**hw9\_1.cpp**

Visual Studio2019

**1. Code Explanation**

1) int transform(char\* key)

: transform the string key into and integer by summing ASCII codes of an each character

-key: pointer of the string

-number: sum of the ASCII codes

2) int hash\_function(char\* key)

: division function (key mod TABLE SIZE)

-key: pointer of the string

3) void hash\_chain\_add(element item, ListNode\* ht[])

: add item to the hash table

-item: string we will add to the hash table

-ht: deliver hash\_table[TABLE\_SIZE] to this parameter

4) void hash\_chain\_search(element item, ListNode\* ht[])

: search item and print whether the search is successed or not

-item: string we will add to the hash table

-ht: deliver hash\_table[TABLE\_SIZE] to this parameter

5) void hash\_chain\_print(ListNode\* ht[])

: print the hash table

-item: string we will add to the hash table

-ht :deliver hash\_table[TABLE\_SIZE] to this parameter

6) void init\_table(ListNode\* ht[])

: initialize the hash tale with null pointer

-ht: deliver hash\_table[TABLE\_SIZE] to this parameter

->each node is initialized as null

7) void hash\_chain\_delete(element item, ListNode\* ht[])

: delete item in the hash table

-t: currnet node

-node\_before: node which is located right before t

->find node which is equal to item and save it in 't'

->link a node before t and a node after t

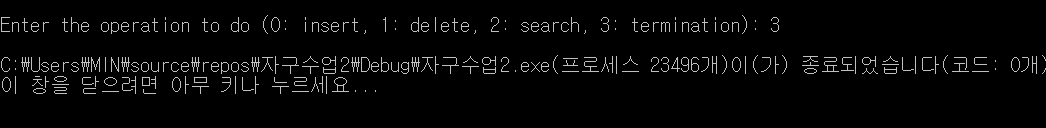
->deallocate the memory of 't' (remove t)

**2. Results**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**hw9\_2.cpp**

**1. Code Explanation**

1) int random(int max\_value)

: generate a random number of range 1 to max\_value using rand() function

-max\_value: maximum range of number we can generate

2) void insert\_node(TreeNode\*\* root, int key)

: Insert the key into the binary search tree. If key is already in tree, it is not inserted.

-root: pointer of the root node

-key: value we will insert

-p: parent node

-t: current node

-n: new node to be inserted

-> search first

while (t != NULL) {

p = t;

if (key < t->key) t = t->left;

else t = t->right;

}

-> allocate memory of ‘n’ and save key value

->insert n to its location

if (p != NULL) //when the tree is not empty

if (key < p->key)

p->left = n;

else p->right = n;

else \*root = n; //when the tree is empty

3) void inorder(TreeNode\* root)

: let's use inorder traversal to print the sorted results(inorder traversal: visit left subtree, root node, right subtree in order)

-root: node we will visit

-> use recursive method

4) int main()

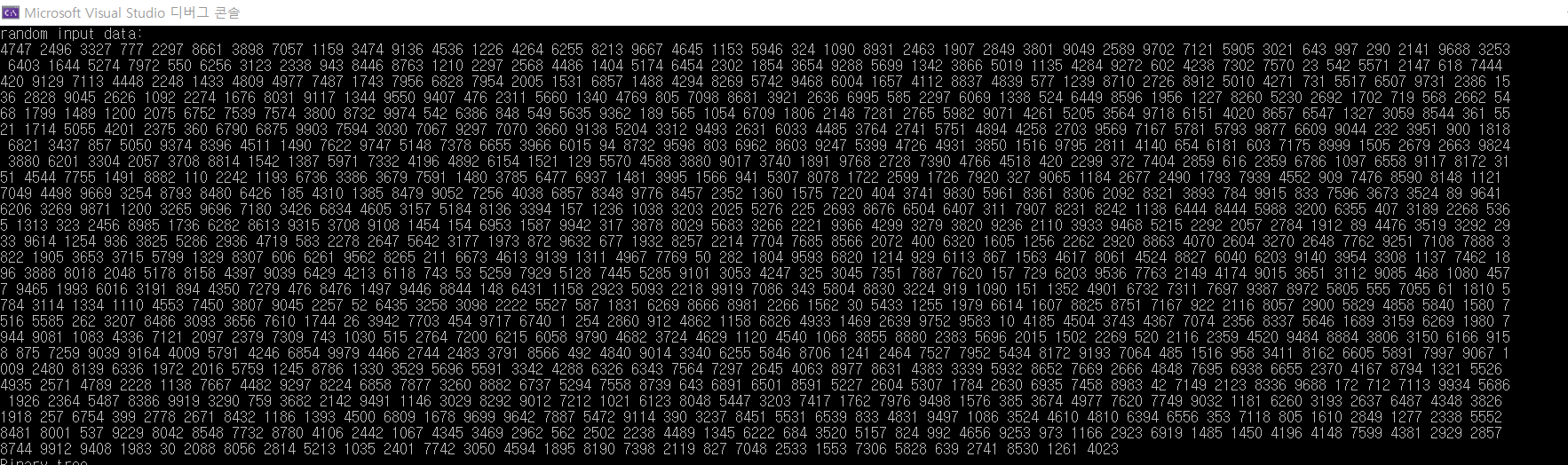
-> generate 1000 input data randomly using random() function

