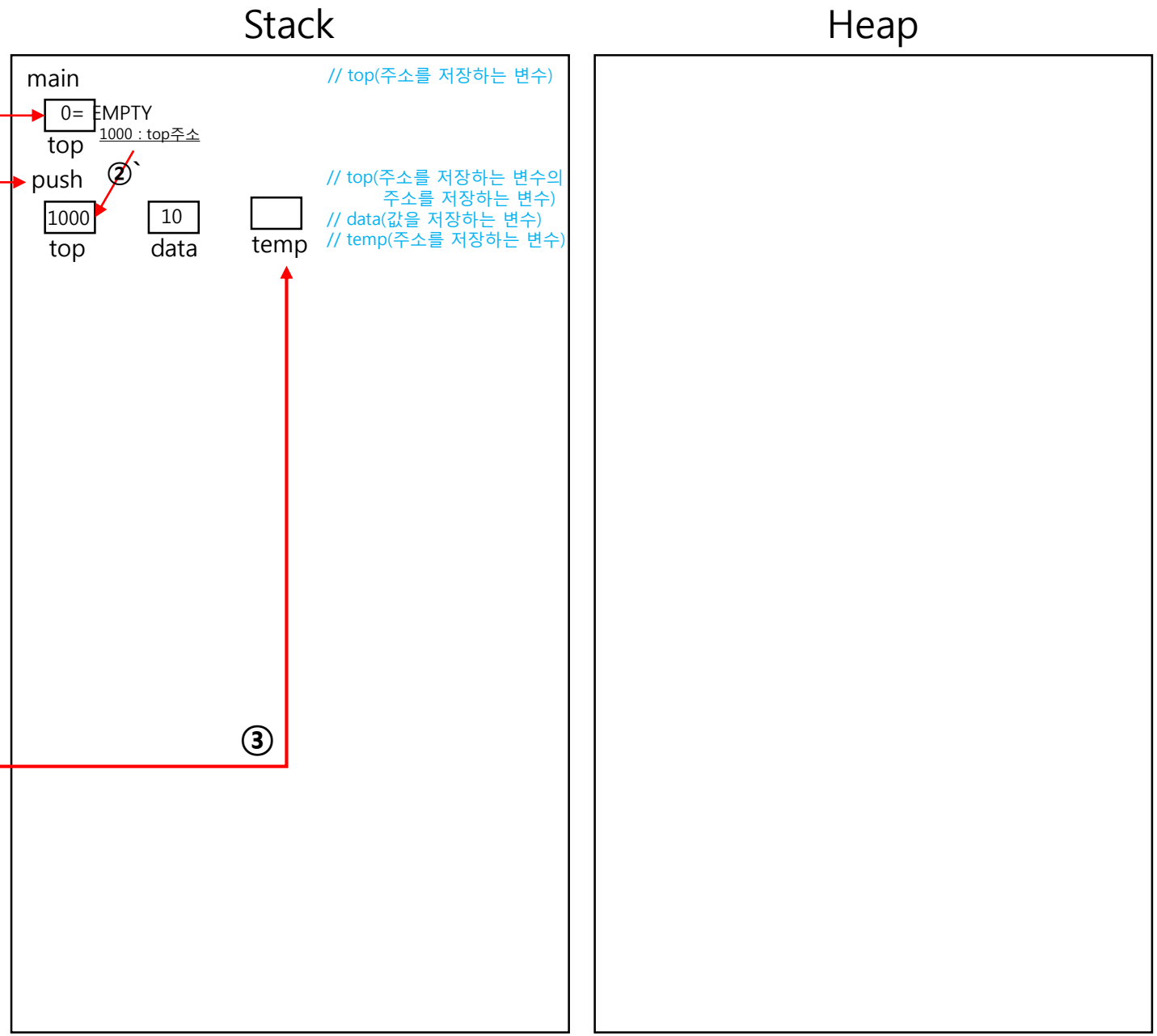
hyunwoopark@hyunwoopark-P65-P67SG: ~/hw

```
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$ ./a.out
aaa called
test2 res = 9.000000
main res = 10.000000
hyunwoopark@hyunwoopark-P65-P67SG:~/hw$
```

