

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 되지 않은 상태에서 넘어간 것 같아 stack 을 다시 공부해보았습니다.

Lecture - avl 재귀 풀어보는것 나감.

자료 구조란?

컴퓨터에 자료를 효율적으로 어떻게 저장할 것인가?

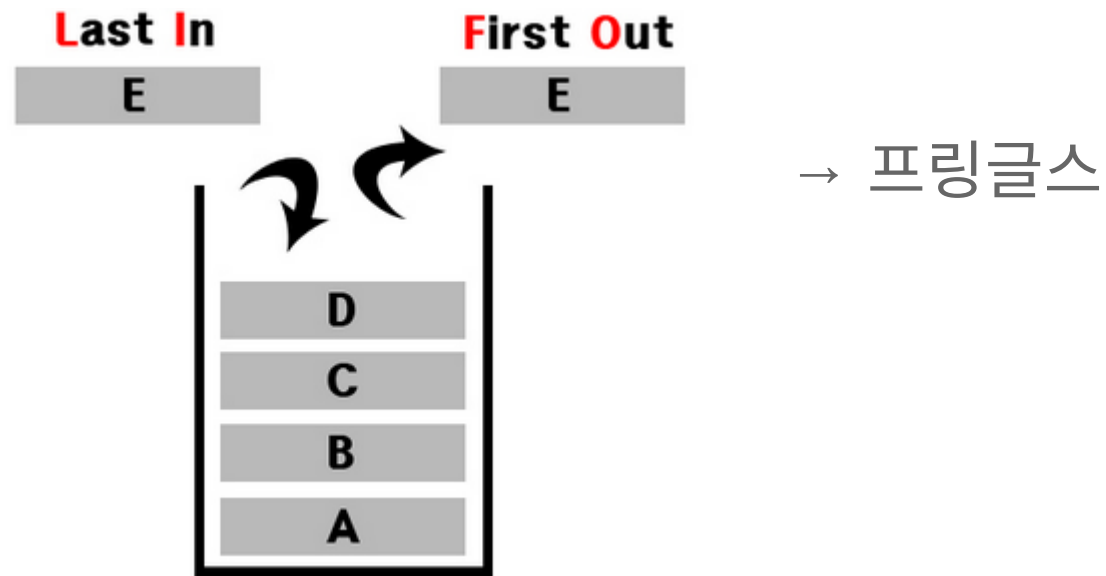
메모리 절약효과와 자료를 저장하는 데 수행하는 시간도 줄일 수 있다.

선형구조 - 큐, 리스트, 스택

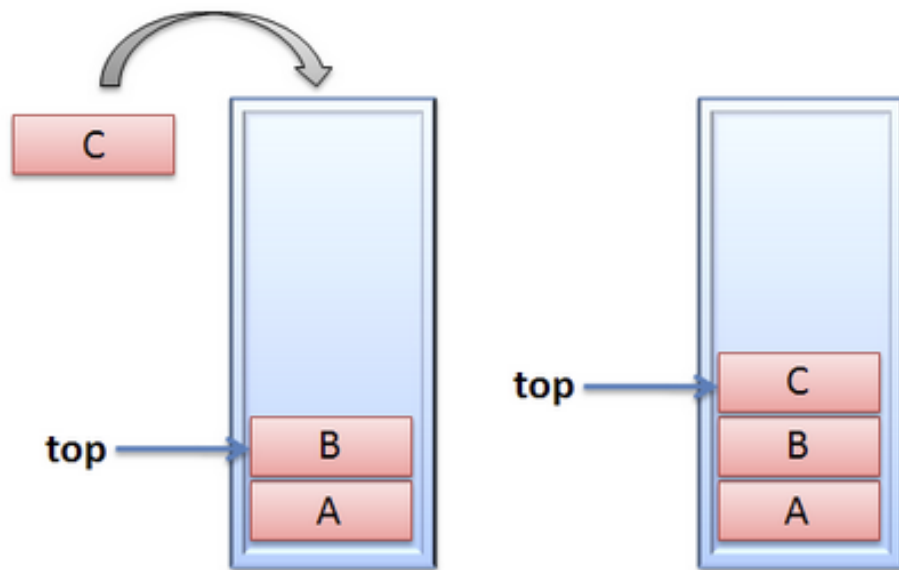
비선형구조 - 트리, 그래프

스택(stack)이란?

