

Assignment 2 V1.0

Session 16

1. [30%] Code Snippet 1

Given snippet code below that you are required to complete. You are not allowed to make a new function or change any given code. Please complete each section that are marked with the notation "INSERT YOUR CODE HERE".

Once you complete the snippet below, your output should have the same result with the given output below.

Descriptions:

a. [10%] generateKey()

This function is built for generating key in the hashTable. The key should be generated by calculate the summation of each ascii character of attribute name, subsequently use division method in corresponding to tableSize.

b. [10%] searchNode()

This function is built for searching whether the current node has the same data with the existing data in the hash table. If it is found, then the location of the current node will be returned. The similary checking by comparing the name and category.

c. [10%] push()

This function is built for inserting a new data in the hashTable. This hashTable could handle the duplicated data by increasing the qty number. If the new data is novel (not found in the existing hashTable), then the new node will be generated in the hashTable.

Assignment 2 V1.0

Session 16

```
//  
// Created by Hanry Ham on 2020-05-24.  
//  
  
#include<stdio.h>  
#include<string.h>  
#include<stdlib.h>  
  
#define tableSize 29  
  
struct storage{  
    int key;  
    int qty;  
    char name[25];  
    char category[20];  
    struct storage *next;  
};  
  
struct hashPool{  
    struct storage *head;  
    struct storage *tail;  
};  
struct hashPool hashTable[tableSize];  
  
void init(){  
    for(int i=0; i<tableSize; i++){  
        hashTable[i].head = NULL;  
        hashTable[i].tail = NULL;  
    }  
}  
  
int generateKey(char *name){  
    int key = 0;  
    int len = strlen(name);  
    // [10%] (1) INSERT YOUR CODE HERE  
    int sum = 0;  
    for(int i=0; i<len; i++){  
        sum += (int)(name[i]);  
    }  
    key = sum%tableSize;  
    return key;  
}  
  
struct storage *newNode(char *name, char *category){  
    struct storage *curr = (struct storage *)malloc(sizeof(struct  
storage));  
    curr->key = generateKey(name);  
    curr->qty = 1;  
    strcpy(curr->name, name);  
    strcpy(curr->category, category);  
    curr->next = NULL;  
    return curr;  
}
```

Assignment 2 V1.0

Session 16

```
struct storage *searchNode(struct storage *node){
    struct storage *search= hashTable[node->key].head;
    struct storage *temp = NULL;
    // [10%] (2) INSERT YOUR CODE HERE
    temp = search;
    while(temp!=NULL && strcmp(temp->name, node->name)){
        temp = temp->next;
    }
    return temp;
}

void push(struct storage *node){
    // [10%] (3) INSERT YOUR CODE HERE
    int key = node->key;
    if(hashTable[key].head == NULL){
        hashTable[key].head = hashTable[key].tail = node;
        return;
    }
    struct storage *tmp = searchNode(node);
    if(tmp==NULL){
        hashTable[key].tail->next = node;
        hashTable[key].tail = node;
    } else { tmp->qty++; }
}

void printAll(){
    for(int i=0; i<tableSize; i++){
        printf("[%2d] : ", i);
        struct storage *curr = hashTable[i].head;
        while(curr != NULL){
            printf("%-10s (%-3d) -> ", curr->name, curr->qty);
            curr = curr->next;
        }
        printf("NULL\n");
    }
}

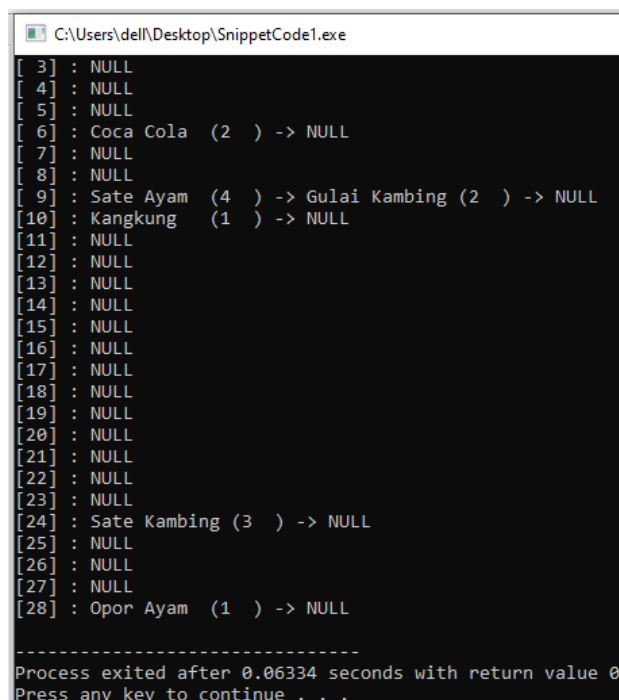
int main(){
    init();
    push(newNode("Sate Ayam", "Daging"));
    push(newNode("Sate Ayam", "Daging"));
    push(newNode("Gulai Kambing", "Daging"));
    push(newNode("Kangkung", "Sayuran"));
    push(newNode("Sate Ayam", "Daging"));
    push(newNode("Coca Cola", "Minuman"));
    push(newNode("Sate Ayam", "Daging"));
    push(newNode("Coca Cola", "Minuman"));
    push(newNode("Gulai Kambing", "Daging"));
    push(newNode("Sate Kambing", "Daging"));
    push(newNode("Opor Ayam", "Daging"));
    push(newNode("Sate Kambing", "Daging"));
    push(newNode("Sate Kambing", "Daging"));
    printAll();
    return 0;
}
```

