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

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

- Stack
- Queue

13 주차 - 3 월 13 일 (화)

12 주차에 tree delect 에서 막혀 stack 부터 완전히 이해가 돼지 않은 상태에서 넘어간 것 같아 stack 을 다시 공부해보았습니다.

Lecture - avl 재귀 풀어보는것 나감.

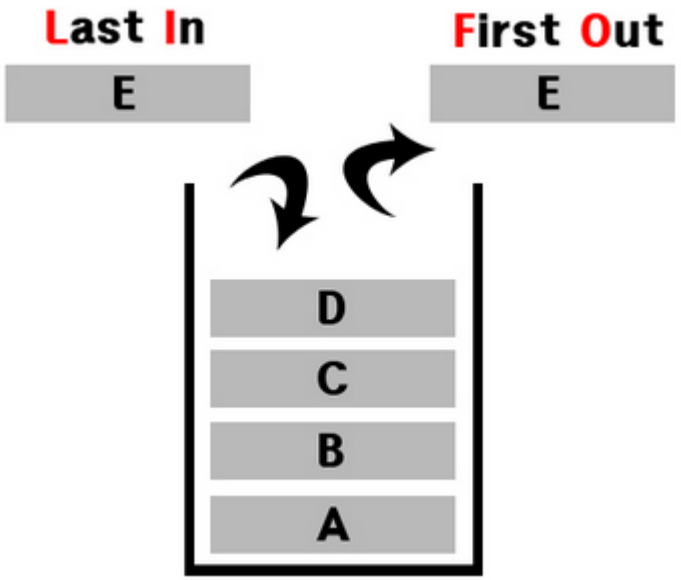
자료 구조란?

컴퓨터에 자료를 효율적으로 어떻게 저장할 것인가?
메모리 절약효과와 자료를 저장하는 데 수행하는 시간도 줄일 수 있다.

선형구조 – 큐, 리스트, 스택
비선형구조 - 트리,그래프

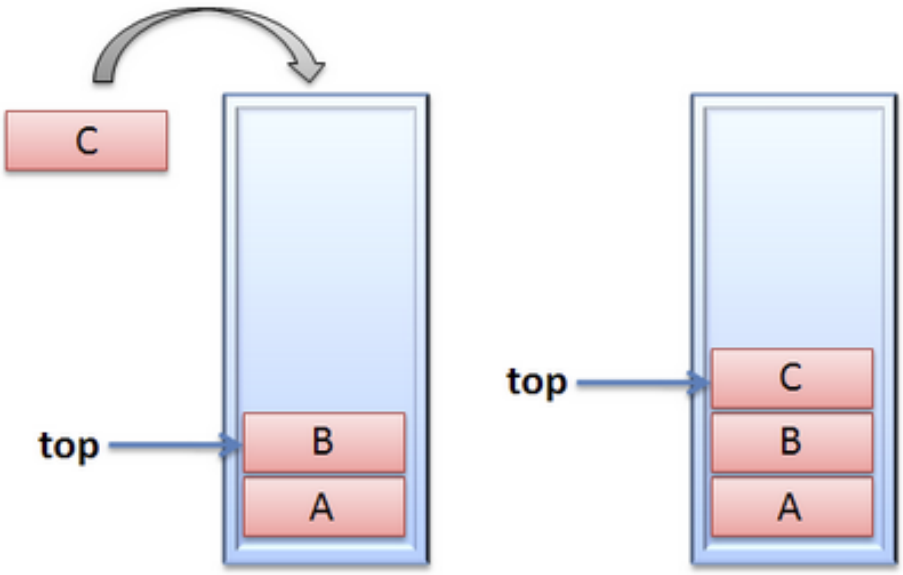
스택(stack)이란?

자료를 한쪽으로 보관하고 꺼내는 **LIFO(Last In First Out)** 자료구조. 스텍에 자료를 보관하는 연산을 **push** 라 말하고, 꺼내는 연산을 **pop** 이라 말한다. 가장 최근에 보관한 위치정보를 **Top or 스택포인터**라 한다.

