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

자료구조

1. Stack
   a) 정의
   b) 예제

2. Queue
   a) 정의
   b) 예제

3. TREE
   a) 정의
   b) 예제

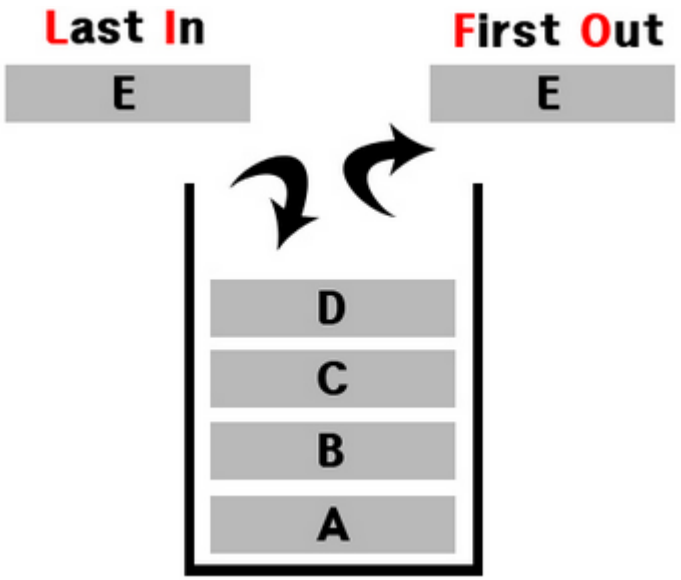
자료 구조란?

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

선형구조 – 큐, 리스트, 스택
비선형구조 - 트리,그래프

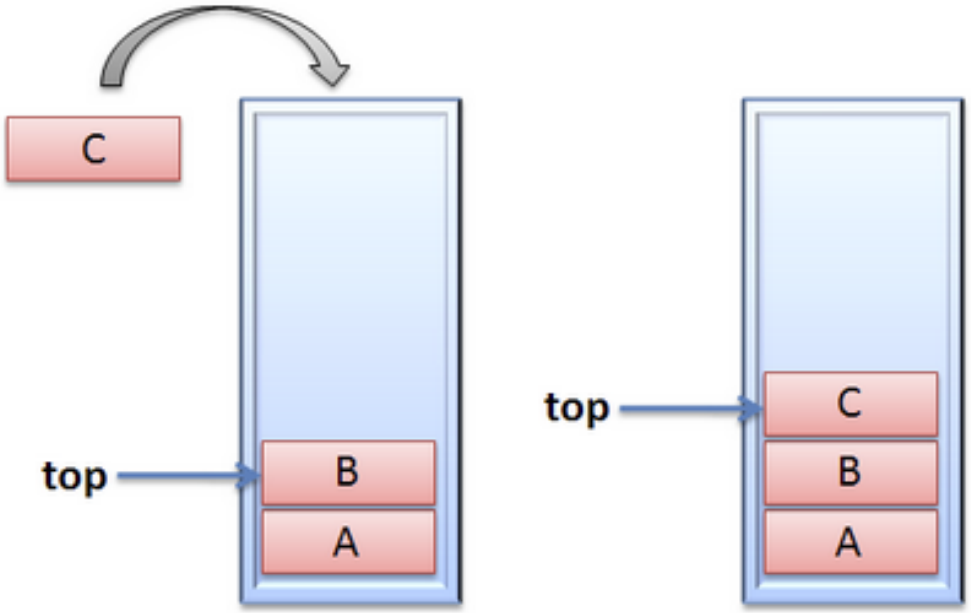
스택(stack)이란?

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.



