A Line-by-line breakdown for a management system using linked-list that covers the purpose and mechanics of each part of the code, explaining how the student management system operates with linked-list.

Management System For Student Using a Linked-List.

C Project | Edges Standard Embedded Diploma Summer 2024 | Eng. Mohamed Tarek

Fady Romany.

Table of Contents

Code Explanation

Structure: student

```
struct student {
    int id;
    char name[100];
    int age;
    float gpa;
    struct student *next;
};
```

- **`struct student { ... }`**: Defines a new structure named `student`.
  - **`int id;`**: An integer to store the student's unique ID.
  - **`char name[100];`**: An array of characters (string) to hold the student's name, allowing for up to 99 characters plus a null terminator.
  - **`int age;`**: An integer to store the student's age.
  - **`float gpa;`**: A floating-point number to store the student's GPA.
  - **`struct student *next;`**: A pointer to the next student in the linked list, enabling the chaining of student records.

Function: addStudent

```
void addStudent(struct student **head) {
```

- **`void addStudent(struct student **head)`**: Declares a function that takes a double pointer to the head of the linked list, allowing for modifications to the head.

```
    int id;
    char name[100];
    int age;
    float gpa;
```

- **Variable Declarations**: Declares local variables to store student details temporarily.

```
    printf("Enter student ID: ");
    scanf("%d", &id);
```

- **Input Prompt**: Asks for the student ID and reads it into the `id` variable.

```
// Check for duplicate IDs
struct student *current = *head;
while (current != NULL) {
    if (current->id == id) {
        printf("ID already exists. Please enter a unique ID.\n");
        return;
    }
    current = current->next;
}
```

- **Duplicate Check**: Iterates through the linked list to check if the entered ID already exists. If it does, prints a message and exits the function.

```
printf("Enter student name: ");
scanf("%s", name);
```

- **Name Input**: Prompts for the student's name and stores it in the `name` variable.

```
printf("Enter student age: ");
scanf("%d", &age);
```

- **Age Input**: Prompts for the student's age and stores it in the `age` variable.

```
printf("Enter student GPA: ");
scanf("%f", &gpa);
```

- **GPA Input**: Prompts for the student's GPA and stores it in the `gpa` variable.

```
struct student *newStudent = malloc(sizeof(struct student));
```

- **Memory Allocation**: Allocates memory for a new `student` structure using `malloc`.

```
newStudent->id = id;
strcpy(newStudent->name, name);
newStudent->age = age;
newStudent->gpa = gpa;
newStudent->next = *head;
```

- **Data Assignment**: Assigns the collected data to the new student structure and links it to the current head.

```
    *head = newStudent;
```

- **Head Update**: Updates the head pointer to point to the new student, making it the first in the list.

Function: displayStudents

```
void displayStudents(struct student *head) {
```

- **Function Declaration**: Declares a function to display all students, taking a pointer to the head of the linked list.

```
    if (head == NULL) {
        printf("No students available.\n");
        return;
    }
```

- **Empty Check**: If the list is empty (`head` is `NULL`), print a message and exit the function.

```
    struct student *stack[MAX_SIZE];
    int top = -1;
```

- **Stack Initialization**: Creates a stack to hold student pointers for reverse order display, initialized to an empty state.

```
    while (head != NULL) {
        stack[++top] = head;
        head = head->next;
    }
```

- **Stack Fill**: Traverses the linked list, pushing each student onto the stack.

```
    while (top != -1) {
        struct student *s = stack[top--];
        printf("ID: %d, Name: %s, Age: %d, GPA: %.2f\n", s->id, s->name, s->age,
s->gpa);
    }
```

- **Reverse Display**: Pops each student from the stack and prints their details in reverse order.

Function: searchStudentByID

```
void searchStudentByID(struct student *head, int id) {
```

- **Function Declaration**: Declares a function to search for a student by ID, taking a pointer to the head and an integer ID.

```
    struct student *current = head;
```

- **Traversal Pointer**: Initializes a pointer to traverse the linked list.

```
    while (current != NULL) {
        if (current->id == id) {
            printf("ID: %d, Name: %s, Age: %d, GPA: %.2f\n", current->id, current
->name, current->age, current->gpa);
            return;
        }
        current = current->next;
    }
```

- **Search Logic**: Iterates through the list, checking if the current student's ID matches. If found, prints their details and exits. If not found, the loop continues.

```
    printf("Student not found.\n");
```

- **Not Found Message**: If the loop completes without finding the student, prints a message indicating the student does not exist.