3. 스택(Stack) 코드 이해(그림 그리기) – (1)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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;
}
```

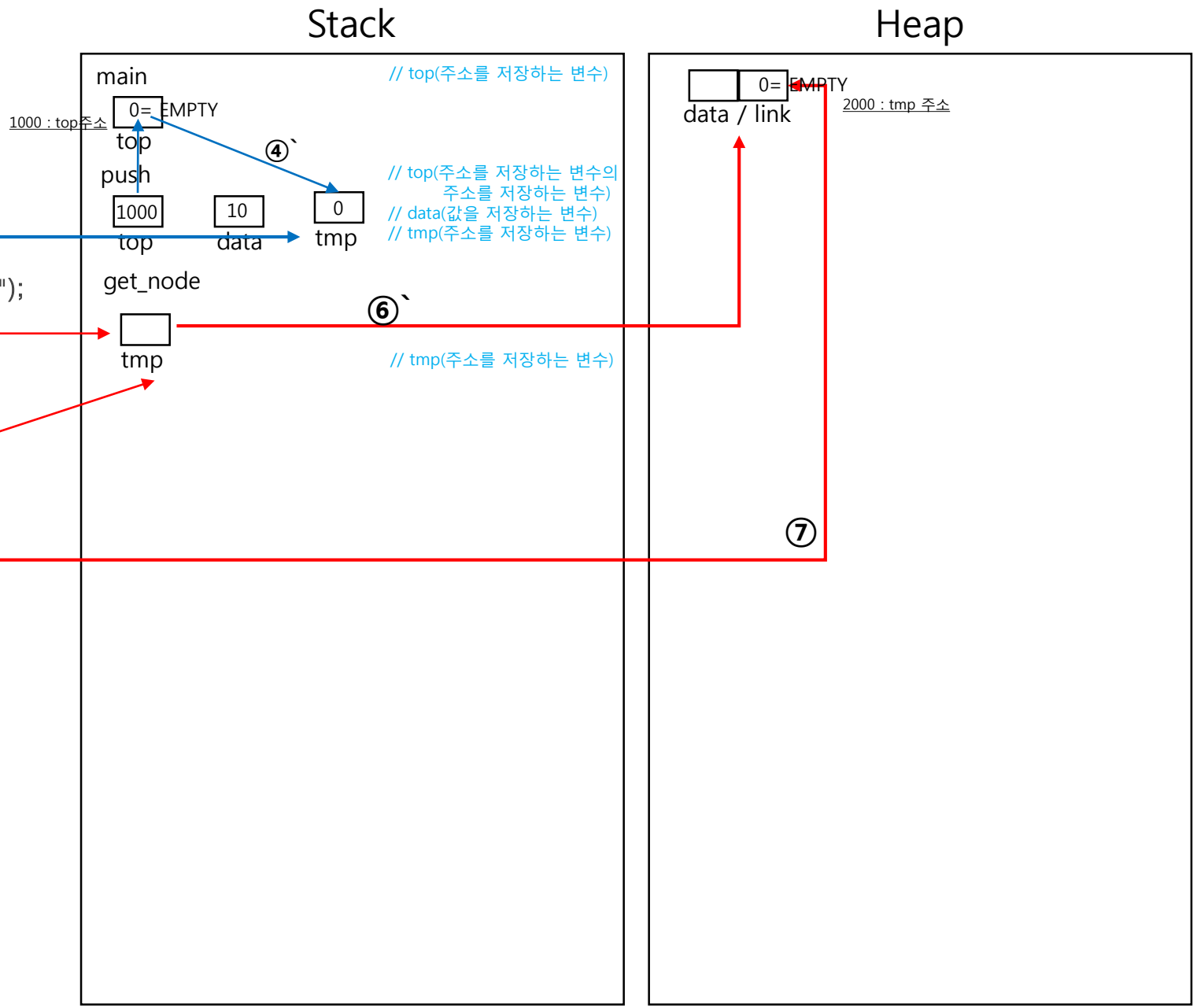
```
int pop(Stack **top){

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));

return 0;

}
```



3. 스택(Stack) 코드 이해(그림 그리기) – (2)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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;
}
```