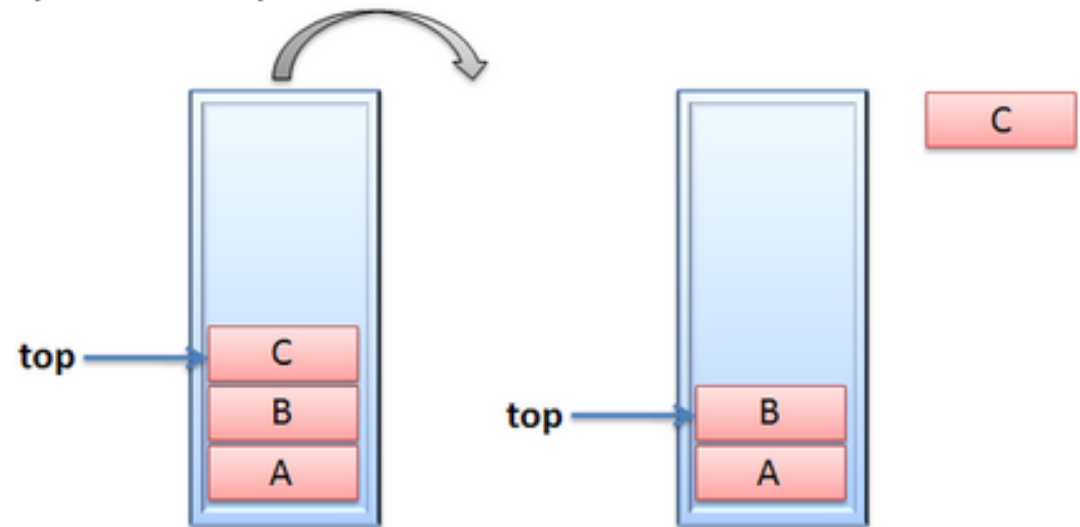
자료를 한쪽으로 보관하고 꺼내는 **LIFO(Last In First Out)** 자료구조. 스택에 자료를 보관하는 연산을 **push** 라 말하고, 꺼내는 연산을 **pop** 이라 말한다. 가장 최근에 보관한 위치정보를 **Top or 스택포인터**라 한다.



