Індивідуальна робота Номер 3
Наденко Олексій МФ-21

Постановка задачі:
Розробіть систему для реєстрації студентів на вибіркові курси, Реалізувавши АТД Відображення (Map) на базі хеш-таблиці у вигляді масиву списків, елементи яких є парами. Про кожного студента зберігається інформація: унікальний ID, список курсів та кількість кредитів за кожний курс.

Ця програма надає повноцінний інструмент для роботи з країнами та командами, дозволяючи завантажувати інформацію, впорядковувати її, виконувати додавання, видалення, редагування та пошук. Основні моменти реалізації:

Структура даних:
Хеш-таблиця:
Хеш-таблиця з відкритою адресацією (ланцюжки) використовується для зберігання студентів.
Індексація здійснюється через хеш-функцію: studentID % size.

Функції програми:
Додавання студентів:
Студенти додаються до хеш-таблиці із перевіркою на дублювання ID.
Під час додавання студента створюється новий вузол StudentInfo.

Додавання курсів студентам:
Курси додаються до списку курсів студента із перевіркою на дублювання.
Реалізована валідація даних курсу (назва не порожня, кількість кредитів більше 0).

Пошук студента:
Пошук здійснюється за хеш-значенням, після чого виконується лінійний обхід відповідного ланцюжка.

Видалення студентів:
Видаляється вузол StudentInfo із хеш-таблиці, а також усі курси студента.
Звільнення пам'яті виконується коректно.

Виведення інформації:
Програма виводить повну інформацію про студента (ID, курси, кредити) або про всіх зареєстрованих студентів.

Очищення пам’яті:
У деструкторі HashMap реалізовано звільнення пам'яті як для списків студентів, так і для їх курсів.

Меню для інтерактивного управління:
Додавання студентів.
Додавання курсів.
Пошук студента.
Видалення студента.
Виведення всіх студентів.
Завершення роботи.

Очищення пам'яті:
Пам’ять звільняється:
Для курсів студента методом deleteCourses у структурі StudentInfo.
Для студентів у деструкторі HashMap.
Після видалення студентів із таблиці.

Обґрунтування вибору структури даних
Операції пошуку, додавання й видалення виконуються за середній час O(1) (для хеш-таблиці), а найгірший час — O(n) (в одному бакеті при конфліктах).
Використання хеш-таблиці дозволяє масштабувати програму, збільшуючи розмір таблиці або оптимізуючи хеш-функцію.