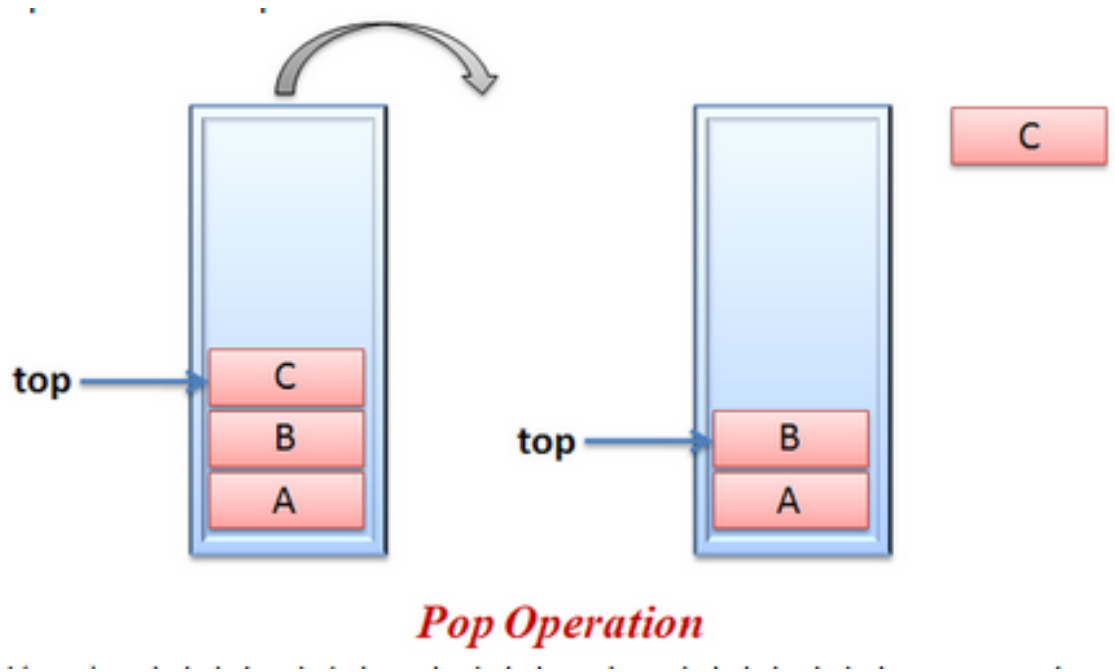
→ 프링글스

push



Push Operation

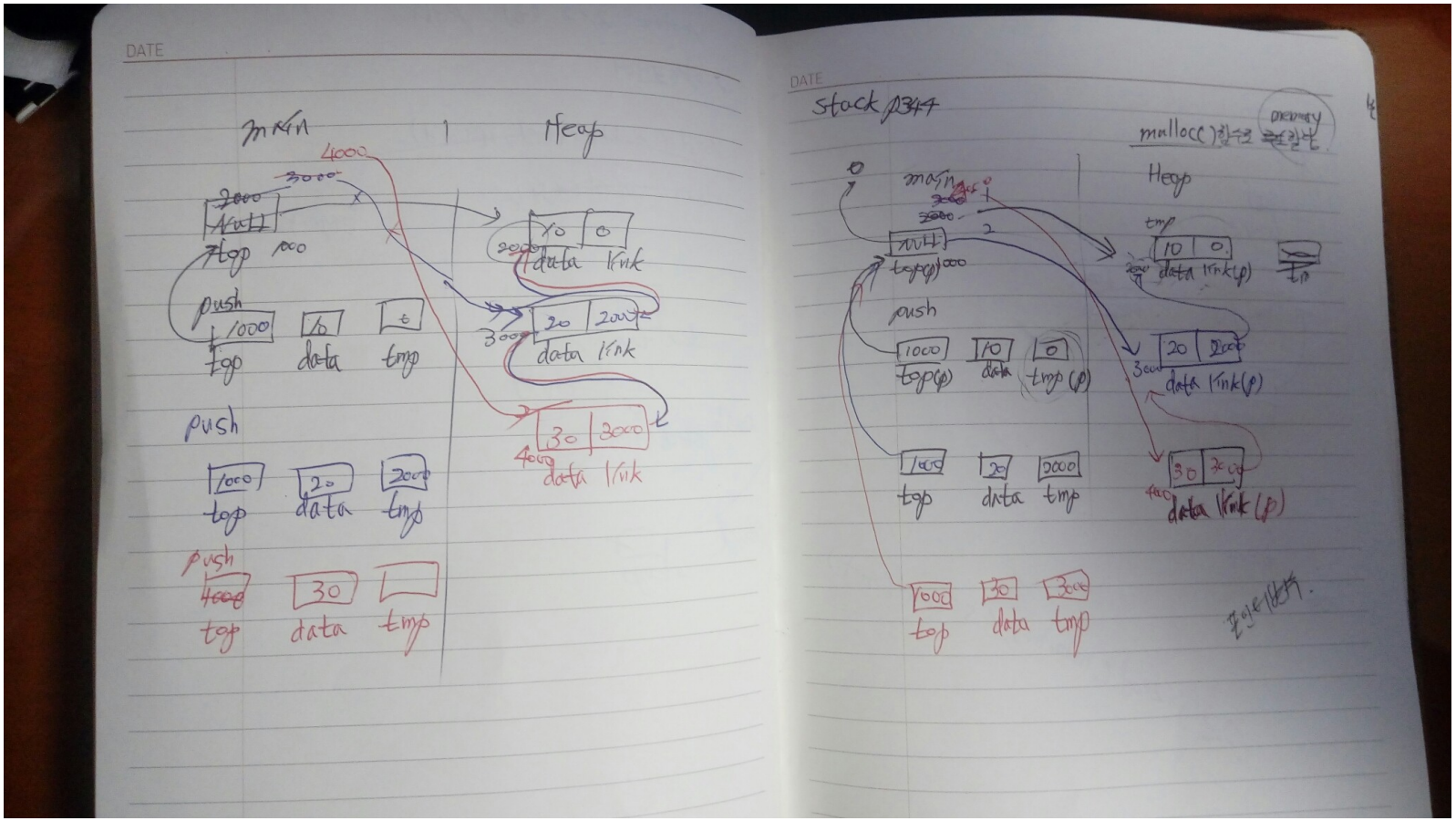
pop



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

#define EMPTY 0

struct node{                                    // 구조체 node 선언
        int data;                               //  int 형 data 변수 선언
        struct node *link;                      //  구조체 포인터 link 변수 선언 ( stack 을 가르킬수 있다.)
};
typedef struct node Stack;                      // 구조체 node → Stack 별칭으로 선언

Stack *get_node()                               // 구조체 포인터 get_node 함수 선언
{
        Stack *tmp;                             // 구조체 포인터  tmp 변수 선언
        tmp=(Stack *)malloc(sizeof(Stack));     // (stack 형)의 동적메모리를 (구조체 stack)의 크기만큼 할당하고,  tmp 에 반환하는 주소값 저장
        tmp→link=EMPTY;                         // 구조체 포인터 tmp 안에 link 에 접근하여 0 을 넣어라.
        return tmp;                             //  tmp 반환. → 할당된 메모리 주소값 반환
}

