자료구조 실습07

Data Structures Lab07

Lab07 예제(1/2)

◎ 목표:

- ☞ Linked structure 를 이용한 Binary Search Tree설계 및 구현
- ☞ 재귀(Recursive) 함수, 포인터(Pointer), 이진 탐색 트리(BST, Binary Search Tree) 이해

◎ 내용:

- ☞ 과제
  - ➢ 재귀함수와 포인터를 이용한 BST 구현

◎ 방법:

- ☞ Binary Search Tree의 메커니즘을 분석하고 BinarySearchTree의 ADT를 바탕으로 구현
- ☞ 재귀함수를 이해하고 이를 바탕으로 각 기능을 구현
- ☞ Global Function 을 BinarySearchTree 멤버 함수에 이용하여 구현

Lab07 예제(2/2)

◎ 내용

☞ 재귀 함수를 사용하는 Binary Search Tree를 작성

☞ ItemType은 list에서 사용하던 ItemType을 사용

➢ 파일명을 기준으로 정렬하는 Binary Search Tree를 구성

◎ 고려사항

☞ Binary Search Tree를 특정 자료형과 무관(Generic)하게 정의

➢ 비교 연산자 (>, < , ==, != 등)와 출력연산자 재정의(overloading)를 사용

➢ Binary Search Tree의 출력은 InOrder, PreOrder, PostOrder의 세가지 방법 모두 출력

✓ 출력 방식은 이론자료(8장)를 참조

예제: Binary Search Tree Class ADT

```
template <typename T>
struct BinaryTreeNode
{
    T data;                          // node data
    TreeNode *left;                  // left node pointer
    TreeNode *right;                 // right node pointer
};

template<typename T>
class BinarySearchTree
{
public:
    BinarySearchTree();              // constructor
    ~BinarySearchTree();             // destructor
    bool IsEmpty() const;            // check tree is empty
    bool IsFull() const;             // check tree is full
    void MakeEmpty();                // make empty tree
    int GetLength() const;           // get number of current node
    void Add(T item);                // add item to tree
    void Delete(T item);             // delete item from tree
    void Retrieve(T &item, bool &found) const;        // retrieve item in tree
    void PrintTree(ostream &out) const;               // display all item in tree, InOrder, PreOrder, PostOrder
private:
    BinaryTreeNode<T> *root;
};
```

Global functions for recursive processing

◎ Binary Search Tree 의 초기화

```
template<typename T> void MakeEmptyTree(BinaryTreeNode<T> *&root);
```

◎ Binary Search Tree 의 노드 개수

```
template<typename T> int CountNodes(BinaryTreeNode<T> *root);
```

◎ Binary Search Tree 에 새로운 노드 추가

```
template<typename T> void Insert(BinaryTreeNode<T> *&root, T item);
```

◎ Binary Search Tree 에 존재하는 노드 삭제

```
template<typename T> void GetPredecessor(BinaryTreeNode<T> *root, T &item);
template<typename T> void DeleteNode(BinaryTreeNode<T> *&root);
template<typename T> void Delete(BinaryTreeNode<T> *&root, T item);
```

◎ Binary Search Tree 검색

```
template<typename T> void Retrieve(BinaryTreeNode<T> *root, T& item, bool &found);
```

◎ Binary Search Tree 출력

```
template<typename T> void PrintInOrderTraversal(BinaryTreeNode<T> *root, ostream &out);
template<typename T> void PrintPreOrderTraversal(BinaryTreeNode<T> *root, ostream &out);
template<typename T> void PrintPostOrderTraversal(BinaryTreeNode<T> *root, ostream &out);
```

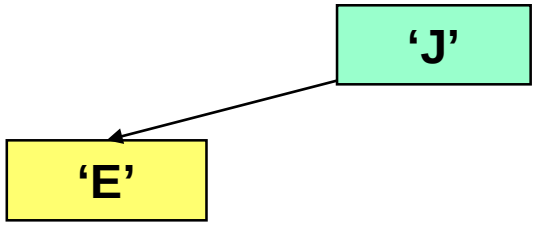
Lab07: Reference Add in BST(1/5)