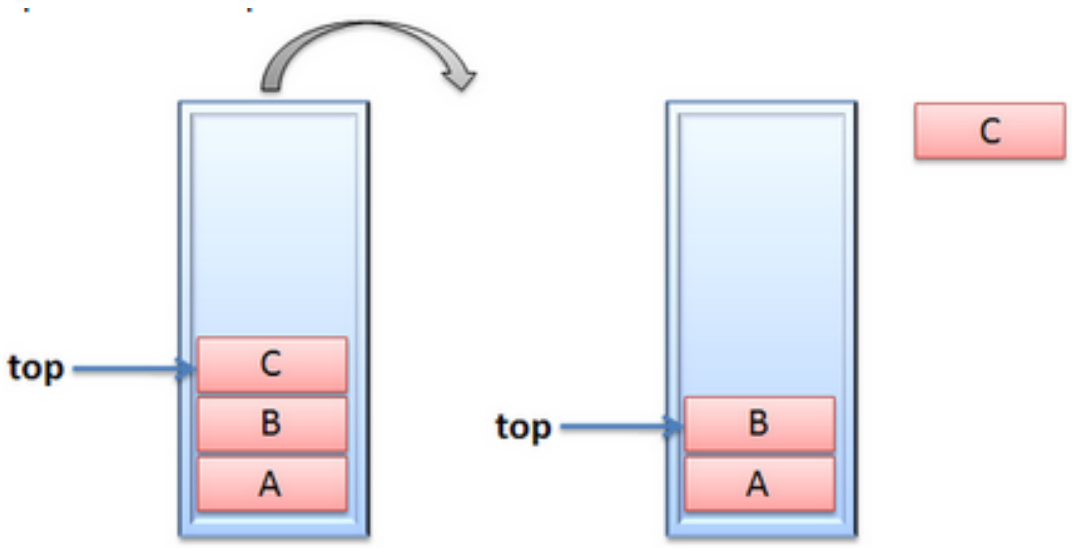


→ 프링글스

push & POP

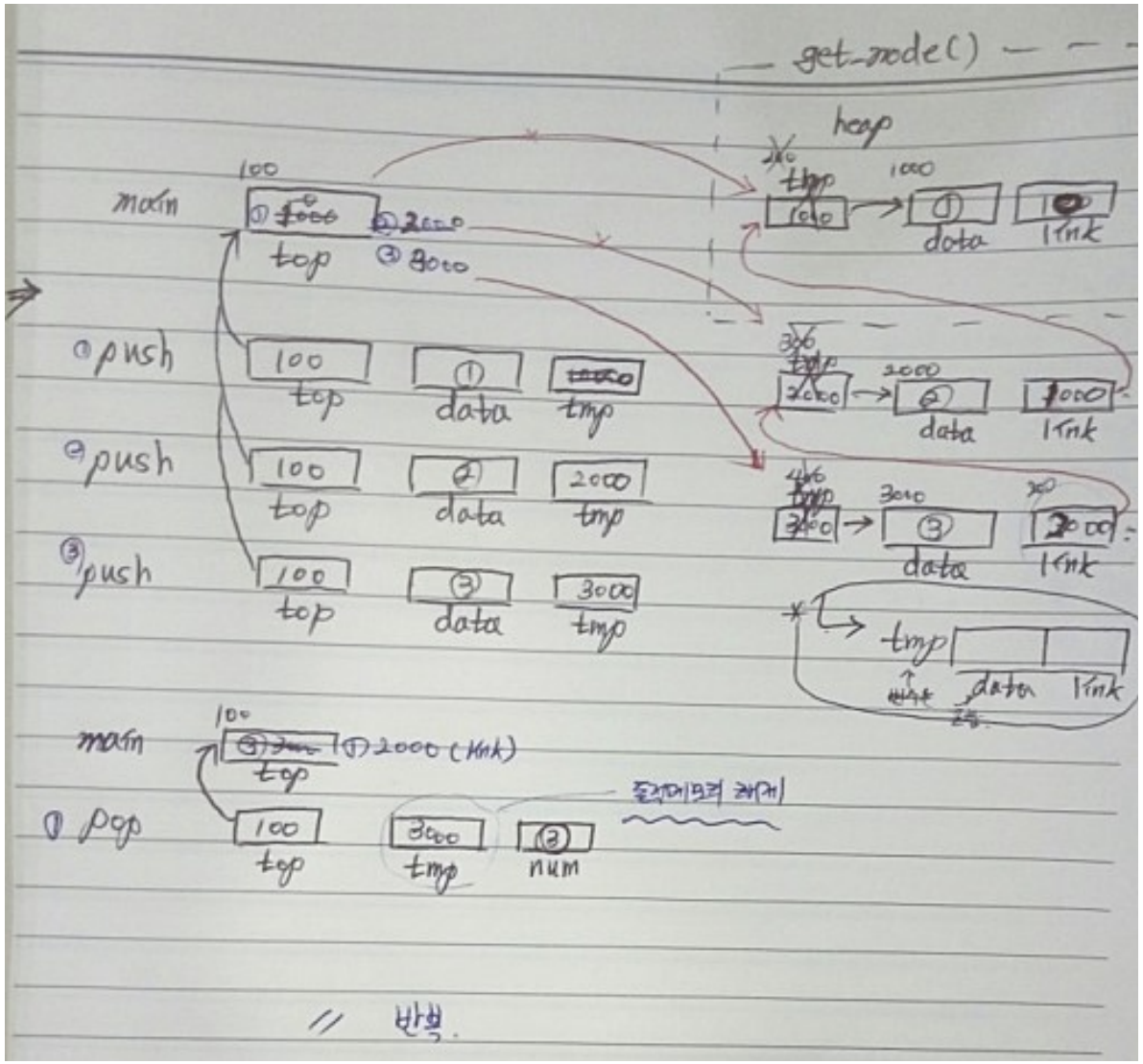


Push Operation



Pop Operation

solv - 그림



stack 중간값을 빼보기

```
#include <stdio.h>
#include <malloc.h>
#define EMPTY 0
struct node{                                    // 구조체 node 정의
        int data;                               // 구조체 멤버 1.
        struct node *link;                      // 구조체 멤버 2. 구조체 포인터 (link)
};
typedef struct node Stack;                      // typedef 를 사용하여 구조체 별칭을 Stack 으로

Stack *get_node()
{
        Stack *tmp;                             // 구조체 포인터 (tmp)
        tmp=(Stack *)malloc(sizeof(Stack));     // 구조체 포인터(tmp) 에 구조체의 크기만큼

        tmp->link=EMPTY;                        //(구조체문구)tmp 가 가르키는 link 에 EMPTY(0)을 넣어
        return tmp;                             //malloc()함수 memoery 할당 후 성공하면 주소값 반환
}                                               //실패하면 NULL 반환.
```

Push ,pop 그대로

```
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;
        }
        else
        {
                *top = (*top)->link;
                free(tmp);
                return num;
        }
}
```

```
int find(Stack **top, int data)
{
	Stack *tmp;
	tmp=*top;
	int num;
	if((*top)->data != data)
	{
		find(&(*top)->link,data);
	}
	else
	{
		num= pop(top);
	}
	return num;
}
```

```
void print_stack(Stack **top)
{
	Stack *tmp=*top;
	while(tmp)
	{
		printf("%d\n",tmp->data);
		tmp = tmp->link;
	}
}
```

```
int main(void)
{
	Stack *top = EMPTY;
	int i;
	for(i=10;i<100;i+=10)
	{
		push(&top, i);
	}
	print_stack(&top);
	find(&top,70);
	printf("\n");
	print_stack(&top);
	return 0;
}
```