void push(Stack **top, int data)                // 반환값이 없는 push 함수 생성,  **top, int 형 data 변수 선언
{
        Stack *tmp;                             // 구조체 포인터 선언 ( tmp 는 stack 을 가르킬 수 있다)
        tmp = *top;                             // push 함수의 top 변수가 가르키는 main top 의 값을  tmp 변수에 넣어라.
        *top = get_node();                      // push 함수의 top 변수가 가르키는 곳에 get_node()함수의 리턴값을 넣어라.
        (*top)->data = data;                    // *top 이 가르키고 있는 할당된 heap memoy 의 (tmp) data 에 접근하여 data 값을 넣어라.
        (*top)->link = tmp;                     // “
```

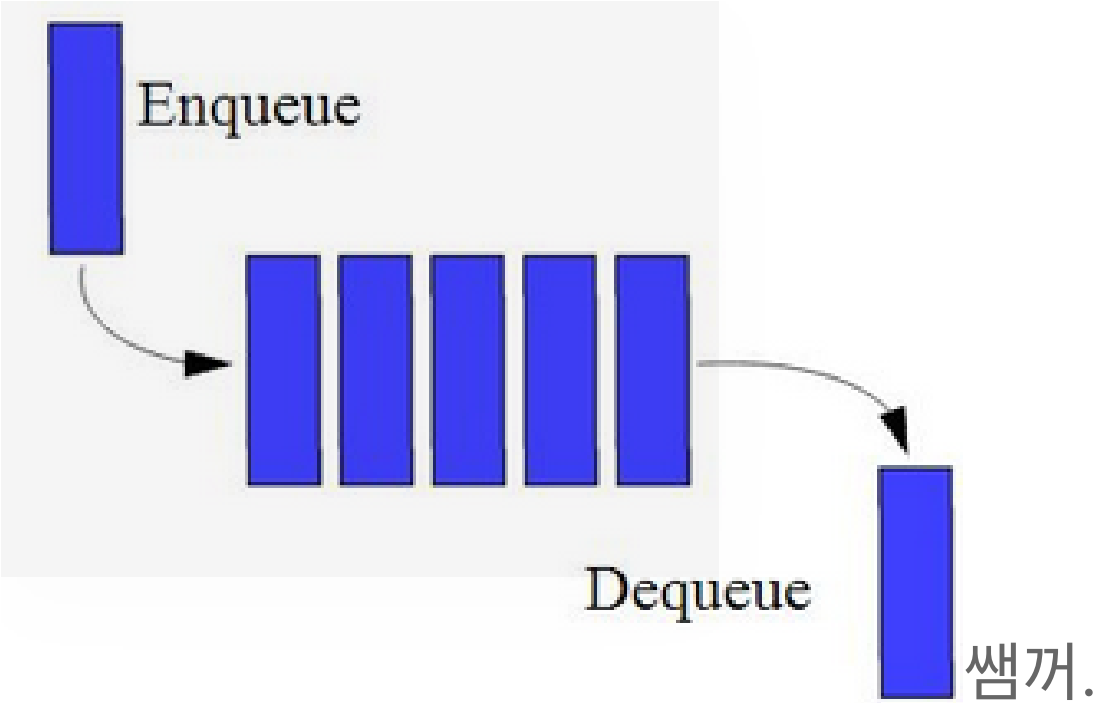
```
}
int pop(Stack **top)                    // 반환값 int 형을 가진 pop 함수 생성, 이중포인터 top 변수 선언
{       Stack *tmp;                     // 구조체 포인터 변수 선언, tmp
        int num;                        // int 형 변수 선언, num
        tmp = *top;                     // pop 함수의 top 변수가 가르키는 곳의 값을 tmp 변수에 넣어라.
        if(*top == EMPTY)
        {
                printf("Stack is empty!!!\n");
                return 0;
        }
        num = tmp→data;                 //  구조체 포인터 tmp 변수가 가르키는 곳 안에 data 에 접근
        *top = (*top)→link;             //  구조체포인터 top 변수가 가르키는 곳 안의 data 에 접근
        free(tmp);                      //  동적 메모리 해제
        return num;
}
int main(void)
{
        Stack *top = EMPTY;    // top          // main 의 구조체포인터 변수 생성 ( top 은 stack 을  가르킬 수 있다.)
        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;
}
```

solv


DATE
main
Heap
4000
3000
2000
NULL
top
push
1000
10
0
data
tmp
20
2000
link
30
push
top
data
tmp
DATE
stack 영역
malloc() 함수로 할당
Heap
main
tmp
top(p)
tmp(p)
data link(p)
free

큐란?

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.



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

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

queue *get_node()
{
	  queue * node;
	  node = (queue*)malloc(sizeof(queue));
	  node->link = NULL;
	  return node;
}

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

