МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Індивідуальна робота № 3

з курсу «Алгоритми і структури даних»
(назва дисципліни)

Виконала: студентка 2 курсу групи мф-21
напряму підготовки (спеціальності)
Комп'ютерні науки
122 Комп'ютерні науки
(шифр і назва напряму підготовки (спеціальності))
Лобанова Д.В.
(прізвище й ініціали студента)

Харків – 2024

Мета програми:
У багатокористувацькій онлайн-грі кожен гравець має свій унікальний ідентифікатор та кількість очок. Реалізуйте АТД Відображення (Map) на базі хеш-таблиці для:
1. Додавання гравця з його початковим рахунком.
2. Оновлення рахунку існуючого гравця.
3. Видалення гравця з таблиці.
Швидкого пошуку поточного рахунку гравця за його ідентифікатором

Вхідні дані:

1. ID гравця (унікальний ідентифікатор гравця).
2. Рахунок гравця (початковий або оновлений рахунок).

Вихідні дані:

1. Повідомлення про успішне додавання, оновлення чи видалення гравця.
2. Значення рахунку гравця за його ID або повідомлення про його відсутність.
3. Список усіх гравців у хеш-таблиці.

Обґрунтування вибору структури даних:

1. Хеш-таблиця дозволяє працювати з даними за допомогою хеш-функції, що дозволяє:
    - Швидко додавати гравців
    - Швидко оновлювати рахунок гравців
    - Швидко видаляти гравців
    - Швидко знаходити рахунок гравця за його ID
2. Структура використовує масив для зберігання даних, що дозволяє ефективно працювати навіть із великою кількістю гравців.
3. У разі колізій (коли декілька гравців мають однаковий хеш) значення зберігаються у вигляді зв'язаного списку.

Опис програми

```
#include <iostream>
using namespace std;

const int TABLE_SIZE = 100; // Розмір хеш-таблиці
```

Структура для зберігання (ID гравця, рахунок)

```
struct Player {
    int playerId;
    int score;
    Player* next; // Вказівник на наступного гравця в ланцюзі при колізії
};
```

Клас PlayerScoreMap

```
class PlayerScoreMap {
```

```
private:
    Player* table[TABLE_SIZE]; // Масив для зберігання елементів хеш-таблиці
```

Хеш-функція для визначення індексу в таблиці

```
    int hashFun(int key) const {
        return key % TABLE_SIZE;
    }

public:
    // Конструктор
    PlayerScoreMap() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            table[i] = nullptr; // Ініціалізація масиву
        }
    }

     // Деструктор
    ~PlayerScoreMap() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            Player* current = table[i];
            while (current != nullptr) {
                Player* temp = current;
                current = current->next;
                delete temp; // Звільняємо пам'ять для кожного вузла
            }
            table[i] = nullptr; // Очищаємо елемент таблиці
        }
        cout << "Memory cleared.\n";
    }
```

Функція для додавання гравця з кількістю очок

```
    void addPlayer(int playerId, int initialScore) {
        int index = hashFun(playerId); // Отримуємо індекс за допомогою хеш-функції

        // Перевіряємо, чи існує вже гравець з таким ID
        Player* current = table[index];
        while (current != nullptr) {
            if (current->playerId == playerId) {
                cout << "Player with ID " << playerId << " already exists.\n";
                return;
            }
            current = current->next;
        }

        // Створюємо нового гравця та додаємо в таблицю
        Player* newPlayer = new Player{playerId, initialScore, table[index]};
```

```
    table[index] = newPlayer;
    cout << "Player with ID " << playerId << " added with score " << initialScore << ".\n";
}
```

Функція для оновлення рахунку існуючого гравця

```
void updateScore(int playerId, int newScore) {
    int index = hashFun(playerId);  // Отримуємо індекс за допомогою хеш-функції

    Player* current = table[index];
    while (current != nullptr) {
        if (current->playerId == playerId) {
            current->score = newScore; // Оновлюємо рахунок
            cout << "Player with ID " << playerId << " score updated to " << newScore << ".\n";
            return;
        }
        current = current->next; // Переходимо до наступного гравця
    }
    cout << "Player with ID " << playerId << " not found.\n";
}
```