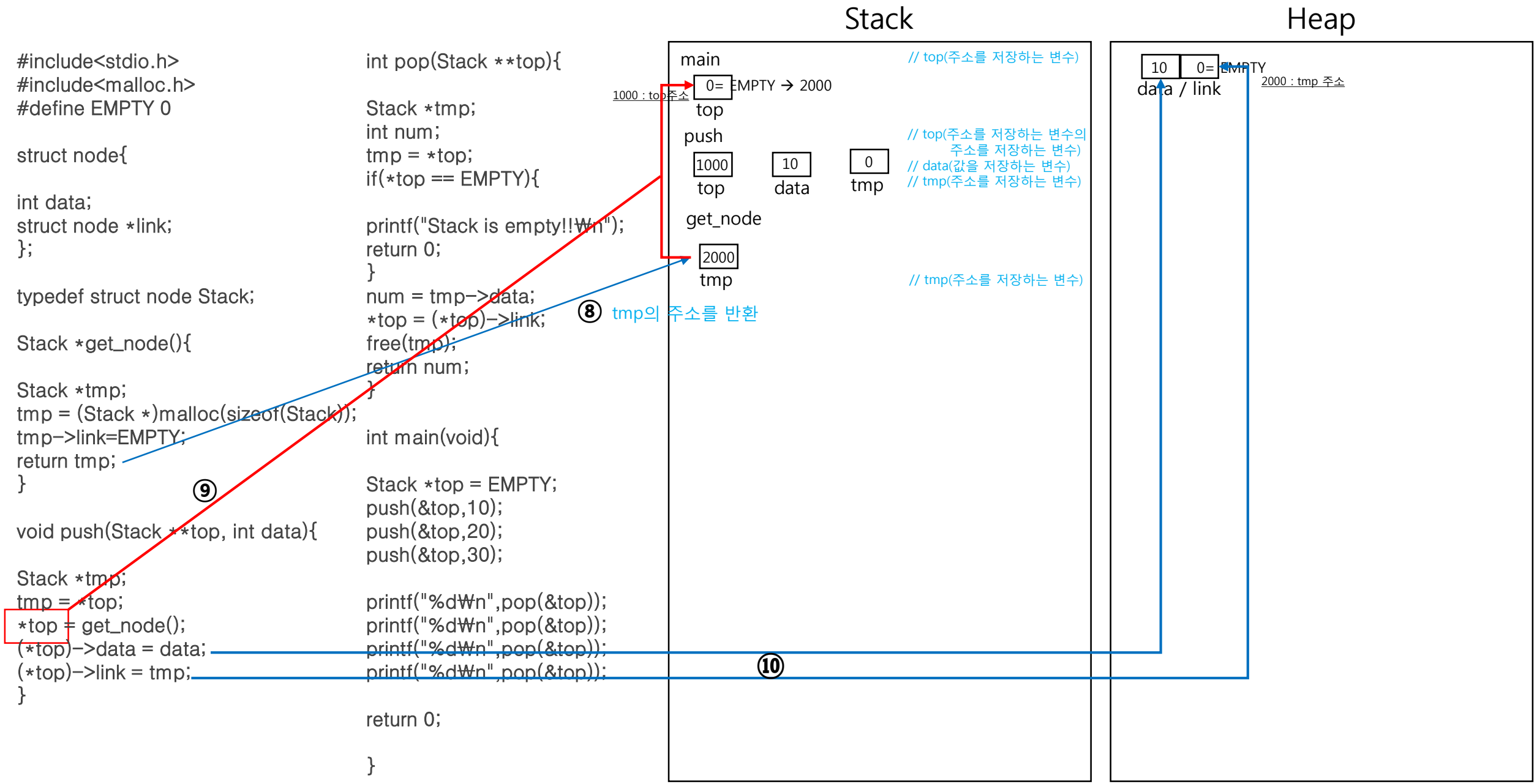
```
int pop(Stack **top){

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));

return 0;

}
```



3. 스택(Stack) 코드 이해(그림 그리기) – (3)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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));

return 0;

}
```



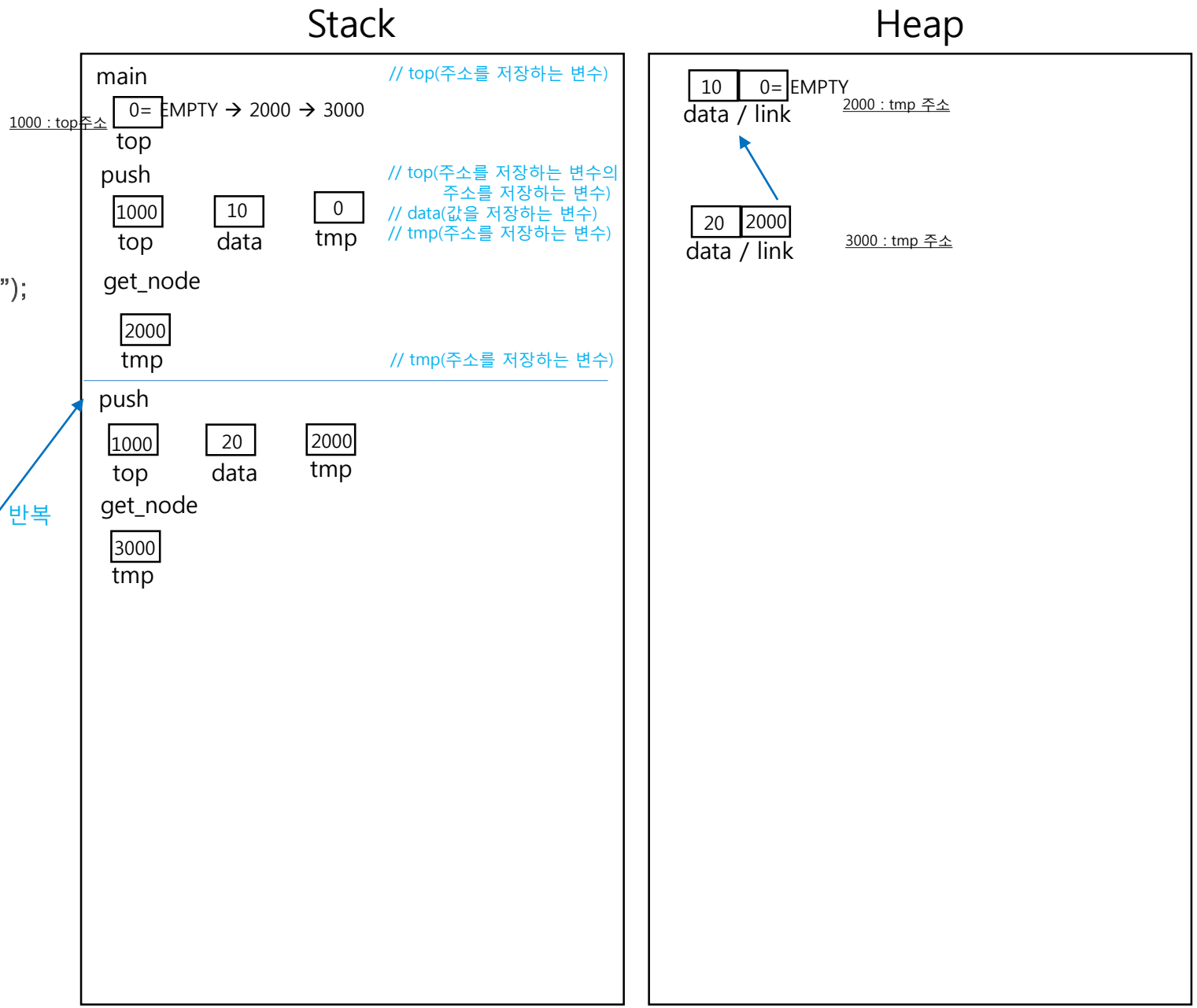
3. 스택(Stack) 코드 이해(그림 그리기) – (4)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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;
}
```