```
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,20);
	print_queue(head);
	return 0;
}
```

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

```
typedef struct node{
    int data;
    struct node *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;
          printf("data=%d 을 저장합니다.  heap/data= %d,heap/link=%p 입니다.\n",data,(*head)->data,*head);
          return;
       }
       ENQUEUE(&(*head)->link,data);
}

Queue *DEQUEUE(Queue *head,int data)
{
```

```
    Queue *tmp;
    if(head->data!=data)
    {
      head->link= DEQUEUE(head->link,data);
      printf("data 가 %d 가 아닐때,다음 주소%p 를 가르키고, heap/data= %d,heap/link=%p 입니다 \n",data,&(head->data),head-
>data,head->link);
    }
    else
    {
//           tmp=head->link;
           printf(" data = %d 삭제합니다.\n",data);
           free(head);
           printf("삭제후 heap/link=%p 를 가르킵니다.\n",head->link);

    }
    return tmp;
}

int main(void)
{
  Queue *head=NULL;
  ENQUEUE(&head,10);
  ENQUEUE(&head,20);
  ENQUEUE(&head,30);
  printf("data=%d 을 삭제하고 싶어요.\n",30);
  head=DEQUEUE(head,30);
  return 0;
 }
```

재귀호출하지않고 내가짠대로하면 쓸때없는 레지스터가 많이 들어감.. head 를 계속 유지하고 link 를 head 가 가르키는 다음주소를 넣으려고하니 스택에서 못벗어났음..
–> ENQUEUE 는 쌤껄로..

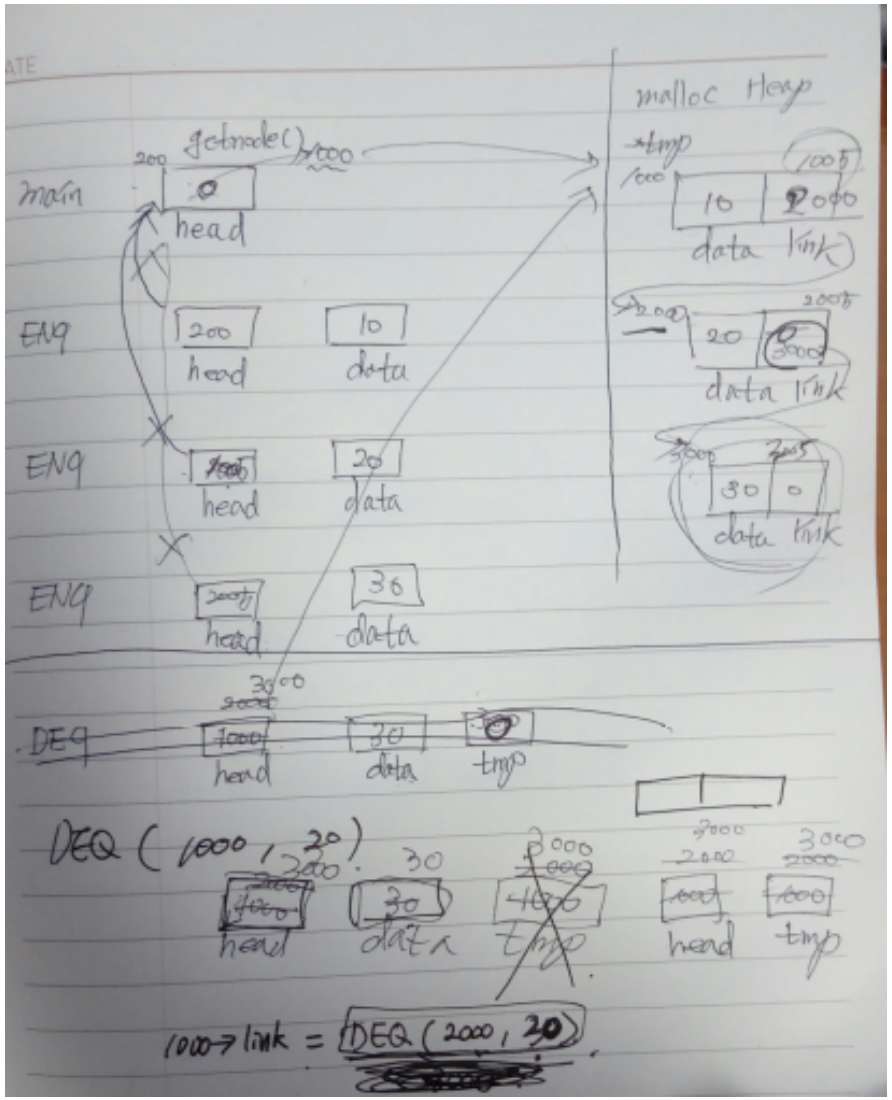
이중포인터 dequeue

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

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

if((*head)→data !=data)
    dequeue(&(*head)→link,data);

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

int main(void)
{
```

```
int I;
}



////////////////////////////////////find max/////////////////////////////

tree *find_max(tree *root, int *data)
{
  if(root → right)
      root → right = find_max(root→right, data);
  else
  {
        *data = root→data;
        root = chg_node(root);
  }

  return root;
}

tree *delete_tree(tree *root, int data)
{

 int num;
tree *tmp;
if(root == NULL)
{
 printf(“Not Found \n”);
return NULL;
```

}

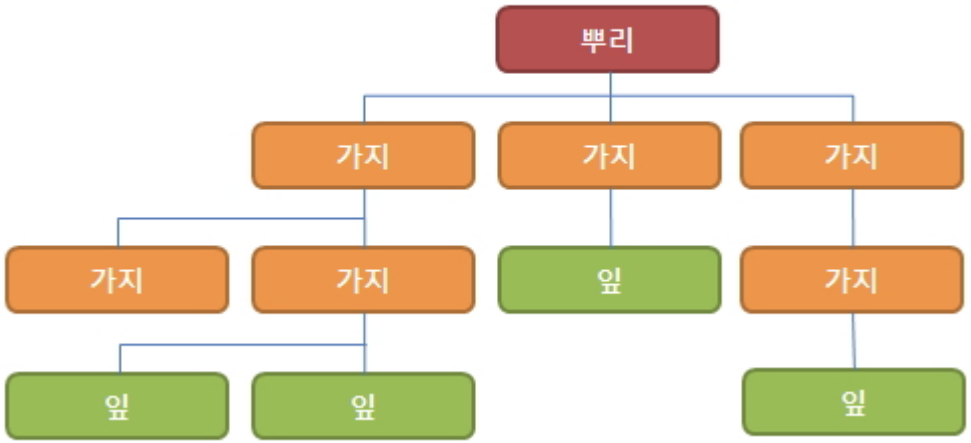
solv


malloc Heap
getnode()
main
head
ENQ
head
data
ENQ
head
data
ENQ
head
data
DEQ
head
data
tmp
DEQ (1000 , 20)
head
data
tmp
head
tmp
1000→link = DEQ (2000 , 20)

트리란?

