Chapter 08.
트리

14주차

이진 트리
(노드 방문을 자유롭게 구성하기. 314~318p)

[아래를 추가!]

```
typedef void VisitFuncPtr(BTData data);

void ShowIntData(BTData data) {
        printf("%3c", data);
}
```

[아래 부분을 수정!]

```
int main() {
        .....
        PreorderTraverse(btA, ShowIntData);
        InorderTraverse(btA, ShowIntData)
        PostorderTraverse(btA, ShowIntData);
        return 0;
}
```

[아래 부분을 수정!]

```
void PreorderTraverse(BTreeNode *bt, VisitFuncPtr action) {
        if (bt == NULL)
                return;
        action(bt->data);
        PreorderTraverse(bt->left, action);
        PreorderTraverse(bt->right, action);

}
void PostorderTraverse(BTreeNode *bt, VisitFuncPtr action) {
        if (bt == NULL)
                return;
        PostorderTraverse(bt->left, action);
        PostorderTraverse(bt->right, action);
        action(bt->data);
}
void InorderTraverse(BTreeNode *bt, VisitFuncPtr action) {
        if (bt == NULL)
                return;
        InorderTraverse(bt->left, action);
        action(bt->data);
        InorderTraverse(bt->right, action);

}
```

이진 트리
(노드 방문을 자유롭게 구성하기. 314~318p)

이진 트리의 소멸
(빨간색 부분 추가 : 319p)

```
int malloc_count = 0;
BTreeNode *MakeBTreeNode() {
        BTreeNode * nd = (BTreeNode*)malloc(sizeof(BTreeNode));
        malloc_count++;
        nd->left = NULL;
        nd->right = NULL;
        return nd;
}
void DeleteTree(BTreeNode *bt) {
        if (bt == NULL)
                return;
        if (bt->left != NULL)
                DeleteTree(bt->left);
        if (bt->right != NULL)
                DeleteTree(bt->right);
        free(bt);
        malloc_count--;
}
void main() {
        .....
        DeleteTree(btA);
        printf("malloc_count=%d", malloc_count);
}
```

수식 트리의 구현 (320p)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>                    // malloc, free, exit
#include <string.h>                    // strlen
```

수식 트리의 구현
(btree 관련)

```
typedef int BTData;
typedef struct _bTreeNode
{
        BTData data;
        struct _bTreeNode * left;
        struct _bTreeNode * right;
} BTreeNode;
typedef BTreeNode * Data;

typedef void VisitFuncPtr(BTData data);

BTreeNode * MakeBTreeNode(void)
{
        BTreeNode * nd = (BTreeNode*)malloc(sizeof(BTreeNode));

        nd->left = NULL;
        nd->right = NULL;
        return nd;
}
```

수식 트리의 구현
(btree 관련)

```
BTData GetData(BTreeNode * bt)
{
	return bt->data;
}

void SetData(BTreeNode * bt, BTData data)
{
	bt->data = data;
}

BTreeNode * GetLeftSubTree(BTreeNode * bt)
{
	return bt->left;
}

BTreeNode * GetRightSubTree(BTreeNode * bt)
{
	return bt->right;
}
```

수식 트리의 구현
(btree 관련)

```
void MakeLeftSubTree(BTreeNode * main, BTreeNode * sub)
{
	if (main->left != NULL)
		free(main->left);

	main->left = sub;
}

void MakeRightSubTree(BTreeNode * main, BTreeNode * sub)
{
	if (main->right != NULL)
		free(main->right);

	main->right = sub;
}
```

수식 트리의 구현
(btree 관련)

```
void PreorderTraverse(BTreeNode * bt, VisitFuncPtr action) {
        if (bt == NULL)
                return;
        action(bt->data);
        PreorderTraverse(bt->left, action);
        PreorderTraverse(bt->right, action);
}
void InorderTraverse(BTreeNode * bt, VisitFuncPtr action) {
        if (bt == NULL)
                return;
        InorderTraverse(bt->left, action);
        action(bt->data);
        InorderTraverse(bt->right, action);
}
void PostorderTraverse(BTreeNode * bt, VisitFuncPtr action) {
        if (bt == NULL)
                return;
        PostorderTraverse(bt->left, action);
        PostorderTraverse(bt->right, action);
        action(bt->data);
}
```

수식 트리의 구현
(stack 관련)

