

TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – GJ (박현우)
uc820@naver.com

1. 트리(Tree) 코드 이해(그림 그리기) - (ins_tree 1)

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

#define EMPTY 0

struct node{
    int data;

    struct node *l_link;
    struct node *r_link;
};

typedef struct node Tree;

Tree *getnode(){
    Tree *tmp = (Tree *)malloc(sizeof(Tree));
    tmp->l_link = EMPTY;
    tmp->r_link = EMPTY;

    return tmp;
}

void ins_tree(Tree **root, int data){
    if( *root == EMPTY){
        *root = getnode();
        (*root)->data = data;
        return;
    }

    if( (*root)->data < data)
        ins_tree( &((*root)->r_link), data);

    if( (*root)->data > data)
        ins_tree( &((*root)->l_link), data);
}

void print_tree(Tree *root){
    Tree *tmp = root;

    if(tmp != NULL){
        printf("%d\n", tmp->data);

        /* if(root->l_link)
            printf("left = %d, ", root->l_link->data);
        else
            printf("left = NULL, ");

        if(root->r_link)
            printf("right = %d, ", root->r_link->data);
        else
            printf("right = NULL, ");
        */

        print_tree(tmp->l_link);
        print_tree(tmp->r_link);
    }
}

int main(void){
    Tree *root = EMPTY;
    int data[13] = {50, 45, 73, 32, 48, 46, 16, 37, 120, 47, 130, 127, 124};
    int i, num;

    num = sizeof(data)/sizeof(int);

    for( i=0; i<num; i++){
        ins_tree(&root, data[i]);
    }

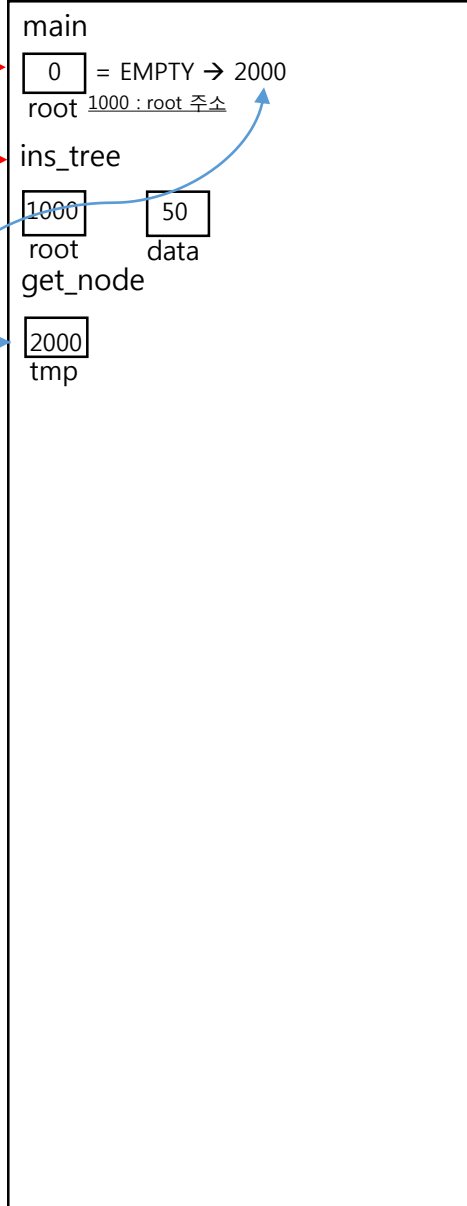
    print_tree(root);

    printf("after del\n");
    root = del_tree(root,32);

    print_tree(root);

    return 0;
}
```

Stack



Heap

	0	0	= EMPTY
data	l_link	r_link	2000 : tmp 주소

1. 트리(Tree) 코드 이해(그림 그리기) - (ins_tree 2)

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