Код програми:
```
#include <iostream>
#include <string>

using namespace std;

// Структура для хранения информации о курсе
struct Course {
    string name;
    int credits;
    Course* next;  // указатель на следующий курс
    Course(const string& courseName, int courseCredits) : name(courseName),
credits(courseCredits), next(nullptr) {}
};

// Структура для хранения информации о студенте
struct StudentInfo {
    int studentID;
    Course* courses;  // указатель на первый элемент списка курсов
    StudentInfo* next;  // указатель на следующего студента в бакете

    StudentInfo(int id) : studentID(id), courses(nullptr), next(nullptr) {}

    // Метод для добавления курса
    void addCourse(const string& courseName, int credits) {
        if (courseName.empty() || credits <= 0) {
            cout << "Invalid course details: name cannot be empty and credits must be
positive.\n";
```

```
            return;
        }

        // Проверка на дублирование курса
        Course* current = courses;
        while (current) {
            if (current->name == courseName) {
                cout << "Course " << courseName << " already exists for this student.\n";
                return;
            }
            current = current->next;
        }

        Course* newCourse = new Course(courseName, credits);
        newCourse->next = courses;
        courses = newCourse;
        cout << "Course " << courseName << " added to student ID " << studentID << ".\n";
    }

    // Метод для вывода информации о студенте
    void display() const {
        if (!courses) {
            cout << "Student ID: " << studentID << " has no registered courses.\n";
            return;
        }

        cout << "Student ID: " << studentID << endl;
        cout << "Courses and credits:\n";
        Course* currentCourse = courses;
        while (currentCourse) {
            cout << "  Course: " << currentCourse->name << ", Credits: " <<
currentCourse->credits << endl;
            currentCourse = currentCourse->next;
        }
    }

    // Метод для удаления всех курсов студента (для очистки памяти)
    void deleteCourses() {
        while (courses) {
            Course* temp = courses;
            courses = courses->next;
            delete temp;
        }
    }
};

// Класс хеш-таблицы
class HashMap {
```

```
private:
    StudentInfo** table;  // массив указателей на StudentInfo
    int size;

    // Хеш-функция для вычисления индекса
    int hash(int studentID) const {
        return studentID % size;
    }

public:
    // Конструктор для инициализации таблицы
    HashMap(int tableSize) : size(tableSize) {
        table = new StudentInfo*[size];
        for (int i = 0; i < size; ++i) {
            table[i] = nullptr;
        }
    }

    // Деструктор для очистки памяти
    ~HashMap() {
        for (int i = 0; i < size; ++i) {
            StudentInfo* current = table[i];
            while (current) {
                StudentInfo* temp = current;
                current->deleteCourses();
                current = current->next;
                delete temp;
            }
        }
        delete[] table;
    }

    // Метод для добавления студента
    void addStudent(int studentID) {
        if (studentID <= 0) {
            cout << "Invalid student ID: must be positive.\n";
            return;
        }

        int index = hash(studentID);
        StudentInfo* current = table[index];

        while (current) {
            if (current->studentID == studentID) {
                cout << "Student with ID " << studentID << " already exists.\n";
                return;
            }
            current = current->next;
```

```
        }

        StudentInfo* newStudent = new StudentInfo(studentID);
        newStudent->next = table[index];
        table[index] = newStudent;
        cout << "Student with ID " << studentID << " added.\n";
    }

    // Метод для добавления курса студенту
    void addCourse(int studentID, const string& courseName, int credits) {
        if (studentID <= 0) {
            cout << "Invalid student ID: must be positive.\n";
            return;
        }

        if (courseName.empty() || credits <= 0) {
            cout << "Invalid course details: name cannot be empty and credits must be
positive.\n";
            return;
        }

        int index = hash(studentID);
        StudentInfo* current = table[index];

        while (current) {
            if (current->studentID == studentID) {
                current->addCourse(courseName, credits);
                return;
            }
            current = current->next;
        }
        cout << "Student with ID " << studentID << " not found.\n";
    }

    // Метод для поиска студента
    void findStudent(int studentID) const {
        if (studentID <= 0) {
            cout << "Invalid student ID: must be positive.\n";
            return;
        }

        int index = hash(studentID);
        StudentInfo* current = table[index];

        while (current) {
            if (current->studentID == studentID) {
                current->display();
                return;
```

```
            }
            current = current->next;
        }
        cout << "Student with ID " << studentID << " not found.\n";
    }

    // Метод для удаления студента
    void removeStudent(int studentID) {
        if (studentID <= 0) {
            cout << "Invalid student ID: must be positive.\n";
            return;
        }

        int index = hash(studentID);
        StudentInfo* current = table[index];
        StudentInfo* prev = nullptr;

        while (current) {
            if (current->studentID == studentID) {
                if (prev) {
                    prev->next = current->next;
                } else {
                    table[index] = current->next;
                }
                current->deleteCourses();
                delete current;
                cout << "Student with ID " << studentID << " removed.\n";
                return;
            }
            prev = current;
            current = current->next;
        }
        cout << "Student with ID " << studentID << " not found.\n";
    }

    // Метод для вывода всех студентов
    void displayAllStudents() const {
        cout << "\n--- All Registered Students ---\n";
        for (int i = 0; i < size; ++i) {
            StudentInfo* current = table[i];
            while (current) {
                current->display();
                cout << "--------------------------\n";
                current = current->next;
            }
        }
    }
};
```

```
// Функция для отображения меню
void displayMenu() {
    cout << "\n--- Student Registration System ---\n";
    cout << "1. Add Student\n";
    cout << "2. Add Course to Student\n";
    cout << "3. Find Student\n";
    cout << "4. Remove Student\n";
    cout << "5. Display All Students\n";
    cout << "6. Exit\n";
    cout << "Choose an option: ";
}

int main() {
    int tableSize = 10;  // размер хеш-таблицы
    HashMap studentMap(tableSize);

    // Додаємо приклади студентів та курсів
    studentMap.addStudent(1001);
    studentMap.addCourse(1001, "Mathematics", 5);
    studentMap.addCourse(1001, "Physics", 4);

    studentMap.addStudent(1002);
    studentMap.addCourse(1002, "Chemistry", 3);
    studentMap.addCourse(1002, "Biology", 4);

    studentMap.addStudent(1003);
    studentMap.addCourse(1003, "Literature", 2);
    studentMap.addCourse(1003, "History", 3);

    // Виведення всіх доданих студентів для демонстрації
    studentMap.displayAllStudents();

    int choice;
    while (true) {
        displayMenu();
        cin >> choice;

        if (choice == 1) {
            int studentID;
            cout << "Enter student ID: ";
            cin >> studentID;
            studentMap.addStudent(studentID);
        } else if (choice == 2) {
            int studentID, credits;
            string courseName;
            cout << "Enter student ID: ";
            cin >> studentID;
```

```
            cout << "Enter course name: ";
            cin >> ws;  // очистка буфера
            getline(cin, courseName);
            cout << "Enter credits: ";
            cin >> credits;
            studentMap.addCourse(studentID, courseName, credits);
        } else if (choice == 3) {
            int studentID;
            cout << "Enter student ID: ";
            cin >> studentID;
            studentMap.findStudent(studentID);
        } else if (choice == 4) {
            int studentID;
            cout << "Enter student ID: ";
            cin >> studentID;
            studentMap.removeStudent(studentID);
        } else if (choice == 5) {
            studentMap.displayAllStudents();
        } else if (choice == 6) {
            cout << "Exiting the program.\n";
            break;
        } else {
            cout << "Invalid choice. Please try again.\n";
        }
    }

    return 0;
}
```