Function: updateStudent

```
void updateStudent(struct student *head, int id) {
```

- **Function Declaration**: Declares a function to update a student's details, taking a pointer to the head and an ID.

```
    struct student *current = head;
```

- **Traversal Pointer**: Initializes a pointer to traverse the linked list.

```
    while (current != NULL) {
        if (current->id == id) {
```

- **Search Logic**: Iterates through the list to find the student with the specified ID.

```
            printf("Enter new name: ");
            scanf("%s", current->name);
```

- **Name Update**: If found, prompts for a new name and updates the current student's name.

```
            printf("Enter new age: ");
            scanf("%d", &current->age);
            printf("Enter new GPA: ");
            scanf("%f", &current->gpa);
```

- **Age and GPA Update**: Prompts for new age and GPA, updating the student's fields accordingly.

```
            printf("Student information updated successfully.\n");
            return;
        }
        current = current->next;
    }
```

- **Success Message**: If updated, prints a success message. If not found, continues searching.

```
    printf("Student not found.\n");
```

- **Not Found Message**: If the student is not found after the loop, prints a message indicating so.

Function: deleteStudent

```
void deleteStudent(struct student **head, int id) {
```

- **Function Declaration**: Declares a function to delete a student, taking a double pointer to the head and an ID.

```
    struct student *current = *head;
    struct student *previous = NULL;
```

- **Pointers Initialization**: Initializes pointers for the current and previous nodes.

```
    while (current != NULL) {
        if (current->id == id) {
```

- **Search Logic**: Iterates through the list to find the student with the specified ID.

```
            if (previous == NULL) {
                *head = current->next;
            } else {
                previous->next = current->next;
            }
```

- **Deletion Logic**: If found, adjusts the `next` pointer of the previous node to skip the current node. If the student is the head, updates the head pointer.

```
            free(current);
            printf("Student deleted successfully.\n");
            return;
        }
        previous = current;
        current = current->next;
    }
```

- **Memory Deallocation**: Frees the memory for the deleted student and prints a success message. Updates pointers for traversal.

```
    printf("Student not found.\n");
```

- **Not Found Message**: If the student is not found after the loop, prints a message indicating so.

Function: calculateAverageGPA

```
float calculateAverageGPA(struct student *head) {
```

- **Function Declaration**: Declares a function to calculate the average GPA, taking a pointer to the head.

```
    if (head == NULL) {
        return 0.0f;
    }
```

- **Empty Check**: If the list is empty, returns 0.0 as the average GPA.

```
    float totalGPA = 0.0f;
    int count = 0;
```

- **Initialization**: Initializes variables to hold the total GPA and count of students.

```
    struct student *current = head;
    while (current != NULL) {
        totalGPA += current->gpa;
        count++;
        current = current->next;
    }
```

- **Summation Loop**: Iterates through the list, summing the GPAs and counting students.

```
    return totalGPA / count;
```

- **Average Calculation**: Returns the average GPA by dividing the total GPA by the count of students.

Function: searchHighestGPA

```
void searchHighestGPA(struct student *head) {
```

- **Function Declaration**: Declares a function to find the student with the highest GPA, taking a pointer to the head.

```
    if (head == NULL) {
        printf("No students available.\n");
        return;
    }
```

- **Empty Check**: If the list is empty, prints a message and exits.

```
    struct student *highest = head;
```

- **Initialization**: Initializes a pointer to track the student with the highest GPA.

```
    struct student *current = head->next;
    while (current != NULL) {
        if (current->gpa > highest->gpa) {
            highest = current;
        }
        current = current->next;
    }
```

- **Comparison Loop**: Iterates through the list, updating the pointer to the student with the highest GPA when a higher GPA is found.

```
    printf("Student with highest GPA: ID: %d, Name: %s, Age: %d, GPA: %.2f\n", hi
ghest->id, highest->name, highest->age, highest->gpa);
}
```

- **Display**: Prints the details of the student with the highest GPA.

Conclusion

This line-by-line breakdown covers the purpose and mechanics of each part of the code, explaining how the student management system operates.

Code Creation

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SIZE 1000

// Define the structure for student data
struct student {
    int id;
    char name[100];
    int age;
    float gpa;
    struct student *next;
};

// Function prototypes
void addStudent(struct student **head);
void displayStudents(struct student *head);
void searchStudentByID(struct student *head, int id);
void updateStudent(struct student *head, int id);
```

```c
void deleteStudent(struct student **head, int id);
float calculateAverageGPA(struct student *head);
void searchHighestGPA(struct student *head);