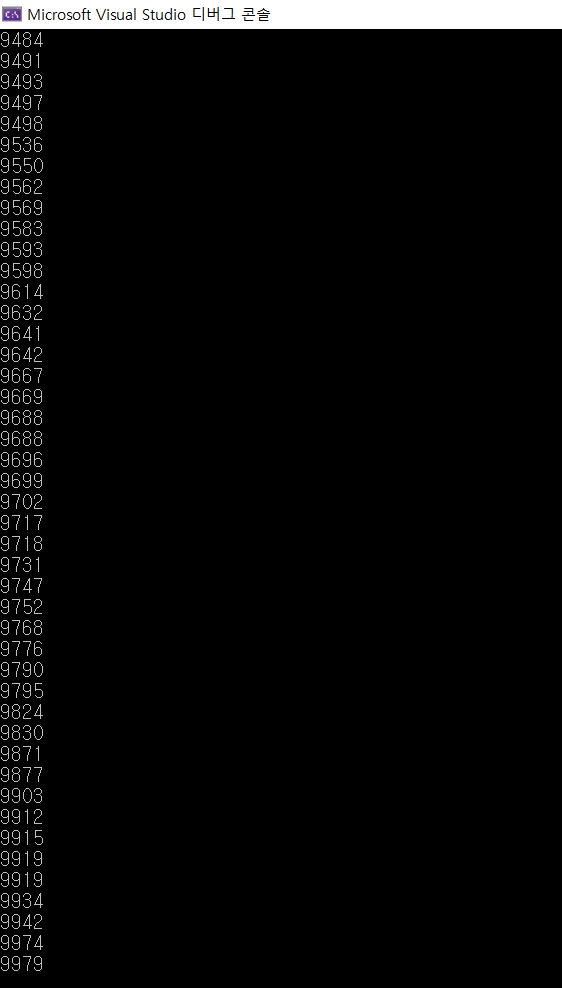
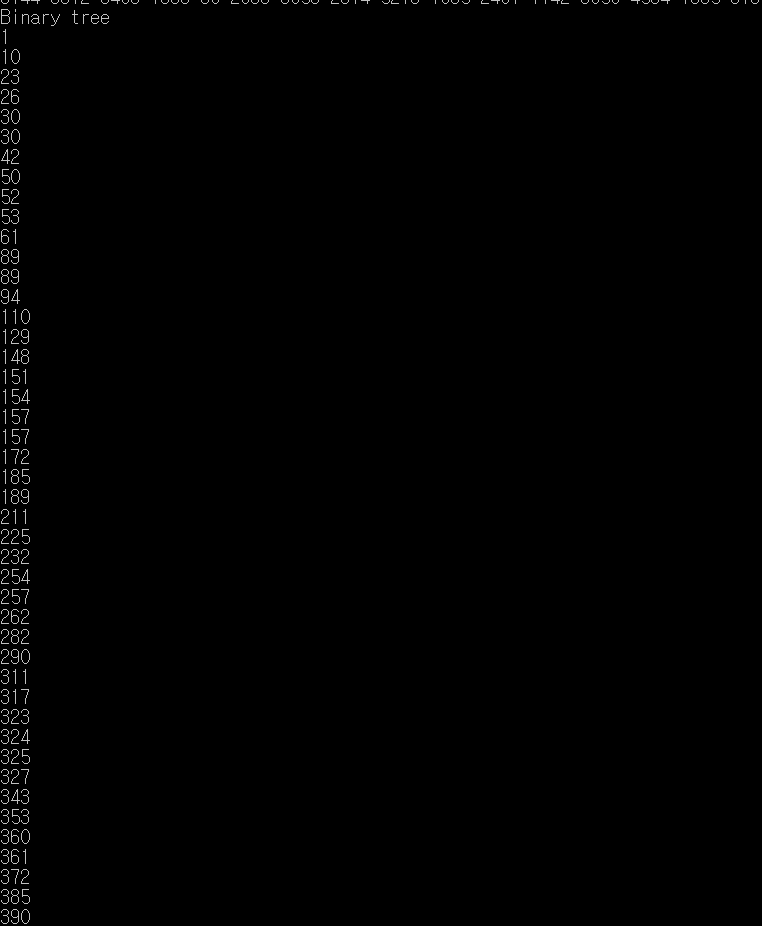
-> print out the input data

->insert elements iteratively into the binary search tree using insert\_node() function

-> print the sorted results using inorder() function

**2. Results**

****randomly generated 1000 inputs

****

Sorted results(중간에 생략함)