◎ Insert **J E F T A**

'J'

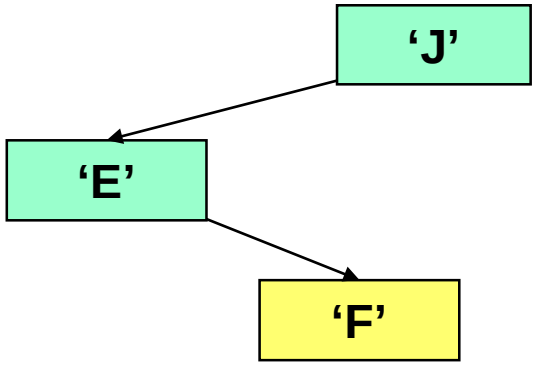
Lab07: Reference Add in BST(2/5)

◎ Insert **J** **E** **F** **T** **A**



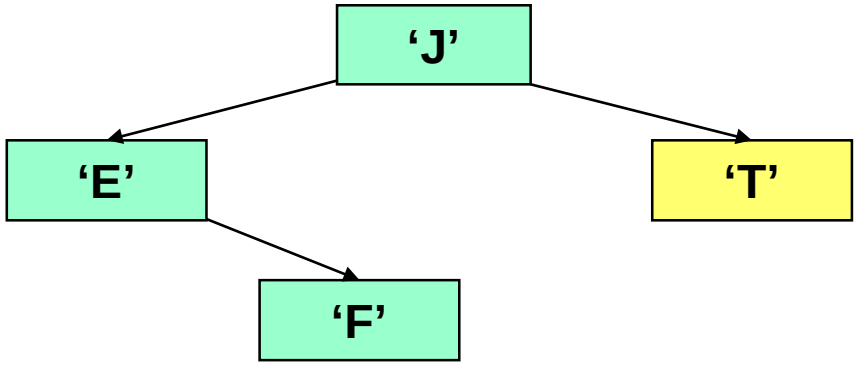
Lab07: Reference Add in BST(3/5)

◎ Insert **J E F T A**



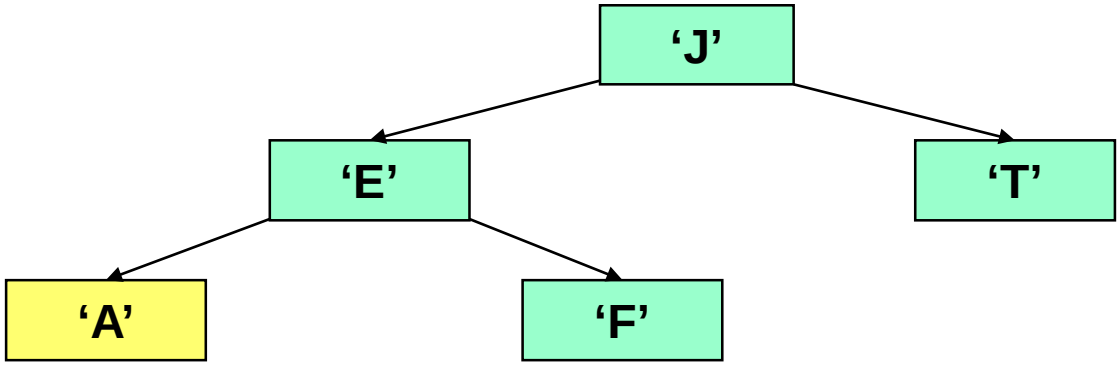
Lab07: Reference Add in BST(4/5)

