Chapter 11.
탐색(Search) 1

7주차

이진 탐색 트리의 삭제 구현

◆ 삭제 후에도 이진 탐색 트리가 유지되어야 함
(아래를 고려하여 삭제의 경우의 수는 최대 6가지)

(1) 삭제할 노드가 단말인 경우
(2) 삭제할 노드가 하나의 자식 노드를 가진 경우
(3) 삭제할 노드가 두 개의 자식 노드를 가진 경우

(1) 삭제 대상이 루트 노드인 경우
(2) 삭제 대상이 루트 노드가 아닌 경우

445-465p

이진탐색 트리의 구현 (삭제 포함)
(타입 선언)

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

typedef int BTData;

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

typedef BTData BSTData;
```

445-465p

이진탐색 트리의 구현 (삭제 포함)
(이진 트리 구현)

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

        if (nd) {
                nd->left = NULL;
                nd->right = NULL;
        }
        return nd;
}

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

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

이진탐색 트리의 구현 (삭제 포함)
(이진 트리 구현)

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

445-465p

이진탐색 트리의 구현 (삭제 포함)
(이진 트리 구현)

```
void InorderTraverse(BTreeNode* bt) {
        if (bt == NULL)
                return;

        InorderTraverse(bt->left);
        printf( "%d  ", bt->data );
        InorderTraverse(bt->right);
}
```

이진탐색트리를 중위 운행하면 : 오름차순

Memory free 없이 왼쪽 또는 오른쪽 자식을 교체.

```
void ChangeLeftSubTree(BTreeNode* main, BTreeNode* sub) {
        main->left = sub;
}

void ChangeRightSubTree(BTreeNode* main, BTreeNode* sub) {
        main->right = sub;
}
```

이진탐색 트리의 구현 (삭제 포함)
(이진 트리 구현 : 왼쪽/오른쪽 자식 제거)

```
BTreeNode *RemoveLeftSubTree(BTreeNode *bt) {
        BTreeNode *delNode = NULL;
        if (bt != NULL) {
                delNode = bt->left;
                bt->left = NULL;
        }
        return delNode;
}

BTreeNode *RemoveRightSubTree(BTreeNode *bt) {
        BTreeNode *delNode = NULL;
        if (bt != NULL) {
                delNode = bt->right;
                bt->right = NULL;
        }
        return delNode;
}
```

이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현)

```
// 이진 탐색 트리의 생성 및 초기화.
void BSTMakeAndInit(BTreeNode** pRoot)
{
	*pRoot = NULL;
}

// 노드에 저장된 데이터 반환.
BSTData BSTGetNodeData(BTreeNode* bst)
{
	return GetData(bst);
}

// 이진 탐색 트리에 저장된 모든 노드의 데이터를 출력.
void BSTShowAll(BTreeNode* bst)
{
	InorderTraverse(bst);
}
```

436-444p

이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현 : 노드 추가. 이전 소스와 같음)

```
void BSTInsert(BTreeNode** pRoot, BSTData data) {
        BTreeNode* pNode = NULL;    // parent node
        BTreeNode* cNode = *pRoot;    // current node
        BTreeNode* nNode = NULL;    // new node

        while (cNode != NULL)        {
                if (data == GetData(cNode))
                        return;
                pNode = cNode;
                if (GetData(cNode) > data)
                        cNode = GetLeftSubTree(cNode);
                else
                        cNode = GetRightSubTree(cNode);
        }

        .....
```

새로운 노드가 추가될 위치 찾기.

키의 중복을 허용하지 않음

436-444p

이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현 : 노드 추가. 이전 소스와 같음)

```
void BSTInsert(BTreeNode** pRoot, BSTData data) {
        .........
        nNode = MakeBTreeNode();
        SetData(nNode, data);

    if (pNode != NULL) {
            if (data < GetData(pNode))
                MakeLeftSubTree(pNode, nNode);
            else
                MakeRightSubTree(pNode, nNode);
    }
    else
    {
            *pRoot = nNode;
    }
}
```

pNode의 자식으로 추가할 새 노드 생성

pNode의 서브 노드에 새 노드를 추가

빈 트리가 아닐 때.

빈 트리일 때.
새 노드가 루트가 됨.

445-465p

이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현 : 데이터 탐색)