Assignment 2 V1.0

Session 16

Output:

```
[ 0] : NULL
[ 1] : NULL
[ 2] : NULL
[ 3] : NULL
[ 4] : NULL
[ 5] : NULL
[ 6] : Coca Cola (2 ) -> NULL
[ 7] : NULL
[ 8] : NULL
[ 9] : Sate Ayam (4 ) -> Gulai Kambing (2 ) -> NULL
[10] : Kangkung (1 ) -> NULL
[11] : NULL
[12] : NULL
[13] : NULL
[14] : NULL
[15] : NULL
[16] : NULL
[17] : NULL
[18] : NULL
[19] : NULL
[20] : NULL
[21] : NULL
[22] : NULL
[23] : NULL
[24] : Sate Kambing (3 ) -> NULL
[25] : NULL
[26] : NULL
[27] : NULL
[28] : Opor Ayam (1 ) -> NULL
```



```
C:\Users\del\Desktop\SnippetCode1.exe
[ 3] : NULL
[ 4] : NULL
[ 5] : NULL
[ 6] : Coca Cola (2 ) -> NULL
[ 7] : NULL
[ 8] : NULL
[ 9] : Sate Ayam (4 ) -> Gulai Kambing (2 ) -> NULL
[10] : Kangkung (1 ) -> NULL
[11] : NULL
[12] : NULL
[13] : NULL
[14] : NULL
[15] : NULL
[16] : NULL
[17] : NULL
[18] : NULL
[19] : NULL
[20] : NULL
[21] : NULL
[22] : NULL
[23] : NULL
[24] : Sate Kambing (3 ) -> NULL
[25] : NULL
[26] : NULL
[27] : NULL
[28] : Opor Ayam (1 ) -> NULL

-----
Process exited after 0.06334 seconds with return value 0
Press any key to continue . . .
```

Assignment 2 V1.0

Session 16

2. [30%] Code Snippet 2

Given snippet code below that you are required to complete. You are not allowed to make a new function or change any given code. Please complete each section that are marked with the notation "INSERT YOUR CODE HERE".

Once you complete the snippet below, your output should have the same result with the given output below.

Descriptions:

a. [15%] linearProbing()

This function is built for executing linearProbing. Basically, linearProbing will check all the index in hashTable, and will put new data in the empty index. If all the indexes are occupied then it will return -1 and if the index in hashTable is still empty, it will return the new key.

b. [15%] insert()

This function is built for inserting a new data by using linearProbing.

