TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 최대성
c3d4s19@naver.com

2018.03.07. – 009일차 수업

---

* 자료구조 이진 트리

구조체 정의

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

typedef int Data;
typedef struct _tree{
	Data data;
	struct _tree* Right;
	struct _tree* Left;
}Tree;
```

새로운 트리 생성 함수

```
Tree* getNewTree(){
	Tree* newTree = (Tree*)malloc(sizeof(Tree));
	newTree->Right = NULL;
	newTree->Left = NULL;
	return newTree;
}
```

새로운 데이터 삽입 함수(재귀 이용)

```
void insertDataToTree(Tree** tree, Data data){
	Data cmpData;
	if( *tree == NULL ){
		*tree = getNewTree();
		(*tree)->data = data;
	}
	else if( data <= (*tree)->data ){
		insertDataToTree( &((*tree)->Left), data );
	}
	else{
		insertDataToTree( &((*tree)->Right), data );
	}
}
```

선택 트리와 모든 자식노드 삭제 함수(재귀 이용)

```
void deleteAllTree(Tree** delTree){
	if (*delTree != NULL)
	{
		if ((*delTree)->Right != NULL)
		{
			deleteAllTree(&((*delTree)->Right));
		}
		if ((*delTree)->Left != NULL)
		{
			deleteAllTree(&((*delTree)->Left));
		}
		Tree *tmpTree = *delTree;
		*delTree = NULL;
		free(tmpTree);
	}
}
```

트리 내용값 프린트 함수(재귀 이용)

```
void printTree(Tree** Tree){
	if( *Tree != NULL ){
		if ((*Tree)->Right != NULL)
		{
			printTree(&((*Tree)->Right));
		}
		if ((*Tree)->Left != NULL)
		{
			printTree(&((*Tree)->Left));
		}
		printf("%d ₩n", (*Tree)->data);
	}
}
```

트리 내부 특정 데이터 제거 함수(재귀 없이 시도함)

```
// 필요 함수 선언
int deleteTreeData(Tree** tree, Data data);
Tree* findRightMinDataParentTree(Tree** tree);
void delRightNode(Tree** tree);
void delLeftNode(Tree** tree);

// 데이터 찾아서 삭제
int deleteTreeData(Tree** tree, Data data){
	Tree* tmpTree = *tree;
	Tree* tmpParentTree = *tree;
	Tree* delTree = NULL;
	Tree* delParentTree = NULL;

	Data cmpData;

	while (tmpTree != NULL)
	{
		cmpData = tmpTree->data;
		if (data < cmpData) //데이터가 작음(왼쪽 이동)
		{
			if(tmpTree->Left == NULL){
				printf("Have no data: %d",data);
				return -1;  //데이터 없음
			}

			if(data == tmpTree->Left->data){
				delLeftNode(&tmpTree);
				return 1; //제거 성공
			}
			tmpParentTree = tmpTree;
			tmpTree = tmpTree->Left;
		}
		else if (data > cmpData)
		{
			if(tmpTree->Right == NULL){
				printf("Have no data: %d",data);
				return -1;  //데이터 없음
			}

			if(data == tmpTree->Right->data){
				delRightNode(&tmpTree);
				return 1;    //제거 성공
			}
			tmpParentTree = tmpTree;
			tmpTree = tmpTree->Right;
		}
		else     //루트노드에서 바로 데이터 찾음(일치)
		{
			delParentTree = findRightMinDataParentTree(&tmpTree);
			delTree = delParentTree->Left;
			(*tree)->data = delTree->data;
			delLeftNode(&delParentTree);
			return 1;    //제거 성공
		}
	}
	return -1;  //데이터 없어서 삭제 실패
}

//트리 오른쪽 자식노드가 없을 경우 자기 자신(트리 왼쪽 자식노드의 부모노드) 주소값 반환
Tree* findRightMinDataParentTree(Tree** tree){
	if((*tree)->Right == NULL){
		return (*tree);
	}
	Tree* tmpTree = (*tree)->Right;
	Tree* tmpParentTree = (*tree);
	Data cmpData = tmpTree->data;
	//루트 노드 기준 오른쪽 최소값의 노드 주소값 찾기
	while(tmpTree->Left != NULL){
		tmpParentTree = tmpTree;
		tmpTree = tmpTree->Left;
	}
	//루트 노드 기준 오른쪽 최소값의 부모 노드 주소값 반환
	return tmpParentTree;
}