① ~ ⑩과 같은 방식으로 반복

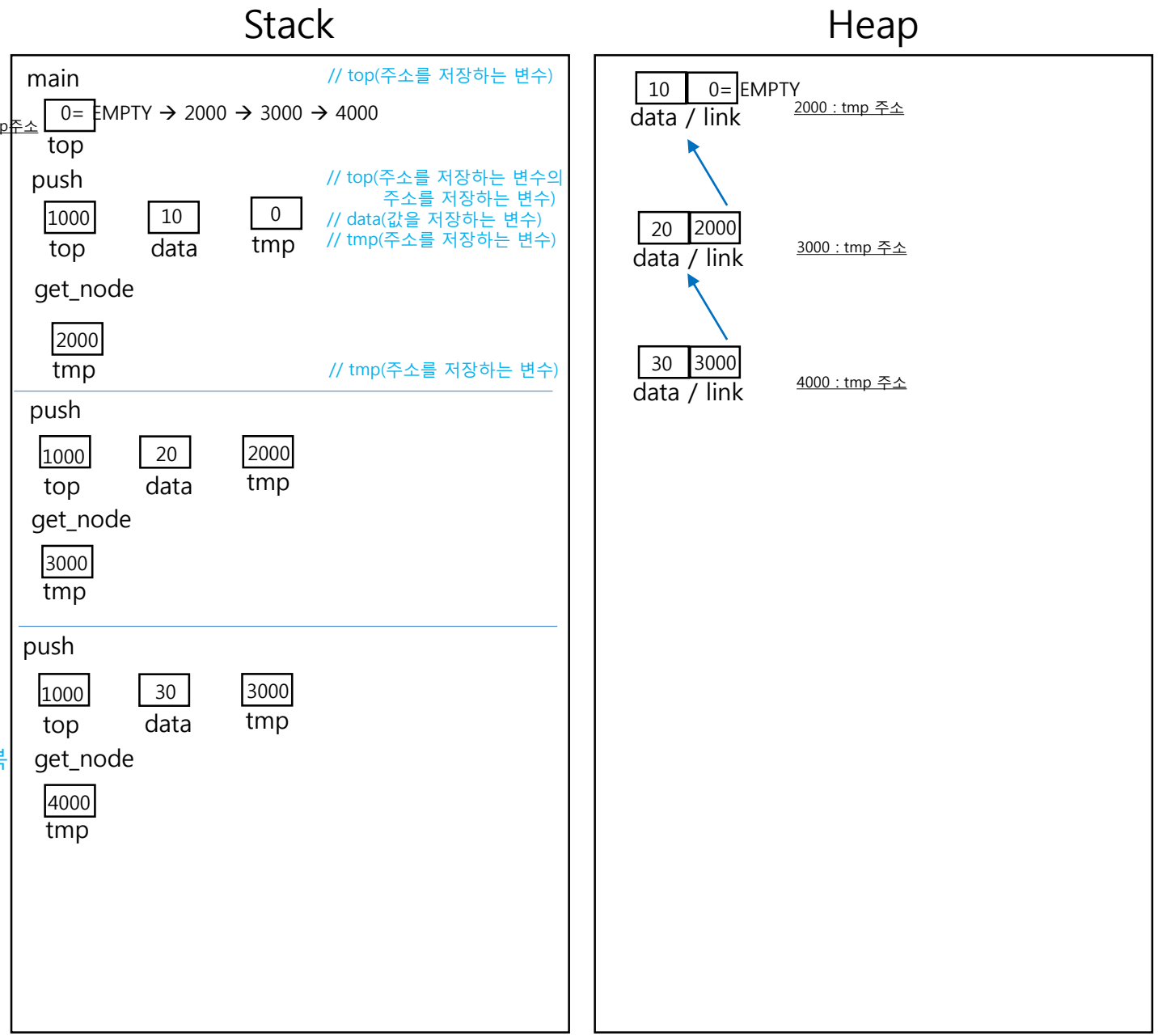
```
Int main(void){

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));

return 0;

}
```



