코드)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define POOL_SIZE 500

struct record {
    int a;
    struct record * left;
    struct record * right;
};

static struct record pool[POOL_SIZE]; // static because pool is strictly private
struct record * top=pool;  // a pointer variable for pool stack top.

int compare(int num, struct record *r)
{
    if(num < r->a)
        return -1;
    else if (num > r->a)
        return 1;
    else
        return 0;
}
struct record * data = NULL;

struct record * new_node()
{
    struct record *r;

    if(top==NULL)
        return NULL;

    r=top;
    top=r->right;  // Again, right instead of next.
    return r;
}
// Push a node to the pool.
void free_node(struct record *r)
{
    r->right=top;  // Again, right instead of next.
    top=r;
}

void init_pool() // Initialize the pool; Use right instead of next!!!
{
    int i;
    struct record *r=pool;
    struct record *s;

    pool[POOL_SIZE-1].right=NULL;

    for(i=1;i<POOL_SIZE;i++) {
        s=r++;
        s->right=r;
    }
}

void add(int randnum)
{
    struct record *p;
    struct record *q;
    struct record *r;
    int chk = 7;
    p = data;


    while(p != NULL)
    {
```

```
69          {
70              chk = compare(randnum, p);
71
72              if(chk == 0 || chk < 0)
73              {
74                  q = p;
75                  p = p->left;
76              }
77              else
78              {
79                  q = p;
80                  p = p->right;
81              }
82          }
83
84          r = new_node();
85          if(r == NULL)
86          {
87              printf("Can't add.  The pool is empty!\n");
88          }
89          else{
90              r->a = randnum;
91              r->left = NULL;
92              r->right = NULL;
93
94              if(data == NULL)
95              {
96                  data = r;
97              }
98              else
99              {
100                 chk = compare(randnum, q);
101                 if(chk <= 0)
102                 {
103                     q->left = r;
104                 }
105                 else
106                 {
107                     q->right = r;
108                 }
109             }
110         }
111     }
112
113     int height(struct record *t)
114      {
115         if(t == NULL)
116         {
117             return -1;
118         }
119         else
120         {
121             if(height(t->left) < height(t->right))
122             {
123                 return height(t->right)+1;
124             }
125             else
126             {
127                 return height(t->left)+1;
128             }
129         }
130     }
131
132     void delete(struct record *t)
133      {
134         if(t == NULL)
135         {
136             return;
137         }
```

```
138         delete(t->left);
139         delete(t->right);
140         free_node(t);
141     }
142
143
144
145     int main()
146     {
147         int num = 0; // 노드의 개수
148         int sum = 0; // 시행된 트리들의 height의 총합
149         int cnt = 1; // 시행횟수 count
150         printf("노드의 개수를 입력하세요 : ");
151         scanf("%d", &num);
152         srand((int)time(NULL));
153         init_pool();
154
155         while(cnt != 10000) {
156             for (int i = 0; i < num; i++) {
157                 int b = rand() % 100; // 랜덤한 데이터
158                 add(b);
159             }
160             int t = height(data);
161             sum += t;
162             if(cnt % 1000 == 0)
163                 printf("\n");
164             printf("%d ", t);
165             cnt++;
166             delete(data); // 트리 비우기
167             data = NULL;
168         }
169         printf("\n");
170         printf("height 평균 : %lf\n", (double)sum / 10000);
171     }
```

결과)

```
노드의 개수를 입력하세요 : 50
10 10 12 7 9 9 8 9 9 12 11 11 9 9 14
9 11 10 12 9 10 9 10 8 9 11 9 8 8 12
10 13 11 8 8 11 11 8 8 10 12 11 11 1
9 10 11 12 12 10 8 8 13 8 11 11 9 9
8 12 9 10 11 11 11 11 8 9 10 7 11 11
10 10 9 10 9 10 9 12 13 10 9 10 11 1
9 8 10 9 8 10 8 9 10 8 10 11 11 11 1
8 13 8 10 12 11 9 10 14 11 10 9 11 1
9 9 9 8 9 10 10 10 10 12 8 11 8 8 12
8 11 8 8 9 10 9 10 8 9 16 11 12 10 8
height 평균 : 9.843800

