TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

7회차 (2018-03-02)

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 - 정유경
ucong@naver.com

1. 함수 포인터 선언문

① **int **a**; int형을 가리키는 포인터의 포인터

② **int *a[3];** int형을 가리키는 포인터를 3개 저장하는 배열

③ **int (*a)[3];** int형을 3개 저장하고있는 배열을 가리키는 포인터

④ **int (*p)(char);** char를 인자로 갖고, int형을 반환하는 함수에 대한 포인터

⑤ **char** (*p)(float, float);** float, float을 인자로 갖고, char을 가리키는 포인터의 포인터를 반환하는 함수에 대한 포인터

⑥ **void* (*a[5])(char* const, char* const);** char* const, char *const를 인자로 갖고, void를 가리키는 포인터를 리턴하는 함수에 대한 포인터를 5개 저장하는 배열

⑦ **int* (*(fp1)(int))[10];** int를 인자로 갖고, int를 가리키는 포인터를 10개 저장하고 있는 배열의 포인터를 리턴하는 함수

⑧ **int* (*(arr[5])( ))( );** int를 가리키는 포인터를 리턴하는 함수에 대한 포인터를 리턴하는 함수 5개를 저장하는 배열

⑨ **float (*(*b( ))[ ])( );** float를 리턴하는 함수에 대한 배열을 저장하고있는 포인터를 리턴하는 함수에 대한 포인터

⑩ **void* (c)(char, int ( )( ));** char, int( )( )를 인자로 갖고, void를 가리키는 포인터를 리턴하는 함수

⑪ **void** (*d)(int &, char**(*) (char*, char**));** int &, char**(*)(char *, char *)를 인자로 갖고, void를 가리키는 포인터의 포인터를 리턴하는 함수에 대한 포인터

⑫ **float (*(*e[10])(int &))[5];** int &를 인자로 갖고, float을 5개 저장하고있는 배열의 포인터를 리턴하는 함수에 대한 포인터를 10개 저장하고 있는 배열

⑬ **int *(*(*fp1)(int))[10];** int를 인자로 갖고, int를 가리키는 포인터 10개를 저장하고있는 배열의 포인터를 리턴하는 함수에 대한 포인터

⑭ **char *(*(**foo[ ][8])( ))[ ];** char를 가리키는 포인터를 저장하고 있는 배열의 포인터를 리턴하는 함수에 대한 포인터의 포인터 8개를 저장할 수 있는 배열에 대한 배열

⑮ **void bbb(void(*p)(void));** 인자로 (void(*p)(void)를 갖고, void를 리턴하는 함수

⑯ **void(*bbb(void))(void);** 인자로 void를 갖고, void를 인자로 갖고 void를 리턴하는 함수에 대한 포인터를 리턴하는 함수

⑰ **void(*bbb(void(*p)(void)))(void);** 인자로 void(*p)(void)를 갖고, void를 인자로 갖고 void를 리턴하는 함수에 대한 포인터를 리턴하는 함수

⑱ **int (*(*bbb(void))(void))[2];** 인자로 void를 갖고, void를 인자로 갖고 int를 2개 저장하고있는 배열의 포인터를 리턴하는 함수에 대한 포인터를 리턴하는 함수

2. 별들의 전쟁

```
#include <stdio.h>
// int형 데이터를 2개 저장하는 배열을 가리키는 포인터를 리턴하는 함수
int ( *aaa(void) )[2] // int (*)[2] aaa(void)
{
	static int a[2][2] = {10, };
	printf("aaa called\n");
	return a;
}

int (* (* bbb(void) )(void) ) [2] // int( *(*)(void) )[2]   bbb (void)
{
/* void를 인자로 갖고, int를 2개 저장하고 있는 배열의 포인터를 리턴하는 함수
에 대한 포인터를 리턴하는 함수 */
	printf("bbb called\n");
	return aaa;
}

int main(void){
	int (*ret)[2]; //int를 2개 저장하고 있는 배열포인터
	int (* (*(*p[][2])(void) )(void))[2] = {{bbb, bbb}, {bbb, bbb}};
	int (*(*(*(*p1)[2])(void))(void))[2] = p;
	ret = ((*(*(*(*p1)[2])))()());
	printf("%d\n", *ret[0]);
	return 0;
}
```

3. 함수 포인터와 인터럽트, 그리고 비동기 처리

함수포인터는 필요에 따라 다른 함수를 실행할 방법을 제공하며, 실행의 순서를 제어하는데 유용하게 사용된다.

4. C언어의 기타 함수들

(1) Memmove

Memory Move의 합성어

메모리의 값을 복사할 때 사용하며 memcpy보다 느리지만 안정적

memmove(dest, src, sizeof(src));

(2) Memcpy

구조체, 배열 등을 일일히 복사하지 않고 메모리 블록처럼 한번에 복사할 수 있다.

memory 공간에 겹치는 부분이 없을때 사용하며 겹치는 부분이 없다면 성능에 좋음
memcpy(dest, src, sizeof(src));

(3) int main(int argc, char**argv);
Main함수는 총 전달된 인자의 수(argc)와 포인터 배열(argv)을 인자로 받는 함수이다.

(4) Put <-> get
printf 대용으로 사용가능
puts()는 string을 인자로 받는다
putchar()는 'A'를 인자로 받는다

(5) get
put() 함수의 상반되는 개념으로 scanf()와 유사하다.