```
BTreeNode* BSTSearch(BTreeNode* bst, BSTData target) {
        BTreeNode* cNode = bst;     // current node
        BSTData cd;     // current data

        while (cNode != NULL) {
                cd = GetData(cNode);

                if (target == cd)
                        return cNode;
                else if (target < cd)
                        cNode = GetLeftSubTree(cNode);
                else
                        cNode = GetRightSubTree(cNode);
        }

        return NULL;
}
```

462-463p

이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현 : 삭제 1/5)

```
BTreeNode* BSTRemove(BTreeNode** pRoot, BSTData target) {
  BTreeNode* pVRoot = MakeBTreeNode();
  BTreeNode* pNode = pVRoot;     // parent node
  BTreeNode* cNode = *pRoot;     // current node
  BTreeNode* dNode;     // delete node

  ChangeRightSubTree(pVRoot, *pRoot);

  while (cNode != NULL && GetData(cNode) != target) {
    pNode = cNode;
    if (target < GetData(cNode))
      cNode = GetLeftSubTree(cNode);
    else
      cNode = GetRightSubTree(cNode);
  }
  if (cNode == NULL)     // 삭제 대상이 존재하지 않는다면,
    return NULL;
```

루트노드 삭제시의 처리를 위해 루트 위에 가상 루트 사용. (462p)

삭제 대상을 저장한 노드 탐색

이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현 : 삭제 2/5)

```
    dNode = cNode;    // 삭제 대상을 dNode가 가리키게 한다.

    // 첫 번째 경우: 삭제할 노드가 단말 노드인 경우
    if (GetLeftSubTree(dNode) == NULL && GetRightSubTree(dNode)
== NULL) {
        if (GetLeftSubTree(pNode) == dNode)
            RemoveLeftSubTree(pNode);
        else
            RemoveRightSubTree(pNode);
    }
```

Remove...() 함수는 child 자리에 NULL을 넣고 free 시켜야 할 노드를 반환. (free는 하지 않음.)

이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현 : 삭제 3/5)

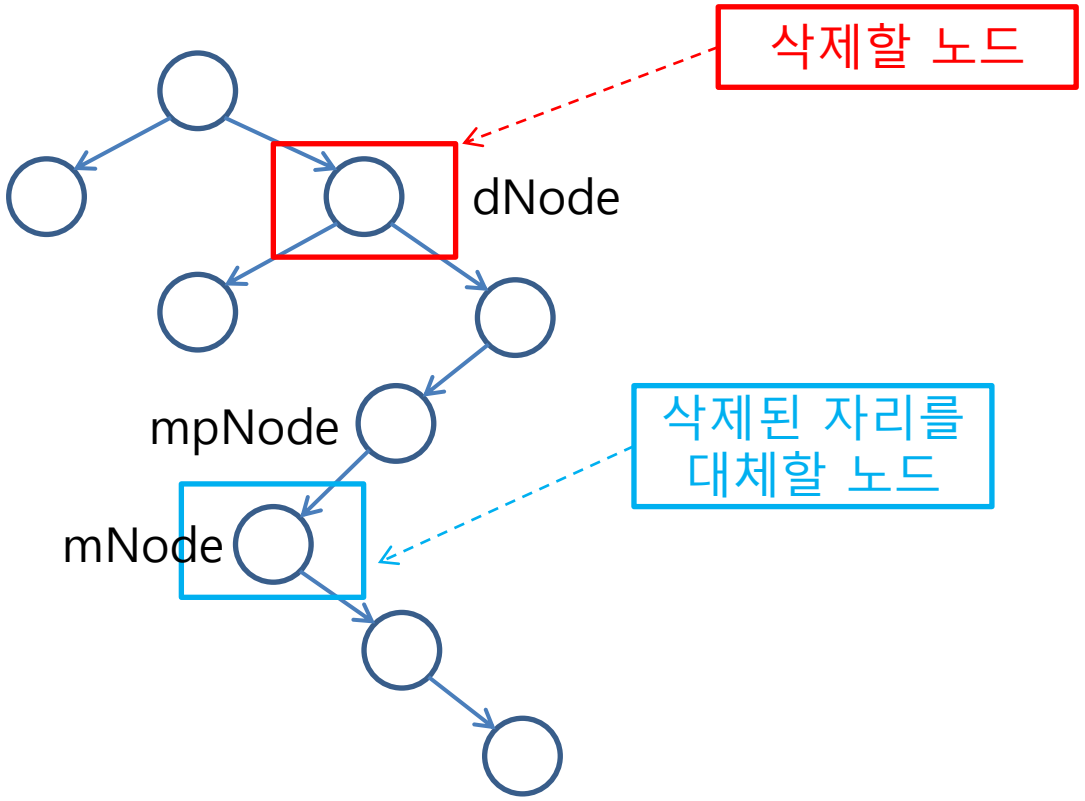
```
// 두 번째 경우: 하나의 자식 노드를 갖는 경우
else if (GetLeftSubTree(dNode) == NULL ||
    GetRightSubTree(dNode) == NULL) {
        BTreeNode* dcNode;    // delete node의 자식 노드