Assignment 2 V1.0

Session 16

```
//  
// Created by Hanry Ham on 2020-05-24.  
//  
  
#include<stdio.h>  
#include<string.h>  
#include<stdlib.h>  
  
struct storage{  
    char name[25];  
    char category[20];  
};  
const int tableSize = 7;  
struct storage hashTable[tableSize];  
  
void init(){  
    for(int i=0; i<tableSize; i++){  
        strcpy(hashTable[i].name, "");  
        strcpy(hashTable[i].category, "");  
    }  
}  
int generateKey(const char *name){  
    int key;  
    key = name[0] - 'A';  
    return key % tableSize;  
}  
int linearProbing(int key){  
    // [15%] (1) INSERT YOUR CODE HERE  
    while (key < tableSize){  
        if (strcmp(hashTable[i].name, "") == 0){  
            return i;  
        }  
        i++;  
        if (i == tableSize){  
            i = 0;  
        }  
        if (i == key){  
            return -1;  
        }  
    }  
    return -1;  
}  
void insert(const char *name, const char *category){  
    // [15%] (2) INSERT YOUR CODE HERE  
    int index = generateKey(name);  
    if (strlen(hashTable[index].name) > 0){  
        index = linearProbing(index);  
    }  
    if (index == -1){  
        printf("The hashTable is full!\n");  
    }  
    strcpy(hashTable[index].name, name);  
    strcpy(hashTable[index].category, category);  
}
```

Assignment 2 V1.0

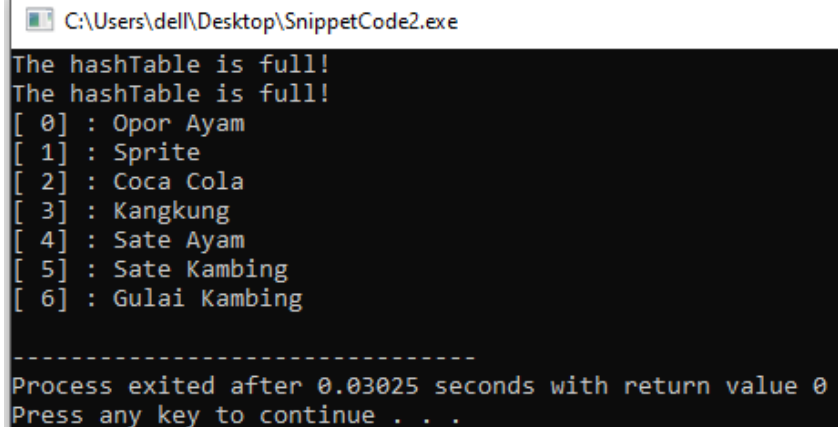
Session 16

```
void printAll(){
    for(int i=0; i<tableSize; i++){
        printf("[%2d] : ", i);
        if(strcmp(hashTable[i].name, "") != 0){
            printf("%s", hashTable[i].name);
        }else{
            printf("NULL");
        }
        printf("\n");
    }
}

int main(){
    init();
    insert("Sate Ayam", "Daging");
    insert("Gulai Kambing", "Daging");
    insert("Kangkung", "Sayuran");
    insert("Coca Cola", "Minuman");
    insert("Sate Kambing", "Daging");
    insert("Opor Ayam", "Daging");
    insert("Sprite", "Minuman");
    insert("Fanta", "Minuman");
    insert("Ayam Kalasan", "Daging");
    printAll();
    return 0;
}
```

Output:

```
The hashTable is full!
The hashTable is full!
[ 0] : Opor Ayam
[ 1] : Sprite
[ 2] : Coca Cola
[ 3] : Kangkung
[ 4] : Sate Ayam
[ 5] : Sate Kambing
[ 6] : Gulai Kambing
```



```
C:\Users\del\Desktop\SnippetCode2.exe
The hashTable is full!
The hashTable is full!
[ 0] : Opor Ayam
[ 1] : Sprite
[ 2] : Coca Cola
[ 3] : Kangkung
[ 4] : Sate Ayam
[ 5] : Sate Kambing
[ 6] : Gulai Kambing

-----
Process exited after 0.03025 seconds with return value 0
Press any key to continue . . .
```

Assignment 2 V1.0

Session 16

3. [40%] Code Snippet 3

Given snippet code below that you are required to complete. You are not allowed to make a new function or change any given code. Please complete each section that are marked with the notation "INSERT YOUR CODE HERE".

Once you complete the snippet below, your output should have the same result with the given output below.

Descriptions:

a. [10%] insert()

This function is built for inserting a new node in our existing Binary Search Tree (BST). In this insert function, you should handle if any duplicated data found, then the quantity should be increase as well.

b. [5%] predecessor()

This function is built for finding the left right most node in the existing BST.

c. [5%] successor()

This function is built for finding the right left most node in the existing BST.

d. [15%] deleteKey()

This function is built for deleting a node in our BST. If the qty of the interested node > 1, then program will decrease the number of qty by 1. On the other hand, if the qty = 1, then you are allowed to delete the node.

e. [5%] inOrder()

This function is built for printing out the existing BST inOrder way.