3. 스택(Stack) 코드 이해(그림 그리기) – (5)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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;
}
```

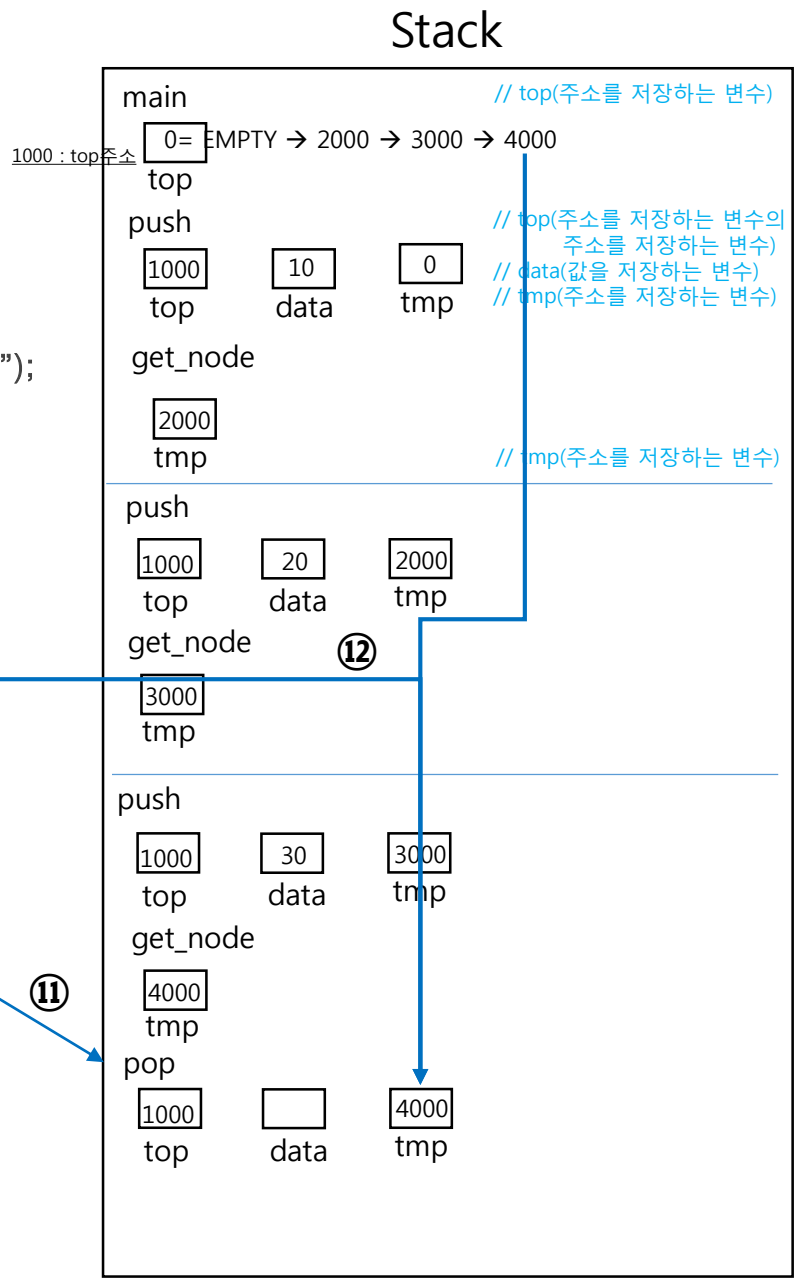
```
int pop(Stack **top){

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);
        ① ~ ⑩과 같은 방식으로 반복
printf("%d₩n",pop(&top));
printf("%d₩n",pop(&top));
printf("%d₩n",pop(&top));
printf("%d₩n",pop(&top));

return 0;

