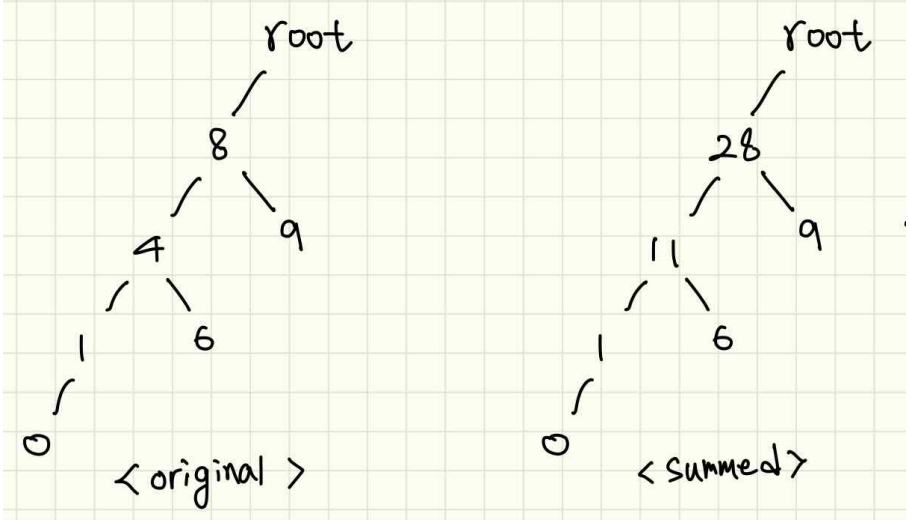
자료구조 12주차 과제

제출일: 2022-11-22
학번/이름: 2016121150 / 윤준영

문1) 8,4,9,1,6,0을 순서대로 삽입하여 이진 검색 나무 트리를 만드는 프로그램 tree.c 프로그램을 사용하여 다음을 수행하시오.
tree 함수를 실행한 결과와 sum한 결과를 tree로 표현하면 다음과 같다.



1) 주어진 트리를 중위 순회를 하면서 방문 순서대로 노드의 값들을 출력하는 코드를 작성하시오.(1점)
중위 순회하는 inorder_print 함수를 작성하여 출력하였다. 뿌리 tree의 left에 모든 값이 저장되어 있으므로 print_tree 매크로를 작성하여 inorder_print 함수를 tree->left부터 시작하게 하였다.

```c
#define print_tree(tree) inorder_print(tree->left)
void inorder_print(node *t) {
    if (t != NULL) {
        inorder_print(t->left);
        printf("%d ",t->key);
        inorder_print(t->right);
    }
}
```

```c
main() {
    node* tree;
    int num = 0;
    init_tree(&tree);
    bti_insert(8, tree, &num);
    bti_insert(4, tree, &num);
    bti_insert(9, tree, &num);
    bti_insert(1, tree, &num);
    bti_insert(6, tree, &num);
    bti_insert(0, tree, &num);
    // inorder_print(tree->left);
    print_tree(tree);
}
```

출력 결과는 다음과 같다. 정렬된 순서대로 출력되는 것을 확인할 수 있다.

```
0 1 4 6 8 9 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

2) 트리의 루트 노드 주소에 대하여 트리 안에 있는 원소들의 합을 출력하는 함수를 작성하고, 이 함수를 만든 트리에 적용한 결과를 출력하는 프로그램을 작성하시오.(2점)
후위 순회하며 하위 노드의 key의 합을 저장하는 함수 postorder_sum과 sum_tree를 작성하여 수행하였다.