Assignment 2 V1.0

Session 16

```
//
// Created by Hanry Ham on 2020-05-25.
//

#include<stdio.h>
#include<string.h>
#include<stdlib.h>

struct storage{
    int qty;
    char name[25];
    char category[20];
    struct storage *left;
    struct storage *right;
};

struct storage *newNode(const char *name, const char *category){
    struct storage *curr = (struct storage *) malloc(sizeof(struct
storage));
    curr->qty = 1;
    strcpy(curr->name, name);
    strcpy(curr->category, category);
    curr->left = NULL;
    curr->right = NULL;
    return curr;
}

struct storage *insert(struct storage *root, const char *name, const
char *category){
    // [10%] (1) INSERT YOUR CODE HERE
    if (root == NULL){
        return(newNode(name, category));
    }
    if(strcmp(name , root->name) == 0) {
        root->qty += 1;
        return root;
    }else if (strcmp(name, root->name) < 0){
        root->left = insert(root->left, name, category);
    }else if (strcmp(name, root->name) > 0){
        root->right = insert(root->right, name, category);
    }
    return root;
}

struct storage *predecessor(struct storage *root){
    // [5%] (2) INSERT YOUR CODE HERE
    struct storage* current = root->left;
    while (current->right != NULL){
        current = current->right;
    }
    return current;
}
```

Assignment 2 V1.0

Session 16

```
struct storage *successor(struct storage *root){
    // [5%] (3) INSERT YOUR CODE HERE
    struct storage* current = root->right;
    while (current->left != NULL){
        current = current->left;
    }
    return current;
}

struct storage *deleteKey(struct storage *root, const char *name){
    // [15%] (4) INSERT YOUR CODE HERE
    int check = 0;
    if (root == NULL){
        return root;
    }
    if (strcmp(name, root->name) < 0){
        root->left = deleteKey(root->left, name);
    }else if (strcmp(name, root->name) > 0){
        root->right = deleteKey(root->right, name);
    }else if (strcmp(name, root->name)==0){
        root->qty = root->qty - 1;
        return root;
    }else{
        if((root->left == NULL) || (root->right == NULL)){
            struct storage *temp = root->left ? root->left : root->right;
            if (temp == NULL){
                temp = root;
                root = NULL;
            }else{
                *root = *temp;
            }
            free(temp);
            check = 1;
        }else{
            struct storage* temp = successor(root->right);
            strcpy(root->name, temp->name);
            strcpy(root->category, temp->category);
            root->right = deleteKey(root->right, temp->name);
        }
    }
    return root;
}

void inOrder(struct storage *root){
    if(root){
        // [5%] (5) INSERT YOUR CODE HERE
        inOrder(root->left);
        if(root->qty > 0){
            printf("%s (%d)\n", root->name, root->qty);
        }
        inOrder(root->right);
    }
}
```

Assignment 2 V1.0

Session 16

```
struct storage *freeAll(struct storage *root){
    if(root){
        freeAll(root->left);
        freeAll(root->right);
        free(root);
        root = NULL;
    }
    return root;
}

int main(){
    struct storage *root = NULL;

    root = insert(root, "Sate Ayam", "Daging");
    root = insert(root, "Gulai Kambing", "Daging");
    root = insert(root, "Kangkung", "Sayuran");
    root = insert(root, "Coca Cola", "Minuman");
    root = insert(root, "Sate Kambing", "Daging");
    root = insert(root, "Opor Ayam", "Daging");
    root = insert(root, "Sprite", "Minuman");
    root = insert(root, "Fanta", "Minuman");
    root = insert(root, "Ayam Kalasan", "Daging");
    root = insert(root, "Kangkung", "Sayuran");
    root = insert(root, "Fanta", "Minuman");
    root = insert(root, "Coca Cola", "Minuman");
    root = insert(root, "Opor Ayam", "Daging");

    printf("Predecessor : %s\n", predecessor(root)->name);
    printf("Successor : %s\n", successor(root)->name);

    printf("\nInorder : \n");
    inorder(root);

    root = deleteKey(root, "Sate Ayam");
    root = deleteKey(root, "Gulai Kambing");
    root = deleteKey(root, "Coca Cola");
    root = deleteKey(root, "Opor Ayam");
    root = deleteKey(root, "Sate Kambing");
    root = deleteKey(root, "Ayam Kalasan");
    printf("\nAfter Del Inorder : \n");
    inorder(root);

    freeAll(root);
    return 0;
}
```