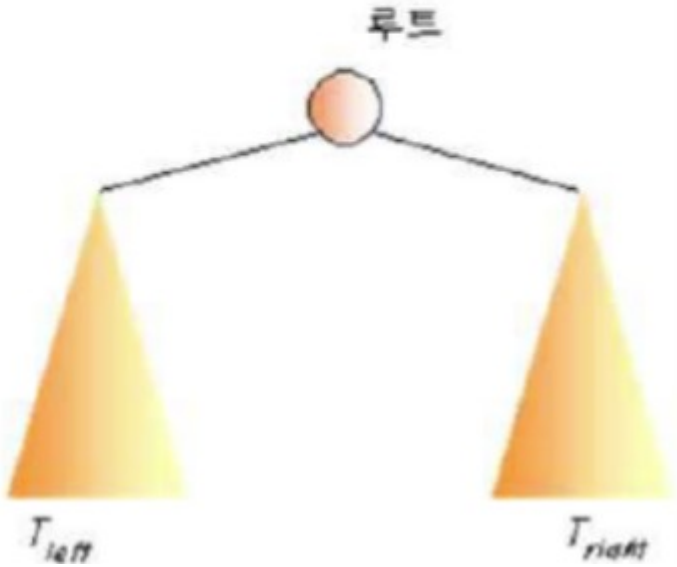
서 보인 큐(Queue), 스택(Stack) 등은 자료구조에서 선형 구조라 한다. 선형 구조란 자료를 구성하고 있는 데이터들이 순차적으로 나열시킨 형태를 의미합니다. 이번에 배울 트리는 비선형 구조이다. 비선형 구조는 선형구조와는 다르게 데이터가 계층적(혹은 망)으로 구성되어있다. 선형구조와 비선형구조의 차이점은 구성형태뿐만 아니라 용도에서도 차이점이 들어나는데, 선형구조는 자료를 저장하고 꺼내는 것에 초점이 맞춰져 있고, 비선형구조는 표현에 초점이 맞춰져 있다. 스택이나 큐 구조는 자료의 갯수가 많을 경우에 탐색 시간이 너무 많기 때문에 트리구조를 배운다.

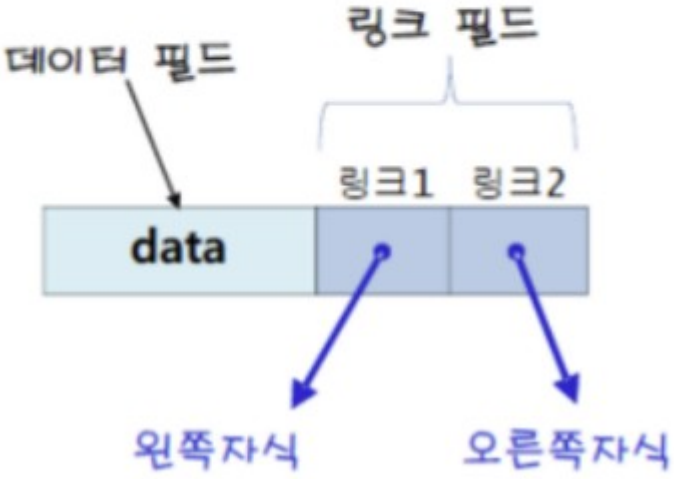
트리는 뿌리(Root), 가지(Branch), 잎(Leaf) 세가지 요소로 이루어져 있다.



이진트리

- 각 노드에는 최대 2개까지의 자식 노드가 존재
- 모든 노드의 차수가 2 이하가 된다-> 구현하기가 편리함
- 서브 트리간의 순서가 존재 (왼쪽, 오른쪽)



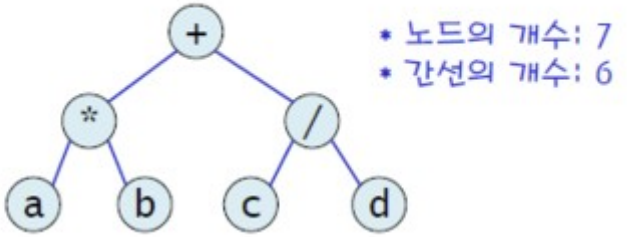


→ 모든 노드가 2 개의 서브트리를 갖는 트리, 서브트리는 공집합일 수 있다.

→ 왼쪽과 오른쪽이 반드시 서로 구별되어야함.