노드의 개수를 입력하세요 : 100
14 12 15 13 15 15 13 11 13 10 13 12
14 11 15 12 15 13 13 14 16 10 15 12
13 11 10 12 14 10 11 12 12 12 13 9 1
11 11 15 14 13 12 13 15 11 11 13 13
13 12 11 18 14 12 13 12 14 13 11 11
14 12 10 12 14 12 11 15 11 10 11 11
13 12 11 12 10 10 10 13 11 14 11 12
11 14 11 11 10 13 11 13 12 15 11 12
13 15 11 12 11 10 18 12 13 11 11 9 1
16 14 11 13 12 11 13 16 13 13 12 13
height 평균 : 12.365000

노드의 개수를 입력하세요 : 200
20 14 18 14 15 14 17 14 14 15 14 15
14 15 16 16 16 16 14 14 13 14 15 13
14 17 14 15 14 13 15 15 14 13 16 15
16 13 15 16 13 16 19 14 16 15 16 12
16 17 19 13 16 15 15 13 13 17 16 15
19 15 15 15 15 12 19 13 14 14 17 15
17 17 17 15 14 14 14 13 13 20 15 18
18 15 12 13 14 17 17 18 17 15 14 14
15 15 16 19 14 14 14 15 14 18 16 15
16 14 14 15 17 16 14 18 16 15 15 17
height 평균 : 15.257800

노드의 개수를 입력하세요 : 500
21 19 18 23 21 18 19 21 19 19 24 18
21 25 18 19 17 22 23 22 19 24 22 24
23 23 20 17 22 19 19 21 24 22 21 22
20 20 22 23 18 20 18 21 23 20 19 19
19 18 22 23 20 21 22 22 21 19 20 21
21 20 21 19 21 23 19 22 21 20 21 22
19 20 19 19 20 22 19 18 21 17 18 21
24 19 20 19 18 19 21 18 19 22 25 20
19 23 21 19 21 22 21 21 19 20 21 22
22 22 19 22 20 20 18 20 18 22 21 20
height 평균 : 20.545400
```

토의)

이번과제는 노드의 개수 N이 주어졌을 때 N개의 랜덤데이터를 통해 만들 수있는 BST의 높이를 구하는 값을 반복하여 각 트리들의 높이의 평균을 구하는 과제다. 과제를 하기위해 이전과제와 다르게 개별구조체의 데이터형을 정수형으로 고치고 pool size도 500으로 하여 과제에서 요구한 노드의 개수 500개 도 받을 수 있게 설정하였다. 그리고 add함수의 인자를 정수형으로 하여 새로운 노드가 생성될때 정수형 데이터를 받을 수 있게 하고 compare 함수도 수정하여 수의크기비교로 트리를 구성할 수 있도록 만들었다. main함수에 들어가서 무작위의 데이터를 받을 수있도록 rand, srand, time 함수를 사용했다. 시행횟수는 10000번으로 수행하고 과제의 별개로 200번으로도 수행을 해보았다. 수행과정중 delete 함수를 구현하지 않아 첫수행이 끝나고 난뒤 다시 수행을 반복할때 노드의 반환이 이뤄지지 않아 poolsize 이상의 노드를 필요로하여 에러가 발생하였고 이를 해결하기 위해 Binary tree를 탐색하기 위한 3가지 방식중 post order를 사용하여 트리의 모든 노드를 반환시켰고 이후 원활한 수행이 가능했다.

결과 10000번과 200번 시행모두 $2\lg N$ ~ $3\lg N$ 사이의 값이 노드의 평균으로 구해졌다. 하지만 두 시행모두 결괏값에 큰 차이가 없었지만 결과가 나오는데 까지의 시간은 10000번 시행이 더 오래걸리는 단점이 있었지만 10000번시행이 더 정확한 결괏값을 나타내는 장점이 있다. 그리고 알 수 있던 점은 대다수의 트리들이 complete 한 트리가 아니고 대체로 $\lg N$에 비교적 가까운 height가 나왔고 반면에 N수에 가까운 height는 볼 수 없었다. 이는 사용자가 임의로 데이터를 큰수부터 작은수 혹은 작은수부터 큰수 까지 차례로 넣는 것 또는 의도한 height를 구현하기 위해 넣는 것이 아니라면 임의의 무작위한 데이터를 넣는 프로그램에서는 BST의 특성(작은 수는 왼쪽 서브트리 큰수는 오른쪽 서브트리) 때문에 과제 결과와 비교했을때 비교적 다양한 결과는 얻기 어려울 것같다.