Assignment 2 V1.0

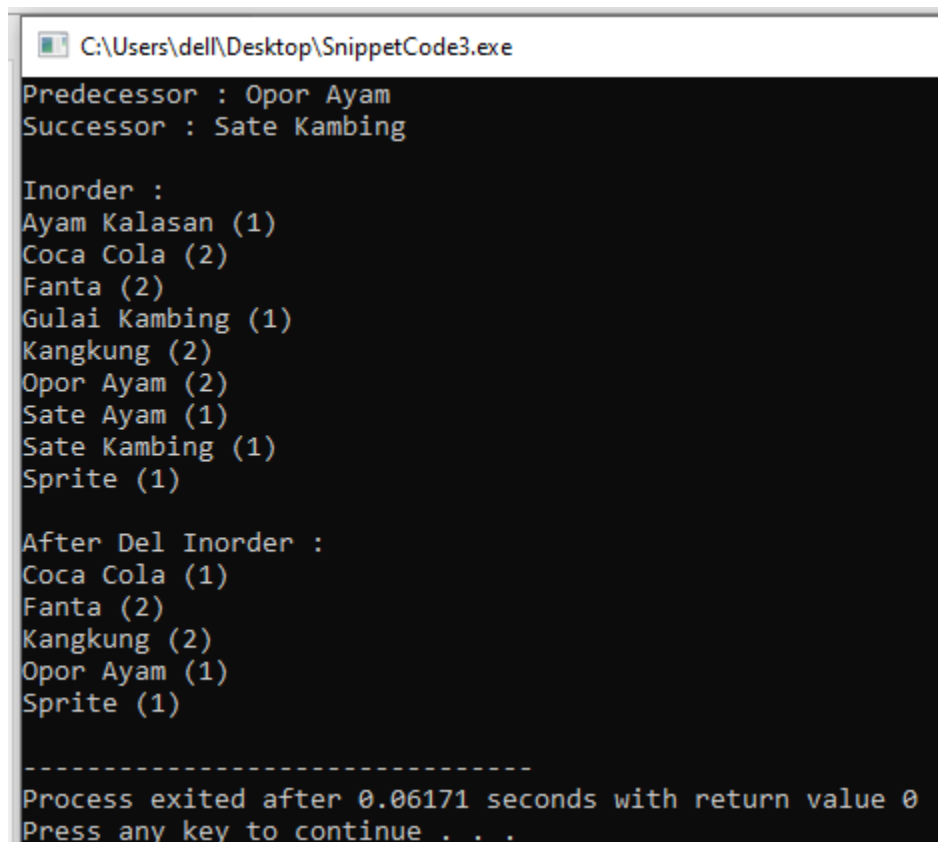
Session 16

Output:

```
Predecessor : Opor Ayam  
Successor : Sate Kambing
```

```
Inorder :  
Ayam Kalasan(1 )  
Coca Cola (2 )  
Fanta (2 )  
Gulai Kambing(1 )  
Kangkung (2 )  
Opor Ayam (2 )  
Sate Ayam (1 )  
Sate Kambing(1 )  
Sprite (1 )
```

```
After Del Inorder :  
Coca Cola (1 )  
Fanta (2 )  
Kangkung (2 )  
Opor Ayam (1 )  
Sprite (1 )
```



```
C:\Users\del\Desktop\SnippetCode3.exe  
Predecessor : Opor Ayam  
Successor : Sate Kambing  
  
Inorder :  
Ayam Kalasan (1)  
Coca Cola (2)  
Fanta (2)  
Gulai Kambing (1)  
Kangkung (2)  
Opor Ayam (2)  
Sate Ayam (1)  
Sate Kambing (1)  
Sprite (1)  
  
After Del Inorder :  
Coca Cola (1)  
Fanta (2)  
Kangkung (2)  
Opor Ayam (1)  
Sprite (1)  
  
-----  
Process exited after 0.06171 seconds with return value 0  
Press any key to continue . . .
```