}
```

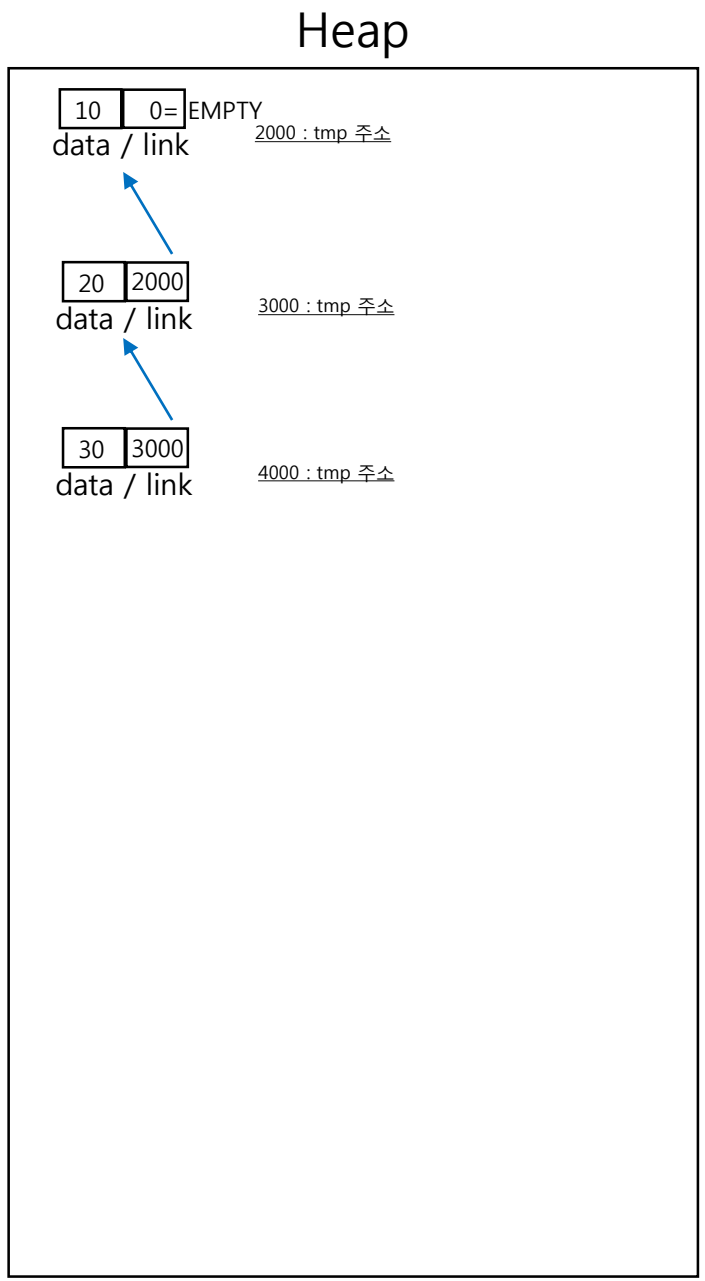


3. 스택(Stack) 코드 이해(그림 그리기) – (6)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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));

return 0;

}
```