Виконання програми

```
Course Mathematics added to student ID 1001.
Course Physics added to student ID 1001.
Student with ID 1002 added.
Course Chemistry added to student ID 1002.
Course Biology added to student ID 1002.
Student with ID 1003 added.
Course Literature added to student ID 1003.
Course History added to student ID 1003.

--- All Registered Students ---
Student ID: 1001
Courses and credits:
  Course: Physics, Credits: 4
  Course: Mathematics, Credits: 5
--------------------------
Student ID: 1002
Courses and credits:
  Course: Biology, Credits: 4
  Course: Chemistry, Credits: 3
--------------------------
Student ID: 1003
Courses and credits:
  Course: History, Credits: 3
  Course: Literature, Credits: 2
--------------------------

--- Student Registration System ---
1. Add Student
2. Add Course to Student
3. Find Student
4. Remove Student
5. Display All Students
6. Exit
```

```
--- Student Registration System ---
1. Add Student
2. Add Course to Student
3. Find Student
4. Remove Student
5. Display All Students
6. Exit
Choose an option: 1
Enter student ID: 234
Student with ID 234 added.
```

```
--- Student Registration System ---
1. Add Student
2. Add Course to Student
3. Find Student
4. Remove Student
5. Display All Students
6. Exit
Choose an option: 2
Enter student ID: 234
Enter course name: English
Enter credits: 60
Course English added to student ID 234.
```

```
--- Student Registration System ---
1. Add Student
2. Add Course to Student
3. Find Student
4. Remove Student
5. Display All Students
6. Exit
Choose an option: 3
Enter student ID: 234
Student ID: 234
Courses and credits:
  Course: English, Credits: 60
```

```
--- Student Registration System ---
1. Add Student
2. Add Course to Student
3. Find Student
4. Remove Student
5. Display All Students
6. Exit
Choose an option: 4
Enter student ID: 234
Student with ID 234 removed.
```

```
--- Student Registration System ---
1. Add Student
2. Add Course to Student
3. Find Student
4. Remove Student
5. Display All Students
6. Exit
Choose an option: 5

--- All Registered Students ---
Student ID: 1001
Courses and credits:
  Course: Physics, Credits: 4
  Course: Mathematics, Credits: 5
--------------------------
Student ID: 1002
Courses and credits:
  Course: Biology, Credits: 4
  Course: Chemistry, Credits: 3
--------------------------
Student ID: 1003
Courses and credits:
  Course: History, Credits: 3
  Course: Literature, Credits: 2
--------------------------
```

```
--- Student Registration System ---
1. Add Student
2. Add Course to Student
3. Find Student
4. Remove Student
5. Display All Students
6. Exit
Choose an option: 6
Exiting the program.
```