Функція для видалення гравця

```
void removePlayer(int playerId) {
    int index = hashFun(playerId); // Отримуємо індекс за допомогою хеш-функції

    Player* current = table[index];
    Player* prev = nullptr;

    // Шукаємо гравця для видалення
    while (current != nullptr) {
        if (current->playerId == playerId) {
            if (prev == nullptr) {
                table[index] = current->next; // Видалення з початку ланцюга
            } else {
                prev->next = current->next; // Видалення зі середини або кінця ланцюга
            }
            delete current;
            cout << "Player with ID " << playerId << " removed.\n";
            return;
        }
        prev = current;
        current = current->next;
    }
    cout << "Player with ID " << playerId << " not found.\n";
}
```

Функція для отримання рахунку гравця за його ID

```
    void getScore(int playerId) const {
        int index = hashFun(playerId); // Отримуємо індекс за допомогою хеш-функції

        Player* current = table[index];
        while (current != nullptr) {
            if (current->playerId == playerId) {
                cout << "Player with ID " << playerId << " score: " << current->score << ".\n";
                return;
            }
            current = current->next; // Переміщаємо до наступного гравця в ланцюгу
        }
        cout << "Player with ID " << playerId << " not found.\n";
    }
```

Функція для виведення всієї таблиці

```
    void printTable() const {
        bool isEmpty = true; // Прапор для перевірки, чи є хоча б один елемент в таблиці
        for (int i = 0; i < TABLE_SIZE; i++) {
            if (table[i] != nullptr) {
                isEmpty = false; // Якщо знайдений хоча б один елемент, таблиця не порожня
                cout << "Index " << i << "| ";
                Player* current = table[i];
                while (current != nullptr) {
                    cout << "[ID: " << current->playerId << ", Score: " << current->score << "] -> ";
                    current = current->next;
                }
                cout << "nullptr\n"; // Позначення що кінць ланцюга
            }
        }

        if (isEmpty) {
            cout << "The table is empty.\n"; // Якщо таблиця порожня
        }
    }
};
```

Функція для перевірки вводу числа

```
int getValidInt(const string& prompt) {
    string input;
    int result;

    while (true) {
        cout << prompt;
        getline(cin, input);
```

```
        try {
            result = stoi(input); // Перетворення рядка на число
            return result; // Повертаємо результат
        } catch (const invalid_argument&) {
            cout << "Invalid input. Please enter a valid number.\n";
        } catch (const out_of_range&) {
            cout << "The number is out of range. Please try again.\n";
        }
    }
}
```

Основна функція програми

```
int main() {
    PlayerScoreMap game;
    int choice;

    do {
        // Меню для користувача
        cout << "\n1. Add player\n";
        cout << "2. Update player score\n";
        cout << "3. Remove player\n";
        cout << "4. Get player score\n";
        cout << "5. Print all players\n";
        cout << "6. Exit\n";

        choice = getValidInt("Enter your choice: ");

        switch (choice) {
            case 1: {
                int playerId = getValidInt("Enter player ID: ");
                int score = getValidInt("Enter score: ");
                game.addPlayer(playerId, score);
                break;
            }
            case 2: {
                int playerId = getValidInt("Enter player ID: ");
                int newScore = getValidInt("Enter new score: ");
                game.updateScore(playerId, newScore);
                break;
            }
            case 3: {
                int playerId = getValidInt("Enter player ID: ");
                game.removePlayer(playerId);
                break;
            }
            case 4: {
                int playerId = getValidInt("Enter player ID: ");
```

```
            game.getScore(playerId);
            break;
        }
        case 5:
            game.printTable();
            break;
        case 6:
            cout << "Exiting program...\n";
            break;
        default:
            cout << "Invalid choice. Please select a valid option (1-6).\n";
    }
} while (choice != 6);

return 0;
}
```