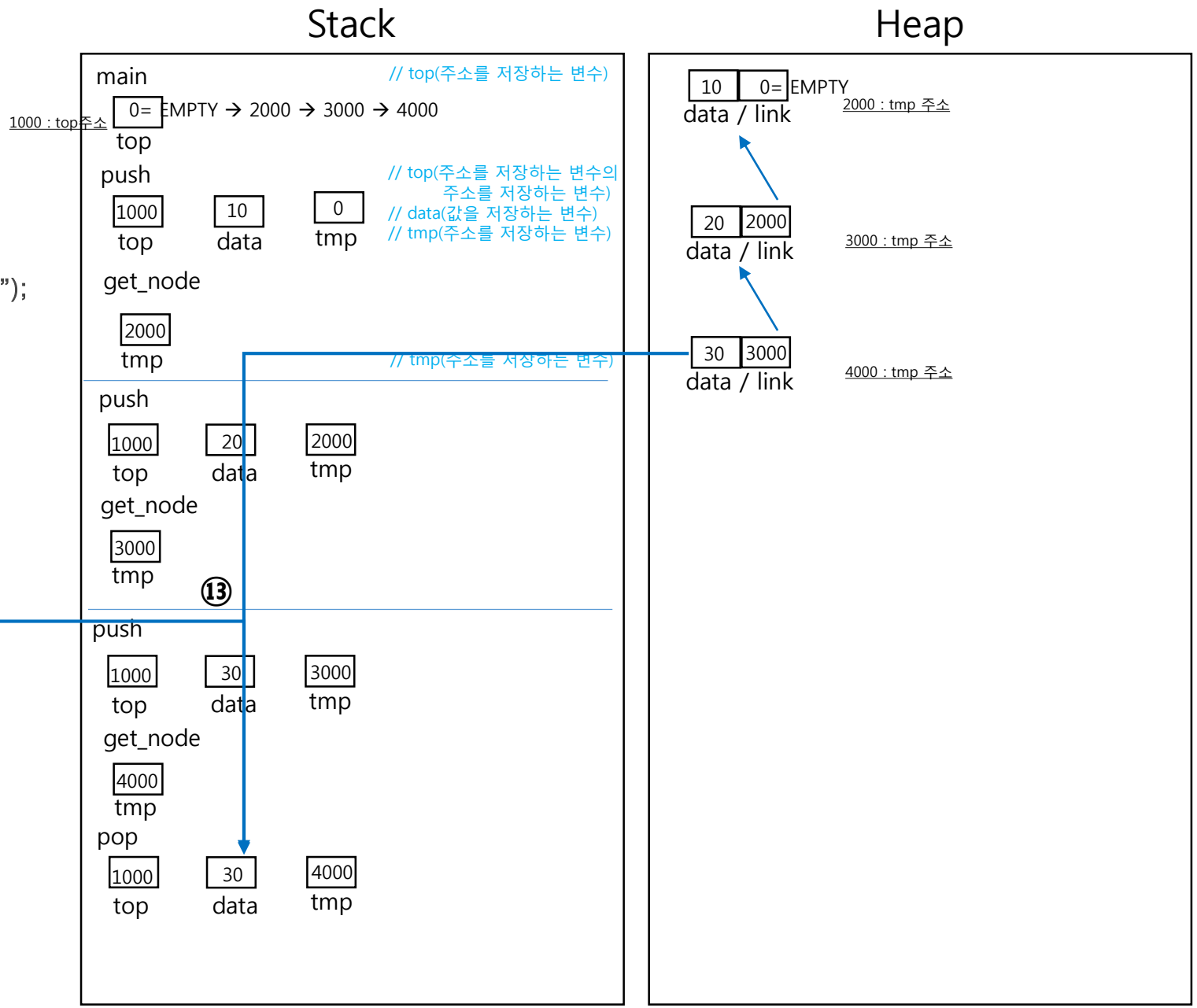




3. 스택(Stack) 코드 이해(그림 그리기) – (7)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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;
}
```

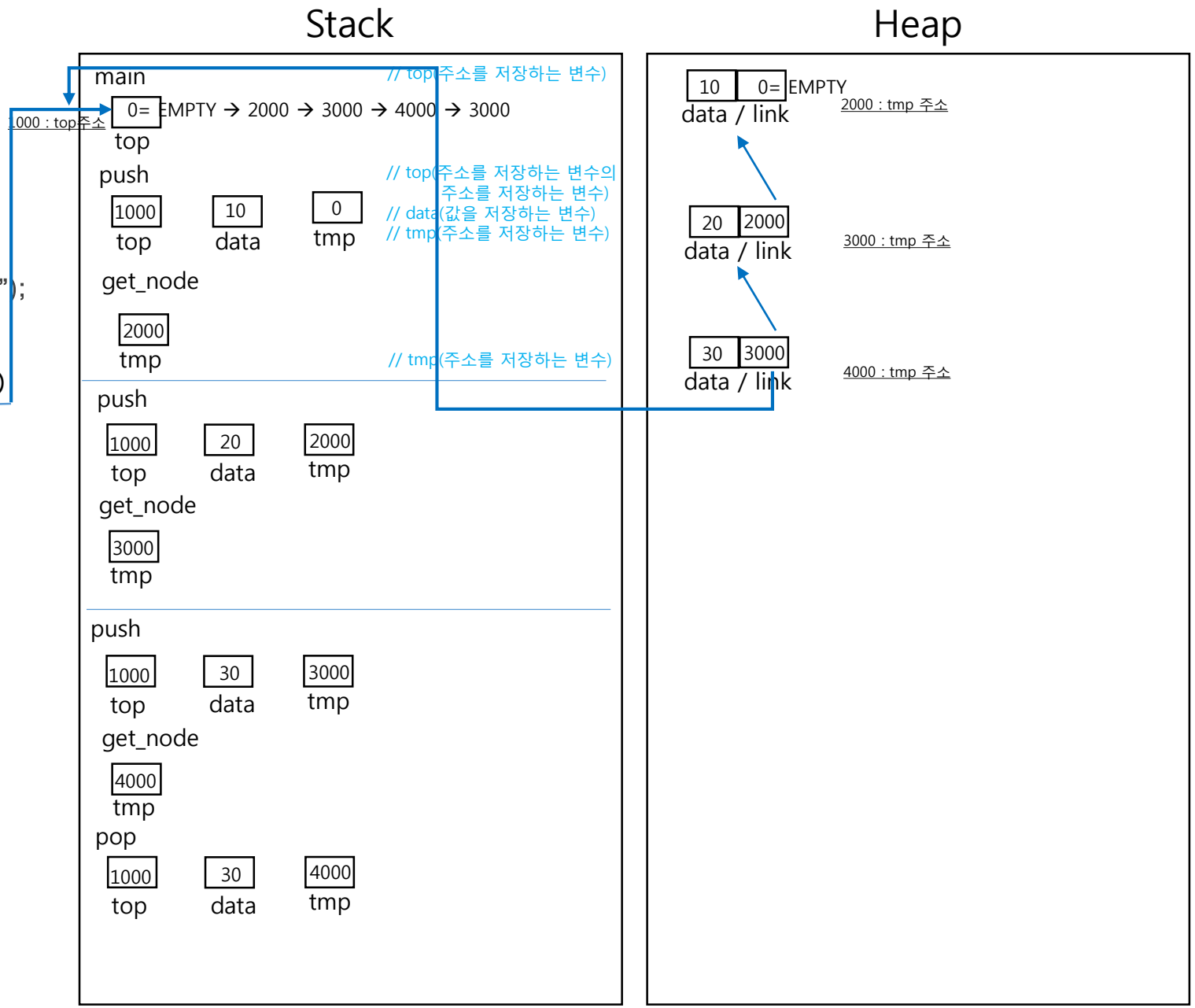
```
int pop(Stack **top){