```
#define EMPTY 0
```

```
struct node{
```

```
int data;
```

```
struct node *l_link;
```

```
struct node *r_link;
```

```
};
```

```
typedef struct node Tree;
```

```
Tree *getnode(){
```

```
Tree *tmp = (Tree *)malloc(sizeof(Tree));
```

```
tmp->l_link = EMPTY;
```

```
tmp->r_link = EMPTY;
```

```
return tmp;
```

```
}
```

```
void ins_tree(Tree **root, int data){
```

```
if( *root == EMPTY){
```

```
    *root = getnode();
```

```
    (*root)->data = data;
```

```
    return;
```

```
}
```

```
if( (*root)->data < data)
```

```
    ins_tree( &((*root)->r_link), data);
```

```
if( (*root)->data > data)
```

```
    ins_tree( &((*root)->l_link), data);
```

```
}
```

```
void print_tree(Tree *root){
```

```
Tree *tmp = root;
```

```
if(tmp != NULL){
```

```
    printf("%d\n", tmp->data);
```

```
    /* if(root->l_link)
```

```
    printf("left = %d, ", root->l_link->data);
```

```
    else
```

```
    printf("left = NULL, ");
```

```
    if(root->r_link)
```

```
    printf("left = %d, ", root->r_link->data);
```

```
    else
```

```
    printf("left = NULL, ");
```

```
    */
```

```
    print_tree(tmp->l_link);
```

```
    print_tree(tmp->r_link);
```

```
}
```

```
}
```

```
int main(void){
```

```
Tree *root = EMPTY;
```

```
int data[13] = {50, 45, 73, 32, 48, 46, 16, 37,
```

```
120, 47, 130, 127, 124};
```

```
int i, num;
```

```
num = sizeof(data)/sizeof(int);
```

```
for( i=0; i<num; i++){
```

```
    ins_tree(&root, data[i]);
```

```
}
```

```
printf("after del\n");
```

```
root = del_tree(root,32);
```

```
print_tree(root);
```

```
return 0;
```

```
}
```

Stack

main

0 = EMPTY → 2000

root 1000 : root 주소

ins_tree

1000

root

50

data

get_node

2000

tmp

⑤

ins_tree는 ⑤ 이후 스택에서 빠짐

Heap

50 0 0 = EMPTY
data l_link r_link 2000 : tmp 주소

1. 트리(Tree) 코드 이해(그림 그리기) - (ins_tree 3)

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

```
#define EMPTY 0
```

```
struct node{
```

```
int data;
```

```
struct node *l_link;
```

```
struct node *r_link;
```

```
};
```

```
typedef struct node Tree;
```

```
Tree *getnode(){
```

```
Tree *tmp = (Tree *)malloc(sizeof(Tree));
```

```
tmp->l_link = EMPTY;
```

```
tmp->r_link = EMPTY;
```

```
return tmp;
```

```
}
```

```
void ins_tree(Tree **root, int data){
```

```
if( *root == EMPTY){
```

```
*root = getnode();
```

```
(*root)->data = data;
```

```
return;
```

```
}
```

```
if( (*root)->data < data)
```

```
ins_tree( &((*root)->r_link), data);
```

```
if( (*root)->data > data)
```

```
ins_tree( &((*root)->l_link), data);
```

```
}
```

```
void print_tree(Tree *root){
```

```
Tree *tmp = root;
```

```
if(tmp != NULL){
```

```
printf("%d\n", tmp->data);
```

```
/* if(root->l_link)
```

```
printf("left = %d, ", root->l_link->data);
```

```
else
```

```
printf("left = NULL, ");
```

```
if(root->r_link)
```

```
printf("left = %d, ", root->r_link->data);
```

```
else
```

```
printf("left = NULL, ");
```

```
*/
```

```
print_tree(tmp->l_link);
```

```
print_tree(tmp->r_link);
```

```
}
```

```
}
```

```
int main(void)
```

```
Tree *root = EMPTY;
```

```
int data[13] = {50, 45, 73, 32, 48, 46, 16, 37,
```

```
120, 47, 130, 127, 124};
```

```
int i, num;
```

```
num = sizeof(data)/sizeof(int);
```

```
for( i=0; i<num; i++){
```

```
ins_tree(&root, data[i]);
```

```
}
```

```
print_tree(root);
```

```
printf("after del\n");
```

```
root = del_tree(root,32);
```

```
print_tree(root);
```

```
return 0;
```

```
}
```

Stack

main

0 = EMPTY → 2000

root 1000 : root 주소

ins_tree

1000

root

45

data

ins_tree

2004

root

45

data

get_node

3000

tmp

⑧

⑨

ins_tree는 ⑨ 이후 스택에서 빠짐

Heap

50 3000 0 = EMPTY

data l_link r_link

2000 = tmp 주소

2004 = l link 주소

2012 = r link 주소

45 0 0 = EMPTY

data l_link r_link

3000 = tmp 주소

1. 트리(Tree) 코드 이해(그림 그리기) - (ins_tree 4)

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