Код програми

```
/* У багатокористувацькій онлайн-грі кожен гравець має
* свій унікальний ідентифікатор та кількість очок.
* Реалізуйте АТД Відображення (Map) на базі хеш-таблиці для:
* 1. Додавання гравця з його початковим рахунком.
* 2. Оновлення рахунку існуючого гравця.
* 3. Видалення гравця з таблиці.
* Швидкого пошуку поточного рахунку гравця за його ідентифікатором
* Daryna Lobanova mf-21
*/

#include <iostream>
using namespace std;

const int TABLE_SIZE = 100; // Розмір хеш-таблиці

// Структура для зберігання (ID гравця, рахунок)
struct Player {
    int playerId;
    int score;
    Player* next; // Вказівник на наступного гравця в ланцюзі при колізії
};

class PlayerScoreMap {
private:
    Player* table[TABLE_SIZE]; // Масив для зберігання елементів хеш-таблиці

    // Хеш-функція для визначення індексу в таблиці
    int hashFun(int key) const {
        return key % TABLE_SIZE;
    }

public:
    // Конструктор
    PlayerScoreMap() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            table[i] = nullptr; // Ініціалізація масиву
        }
    }

    // Деструктор
    ~PlayerScoreMap() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            Player* current = table[i];
```

```
        while (current != nullptr) {
            Player* temp = current;
            current = current->next;
            delete temp; // Звільняємо пам'ять для кожного вузла
        }
        table[i] = nullptr; // Очищаємо елемент таблиці
    }
    cout << "Memory cleared.\n";
}

// Функція для додавання гравця з кількістю очок
void addPlayer(int playerId, int initialScore) {
    int index = hashFun(playerId); // Отримуємо індекс за допомогою хеш-функції

    // Перевіряємо, чи існує вже гравець з таким ID
    Player* current = table[index];
    while (current != nullptr) {
        if (current->playerId == playerId) {
            cout << "Player with ID " << playerId << " already exists.\n";
            return;
        }
        current = current->next;
    }

    // Створюємо нового гравця та додаємо в таблицю
    Player* newPlayer = new Player{playerId, initialScore, table[index]};
    table[index] = newPlayer;
    cout << "Player with ID " << playerId << " added with score " << initialScore << ".\n";
}

// Функція для оновлення рахунку існуючого гравця
void updateScore(int playerId, int newScore) {
    int index = hashFun(playerId);  // Отримуємо індекс за допомогою хеш-функції

    Player* current = table[index];
    while (current != nullptr) {
        if (current->playerId == playerId) {
            current->score = newScore; // Оновлюємо рахунок
            cout << "Player with ID " << playerId << " score updated to " << newScore << ".\n";
            return;
        }
        current = current->next; // Переходимо до наступного гравця
    }
    cout << "Player with ID " << playerId << " not found.\n";
}

// Функція для видалення гравця
void removePlayer(int playerId) {
    int index = hashFun(playerId); // Отримуємо індекс за допомогою хеш-функції

    Player* current = table[index];
    Player* prev = nullptr;

    // Шукаємо гравця для видалення
    while (current != nullptr) {
        if (current->playerId == playerId) {
            if (prev == nullptr) {
                table[index] = current->next; // Видалення з початку ланцюга
            } else {
                prev->next = current->next; // Видалення зі середини або кінця ланцюга
            }
            delete current;
            cout << "Player with ID " << playerId << " removed.\n";
            return;
        }
        prev = current;
        current = current->next;
    }
    cout << "Player with ID " << playerId << " not found.\n";
}

// Функція для отримання рахунку гравця за його ID
void getScore(int playerId) const {
    int index = hashFun(playerId); // Отримуємо індекс за допомогою хеш-функції

    Player* current = table[index];
```

```
        while (current != nullptr) {
            if (current->playerId == playerId) {
                cout << "Player with ID " << playerId << " score: " << current->score << ".\n";
                return;
            }
            current = current->next; // Переміщаємо до наступного гравця в ланцюгу
        }
        cout << "Player with ID " << playerId << " not found.\n";
    }

    // Функція для виведення всієї таблиці
    void printTable() const {
        bool isEmpty = true; // Прапор для перевірки, чи є хоча б один елемент в таблиці
        for (int i = 0; i < TABLE_SIZE; i++) {
            if (table[i] != nullptr) {
                isEmpty = false; // Якщо знайдений хоча б один елемент, таблиця не порожня
                cout << "Index " << i << "| ";
                Player* current = table[i];
                while (current != nullptr) {
                    cout << "[ID: " << current->playerId << ", Score: " << current->score << "] -> ";
                    current = current->next;
                }
                cout << "nullptr\n"; // Позначення що кінць ланцюга
            }
        }

        if (isEmpty) {
            cout << "The table is empty.\n"; // Якщо таблиця порожня
        }
    }
};

// Функція для перевірки вводу числа
int getValidInt(const string& prompt) {
    string input;
    int result;

    while (true) {
        cout << prompt;
        getline(cin, input);

        try {
            result = stoi(input); // Перетворення рядка на число
            return result; // Повертаємо результат
        } catch (const invalid_argument&) {
            cout << "Invalid input. Please enter a valid number.\n";
        } catch (const out_of_range&) {
            cout << "The number is out of range. Please try again.\n";
        }
    }
}

int main() {
    PlayerScoreMap game;
    int choice;

    do {
        // Меню для користувача
        cout << "\n1. Add player\n";
        cout << "2. Update player score\n";
        cout << "3. Remove player\n";
        cout << "4. Get player score\n";
        cout << "5. Print all players\n";
        cout << "6. Exit\n";

        choice = getValidInt("Enter your choice: ");

        switch (choice) {
            case 1: {
                int playerId = getValidInt("Enter player ID: ");
                int score = getValidInt("Enter score: ");
                game.addPlayer(playerId, score);
                break;
            }
            case 2: {
                int playerId = getValidInt("Enter player ID: ");
```

```
            int newScore = getValidInt("Enter new score: ");
            game.updateScore(playerId, newScore);
            break;
        }
        case 3: {
            int playerId = getValidInt("Enter player ID: ");
            game.removePlayer(playerId);
            break;
        }
        case 4: {
            int playerId = getValidInt("Enter player ID: ");
            game.getScore(playerId);
            break;
        }
        case 5:
            game.printTable();
            break;
        case 6:
            cout << "Exiting program...\n";
            break;
        default:
            cout << "Invalid choice. Please select a valid option (1-6).\n";
    }
  } while (choice != 6);

  return 0;
}
```