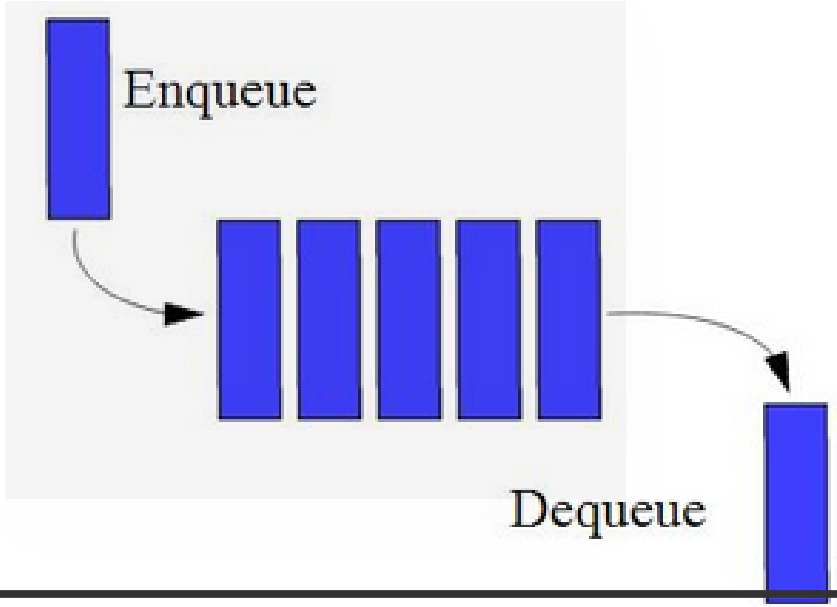
결과 :

```
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs$ vi stack2.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs$ gcc -g -O0 -o debug stack2.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs$ ./debug
90
80
70
60
50
40
30
20
10

90
80
60
50
40
30
20
10
```

Queue

큐는 자료를 한쪽으로 보관하고 다른쪽에서 꺼내는 **FIFO(First In First Out)** 방식의 자료구조이다. 큐에 자료를 보관하는 연산을 **ENQUEUE** or PUT 이고, 꺼내는 연산을 **DEQUEUE** or GET 라고 말한다. 그리고 보관할 위치 정보를 **tail** or rear, 꺼낼 위치 정보를 **head** or front 라고 말한다.



Queue

lec-코드

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct __queue
{
        int data;
        struct __queue *link;

} queue;
queue *get_node(void)
{
        queue *tmp;
        tmp = (queue *)malloc(sizeof(queue));
        tmp->link = NULL;
        return tmp;
                                                }
  void enqueue(queue **head, int data)
  {
        if(*head == NULL)
        {
                *head = get_node();
                (*head)->data = data;
                return;
        }

        enqueue(&(*head)->link, data);
  }
```

내코드

```
#include<stdio.h>
#include <stdlib.h>

typedef struct __queue
{
        int data;
        struct __queue *link;
}Queue;

Queue *get_node()
{
        Queue *tmp;
        tmp=(Queue *)malloc(sizeof(Queue));
        tmp->link = NULL;
        return tmp;
}

void Enqueue(Queue **head,int data)
{

        if(*head==NULL)
        {
                *head=get_node();
                (*head)->data=data;
                return;
        }
        Enqueue(&(*head)->link,data);
        return;
}
```

```
void print_queue(queue *head)
{
        queue *tmp = head;

        while(head)
        {
                printf("head->data = %d\n", head->data);
                head = head->link;
        }
}
```

→ **while 문 이용**

main, head 는 꺼낼 위치 정보, 즉 제일 먼저 들어간 데이터를 가르킨다. 데이터를 가르키는 주소를 print, head 에 넣고, head 에 데이터가 있다면 printf 하여 data 를 꺼내고, head 자리에 head 가 가르키는 link 의 값을 넣는다. link 에는 다음에 쌓인 데이터를 가르키고 있다.