// 지울 오른쪽 자식노드 포인터 인자
void delRightNode(Tree** tree){
	printf("check-1₩n");
	//트리 자체가 NULL값 또는 트리 오른쪽
```

```
삭제할 값이 NULL
    if(*tree == NULL || (*tree)->Right == NULL){
        printf("*tree = NULL || (*tree)->Right ==
NULL₩n");
    }
    Tree* rootTree = *tree;
    Tree* tmpTree = *tree;
    Tree* delTree = (*tree)->Right;

    //삭제할 노드의 자식노드가 없는경우
    if (delTree->Left == NULL && delTree->Right
== NULL)
    {
        printf("check0₩n");
        tmpTree->Right = NULL;
        free(delTree);
    }
    //삭제할 노드의 자식 노드가 2개인 경우
    else if (delTree->Left != NULL && delTree-
>Right != NULL)
    {
        printf("check1₩n");

        tmpTree =
findRightMinDataParentTree(&delTree);
        rootTree->Right->data = tmpTree->Left-
>data;  //삭제할 노드의 데이터 값만 교환
        delTree = tmpTree->Left;
        tmpTree->Left = tmpTree->Left->Right;
        free(delTree);
    }
    //삭제할 노드의 자식 노드가 1개인 경우
    else
    {
        printf("check2₩n");
        //자식노드가 오른쪽 노드인 경우
        if (delTree->Right != NULL)
        {
            tmpTree->Right = delTree->Right;
            free(delTree);
        }
        //자식노드가 왼쪽 노드인 경우
        else{
            tmpTree->Right = delTree->Left;
            free(delTree);
        }
    }
}

// 지울 왼쪽 자식노드 포인터 인자
void delLeftNode(Tree** tree){
    printf("check-2₩n");
    //트리 자체가 NULL값 또는 트리 왼쪽 삭제할
값이 NULL
    if(*tree == NULL || (*tree)->Left == NULL){
        printf("*tree = NULL || (*tree)->Right ==
NULL₩n");
    }
    Tree* rootTree = *tree;
    Tree* tmpTree = *tree;
    Tree* delTree = (*tree)->Left;

    //삭제할 노드의 자식노드가 없는경우
    if (delTree->Left == NULL && delTree->Right
== NULL)
    {
        printf("check0₩n");
        tmpTree->Left = NULL;
        free(delTree);
    }
    //삭제할 노드의 자식 노드가 2개인 경우
    else if (delTree->Left != NULL && delTree-
>Right != NULL)
    {
        printf("check1₩n");

        tmpTree =
findRightMinDataParentTree(&delTree);
        rootTree->Left->data = tmpTree->Left-
>data;  //삭제할 노드의 데이터 값만 교환
        delTree = tmpTree->Left;     //delTree
변수에 삭제할 노드 저장
        tmpTree->Left = tmpTree->Left->Right;
        free(delTree);
    }
    //삭제할 노드의 자식 노드가 1개인 경우
    else
    {
```

```
            printf("check2₩n");
            //자식노드가 오른쪽 노드인 경우
            if (delTree->Right != NULL)
            {
                tmpTree->Left = delTree->Right;
                free(delTree);
            }
            //자식노드가 왼쪽 노드인 경우
            else{
                tmpTree->Left = delTree->Left;
                free(delTree);
            }
        }
}
```

위의 함수들을 테스트할 메인함수

```
int main(){
    Tree* tree = NULL;
    Data d[13] = { 50, 45, 73, 32, 48, 46, 16, 37,
120, 47, 130, 127, 124 };
    for(int i = 0; i < 13; i++){
        insertDataToTree(&tree, d[i]);
    }
    printTree(&tree);
    printf("₩n");

    deleteTreeData(&tree, 999);
        deleteTreeData(&tree, 50);
        deleteTreeData(&tree, 120);
        deleteTreeData(&tree, 32);
        printTree(&tree);
        printf("₩n");

    deleteAllTree(&((*tree)->Right));
        printTree(&tree);
    printf("₩n");

    return 0;
}
```