        if (GetLeftSubTree(dNode) != NULL)
                dcNode = GetLeftSubTree(dNode);
        else
                dcNode = GetRightSubTree(dNode);

        if (GetLeftSubTree(pNode) == dNode)
                ChangeLeftSubTree(pNode, dcNode);
        else
                ChangeRightSubTree(pNode, dcNode);
}
```

449-452p

세 번째 경우:
두 개의 자식 노드를 모두 갖는 경우



이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현 : 삭제 4/5)

449-452p
460-461p

세 번째 경우: 두 개의 자식 노드를 모두 갖는 경우

```
else {
    BTreeNode* mNode = GetRightSubTree(dNode);
    BTreeNode* mpNode = dNode;
    int delData;
    while (GetLeftSubTree(mNode) != NULL) {
        mpNode = mNode;
        mNode = GetLeftSubTree(mNode);
    }
    delData = GetData(dNode);
    SetData(dNode, GetData(mNode));
    // 대체할 노드의 부모 노드와 자식 노드를 연결한다.
    if (GetLeftSubTree(mpNode) == mNode)
        ChangeLeftSubTree(mpNode, GetRightSubTree(mNode));
    else
        ChangeRightSubTree(mpNode, GetRightSubTree(mNode));
    dNode = mNode;
    SetData(dNode, delData);
}
```

대체할 노드 (오른쪽 서브 트리의 최소값) 찾기.
- mpNode : 대체할 노드의 부모노드
- mNode : 대체할 노드.

대체할 노드와 삭제할 노드의 데이터를 맞바꾸기.

값을 교환했으므로 실제 지워질 노드는 mNode가 되게 하기.

이진탐색 트리의 구현 (삭제 포함)
(이진 탐색 트리 구현 : 삭제 5/5)

```
        // 삭제된 노드가 루트 노드인 경우에 대한 처리
        if (GetRightSubTree(pVRoot) != *pRoot)
                *pRoot = GetRightSubTree(pVRoot);

        free(pVRoot);
        return dNode;
}
```

이진탐색 트리의 구현 (삭제 포함)
(main 함수)


```
int main(void) {
  BTreeNode* bstRoot;
  BTreeNode* sNode;    // search node
  BSTMakeAndInit(&bstRoot);
  BSTInsert(&bstRoot,5); BSTInsert(&bstRoot,8); BSTInsert(&bstRoot,1);
  BSTInsert(&bstRoot,6); BSTInsert(&bstRoot,4); BSTInsert(&bstRoot,9);
  BSTInsert(&bstRoot,3); BSTInsert(&bstRoot,2); BSTInsert(&bstRoot,7);

  BSTShowAll(bstRoot); printf("₩n");
  sNode = BSTRemove(&bstRoot, 3); free(sNode);
  BSTShowAll(bstRoot); printf("₩n");
  sNode = BSTRemove(&bstRoot, 8); free(sNode);
  BSTShowAll(bstRoot); printf("₩n");
  sNode = BSTRemove(&bstRoot, 1); free(sNode);
  BSTShowAll(bstRoot); printf("₩n");
  sNode = BSTRemove(&bstRoot, 6); free(sNode);
  BSTShowAll(bstRoot); printf("₩n");
  return 0;
}
```

[실행결과]

```
1 2 3 4 5 6 7 8 9
1 2 4 5 6 7 8 9
1 2 4 5 6 7 9
2 4 5 6 7 9
2 4 5 7 9
```