```c
void postorder_sum(node *t) {
    if (t != NULL) {
        postorder_sum(t->left);
        postorder_sum(t->right);
        if (t->left != NULL) t->key += t->left->key;
        if (t->right != NULL) t->key += t->right->key;
        printf("%d ", t->key);
    }
}
void sum_tree(node *t) {
    postorder_sum(t->left);
    printf("\nsum: %d", t->left->key);
}
```

```c
main() {
    node* tree;
    int num = 0;
    init_tree(&tree);
    bti_insert(8, tree, &num);
    bti_insert(4, tree, &num);
    bti_insert(9, tree, &num);
    bti_insert(1, tree, &num);
    bti_insert(6, tree, &num);
    bti_insert(0, tree, &num);
    printf("\ninorder: ");
    print_tree(tree);
    printf("\n");
    sum_tree(tree);
}
```

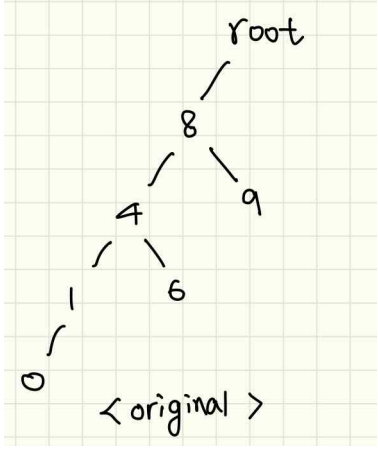
실행 결과는 다음과 같다. 합 28이 잘 출력되는 것을 확인할 수 있다.

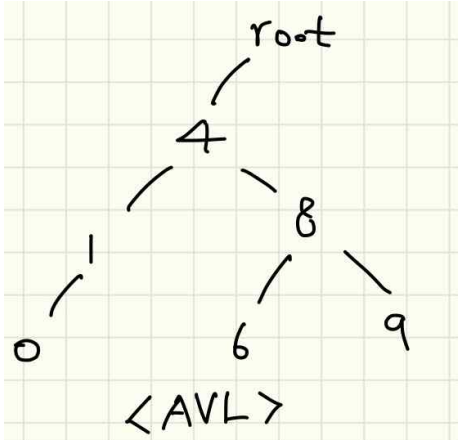
```
inorder: 0 1 4 6 8 9
0 1 6 11 9 28
sum: 28* 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

문2) tree.c 프로그램에 대하여 다음을 수행하시오.

1) 생성된 트리는 AVL 트리가 아니다. 트리의 모양을 그리고, 이를 AVL 트리로 바꾸기 위해서는 어떻게 회전을 하면 되는지 설명하시오. (1점)

생성된 트리는 좌측과 같으며 AVL 트리가 되기 위하여 8인 노드를 기준으로 L회전을 하여야 한다. 회전하게 되면 우측과 같다.





2) 1)에서 설명한 동작을 수행하는 코드를 main 함수 후반부에 작성하고(함수는 추가하지 마시오), 해당 트리를 후위 순회하여 결과를 출력하는 프로그램을 작성하시오. (2점)

main 함수 후반부에 다음과 같은 명령을 추가하여 L회전을 수행하였다.

```c
    node *tmp, *p, *s;
    p = tree->left;
    s = p->left;
    p->left = s->right;
    s->right = p;
    tree->left = s;
```

후위 순회한 결과는 문1) 1번과 같이 postorder_print 함수와 print_tree_post 매크로를 사용하여 구현하였다.

```c
#define print_tree_post(tree) postorder_print(tree->left)
void postorder_print(node *t) {
    if (t != NULL) {
        postorder_print(t->left);
        postorder_print(t->right);
        printf("%d ",t->key);
    }
}
```

