Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
Образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

Факультет компьютерных технологий

Кафедра «Проектирование, управление и разработка информационных систем»

ЛАБОРАТОРНАЯ РАБОТА № 9

по дисциплине «Алгоритмизация и языки программирования»

Динамические структуры данных

Студент группы 3ИТб-1 А.П. Березовский

Преподаватель Е.Э. Шаповалов

2024

Содержание

Структуры

1.1 Задание

Составить программу, которая содержит текущую информацию о заявках на авиабилеты. Каждая заявка включает:

- пункт назначения; номер рейса;

- фамилию и инициалы пассажира; желаемую дату вылета.

Программа должна обеспечивать:

- хранение всех заявок в виде двоичного дерева;

- добавление и удаление заявок;

- по заданному номеру рейса и дате вылета вывод заявок с их последующим удалением;

- вывод всех заявок.

Математическая модель

1) Программа для пользователя, позволяющая узнать всю информацию о рейсах.
2) Программа должна хранить и изменять списки-заявки.

1.2 Блок-схема

На рисунке 1 изображена блок схема, которая отображает исполняемый файл main.cpp, в котором в свою очередь происходит ввод данных с клавиатуры для передачи их в вызываемую функцию, которая изображается в качестве предопределенного процесса.

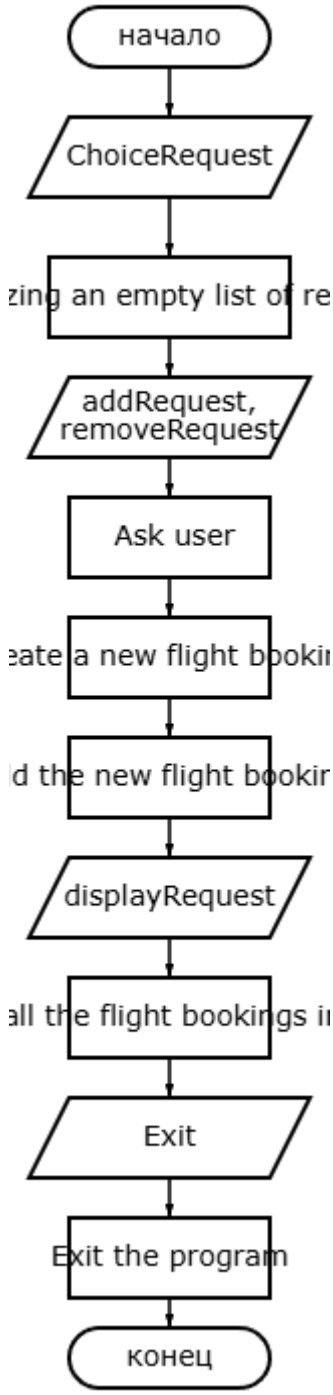

Рисунок 1 – Блок-схема

1.3 Входные и выходные параметры

Входные и выходные данные представлены в таблицах 1 и 2 соответственно.

Таблица 1 – Входные данные

Переменная	Описание
ChoiceRequest	Аргумент, задаваемый пользователем

Таблица 2 – Выходные данные

Переменная	Описание
FlightRequest, choice, removeRequest, displayRequests	Аргументы, полученные после завершения выполнения алгоритма.

1.4 Текст программы

Листинг 1 – Реализация задачи

```
#include <iostream>
#include <vector>
#include <string>

struct FlightRequest {
    std::string destination;
    int flightNumber;
    std::string passengerName;
    std::string departureDate;
};

std::vector<FlightRequest> requests;

void addRequest() {
    FlightRequest newRequest;

    std::cout << "Enter destination: ";
```

```
    std::cin >> newRequest.destination;

    std::cout << "Enter flight number: ";
    std::cin >> newRequest.flightNumber;

    std::cout << "Enter passenger name: ";
    std::cin >> newRequest.passengerName;

    std::cout << "Enter departure date: ";
    std::cin >> newRequest.departureDate;

    requests.push_back(newRequest);

    std::cout << "Request added successfully.\n";
}

void removeRequest() {
    int index;

    std::cout << "Enter the index of the request you
want to remove: ";
    std::cin >> index;

    if (index >= 0 && index < requests.size()) {
        requests.erase(requests.begin() + index);
        std::cout << "Request removed successfully.\n";
    }
    else {
        std::cout << "Invalid index.\n";
    }
}

void displayRequests() {
    if (requests.empty()) {
        std::cout << "No requests\n";
    }
    else {
        std::cout << "Flight Requests:\n";
        for (int i = 0; i < requests.size(); ++i) {
            std::cout << "Request " << i << ":\n";
            std::cout << "Destination: " << re-
quests[i].destination << "\n";
            std::cout << "Flight Number: " << re-
quests[i].flightNumber << "\n";
```

```
            std::cout << "Passenger Name: " << re-
quests[i].passengerName << "\n";
            std::cout << "Departure Date: " << re-
quests[i].departureDate << "\n";
            std::cout << "\n";
        }
    }
}

int main() {
    int choice;

    do {
        std::cout << "1. Add request\n";
        std::cout << "2. Remove request\n";
        std::cout << "3. Display requests\n";
        std::cout << "4. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
        case 1:
            addRequest();
            break;
        case 2:
            removeRequest();
            break;
        case 3:
            displayRequests();
            break;
        case 4:
            std::cout << "Exiting program\n";
            break;
        default:
            std::cout << "Invalid choice\n";
            break;
        }
    } while (choice != 4);

    return 0;
}
```

1.5 Результат

На рисунке 2 изображен результат выполнения программы.

