Індивідуальна робота Номер 2
Наденeнко Олексій МФ-21

Постановка задачі
Скласти програму, яка містить поточну інформацію про заявки на авіаквитки. Кожна заявка включає: пункт призначення, номер рейсу, прізвище та ініціали пасажира, бажану дату вильоту.
Програма має забезпечувати:
• зберігання всіх заявок у вигляді двійкового дерева,
• додавання та видалення заявок,
• за заданим номером рейсу та датою вильоту виведення заявок з їх подальшим видаленням,
• виведення всіх заявок**.**

Структура програми
Програма реалізує управління заявками на авіаквитки, включаючи функції додавання, видалення, пошуку та виведення заявок.

Структура даних
Двійкове дерево:
Кожна заявка зберігається у вигляді вузла дерева, який містить такі поля:
Пункт призначення.
Номер рейсу.
Прізвище та ініціали пасажира.
Дата вильоту.
Вказівники на лівого та правого нащадків.

Функції програми:
1.Зберігання всіх заявок у вигляді двійкового дерева.
2.Додавання заявки: заявка автоматично додається в дерево згідно з її номером рейсу, а при однакових номерах – за датою.
3.Видалення заявки: підтримується видалення конкретної заявки за номером рейсу та датою вильоту.
4.Пошук заявок: можливість знайти всі заявки за заданими номером рейсу та датою, а також автоматично видалити їх після виведення.
5.Виведення всіх заявок: структуроване виведення всіх наявних заявок.
6.Інтерактивне меню:Забезпечує користувачеві простий інтерфейс для роботи з функціями програми.
7.Перевірка вхідних даних: Підтримується валідація номера рейсу та формату дати (YYYY-MM-DD), щоб уникнути некоректного вводу.

Обґрунтування вибору двійкового дерева

Динамічна структура:
Двійкове дерево дозволяє легко додавати нові заявки та видаляти старі без необхідності перерозподілу пам’яті, що робить його ідеальним для задач із невідомою кількістю елементів.

Інтуїтивна реалізація пошуку:
Завдяки впорядкованій структурі, дерево забезпечує швидкий пошук за заданими параметрами.

Ефективність управління пам’яттю:
Дерево займає стільки пам’яті, скільки потрібно для зберігання вузлів. У разі видалення пам’ять звільняється автоматично.

Автоматичне впорядкування:Під час додавання нових заявок вони розташовуються в потрібному порядку, що дозволяє уникнути додаткового сортування.

