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

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

* TREE

스택이나 큐 같은 구조는 자료의 갯수가 많을 경우에 탐색의 시간이 너무 많다.
따라서 트리구조를 만들게 된다.

기본트리
AVL 트리 - 검색 속도가 가장 빠르다.
RB 트리 - 구글 등 사용

```
void tree_ins(tree **root, int data){
        if(*root==NULL){
                *root = get_node();
                (*root)→data =data;
                return;
        }
        else if((*root)→data >data)
                tree_ins(&(*root)->left,data);
        else if((*root)→data < data)
                tree_ins(&(*root)→right , data);
} //트리 삽입

void print_tree(tree *root){
        if(root){
                printf("data = %d, ",root→ data);
                if(root->left)
                        printf("left=%d, ",root->left->data);
                else
                        printf("left = NULL, ");

                if(root->right)
                        printf("left=%d, ",root->right->data);
                else
                        printf("left = NULL, ");
```

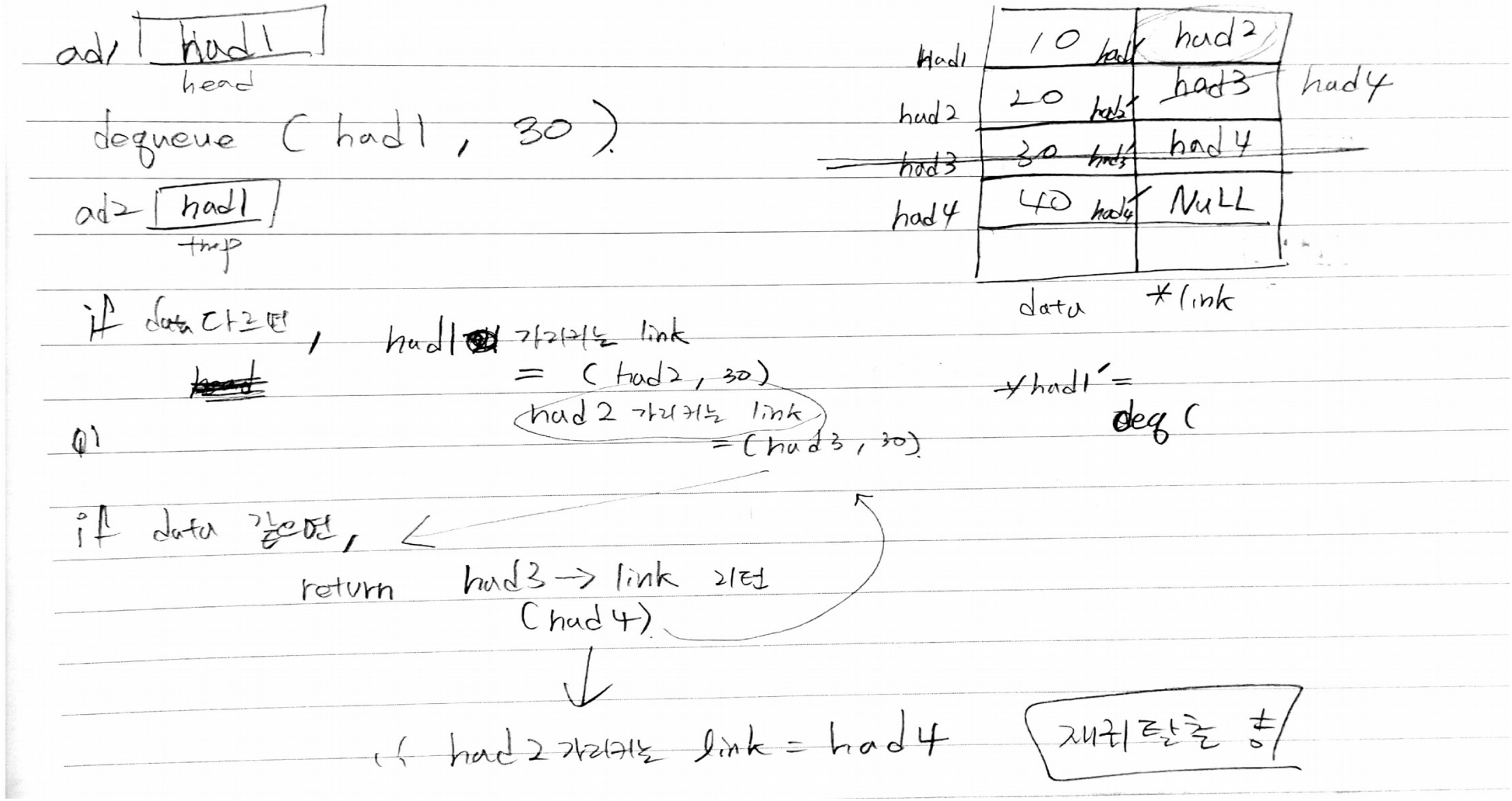
```
                print_tree(root→left);
                print_tree(root→right);
        }
} // 트리 표현

int main(void){
int I;
int data[14]={…};
tree *root = NULL;
for(i=0;data[i];i++){
        tree_ins(&root,data[i]);

print_tree(root);
delete_tree(root,50);

printf_tree(root);
return 0;
}
```