```
void print_queue(Queue *head)
{
        if((head)!=NULL)
        {
                printf("%d\n",(head)->data);
                print_queue(head->link);
        }
}
```

→ **if 문 이용**

```
queue *dequeue(queue *head, int data)
{
        queue *tmp = head;

        if(tmp == NULL)
                printf("There are no data that you delete\n");

        if(head->data != data)
                head->link = dequeue(head->link, data);
        else
        {
                queue *res = head->link;
                printf("Now you delete %d\n", data);
                free(tmp);
                return res;
        }

        return head;
}
```

→ 여기서 else 문을 봐라.
free(tmp) 를 통해 먼저 메모리를 해제했다. 그러면 이상하게도 컴파일이 잘된다. 이는 컴파일러가 좋아져 메모리를 해제하기전에 그 값을 알아서 보낸 것으로 보인다.

```
Queue *Dequeue(Queue *head,int data)
{
        Queue *tmp;
        tmp=head;
        if(head->data!=data)
                head->link=Dequeue(head->link,data);
        else
        {
                tmp=head->link;
                free(head);
        }
        return tmp;

}
```

-> lec 코드의 res 변수처럼 링크값을 저장할 변수 tmp 를 만들고, 그 후에 메모리를 해제한후 tmp 를 리턴한다.

```c
int main(void)
{
	int i;

	queue *head = NULL;

	srand(time(NULL));

	for(i = 0; i < 3; i++)
		enqueue(&head, (i + 1) * 10);

	print_queue(head);

	head = dequeue(head, 10);

	print_queue(head);

	return 0;
}
```

```c
int main(void)
{
        Queue *head=NULL;
        int i;
        for(i=0;i<80;i+=10)
        Enqueue(&head,i);
        print_queue(head);
        head=Dequeue(head,40);
        printf("\n");
        print_queue(head);
        return 0;

}
```

결과

```
head->data = 10
head->data = 20
head->data = 30
head->data = 40
head->data = 50
Now you delete 30
head->data = 10
head->data = 20
head->data = 40
head->data = 50
```

enq

```
0
10
20
30
40
50
60
70
```

deq

```
0
10
20
30
50
60
70
```

Queue 재귀풀기

```
void enqueue(queue **head, int data)
{
        if(*head == NULL)
        {
                *head = get_node();
                (*head)->data = data;
                return;
        }

        enqueue(&(*head)->link, data);
}
```

```
void Enqueue(Queue **head,int data)
{
        while(1)
        {

        if(*head==NULL)
        {
                *head = get_node();
                (*head)->data=data;
                 break;
        }
        else
                head=&(*head)->link;
        }
}
```

→ 재귀함수가 어떤 조건일 때, 재귀를 벗어나는지 생각해보자.
head 가 가르키는 값이 NULL 일때, 재귀를 벗어난다.

```
queue *dequeue(queue *head, int data)
{
	queue *tmp = he:ad;

	if(tmp == NULL)
		printf("There are no data that you delete\n");

	if(head->data != data)
		head->link = dequeue(head->link, data);
	else
	{
		//queue *res = head->link;
		printf("Now you delete %d\n", data);
		free(tmp);
		return head ->link;
	}

	return head;
}
```

```
Queue *Dequeue(Queue *head,int data)
 36 {
 37         Queue *tmp;
 38         tmp=head;
 39         while(head->data!=data)
 40         {
 41                 Queue *res=head->link
 42                 head->link=res->link;
 43         }
 44         if(head->data==data)
 45         {
 46
 47                 tmp->link=head->link;
 48                 free(head);
 49         }
 50
 51         return  ;
 52
 53 }
 54
```

과목 번호표기

c 문제 30 문제
자료구조 10 문제

주는문제 10 개

github 제출할 내용 : 문제푼것 pdf 로 제출 5 시 45 분까지
TMS 에도 올릴것