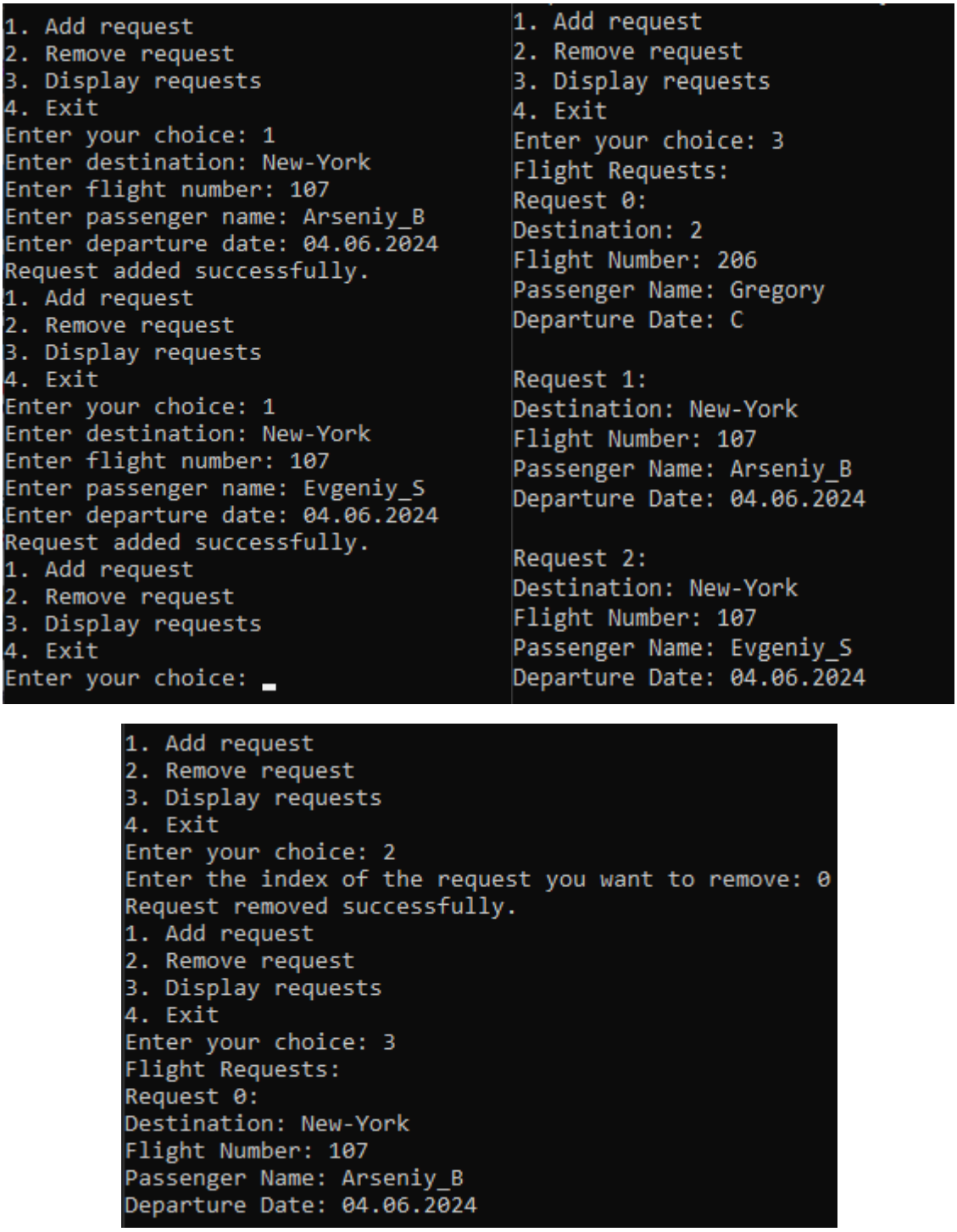

Рисунок 2 – Вывод результата программы

Заключение

Динамические структуры данных на языке C++ представляют собой мощный инструмент для эффективной работы с памятью и хранения данных различных типов. Использование динамических структур данных, таких как динамические массивы, связанные списки, деревья и графы, позволяет создавать гибкие и масштабируемые программы.

Основные преимущества динамических структур данных в C++ включают в себя возможность динамического выделения памяти во время выполнения программы, гибкость в управлении данными и поддержку различных операций вставки, удаления и поиска элементов. Однако необходимо помнить о том, что правильная работа с динамическими структурами данных требует аккуратного управления памятью, чтобы избежать утечек и ошибок.

Использование динамических структур данных на языке C++ позволяет разработчикам создавать эффективные и оптимизированные программы, способные решать разнообразные задачи. Понимание основных принципов работы с динамическими структурами данных позволяет улучшить качество кода и повысить производительность программного обеспечения.

Список использованных источников

1 Павловская Т. А., C/C++. Структурное и объектно-ориентированное программирование: Практикум. / Т. А. Павловская. – СПб.: Питер, 2011. – 352 с.

2 РД ФГБОУ ВО «КнАГУ» 013-2016. Текстовые студенческие работы. Правила оформления : дата введения 2016-03-10. – Комсомольск-на-Амуре : ФГБОУ ВО «КнАГТУ», 2016. – 55 с.