push & POP

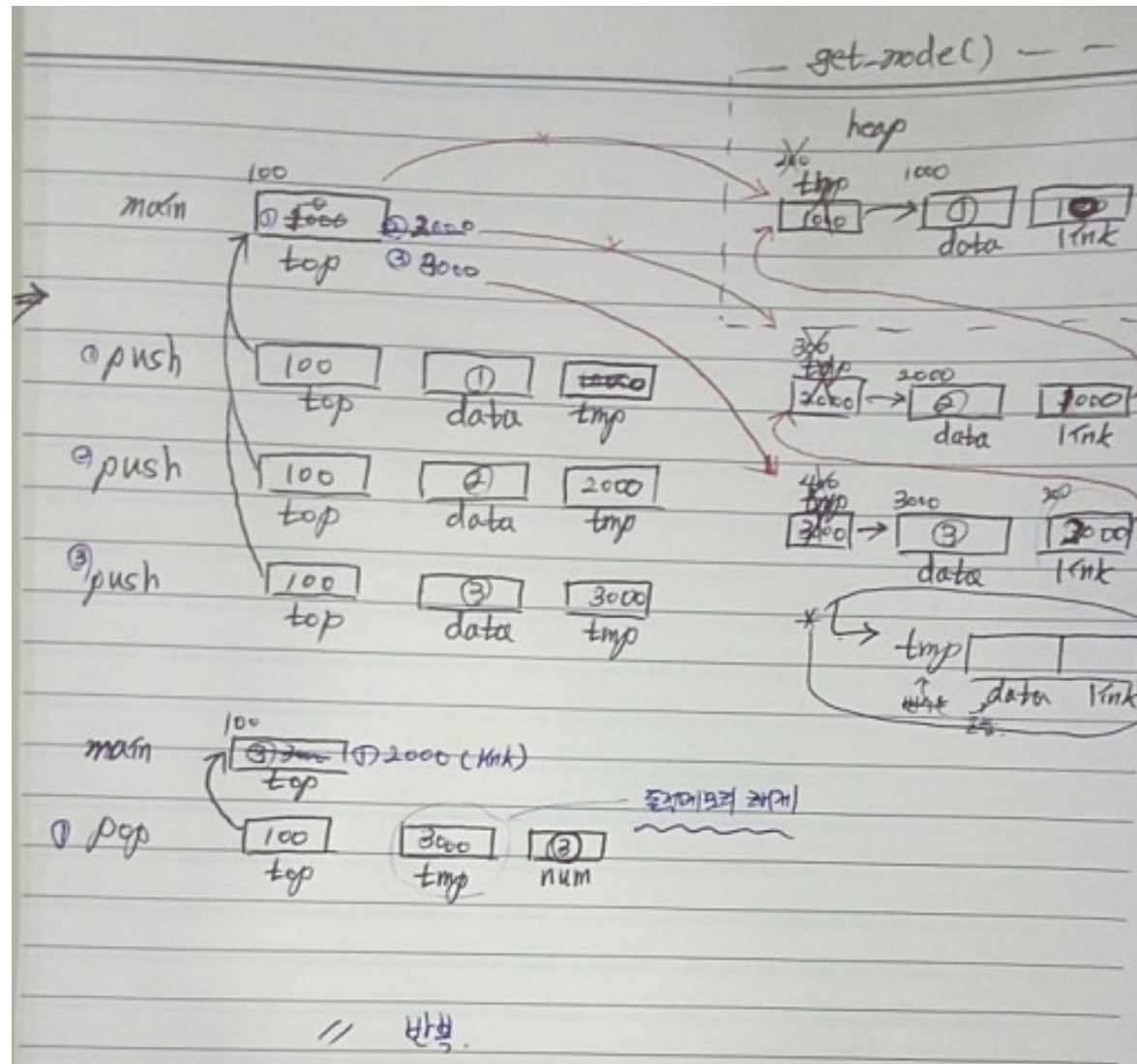


Push Operation



Pop Operation

solv - 그림



stack 중간값을 빼보기

```
#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;
}
```

// 구조체 node 정의
// 구조체 멤버 1.
// 구조체 멤버 2. 구조체 포인터 (link)

// typedef 를 사용하여 구조체 별칭을 Stack 으로

// 구조체 포인터 (tmp)
// 구조체 포인터(tmp) 에 구조체의 크기만큼

//(구조체문구)tmp 가 가르키는 link 에 EMPTY(0)을 넣어
//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 :

```

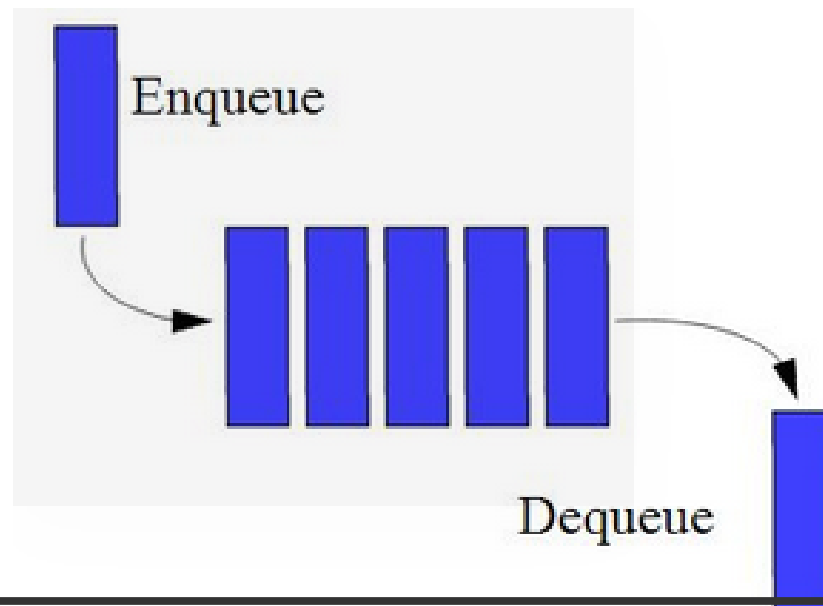
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs$ vi stack2.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs$ gcc -g -O0 -o debug stack2.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs$ ./debug
90
80
70
60
50
40
30
20
10

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

```
#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);
}
```

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

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

```

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 head->link;
    }

    return head;
}

```

```

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;
}

```

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
```

재귀 푸는 방식 진행하겠습니다.