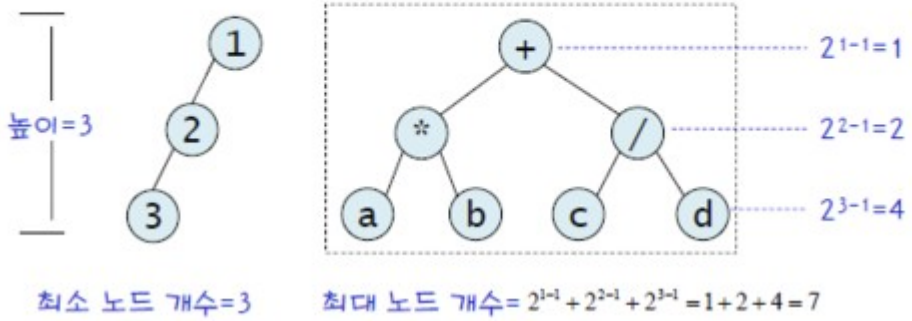
성질 1

노드의 개수가 n개이면 간선의 개수는 n-1



성질 2

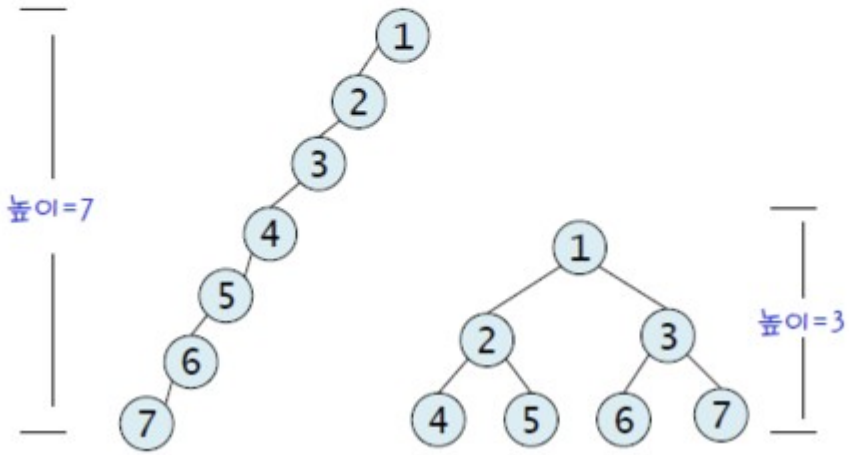
높이가 h인 이진트리의 경우, 최소 h개의 노드를 가지며
최대 2h-1개의 노드를 가진다.



성질 3

n개의 노드를 가지는 이진트리의 높이는 최대 n이거나 최소

$$\lceil \log_2(n+1) \rceil$$



참고: https://blog.naver.com/qwdqwd910/221082859112

이진트리

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

#define EMPTY 0

typedef struct tree
{
	int data;
	struct tree *left;
	struct tree *right;
}Queue;

Queue *get_node()
{
	Queue *tmp;
	tmp=(Queue *)malloc(sizeof(Queue));
	tmp→left = EMPTY;
	tmp→right= EMPTY;
	return tmp;
}

void ENQUEUE(Queue **head,int data)
{
	if(*head == EMPTY)
	{
		*head=get_node();
		(*head)→data=data;
		return;
```

```
        }

        else if(data>(*head)->data)
        {
                EQUEUE(&(*head)→right,data);
        }
        else if(data<(*head)->data)
        {
                EQUEUE(&(*head)→left,data);
        }
        else if(data==(*head)→data)
        {
                return;
        }
}

int print_queue(queue *head)
{
        if(head)
        {
                printf(“data=%d\n”,head→data);
                print_queue(head→left);
                printf_queue(head->right);
        }
}
```

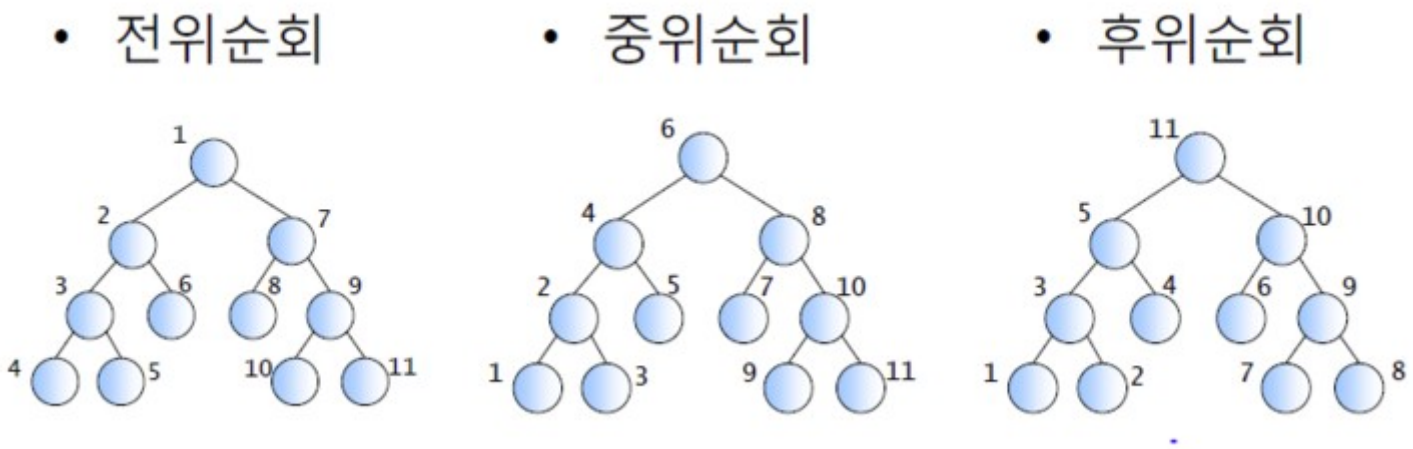
```
int main(void)
{
	Queue *head = EMPTY;

