

第 11 回 プログラミング応用レポート

15302114 番 山下尚人

提出日：2018 年 1 月 25 日

課題

● ソースコード

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sysexit.h>
4
5  typedef struct node {
6      struct node *left;
7      struct node *right;
8      int label;
9  } node_t;
10
11 node_t *allocNode(void);
12 void setNode(node_t *n, int x, node_t *left, node_t *right);
13 node_t *insertNode(node_t *n, int x);
14 void printTree(node_t *n);
15
16 int main(void){
17     //3,8,6,5,4,9,2,7,1
18     node_t *root;
19     root = allocNode();
20     setNode(root, 3, NULL, NULL);
21
22     insertNode(root,8);
23     printf("*****TREE:add_8*****\n");
24     printTree(root);
25
26     insertNode(root,6);
27     printf("*****TREE:add_6*****\n");
28     printTree(root);
29
30     insertNode(root,5);
31     printf("*****TREE:add_5*****\n");
32     printTree(root);
33
34     insertNode(root,4);
35     printf("*****TREE:add_4*****\n");
36     printTree(root);
37
38     insertNode(root,9);
39     printf("*****TREE:add_9*****\n");
40     printTree(root);
41
42     insertNode(root,2);
43     printf("*****TREE:add_2*****\n");
44     printTree(root);
45
46     insertNode(root,7);
47     printf("*****TREE:add_7*****\n");
48     printTree(root);
49
50     insertNode(root,1);
51     printf("*****TREE:add_1*****\n");
52     printTree(root);
53
54     return EXIT_SUCCESS;
55 }
56
57 node_t *allocNode(void){
58     //大きさsizeof(node_t)バイトを1個分をメモリ領域に確保。
59     //mallocではなくcallocは、確保した領域を0で初期化する。
60     return (node_t *)calloc(1, sizeof(node_t));
61 }
62
```

```

63 void setNode(node_t *n, int x, node_t *left, node_t *right){
64     n->left = left;
65     n->right = right;
66     n->label = x;
67 }
68
69 node_t *insertNode(node_t *n, int x){
70     if (NULL == n) {
71         n = allocNode();
72         setNode(n, x, NULL, NULL);
73     } else {
74         if (x < n->label) {
75             n->left = insertNode(n->left, x);
76         } else if (x > n->label) {
77             n->right = insertNode(n->right, x);
78         } else {
79             // x == n->label
80             printf("ERROR:Registered\n");
81         }
82     }
83     return n;
84 }
85
86 void printTree(node_t *n){
87     static int level = 0;
88     int i;
89     level++;
90
91     if (n->right != NULL) {
92         printTree(n->right);
93     }
94
95     for (i=1; i<level; i++) {
96         printf("  ");
97     }
98     printf("%d\n",n->label);
99
100    if (n->left !=NULL) {
101        printTree(n->left);
102    }
103
104    level--;
105 }

```

● 実行結果

```

1 ***** TREE:add 8 *****
2     8
3
4 ***** TREE:add 6 *****
5     8
6     6
7
8 ***** TREE:add 5 *****
9     8
10    6
11    5
12
13 ***** TREE:add 4 *****
14     8
15     6
16     5
17     4
18
19 ***** TREE:add 9 *****
20     9
21     8
22     6
23     5
24     4
25
26 ***** TREE:add 2 *****
27     9
28     8
29     6
30     5
31     4
32
33     2
34 ***** TREE:add 7 *****
35     9
36     8
37     7
38     6
39     5
40     4
41
42     3
43     2

```

```

43 ***** TREE:add 1 *****
44     9
45    8 7
46   6 5 4
47  3 2 1
48
49
50
51
52

```

- プリントアウト関数の動作説明

画面の垂直方向 (行) の順番は、ソースの 92,101 行目の再帰呼び出しと 98 行目の printf 関数により決まる。

右側のノードの再帰呼び出し、ノードの画面出力、左側のノードの再帰呼び出しの順で処理している。よって、「そのノードの右の子の末端へ移動し出力。一つ親のノードへ移動して出力し、ノードの左の子へ移動。」を繰り返している。

91,101 行目の if 文による条件分岐は再帰呼び出しを終わらせるため。

画面の水平方向 (列) の空白の数は、ソースの 89,104 行目の変数 level の増減により処理し、95～97 行目の for 文により実際に出力している。

printTree 関数内のローカル変数 level は、static 装飾子により printTree 関数が呼ばれるたびに初期化されず、値が保持される。

printTree 関数は 92,101 行目で再帰呼び出しされているので、level が 1 増えた後に再帰呼び出しされ、呼び出しが終わるごとに level が 1 減っていく。

これにより、95～97 行目でノードのレベルの数だけ空白を画面出力した後、98 行目で label を出力して改行している。