---

dequeue 그림,

```
queue *dequeue(queue *head, int data){
 40    queue *tmp = head;
 41    if(tmp == NULL)
 42       printf("There are no data that you delete\n");
 43    if(head->data != data)
 44       head->link = dequeue(head->link,data);
 45    else{
 46
 47       printf("Now you delete %d \n",data);
 48       free(tmp);
 49       return head->link;
```

```
50    }
51 }
```


add
had1
head
dequeue (had1, 30)
add
had1
temp
if data 다르면, had1 가리키는 link = (had2, 30)
had2 가리키는 link = (had3, 30)
if data 같으면,
return had3 -> link 리턴 (had4)
if had2 가리키는 link = had4
재귀호출을 통해
had1
had2
had3
had4
10
20
30
40
had2
had3
had4
NULL
had4
data
*link
*had1' =
deq (

TREE

```
void tree_ins(tree **root, int data){
        if(*root==NULL){
                *root = get_node();
                (*root)→data =data;
                return;
        }
        else if((*root)→data >data)
                tree_ins(&(*root)->left,data);
        else if((*root)→data < data)
                tree_ins(&(*root)→right , data);
} //트리 삽입
```

void tree_ins (tree **root, int data)

add1 , n1 had2

if (*root == null)

NULL

add1 NULL

*root

add1 had1

*root

had1 n1 NULL null

had1' had1''

Void tree_ins (tree **root , int data)

add1 n2

else if (had1→data > data)

n1 n2

~~Void~~ tree_ins (&(*root)→left , data)

had1→left

&(NULL)

had1' , data

~~if (*root == null)~~

NULL

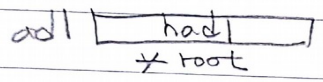
had2 n2 null/null

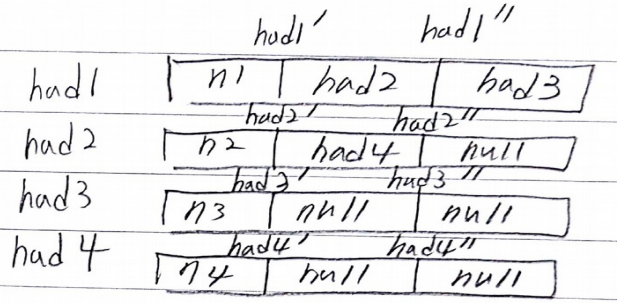
had2' had2''

반복

```
void print_tree(tree *root){
        if(root){
                printf("data = %d, ",root→ data);
                if(root->left)
                        printf("left=%d, ",root->left->data);
                else
                        printf("left = NULL, ");

                if(root->right)
                        printf("left=%d, ",root->right->data);
                else
                        printf("left = NULL, ");

                print_tree(root→left);
                print_tree(root→right);
        }
} // 트리 표현
```

```
void print_tree (tree *root)
                 had1
```





~~Print~~

had1 -> data , (n1) 출력

print_tree (had2)

had2 -> data , (n2) 출력

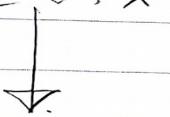
print_tree (had4)

had4 -> data (n4) 출력

print_tree (null)

print

null일 경우 X



리턴, return

print_tree (root -> right)

had3

had3 -> data (n3) 출력

∴ n1, n2, n4, n3 출력