	int I;
	int arr[13] = {50,45,73,32,48,46,16,37,120,47,130,127,124};
	for(i=0;i<13;i++)
	{
		ENQUEUE(&head,arr[i]);
	}

	printf_queue(head);
	return 0;
}
```

sol

Queue 와 비슷하다.

DETREE → 진행중

→ DEQUEUE 처럼 짜보았지만, 반복루트를 타고 꼬여버리게된다.
CASE 문을 사용해서 나눠주고 재귀호출해봐야겠다.

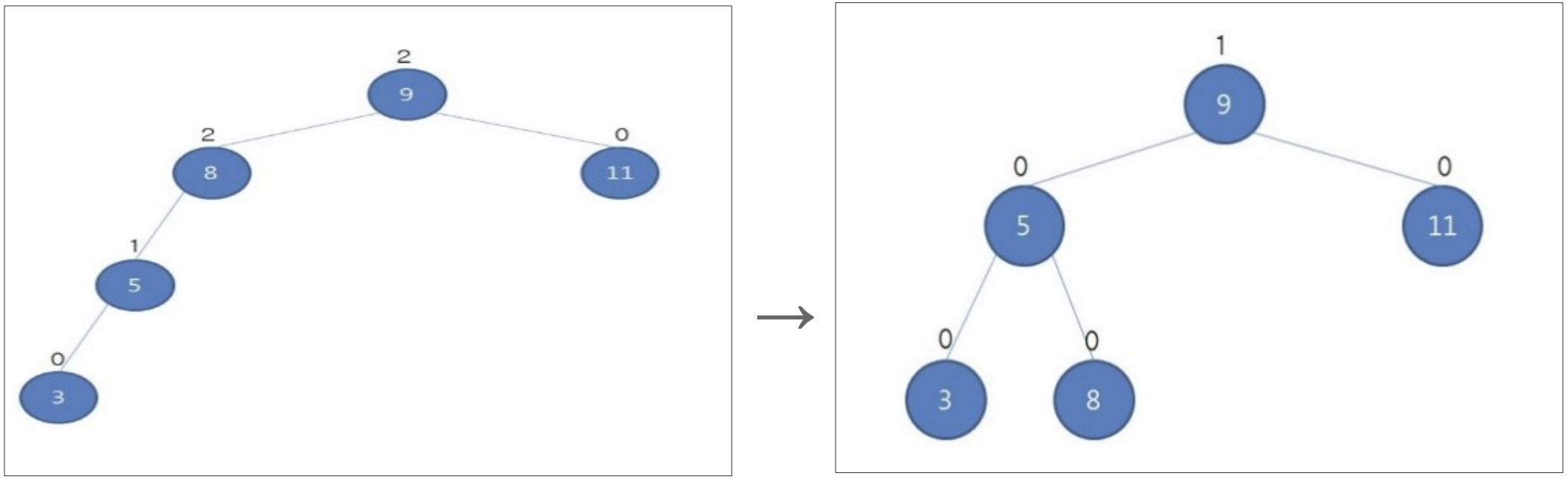


트리를 구현하는 방법은 주로 링크 표현법을 이용한다.
이진트리의 순회라는 개념이다. 트리를 순회한다는 것은 트리에 속하는 모든 노드를 한번씩 방문하여 노드가 가지고 있는 데이터를 목적에 맞게 처리하는 것을 의미한다. 우리가 트리를 사용하는 목적은 트리 노드에 자료를 저장하고, 필요에따라 이 자료를 처리하기 위해서이다.

AVL 트리

이진트리의 찾는 효율을 올리기위해 만듬.

모든 노드의 서브트리 높이 차이가 1 이하라는 점이다. 만약 높이 차이가 2 이상이 된다면 노드들은 스스로 재배치하여 균형 상태를 유지한다. (=1 의 차이가 나게 만든다)



참고: https://blog.naver.com/ellay06/120171022413