Код програми:

```
#include <iostream>
#include <string>
#include <cctype>

using namespace std;

// Структура, яка представляє квиток.
struct Ticket {
    string dest;   // Пункт призначення
    int flight;    // Номер рейсу
    string name;   // Ім'я пасажира
    string date;   // Дата рейсу (у форматі YYYY-MM-DD)

    Ticket* left;  // Вказівник на лівого нащадка
    Ticket* right; // Вказівник на правого нащадка

    // Конструктор для створення нового квитка
    Ticket(string d, int f, string n, string dt)
        : dest(d), flight(f), name(n), date(dt), left(nullptr), right(nullptr) {}
};

// Клас, який представляє дерево
class Tree {
private:
    Ticket* root; // Корінь дерева

    // Рекурсивна функція для додавання квитка
    Ticket* add(Ticket* node, string d, int f, string n, string dt) {
        if (node == nullptr)
            return new Ticket(d, f, n, dt); // Якщо місце порожнє, створюємо новий квиток

        if (f < node->flight)
            node->left = add(node->left, d, f, n, dt); // Додаємо у ліве піддерево
        else if (f > node->flight)
            node->right = add(node->right, d, f, n, dt); // Додаємо у праве піддерево
```

```
        else {
            if (dt < node->date)
                node->left = add(node->left, d, f, n, dt); // Якщо номер рейсу однаковий,
порівнюємо за датою
            else
                node->right = add(node->right, d, f, n, dt);
        }
        return node;
    }

    // Рекурсивна функція для видалення квитка
    Ticket* del(Ticket* node, int f, string dt) {
        if (node == nullptr) return node; // Якщо вузол порожній, нічого не робимо

        if (f < node->flight)
            node->left = del(node->left, f, dt); // Шукаємо у лівому піддереві
        else if (f > node->flight)
            node->right = del(node->right, f, dt); // Шукаємо у правому піддереві
        else if (dt == node->date) { // Якщо знайшли квиток
            if (node->left == nullptr) { // Вузол має тільки правого нащадка
                Ticket* temp = node->right;
                delete node;
                return temp;
            } else if (node->right == nullptr) { // Вузол має тільки лівого нащадка
                Ticket* temp = node->left;
                delete node;
                return temp;
            }
            Ticket* temp = min(node->right); // Знаходимо мінімальний вузол у правому
піддереві
            node->flight = temp->flight; // Замінюємо дані
            node->date = temp->date;
            node->dest = temp->dest;
            node->name = temp->name;
            node->right = del(node->right, temp->flight, temp->date); // Видаляємо дубль
        }
        return node;
    }

    // Знаходження вузла з мінімальним значенням
    Ticket* min(Ticket* node) {
        Ticket* cur = node;
        while (cur && cur->left != nullptr)
            cur = cur->left; // Йдемо ліворуч, поки є вузли
        return cur;
    }

    // Рекурсивна функція для виведення всіх квитків
```

```
    void show(Ticket* node) {
        if (node != nullptr) {
            show(node->left); // Виводимо ліве піддерево
            cout << "Dest: " << node->dest << ", Flight: " << node->flight
                 << ", Name: " << node->name << ", Date: " << node->date << endl;
            show(node->right); // Виводимо праве піддерево
        }
    }

    // Рекурсивна функція для пошуку квитків
    void find(Ticket* node, int f, string dt) {
        if (node == nullptr) return;

        if (node->flight == f && node->date == dt) { // Якщо знайдено відповідний квиток
            cout << "Dest: " << node->dest << ", Flight: " << node->flight
                 << ", Name: " << node->name << ", Date: " << node->date << endl;
        }
        find(node->left, f, dt); // Продовжуємо пошук у лівому піддереві
        find(node->right, f, dt); // Продовжуємо пошук у правому піддереві
    }

public:
    Tree() : root(nullptr) {
        // Додавання квитків
        add("New York", 1001, "Ivanov I.I.", "2024-12-01");
        add("Los Angeles", 1002, "Petrov P.P.", "2025-05-15");
        add("Chicago", 1003, "Sidorov S.S.", "2026-07-20");
        add("Miami", 1004, "Smirnov A.A.", "2024-03-10");
        add("San Francisco", 1005, "Kuznetsov K.K.", "2025-11-30");
    }

    // Додавання квитка
    void add(string d, int f, string n, string dt) {
        if (d.empty() || n.empty() || dt.empty() || f <= 0) {
            cout << "Помилка: Невірні вхідні дані.\n";
            return;
        }
        root = add(root, d, f, n, dt);
    }

    // Видалення квитка
    void del(int f, string dt) {
        if (f <= 0 || dt.empty()) {
            cout << "Помилка: Невірний номер рейсу або дата.\n";
            return;
        }
        root = del(root, f, dt);
    }
```

```
    // Виведення всіх квитків
    void show() {
        if (root == nullptr) {
            cout << "Квитків немає.\n";
            return;
        }
        show(root);
    }

    // Пошук квитків
    void find(int f, string dt) {
        if (f <= 0 || dt.empty()) {
            cout << "Помилка: Невірний номер рейсу або дата.\n";
            return;
        }
        find(root, f, dt);
    }
};

// Перевірка валідності дати
bool validateDate(const string& date) {
    if (date.length() != 10 || date[4] != '-' || date[7] != '-')
        return false;

    int year = stoi(date.substr(0, 4));
    int month = stoi(date.substr(5, 2));
    int day = stoi(date.substr(8, 2));

    if (year < 2024 || year > 2026) return false;
    if (month < 1 || month > 12) return false;
    if (day < 1 || day > 31) return false;

    return true;
}

// Меню програми
void menu() {
    Tree tree;
    int ch;

    do {
        cout << "\n--- Меню ---\n";
        cout << "1. Додати квиток\n";
        cout << "2. Видалити квиток\n";
        cout << "3. Показати всі квитки\n";
        cout << "4. Знайти квитки за рейсом і датою\n";
        cout << "5. Вихід\n";
```

```
        cout << "Ваш вибір: ";
        cin >> ch;

        switch (ch) {
            case 1: {
                string d, n, dt;
                int f;
                cout << "Введіть пункт призначення (або '0' для скасування): ";
                cin.ignore();
                getline(cin, d);
                if (d.empty() || d == "0") break;

                cout << "Введіть номер рейсу (або '0' для скасування): ";
                cin >> f;
                if (f <= 0) break;

                cout << "Введіть ім'я (або '0' для скасування): ";
                cin.ignore();
                getline(cin, n);
                if (n.empty() || n == "0") break;

                do {
                    cout << "Введіть дату (YYYY-MM-DD, або '0' для скасування): ";
                    cin >> dt;
                    if (dt == "0") break;
                    if (!validateDate(dt))
                        cout << "Невірна дата. Введіть дату між 2024 і 2026 роками.\n";
                } while (!validateDate(dt));

                if (dt == "0") break;

                tree.add(d, f, n, dt);
                cout << "Квиток додано.\n";
                break;
            }
            case 2: {
                int f;
                string dt;
                cout << "Введіть номер рейсу для видалення (або '0' для скасування): ";
                cin >> f;
                if (f <= 0) break;

                do {
                    cout << "Введіть дату для видалення (YYYY-MM-DD, або '0' для
скасування): ";
                    cin >> dt;
                    if (dt == "0") break;
                    if (!validateDate(dt))
```

```
                cout << "Невірна дата. Введіть дату між 2024 і 2026 роками.\n";
            } while (!validateDate(dt));

            if (dt == "0") break;

            tree.del(f, dt);
            cout << "Квиток видалено.\n";
            break;
        }
        case 3:
            cout << "Усі квитки:\n";
            tree.show();
            break;
        case 4: {
            int f;
            string dt;
            cout << "Введіть номер рейсу (або '0' для скасування): ";
            cin >> f;
            if (f <= 0) break;

            do {
                cout << "Введіть дату (YYYY-MM-DD, або '0' для скасування): ";
                cin >> dt;
                if (dt == "0") break;
                if (!validateDate(dt))
                    cout << "Невірна дата. Введіть дату між 2024 і 2026 роками.\n";
            } while (!validateDate(dt));

            if (dt == "0") break;

            cout << "Квитки для рейсу " << f << " на " << dt << ":\n";
            tree.find(f, dt);
            break;
        }
        case 5:
            cout << "Вихід.\n";
            break;
        default:
            cout << "Невірний вибір.\n";
      }
   } while (ch != 5);
}

int main() {
   menu();
   return 0;
}
```