(6) strlen
주로 strcpy(), malloc()등의 함수와 함께 사용한다.
문자열의 길이를 구하는데 사용한다.
strlen("문자열")

(7) Strncpy
문자열을 복사하고 싶을 경우 사용
strncpy(destination, source, length)

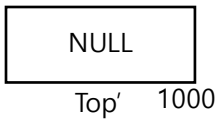
(8) Strcmp
문자열이 서로 같은지 비교, 서로 같은 경우 0을 반환
strcmp(str1, str2), strncmp(str1, str2, len)

5. 자료구조 STACK 이해하기

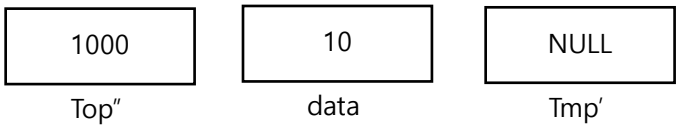
```c
#include <stdio.h>
#include <malloc.h>
#define EMPTY 0

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);
	pop(&top);
	return 0;
}
```

1. `Stack *top = EMPTY;`
구조체 포인터 top을 선언하고 NULL로 초기화한다.



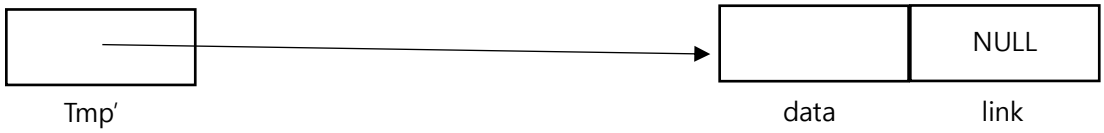
2. `push(&top, 10);`



push함수에 top포인터의 주소를 이중포인터 top에, 정수 10을 정수형 변수 data에 인자로 전달한다.

구조체 포인터 tmp를 선언하고 top포인터가 가리키는 값을 넣는다.

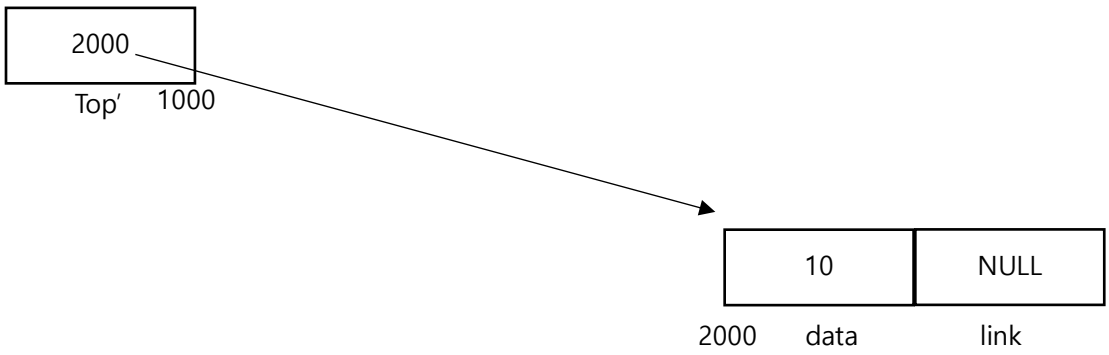
3. `*top = get_node();`



구조체 포인터 tmp를 선언하고 힙영역에 구조체 Stack타입의 동적메모리를 할당한다.

Tmp를 top'에 반환한다.

4. `(*top)->data = data; (*top)->link = tmp;`



5. pop(&top);

top포인터의 주소를 이중포인터 top에 인자로 전달한다.

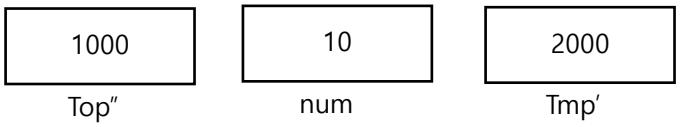
구조체 포인터 tmp를 선언하고 top포인터가 가리키는 값을 넣는다.

6. if (*top == EMPTY)

Top''포인터가 가리키는 값이 0이면, "스택이 비어있다"는 문구를 출력한다.

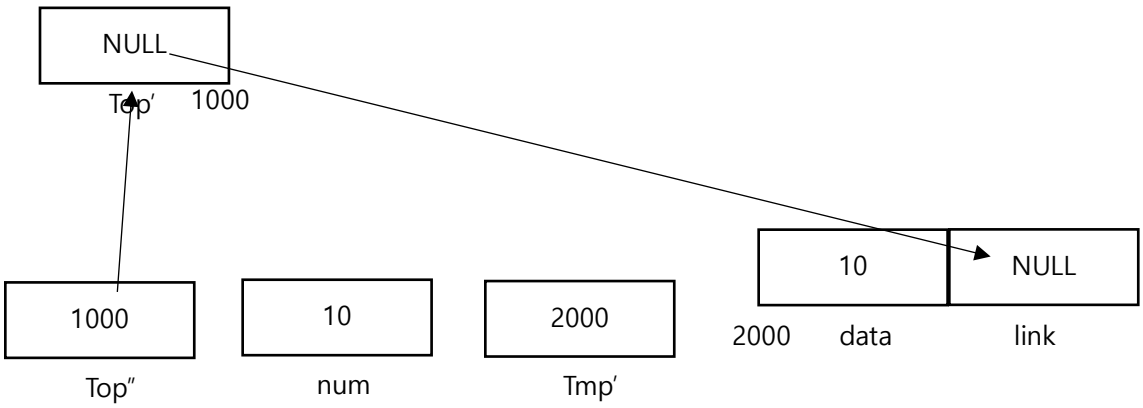
7. num = tmp->data;

스택이 비어있지 않을 경우, 구조체 포인터 tmp'가 가리키는 곳의 data를 변수 num에 저장한다.



8. *top = (*top)->link;

구조체 포인터 Top''가 가리키는 곳에서 link를 참조하여 NULL을 Top''가 가리키는 곳에 넣는다.



9. free(tmp); return num;

구조체 포인터 Tmp'가 가리키는 곳의 할당된 메모리를 해제하고 변수 num을 반환한다.