◎ Insert **J E F T A**



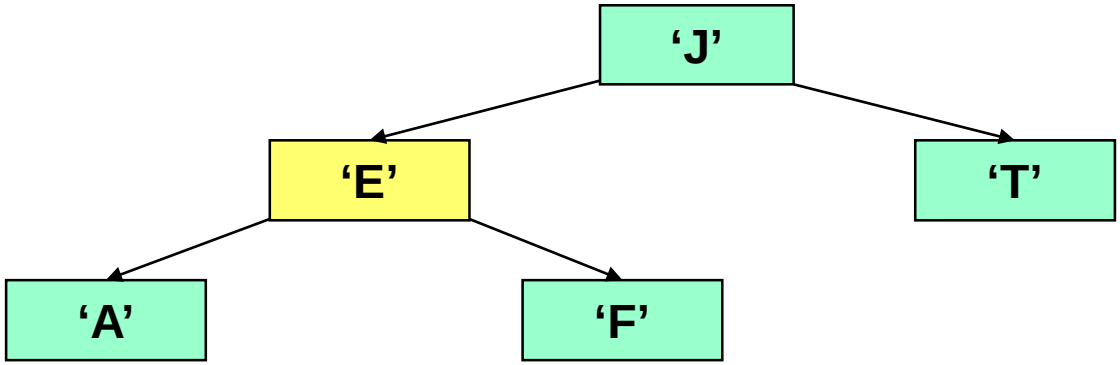
Lab07: Reference Add in BST(5/5)

◎ Insert **J E F T A**



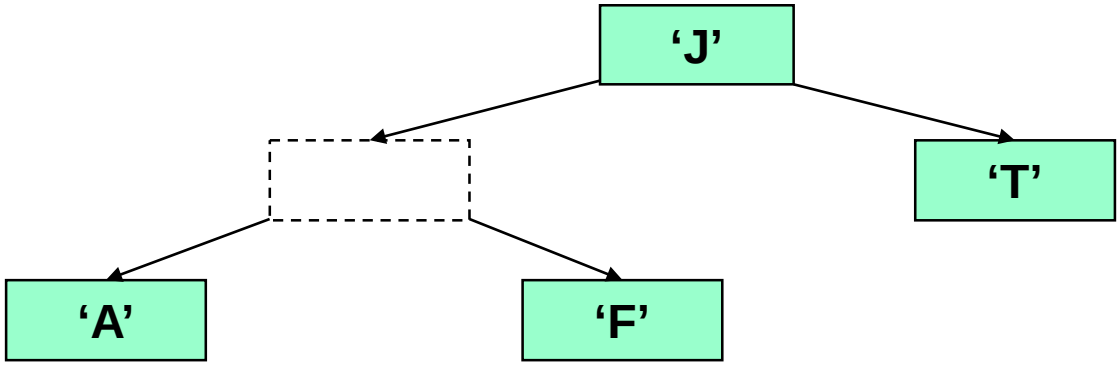
Lab07: Reference Delete in BST(1/4)

◎ Delete **E**



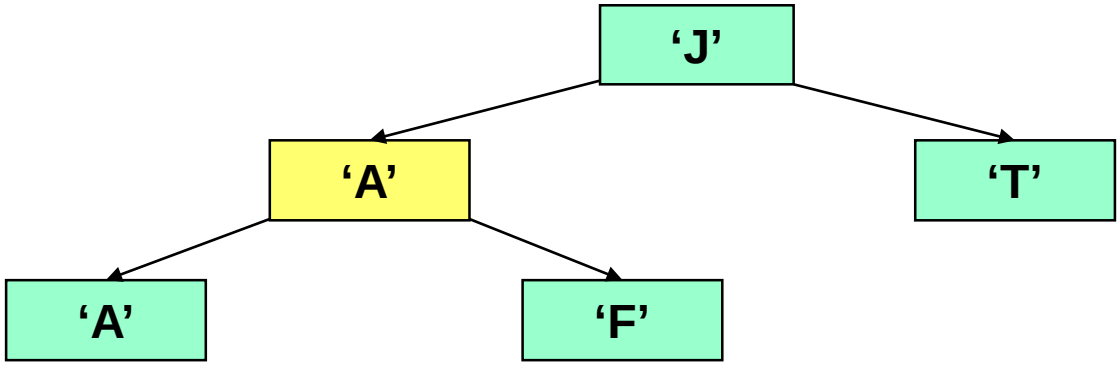
Lab07: Reference Delete in BST(2/4)

◎ Delete **E**



Lab07: Reference Delete in BST(3/4)

◎ Delete **E**



Lab07: Reference Delete in BST(4/4)

◎ Delete **E**