Результат виконання програми

```
--- Menu ---
1. Add Ticket
2. Delete Ticket
3. Show All Tickets
4. Find Tickets by Flight and Date
5. Exit
Choice: 3
All tickets:
Dest: New York, Flight: 1001, Name: Ivanov I.I., Date: 2024-12-01
Dest: Los Angeles, Flight: 1002, Name: Petrov P.P., Date: 2025-05-15
Dest: Chicago, Flight: 1003, Name: Sidorov S.S., Date: 2026-07-20
Dest: Miami, Flight: 1004, Name: Smirnov A.A., Date: 2024-03-10
Dest: San Francisco, Flight: 1005, Name: Kuznetsov K.K., Date: 2025-11-30
```

```
--- Menu ---
1. Add Ticket
2. Delete Ticket
3. Show All Tickets
4. Find Tickets by Flight and Date
5. Exit
Choice: 1
Enter destination (or '0' to cancel): Paris
Enter flight number (or '0' to cancel): 234
Enter name (or '0' to cancel): Alex
Enter date (YYYY-MM-DD, or '0' to cancel): 2024-08-21
Ticket added.
```

```
--- Menu ---
1. Add Ticket
2. Delete Ticket
3. Show All Tickets
4. Find Tickets by Flight and Date
5. Exit
Choice: 4
Enter flight number (or '0' to cancel): 234
Enter date (YYYY-MM-DD, or '0' to cancel): 2024-08-21
Tickets for flight 234 on 2024-08-21:
Dest: Paris, Flight: 234, Name: Alex, Date: 2024-08-21
```

```
--- Menu ---
1. Add Ticket
2. Delete Ticket
3. Show All Tickets
4. Find Tickets by Flight and Date
5. Exit
Choice: 5
Exiting.
```