```
#define TRUE      1
#define FALSE     0

typedef struct _node
{
        Data data;
        struct _node * next;
} Node;

typedef struct _listStack
{
        Node * head;
} ListStack;

typedef ListStack Stack;
```

수식 트리의 구현
(stack 관련)

```
void StackInit(Stack * pstack) {
        pstack->head = NULL;
}

int SIsEmpty(Stack * pstack) {
        if (pstack->head == NULL)
                return TRUE;
        else
                return FALSE;
}

void SPush(Stack * pstack, Data data) {
        Node * newNode = (Node*)malloc(sizeof(Node));

        newNode->data = data;
        newNode->next = pstack->head;

        pstack->head = newNode;
}
```

수식 트리의 구현
(stack 관련)

```
Data SPop(Stack * pstack)
{
	Data rdata;
	Node * rnode;

	if (SIsEmpty(pstack)) {
		printf("Stack Memory Error!");
		exit(-1);
	}

	rdata = pstack->head->data;
	rnode = pstack->head;

	pstack->head = pstack->head->next;
	free(rnode);

	return rdata;
}
```

수식 트리의 구현

(expression tree 관련 : 후위표기식 -> 이진 트리로 변환)

```
BTreeNode * MakeExpTree(char exp[]) {
        Stack stack;
        BTreeNode * pnode;
        int expLen = strlen(exp);
        int i;
        StackInit(&stack);
        for (i = 0; i<expLen; i++)  {
                pnode = MakeBTreeNode();
                if (isdigit(exp[i]))                // 피연산자라면...
                        SetData(pnode, exp[i] - '0');
                else {                              // 연산자라면...
                        MakeRightSubTree(pnode, SPop(&stack));
                        MakeLeftSubTree(pnode, SPop(&stack));
                        SetData(pnode, exp[i]);
                }
                SPush(&stack, pnode);
        }
        return SPop(&stack);
}
```

수식 트리의 구현

(expression tree 관련 : 이진 수식트리의 결과값을 반환.)

```
int EvaluateExpTree(BTreeNode * bt) {
        int op1, op2;

        if (GetLeftSubTree(bt) == NULL && GetRightSubTree(bt) == NULL)
                return GetData(bt);

        op1 = EvaluateExpTree(GetLeftSubTree(bt));
        op2 = EvaluateExpTree(GetRightSubTree(bt));

        switch (GetData(bt)) {
        case '+': return op1 + op2;
        case '-': return op1 - op2;
        case '*': return op1*op2;
        case '/': return op1 / op2;
        }

        return 0;
}
```

수식 트리의 구현
(expression tree 관련)

```
void ShowNodeData(int data) {
        if (0 <= data && data <= 9)
                printf("%d ", data);
        else
                printf("%c ", data);
}

void ShowPrefixTypeExp(BTreeNode * bt) {
        PreorderTraverse(bt, ShowNodeData);
}

void ShowInfixTypeExp(BTreeNode * bt) {
        InorderTraverse(bt, ShowNodeData);
}

void ShowPostfixTypeExp(BTreeNode * bt) {
        PostorderTraverse(bt, ShowNodeData);
}
```

수식 트리의 구현
(main 함수)

```
int main(void)
{
	char exp[] = "12+7*";
	BTreeNode * eTree = MakeExpTree(exp);

	printf("전위 표기법의 수식: ");
	ShowPrefixTypeExp(eTree); printf("₩n");

	printf("중위 표기법의 수식: ");
	ShowInfixTypeExp(eTree); printf("₩n");

	printf("후위 표기법의 수식: ");
	ShowPostfixTypeExp(eTree); printf("₩n");

	printf("연산의 결과: %d ₩n", EvaluateExpTree(eTree));

	return 0;
}
```