S.No: 1	Exp. Name: ***Project Module***	**Date: 2024-06-13**

Aim:

Project Module

Source Code:

hello.c

ID: 2303811710422124

2023-2027-K

K.Ramakrishnan College of Technology

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

// Define a structure to represent a student
struct Student {
    int id;
    char name[50];
    int *grades;
    int num_grades;
    struct Student *left;
    struct Student *right;
};

// Function prototypes
void addStudent(struct Student **root);
void removeStudent(struct Student **root, int id);
struct Student* findMin(struct Student *root);
void displayStudents(struct Student *root);
void displayStudent(struct Student *student);
float calculateAverage(int *grades, int num_grades);
void freeMemory(struct Student *root);
int main() {
    struct Student *root = NULL;
    int choice;
    do {
        printf("\nStudent Grade Checker\n");
        printf("1. Add Student\n");
        printf("2. Remove Student\n");
        printf("3. Display Students\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                addStudent(&root);
                break;
            case 2: {
                int id;
                printf("Enter student ID to remove: ");
                scanf("%d", &id);
                removeStudent(&root, id);
                break;
            }
            case 3:
                displayStudents(root);
                break;
            case 4:
                printf("Exiting program...\n");
                break;
                        default:
                        printf("Invalid choice. Please enter a number between 1 and
4.\n");
        }
    } while (choice != 4);
```

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

// Input Module: Add Student
void addStudent(struct Student **root) {
    struct Student *newStudent = (struct Student *)malloc(sizeof(struct Student));
    if (newStudent == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newStudent->left = NULL;
    newStudent->right = NULL;
    printf("Enter student ID: ");
    scanf("%d", &newStudent->id);
    printf("Enter student name: ");
    scanf("%s", newStudent->name);
    printf("Enter number of grades: ");
    scanf("%d", &newStudent->num_grades);
    newStudent->grades = (int *)malloc(newStudent->num_grades * sizeof(int));
    if (newStudent->grades == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    for (int i = 0; i < newStudent->num_grades; i++) {
        printf("Enter grade %d: ", i + 1);
        scanf("%d", &newStudent->grades[i]);
        }
        // Insert the new student into the BST
    if (*root == NULL) {
        *root = newStudent;
    } else {
        struct Student *current = *root;
        struct Student *parent = NULL;
        while (current != NULL) {
            parent = current;
            if (newStudent->id < current->id) {
                current = current->left;
            } else {
                current = current->right;
            }
        }
        if (newStudent->id < parent->id) {
            parent->left = newStudent;
        } else {
            parent->right = newStudent;
        }
    }
}
// Output Module: Display Students (In-order traversal)
void displayStudents(struct Student *root) {
    printf("\nStudent List\n");
    displayStudent(root);
}
```

```
        displayStudent(student->left);
        printf("ID: %d, Name: %s, Grades: ", student->id, student->name);
        for (int i = 0; i < student->num_grades; i++) {
            printf("%d ", student->grades[i]);
        }
        printf("Average: %.2f\n", calculateAverage(student->grades, student-
>num_grades));
        displayStudent(student->right);
    }
}
// Calculate average grade
float calculateAverage(int *grades, int num_grades) {
    int sum = 0;
    for (int i = 0; i < num_grades; i++) {
        sum += grades[i];
    }
    return (float)sum / num_grades;
}

// Remove a student
void removeStudent(struct Student **root, int id) {
    if (*root == NULL) {
        printf("Student with ID %d not found.\n", id);
        return;
    }
    if (id < (*root)->id) {
        removeStudent(&(*root)->left, id);
    } else if (id > (*root)->id) {
        removeStudent(&(*root)->right, id);
    } else {
        // Node with only one child or no child
        if ((*root)->left == NULL) {
            struct Student *temp = *root;
            *root = (*root)->right;
            free(temp->grades);
            free(temp);
        } else if ((*root)->right == NULL) {
            struct Student *temp = *root;
            *root = (*root)->left;
            free(temp->grades);
            free(temp);
        } else {
            // Node with two children: Get the inorder successor (smallest in the
right subtree)
            struct Student *temp = findMin((*root)->right);
            (*root)->id = temp->id;
            strcpy((*root)->name, temp->name);
            free((*root)->grades);
            (*root)->grades = temp->grades;
            (*root)->num_grades = temp->num_grades;
            removeStudent(&(*root)->right, temp->id);
        }
    }
}
// Find minimum value node in a BST
```

```
    return root;
}

// Free allocated memory
void freeMemory(struct Student *root) {
    if (root != NULL) {
        freeMemory(root->left);
        freeMemory(root->right);
        free(root->grades);
        free(root);
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Hello World