```
#define EMPTY 0
```

```
struct node{
```

```
int data;
```

```
struct node *l_link;
```

```
struct node *r_link;
```

```
};
```

```
typedef struct node Tree;
```

```
Tree *getnode(){
```

```
Tree *tmp = (Tree *)malloc(sizeof(Tree));
```

```
tmp->l_link = EMPTY;
```

```
tmp->r_link = EMPTY;
```

```
return tmp;
```

```
}
```

```
void ins_tree(Tree **root, int data){
```

```
if( *root == EMPTY){
```

```
*root = getnode();
```

```
(*root)->data = data;
```

```
return;
```

```
}
```

```
if( (*root)->data < data)
```

```
ins_tree( &((*root)->r_link), data);
```

```
if( (*root)->data > data)
```

```
ins_tree( &((*root)->l_link), data);
```

```
}
```

```
void print_tree(Tree *root){
```

```
Tree *tmp = root;
```

```
if(tmp != NULL){
```

```
printf("%d\n", tmp->data);
```

```
/* if(root->l_link)
```

```
printf("left = %d, ", root->l_link->data);
```

```
else
```

```
printf("left = NULL, ");
```

```
};
```

```
if(root->r_link)
```

```
printf("left = %d, ", root->r_link->data);
```

```
else
```

```
printf("left = NULL, ");
```

```
*/
```

```
print_tree(tmp->l_link);
```

```
print_tree(tmp->r_link);
```

```
}
```

```
int main(void){
```

```
Tree *root = EMPTY;
```

```
int data[13] = {50, 45, 73, 32, 48, 46, 16, 37,
```

```
120, 47, 130, 127, 124};
```

```
int i, num;
```

```
num = sizeof(data)/sizeof(int);
```

```
for( i=0; i<num; i++){
```

```
ins_tree(&root, data[i]);
```

```
}
```

```
print_tree(root);
```

```
printf("after del\n");
```

```
root = del_tree(root,32);
```

```
print_tree(root);
```

```
return 0;
```

```
}
```

Stack

main

0 = EMPTY → 2000

root 1000 : root 주소

ins_tree

1000

root

73

data

ins_tree

2012

root

73

data

get_node

4000

tmp

⑫

⑬

⑩

⑪

⑨

⑧

⑦

⑥

⑤

④

③

②

①

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

1. 트리(Tree) 코드 이해(그림 그리기) - (ins_tree 5)

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

```
#define EMPTY 0
```

```
struct node{
```

```
int data;
```

```
struct node *l_link;
```

```
struct node *r_link;
```

```
};
```

```
typedef struct node Tree;
```

```
Tree *getNode(){
```

```
Tree *tmp = (Tree *)malloc(sizeof(Tree));
```

```
tmp->l_link = EMPTY;
```

```
tmp->r_link = EMPTY;
```

```
return tmp;
```

```
}
```

```
void ins_tree(Tree **root, int data){
```

```
if( *root == EMPTY){
```

```
*root = getNode();
```

```
(*root)->data = data;
```

```
return;
```

```
}
```

```
if( (*root)->data < data)
```

```
ins_tree( &((*root)->r_link), data);
```

```
if( (*root)->data > data)
```

```
ins_tree( &((*root)->l_link), data);
```

```
}
```

```
void print_tree(Tree *root){
```

```
Tree *tmp = root;
```

```
if(tmp != NULL){
```

```
printf("%d\n", tmp->data);
```

```
/* if(root->l_link)
```

```
printf("left = %d, ", root->l_link->data);
```

```
else
```

```
printf("left = NULL, ");
```

```
if(root->r_link)
```

```
printf("right = %d, ", root->r_link->data);
```

```
else
```

```
printf("right = NULL, ");
```

```
*/
```

```
print_tree(tmp->l_link);
```

```
print_tree(tmp->r_link);
```

```
}
```

```
int main(void){
```

```
Tree *root = EMPTY;
```

```
int data[13] = {50, 45, 73, 32, 48, 46, 16, 37,
```

```
120, 47, 130, 127, 124};
```

```
int i, num;
```

```
num = sizeof(data)/sizeof(int);
```

```
for( i=0; i<num; i++){
```

```
ins_tree(&root, data[i]);
```

```
}
```

```
print_tree(root);
```

```
printf("after del\n");
```

```
root = del_tree(root,32);
```

```
print_tree(root);
```

```
return 0;
```

```
}
```

Stack

main

2000

root 1000 : root 주소

① ~ ⑭

와 같은 방식으로 반복

⑮

Heap

50 3000 4000

data l_link r_link

2000 = tmp 주소

2004 = l link 주소

2012 = r link 주소

45 5000 6000

data l_link r_link

3000 = tmp 주소

3004 = l link 주소

3012 = r link 주소

73 0 9000

data l_link r_link

4000 = tmp 주소

4004 = l link 주소

4012 = r link 주소

32 8000 0

data l_link r_link

5000 = tmp 주소

5004 = l link 주소

5012 = r link 주소

48 7000 0

data l_link r_link

6000 = tmp 주소

6004 = l link 주소

6012 = r link 주소

120 0 0

data l_link r_link

9000 = tmp 주소

9004 = l link 주소

9012 = r link 주소

16 0 0

data l_link r_link

8000 = tmp 주소

8004 = l link 주소

8012 = r link 주소

46 0 0

data l_link r_link

7000 = tmp 주소

7004 = l link 주소

7012 = r link 주소