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));

return 0;

}
```



3. 스택(Stack) 코드 이해(그림 그리기) – (8)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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;
}
```

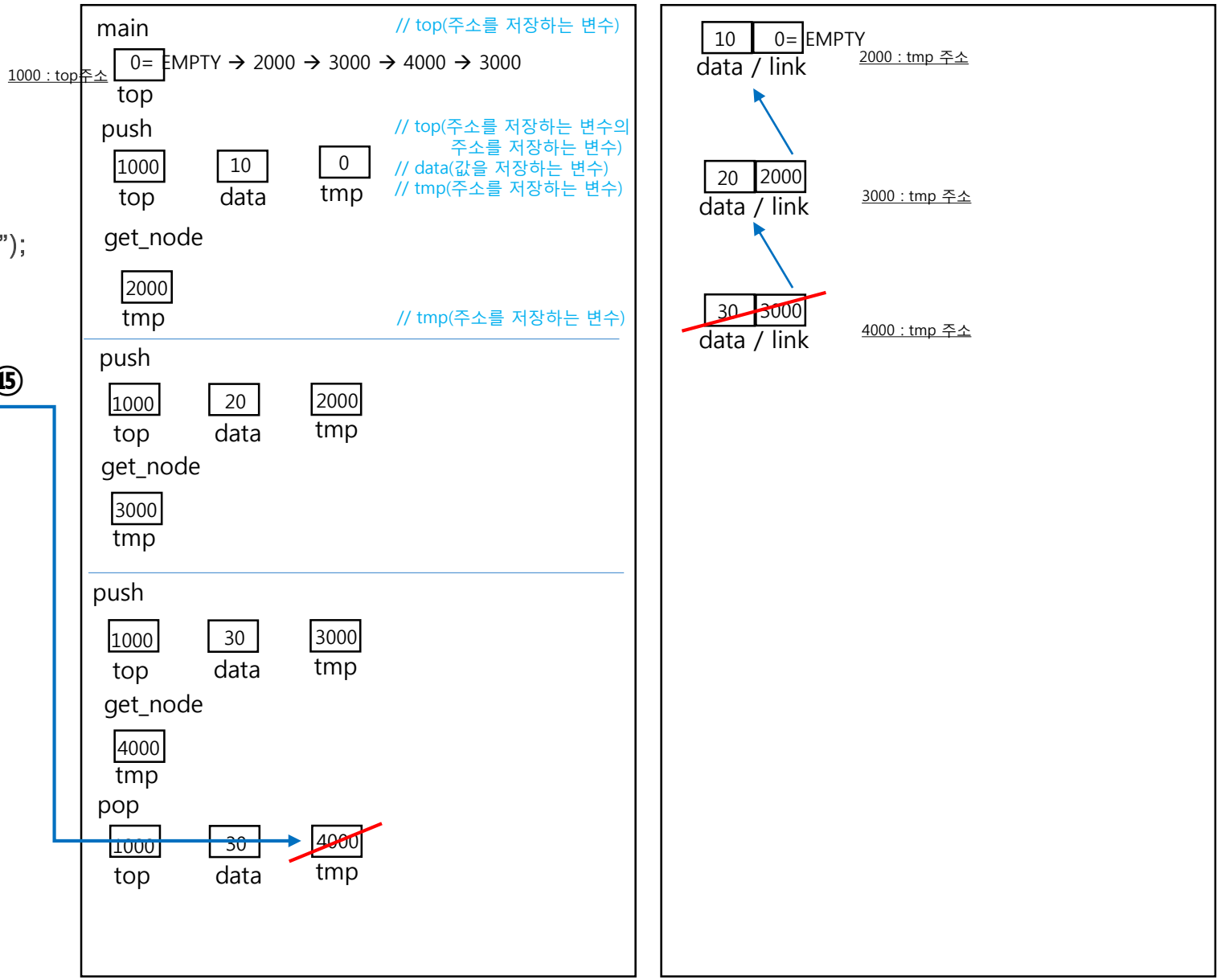
```
int pop(Stack **top){

Stack *tmp;
Int num;
Tmp = *top;
If(*top == EMPTY){

Printf("Stack is empty!!₩n");
Return 0;
}
Num = tmp->data;        ⑭
*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));

return 0;

}
```



3. 스택(Stack) 코드 이해(그림 그리기) – (9)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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));

return 0;

}
```



3. 스택(Stack) 코드 이해(그림 그리기) – (10)

```
#include<stdio.h>
#include<malloc.h>
#define EMPTY 0

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));

return 0;

}
```