TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

8회차 (2018-03-05)

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

학생 - 정유경
ucong@naver.com

1. 자료구조 Stack pop 그림 그리기

```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; // 이전 노드 주소 저장
	*top=get_node(); // top이 새 노드를 가리킨다.
	(*top)->data =data;
	(*top)->link = tmp;
}
int Pop(Stack** top)
{
	Stack* tmp;
	int num;
	tmp=*top; // 동적할당을 해제할 메모리 주소를 tmp에 저장해놓는다
	if(*top==EMPTY)
	{
		printf("Stack is EMPTY!\n");
		return 0;
	}
	num=(*top)->data;
	*top =(*top)->link;
	free(tmp);
	return num;
}

int main(void)
{
	Stack*top=EMPTY;
	Push(&top,10); // top에서 Push하여 10을 넣는다
	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;
}
```

<stack>

main.

push ↔ pop

top'

push①

top" data top'

get node

top' 2000

push②

top" data top'

top' 3000

push③

top" data top'

top' 4000

결과!

top'

pop①

top" num top'

free!

return num

pop②

top" num top'

pop③

top" num top'

pop④

top" num top'

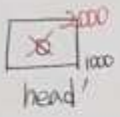
→ stack is Empty! (pop 종료)

2. 재귀호출과 이중 포인터 연동 Queue 구현하기

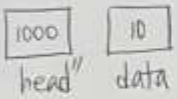
```c
#include <stdio.h>
#include <malloc.h>
#define EMPTY 0

typedef struct node {
	int data;	// 데이터 저장
	struct node * link;	// 연결에 사용
} queue;

queue* get_node()
{
	queue * tmp;	// 구조체초인터 tmp선언
	tmp = (queue*)malloc(sizeof(queue)); // 노드 생성
	tmp->link = EMPTY;	// 노드의 link를 NULL로 초기화
	return tmp;	// 노드의 주소를 리턴한다
}

void print_queue(queue * head)
{
	queue * tmp = head;
	while (tmp)	//	tmp가 가리키는 노드가 있을때 반복한다
	{
		printf("%d\n", tmp->data);	//tmp가 가리키는 노드의 데이터 출력
		tmp = tmp->link;
	}
}

/* Dequeue 구현하기 */

void Enqueue(queue** head, int data)
{
	if (*head == NULL)	// 첫번째 노드라면
	{
		*head = get_node();	// 첫번째 노드를 head가 가리킨다
		(*head)->data = data; // head가 가리키는 노드에 데이터를 저장한다
		return;
	}
	Enqueue( &(*head)->link , data);	// head가 가리키는 link에 접근해서 그 주소와 데이터를 인자로 전달한다
	printf("재귀함수 호출!\n");
}

int main()
{
	queue *head = EMPTY; // 구조체queue를 가리키는 포인터 변수
	Enqueue(&head, 10);
	Enqueue(&head, 20);
	Enqueue(&head, 30);
	print_queue(head);
}
```

<queue>

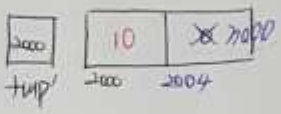
main



Enqueue ①

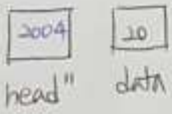


get node

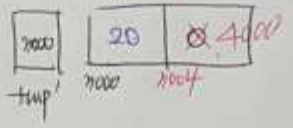


Enqueue ②

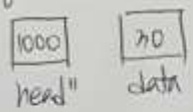
2번째 Enqueue



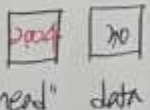
get node



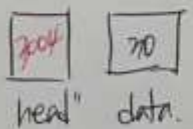
Enqueue ③



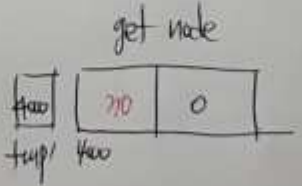
3번째 Enqueue

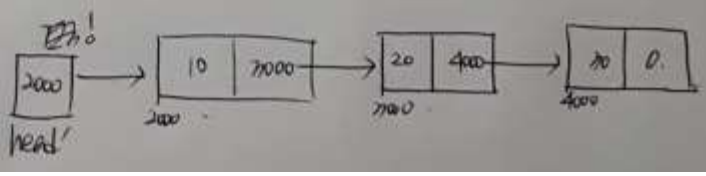


3번째 Enqueue



get node





[Main]
Queue를 가리키는 구조체 포인터 head’를 선언하고 NULL로 초기화 한다.

[Enqueue 1]
Enqueue 함수를 호출하며 인자로 head’를 가리키는 구조체 포인터 head’’과 data=10을 전달한다.
Head’’가 가리키는 값이 0이므로 if제어문 안으로 들어간다.
getnode함수를 이용해서 새로운 노드를 생성한 뒤 노드의 주소를 head’’가 가리키는 head’에 저장한다.
Head’’의 head’의 data에 접근하여 인자로 전달받은 data=10을 저장한다.
return하고 Enqueue 함수를 빠져나온다.

[Enqueue 2]
Enqueue함수를 호출하여 인자로 head’를 가리키는 구조체 포인터 head’’와 data=20을 전달한다.
[재귀Enqueue]
Head’’이 가리키는 값이 0이 아니므로 if제어문을 호출하지 않고 재귀Enqueue함수를 호출하며 인자로 head’’의 head’의 link에 접근하여 그 주소인 2004와 data=20을 전달한다.
Head’’가 가리키는 값이 0이므로 if 제어문 안으로 들어간다.
getnode함수를 이용해서 새로운 노드를 생성한 뒤 노드의 주소를 head’’가 가리키는 이전 노드의 link에 저장한다.
Head’’의 link의 data에 접근하여 인자로 전달받은 data =20을 저장한다.
return하고 Enqueue함수를 빠져나온다.

[Enqueue 3]
Enqueue함수를 호출하여 인자로 head’를 가리키는 구조체 포인터 head’’와 data=30을 전달한다.
[재귀Enqueue 1]
Head’’이 가리키는 값이 0이 아니므로 if제어문을 호출하지 않고 재귀Enqueue함수를 호출하며 인자로 head’’의 head’의 link에 접근하여 그 주소인 2004와 data=30을 전달한다.
[재귀Enqueue 2]
Head’’이 가리키는 값이 0이 아니므로 if제어문을 호출하지 않고 재귀Enqueue함수를 호출하며 인자로 head’’의 link의 link에 접근하여 그 주소인 3004와 data=30을 전달한다.
Head’’가 가리키는 값이 0이므로 if 제어문 안으로 들어간다.
getnode함수를 이용해서 새로운 노드를 생성한 뒤 노드의 주소4000을 head’’가 가리키는 이전 노드의 link에 저장한다.
Head’’의 link의 data에 접근하여 인자로 전달받은 data =30을 저장한다.
return하고 Enqueue함수를 빠져나온다.