사용한 main함수와 결과는 다음과 같다. 중위 순회와 후위 순회로 tree의 모양을 보면 위 그림과 같은 것을 확인할 수 있다.

```
78  main() {
79      node* tree;
80      int num = 0;
81      init_tree(&tree);
82      bti_insert(8, tree, &num);
83      bti_insert(4, tree, &num);
84      bti_insert(9, tree, &num);
85      bti_insert(1, tree, &num);
86      bti_insert(6, tree, &num);
87      bti_insert(0, tree, &num);
88      printf("\ninorder: ");
89      print_tree(tree);
90
```

```
90
91      node *tmp, *p, *s;
92      p = tree->left;
93      s = p->left;
94      p->left = s->right;
95      s->right = p;
96      tree->left = s;
97
98      printf("\ninorder(AVL): ");
99      print_tree(tree);
100     printf("\npostorder(AVL): ");
101     print_tree_post(tree);
102 }
```

```
inorder: 0 1 4 6 8 9
inorder(AVL): 0 1 4 6 8 9
postorder(AVL): 0 1 6 9 8 4 * 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

문3) 이분검색을 사용하는 다음 프로그램을 작성하시오. (교재의 코드를 사용하시오.)

1) random1000.c는 0이상 1000미만의 임의의 수 1000개를 출력하는 프로그램이다. 이 프로그램을 참고하여 "0이상 1000미만의 임의의 수를 생성하여 이분검색 자료구조에 추가하는 작업"을 1000번 하는 코드를 작성하시오. (이미 삽입한 수가 또 생성되면 삽입하지 않으며 이것도 한 번으로 간주한다.) (2점)

이분검색으로 배열 형태의 이진tree를 생성하는 함수 bi_insert를 사용하였다. 만약 배열 내의 원소를 검색하면서 이미 삽입한 수가 또 생성되면 삽입하기 전 -1을 반환하여 삽입하지 않는다.

```
25  int bi_insert(int key, int a[], int *np) {
26      int p = 0;
27      int i;
28      while (key > a[p] && p < *np) p++;
29      if (key == a[p]) return -1; // 중복일 경우 -1 반환
30      else {
31          for (i = *np; i > p; i--) a[i] = a[i-1];
32          a[p] = key;
33          (*np)++;
34          return p;
35      }
36  }
```

2) 1) 수행 후에 111, 222, 333, 444, 555가 삽입된 위치를 출력하는 코드를 작성하시오. (삽입되지 않았으면 해당 코드가 없다고 출력한다.) (2점)

해당값의 위치와 각 숫자를 찾는데 몇 번이나 검사를 했는지(mid위치의 값 검사 반복한 횟수)를 출력하도록 출력하도록 bi_search 함수를 수정한다.

bi_search 함수를 다음과 같이 수정하였다. ex 변수를 추가하여 while문을 수행할 때마다 +1하여 검사 횟수를 검사하였다. 만약 key를 찾았으면 mid위치를 return하며, 찾지 못하였을 경우 없음을 출력한다.

```c
int bi_search(int key, int a[], int n) {
    int mid;
    int left = 0;
    int right = n-1;
    int ex=1;
    while (right >= left) {
        mid = (right + left) / 2;
        if (key == a[mid]) {
            printf("%d의 위치 : %d (검사 %d번)\n", key, mid, ex);
            return mid;
        }
        if (key < a[mid]) right = mid - 1;
        else left = mid + 1;
        ex++;
    }
    printf("%d의 위치 : 없음\n", key);
    return -1;
}
```

작성된 main함수와 수행 결과는 다음과 같다. 잘 수행되는 것을 확인할 수 있다.

```c
main() {
    int a[1000] = {1000};
    int n = 0;
    int i;
    srand(time(NULL));
    for (i = 0; i < 1000; i++) bi_insert(rand()%1000, a, &n);
    // for (i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\nlength of array : %d\n",n);
    bi_search(111, a, n);
    bi_search(222, a, n);
    bi_search(333, a, n);
    bi_search(444, a, n);
    bi_search(555, a, n);
}
```

```
length of array : 652
111의 위치 : 76 (검사 10번)
222의 위치 : 148 (검사 8번)
333의 위치 : 218 (검사 9번)
444의 위치 : 없음
555의 위치 : 362 (검사 8번)
* 터미널이 작업에서 다시 사용됩니다.
```