ECE20010 Data Structures

Chapter 5

- *Recursion Revisited*
- *binary search tree Implementation*
  - *inorder, preorder, postorder*
  - *minTree, maxTree, minKey, maxKey,*
  - *predecessor, successor*
  - *depthTree*
  - *freeTree*
  - *contains*
  - *insert*
  - *deleteNode*

Chapter 5.7 Binary search trees – Recursion Revisited

insertLast(): insert a new node at the last in a singly linked list

```
void insertLast(pList p, int item) {
  if (isEmptyList(p)) {
    p->head = newNode(item);          // returns new head node
    p->tail = p->head;
  }
  else {
    pNode curr = p->head;
    while (curr->next != NULL)        // move to the last node
      curr = curr->next;

    pNode aNode = newNode(item);      // add new node and link
    curr->next = aNode;
    p->tail = aNode;                  // set the last node
  }
  p->size++;
}
```

bunnyEars(): counting bunny ears in recursion

```
// each bunny has two ears.
int bunnyEars(int bunnies) {
  
  return 2 + bunnyEars(bunnies-1);
}
```

funnyEars(): counting funny ears in recursion

```
// even numbered funny has two ears, odd numbered funny three.
int funnyEars(int funnies) {
  if (bunnies == 0) return 0;

  if (funnies % 2 == 0)
    return 
  else
    return 
}
```

getSize(): in singly linked list

```
int getSize(pList p) {
  if (isEmptyList(p)) return 0;
  int count = 0;
  for (pNode x = p->head; x != NULL; x = x->next)
    count++;
  return count;
}
```

```
int getSize(pList p) {
  if (isEmptyList(p)) return 0;
  return getSizeRecur(p->head);
}
```

```
int getSizeRecur(pNode node) {

}
```

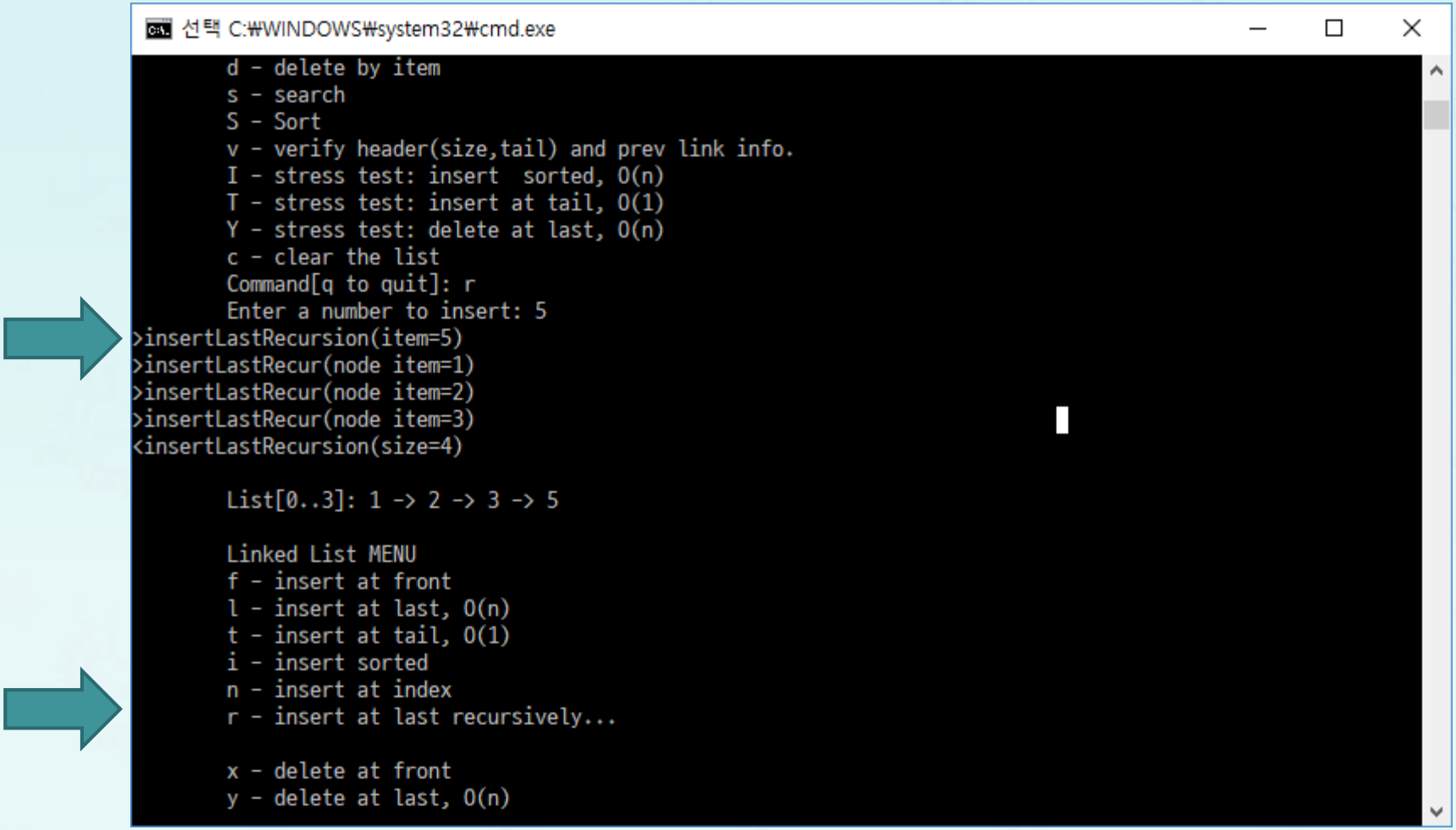
Chapter 5.7 Binary search trees – Recursion Revisited