Результати програми

Тест 1. Додавання гравців

```
1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:1
 Enter player ID:1234
 Enter score:100
 Player with ID 1234 added with score 100.

1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:5
 Index 34| [ID: 1234, Score: 100] -> nullptr
```

Тест 2. Оновлення рахунку гравця

```
1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:2
 Enter player ID:1234
 Enter new score:300
 Player with ID 1234 score updated to 300.

1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:5
 Index 34| [ID: 1234, Score: 300] -> nullptr
```

Тест 3. Видалення гравців

```
1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:3
 Enter player ID:1234
 Player with ID 1234 removed.

1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:5
 The table is empty.
```

Тест 4. Отримання рахунку гравця

```
1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:4
 Enter player ID:11237
 Player with ID 11237 score: 345.
```

Тест 5. Виведення всієї таблиці

```
1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:5
 Index 0| [ID: 200, Score: 466] -> [ID: 100, Score: 367] -> nullptr
Index 23| [ID: 23, Score: 234] -> [ID: 123, Score: 234] -> nullptr
Index 45| [ID: 3345, Score: 3] -> [ID: 345, Score: 657] -> nullptr
Index 65| [ID: 98765, Score: 567] -> nullptr
Index 89| [ID: 23456789, Score: 3456] -> nullptr
```

Тест 6. Некоректний ввід

```
1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:кк
 Invalid input. Please enter a valid number.
Enter your choice:12345678909876543456789045677789000000000000000000000
 The number is out of range. Please try again.
Enter your choice:1
 Enter player ID:цкенрго
 Invalid input. Please enter a valid number.
Enter player ID:1234567888
 Enter score:11111111111111111111111111111111111111111111111111111111111111111111
 The number is out of range. Please try again.
Enter score:3234
 Player with ID 1234567888 added with score 3234.
```

Тест 7. Exit

```
1. Add player
2. Update player score
3. Remove player
4. Get player score
5. Print all players
6. Exit
Enter your choice:6
 Exiting program...
Memory cleared.

Process finished with exit code 0
```