// Main function
int main(void) {
     setvbuf(stdout, NULL, _IONBF, 0);     // Set the buffering mode of stdout to unbuffered
     setvbuf(stderr, NULL, _IONBF, 0);     // Set the buffering mode of stderr to unbuffered

    struct student *head = NULL;
    int choice, id;

    while (1) {
        printf("\n\t\t**Menu Options**\n");
        printf("-------------------------------------------------------------------------\n");
        printf("1. Add a Student\n");
        printf("2. Display All Students\n");
        printf("3. Search for a Student by ID\n");
        printf("4. Update Student Information\n");
        printf("5. Delete a Student\n");
        printf("6. Calculate Average GPA\n");
        printf("7. Find Student with Highest GPA\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        printf("-------------------------------------------------------------------------\n");

        switch (choice) {
            case 1:
                addStudent(&head);
                break;
            case 2:
                displayStudents(head);
                break;
            case 3:
                printf("Enter student ID to search: ");
                scanf("%d", &id);
                searchStudentByID(head, id);
                break;
            case 4:
                printf("Enter student ID to update: ");
                scanf("%d", &id);
                updateStudent(head, id);
                break;
            case 5:
                printf("Enter student ID to delete: ");
```

```
                scanf("%d", &id);
                deleteStudent(&head, id);
                break;
            case 6:
                printf("Average GPA: %.3f\n", calculateAverageGPA(head));
                break;
            case 7:
                searchHighestGPA(head);
                break;
            case 8:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}

// Function to add a student to the linked list
void addStudent(struct student **head) {
    int id;
    printf("Enter student ID: ");
    scanf("%d", &id);

    struct student *current = *head;
    while (current) {
        if (current->id == id) {
            printf("ID is already taken.\n");
            return;
        }
        current = current->next;
    }

    struct student *newStudent = (struct student *)malloc(sizeof(struct
student));
    if (!newStudent) {
        printf("Memory allocation failed.\n");
        return;
    }

    newStudent->id = id;
    getchar(); // to consume the newline character left by scanf
    printf("Enter student name: ");
    fgets(newStudent->name, 100, stdin);
    newStudent->name[strcspn(newStudent->name, "\n")] = '\0'; // Remove trailing
newline character
    printf("Enter student age: ");
    scanf("%d", &newStudent->age);
    printf("Enter student GPA: ");
```

```
    scanf("%f", &newStudent->gpa);

    newStudent->next = *head;
    *head = newStudent;

    printf("Student added successfully.\n");
}

// Function to display all students in the linked list
void displayStudents(struct student *head) {
    if (!head) {
        printf("No students to display.\n");
        return;
    }

    struct student *current = head;
    struct student *stack[MAX_SIZE];  // Assuming a maximum of 1000 students
    int top = -1;

    while (current) {
        stack[++top] = current;
        current = current->next;
    }

    while (top >= 0) {
        current = stack[top--];
        printf("ID: %d, Name: %s, Age: %d, GPA: %.3f\n",
               current->id, current->name, current->age, current->gpa);
    }
}

// Function to search for a student by ID
void searchStudentByID(struct student *head, int id) {
    struct student *current = head;
    while (current) {
        if (current->id == id) {
            printf("ID: %d, Name: %s, Age: %d, GPA: %.2f\n",
                   current->id, current->name, current->age, current->gpa);
            return;
        }
        current = current->next;
    }
    printf("Student not found.\n");
}

// Function to update student information
void updateStudent(struct student *head, int id) {
    struct student *current = head;
    while (current) {
        if (current->id == id) {
```

```
            getchar(); // to consume the newline character left by scanf
            printf("Enter new name: ");
            fgets(current->name, 100, stdin);
            current->name[strcspn(current->name, "\n")] = '\0'; // Remove
trailing newline character
            printf("Enter new age: ");
            scanf("%d", &current->age);
            printf("Enter new GPA: ");
            scanf("%f", &current->gpa);
            printf("Student updated successfully.\n");
            return;
        }
        current = current->next;
    }
    printf("Student not found.\n");
}

// Function to delete a student by ID
void deleteStudent(struct student **head, int id) {
    struct student *current = *head, *prev = NULL;

    while (current && current->id != id) {
        prev = current;
        current = current->next;
    }

    if (!current) {
        printf("Student not found.\n");
        return;
    }

    if (prev) {
        prev->next = current->next;
    } else {
        *head = current->next;
    }

    free(current);
    printf("Student deleted successfully.\n");
}

// Function to calculate the average GPA of all students
float calculateAverageGPA(struct student *head) {
    if (!head) return 0.0;

    int count = 0;
    float totalGPA = 0.0;
    struct student *current = head;

    while (current) {
```

```
        totalGPA += current->gpa;
        count++;
        current = current->next;
    }

    return (count == 0) ? 0.0 : (totalGPA / count);
}

// Function to find the student with the highest GPA
void searchHighestGPA(struct student *head) {
    if (!head) {
        printf("No students to search.\n");
        return;
    }

    struct student *current = head;
    struct student *highest = head;

    while (current) {
        if (current->gpa > highest->gpa) {
            highest = current;
        }
        current = current->next;
    }

    printf("Student with highest GPA:\n");
    printf("ID: %d, Name: %s, Age: %d, GPA: %.2f\n",
           highest->id, highest->name, highest->age, highest->gpa);
}
```