insertLast(): insert a new node at the last in a singly linked list recursively

```
pNode insertLastRecur(pNode node, int item) {
  DPRINT(printf(">insertLastRecur(node item=%d)\n", node->item));
  if ____________________________________
    return ____________________________________
  return ____________________________________
}
void insertLastRecursion(pList p, int item) {
  DPRINT(printf(">insertLastRecursion(item=%d)\n", item));
  if (isEmptyList(p)) {
    p->head = newNode(item);        // returns new head node
    p->tail = p->head;
  }
  else {
    p->tail = insertLastRecur(p->head, item);
  }
  p->size++;
  DPRINT(printf("<insertLastRecursion(size=%d)\n", p->size));
}
```

Homework (2 points)
Use **Extra 2**
Piazza folder

Cut and paste your insertLastRecur() function and your screen capture ☺
Due: 5/7 (Wed. 11:55 PM)


선택 C:₩WINDOWS₩system32₩cmd.exe
d - delete by item
s - search
S - Sort
v - verify header(size,tail) and prev link info.
I - stress test: insert sorted, O(n)
T - stress test: insert at tail, O(1)
Y - stress test: delete at last, O(n)
c - clear the list
Command[q to quit]: r
Enter a number to insert: 5
>insertLastRecursion(item=5)
>insertLastRecur(node item=1)
>insertLastRecur(node item=2)
>insertLastRecur(node item=3)
<insertLastRecursion(size=4)
List[0..3]: 1 -> 2 -> 3 -> 5
Linked List MENU
f - insert at front
l - insert at last, O(n)
t - insert at tail, O(1)
i - insert sorted
n - insert at index
r - insert at last recursively...
x - delete at front
y - delete at last, O(n)

sizeTree: Count the number of nodes in the binary tree recursively.

```
int sizeTree(tree node) {
  if (node == NULL) return 0;
  return 
}
```

sizeTree: Count the number of nodes in the binary tree recursively.

```
int sizeTree(tree node) {
  if (node == NULL) return 0;
  return 1 + sizeTree(node->left) + sizeTree(node->right);
}
```

sizeTree: Count the number of nodes in the binary tree recursively.

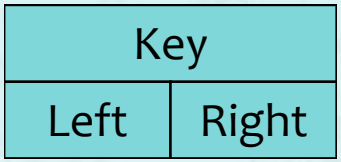
```
int sizeTree(tree node) {
  if (node == NULL) return 0;
  return 1 + sizeTree(node->left) + sizeTree(node->right);
}
```

maxHeightTree: compute the max height(or depth) of a tree.
// It is the number of nodes along the longest path from the root node
// down to the farthest leaf node.

```
int maxHeightTree(tree node) {


}
```

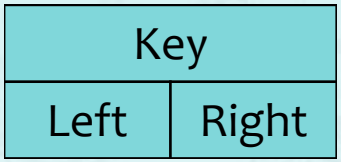
BST Node structure:




```
typedef int Key;// may be replaced by different type

typedef struct tree_struct *tree;
typedef struct tree_struct {
    tree        left;           // left child
    tree        right;          // right child
    Key         key;            // sorted by key
} tree_struct;
```

BST Node structure:




```
tree newNode(int key) {
    tree newNode = (tree)malloc(sizeof(tree_struct));
    assert(newNode != NULL)

    newNode->key = key;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
```

```
tree newTree() {
  return NULL;
}
```

Insert: insert a new node with given key in BST

```
tree insert(tree node, Key key) {
    if (node == NULL)                    // create a new node.

    if (key < node->key)// recur down the tree
        insert(node->left, key);
    else if (key > node->key)
        insert(node->right, key);
    else
        printf("Insert: the same key(%d) is ignored.\n", key);

    return node;
}
```

Insert: insert a new node with given key in BST

```
tree insert(tree node, Key key) {
    if (node == NULL) return newNode(key); // create a new node.

    if (key < node->key)// recur down the tree
        insert(node->left, key);
    else if (key > node->key)
        insert(node->right, key);
    else
        printf("Insert: the same key(%d) is ignored.\n", key);

    return node;
}
```

Something wrong?

Insert: insert a new node with given key in BST

```
tree insert(tree node, Key key) {
    if (node == NULL) return newNode(key); // create a new node.

    if (key < node->key)// recur down the tree
        insert(node->left, key);
    else if (key > node->key)
        insert(node->right, key);
    else
        printf("Insert: the same key(%d) is ignored.\n", key);

    return node;
}
```

Insert: insert a new node with given key in BST

```
tree insert(tree node, Key key) {
    if (node == NULL) return newNode(key); // create a new node.

    if (key < node->key)// recur down the tree
        node->left =  insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        printf("Insert: the same key(%d) is ignored.\n", key);

    return node;
}
```

inorder traversal: do inorder traversal of BST.

```
void inorder(tree node) {
    if (node == NULL) return;

    inorder(node->left);
    printf("%3d ", node->key);
    inorder(node->right);
}
```

minTree, maxTree: returns the node with min or max key.
Note that the entire tree does not need to be searched.

```
Key minIteration(tree node) {
  whil                                  // find the left most leaf

  }
  return node->key;
}
```

```
Key minKey(tree node) {   // returns left-most node key


}
```

```
Key maxKey(tree node) {   // returns right-most node key


}
```

minTree, maxTree: returns the node with min or max key.
Note that the entire tree does not need to be searched.

```
Key minIteration(tree node) {
  while (node->left != NULL) {// find the left-most leaf
       node = node->left;
  }
  return node->key;
}
```

```
Key minKey(tree node) {   // returns left-most node key


}
```

```
Key maxKey(tree node) {   // returns right-most node key


}
```

predecessor, successor:

```
Input: root node, key
output: predecessor node, successor node

1. If root is NULL, then return

2. if key is found then
    a. If its left subtree is not null
        Then predecessor will be the right most
        child of left subtree or left child itself.
    b. If its right subtree is not null
        The successor will be the left most child
        of right subtree or right child itself.
    return
```

deleteNode: delete node with the key and return the new root.

```
tree deleteNode(tree root, Key key) {
  if (root == NULL) return root;// base case
  if (key < root->key)
    root->right = deleteNode(root->left, key);
  else if (key > root->key) {
    root-right = deleteNode(root->right, key);
  else if (root->left && root->right) {
    printf("your code here: node with two children\n");
  }
  else {
    printf("your code here: node with one child or no child)\n");
  }
  return root;
}
```

http://visualgo.net/bst