|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА** - **Российский технологический университет»**  **РТУ МИРЭА** |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1**  Стандартные типы данных для представления многоэлементных не однородных структур данных | |
| **по дисциплине** | |
| «Структуры и алгоритмы обработки данных» | |
| Выполнил студент группы ИКБО-02-22 | Вагулич А.Р. |
|  |  |
| Принял | Сорокин А.В. |
|  |  |

Практическая работа выполнена «\_\_» 2023 г.

(подпись студента)

«Зачтено» «\_\_» 2023 г.

(подпись преподавателя)

Москва 2023

**1 УСЛОВИЕ ЗАДАЧИ**

Вариант №1: Студент: номер зачетной книжки (число), фамилия и инициалы, номер группы (текст длиной 10), оценки (успеваемость) (массив из пяти элементов).

**2 ПОСТАНОВКА ЗАДАЧИ**

Разработать набор операций для управления таблицей, созданной на основе статического/динамического массива и контейнера <vector>.

**3 МОДЕЛЬ РЕШЕНИЯ**

**Условие задания №1:**

Задание №1: Разработать набор операций для управления таблицей, созданной на основе статического массива.

**Структура записи (рис. 1):**

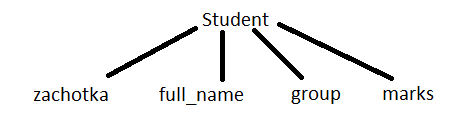


Рисунок 1 – Структура записи

**Структура таблицы (рис. 2):**

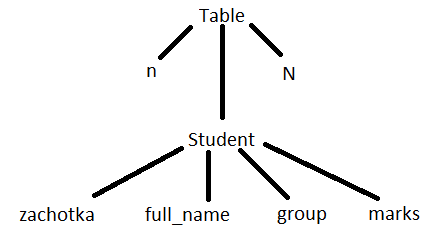
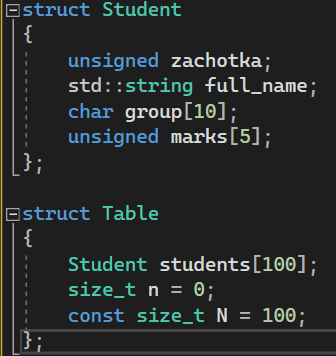


Рисунок 2 – Структура таблицы

**Структура записи и структура таблицы, определённые средствами языка программирования C++:**



**Список подзадач задачи:**

1) Заполнить таблицу с клавиатуры (или из текстового файла)

2) Добавить запись в таблицу

3) Сформировать список студентов (номера зачетных книжек), средний балл которых больше 4.0, вставляя значения в порядке убывания баллов (сортировку не применять).

4) Удалить из таблицы записи о студентах, средний балл которых меньше заданного значения.

**Прототипы функций выделенных подзадач:**

1)

Flight getFlightFromInput() – функция определяет объект структуры flight по типу Flight, далее происходит ввод значений для элементов объекта структуры, после функция возвращает объект структуры flight.

void addFlight(Table& table, const Flight& flight) – функция принимает указатель на таблицу и указатель на элемент записи, далее производится проверка переполнения таблицы, в случае успешного прохождения проверки, в статический массив data вноситься элемент объекта, а текущий размер массива увеличивается на один, если же проверка не пройдена, выводится сообщение об ошибке.

2)

Table deleteFlight(Table& table, int flightNumber) – функция принимает указатель на таблицу и целочисленную переменную номера рейса, после чего определяется объект структуры archive по типу Table, его текущий размер приравнивается к нулю, а так же инициализируется переменная типа bool и её значение приравнивается к false. Далее создаётся цикл for, который будет использоваться в качестве итератор, со вложенным условием проверки на равенство целочисленного значения номера рейса из итерируемого массива data и значения номера рейса, введённого пользователем. Если итерируемый элемент прошёл проверку условием, то элемент записи удаляется из массива data, относящегося к объекту table, и добавляется в массив data, относящегося к объекту archive. Текущий размер таблицы archive увеличивается на один, а текущий размер таблицы table уменьшается на один, а ранее инициализированная переменная типа bool принимает значение true. В случае если переменная типа bool так и осталась false после прохождения цикла (если ни один из элементов не прошёл вложенное условие), пользователю выводится сообщение о том, что не было найдено ни одного рейса с номером, введённым пользователем.

3)

Table findFlights(const Table& table, int weekday, int flightNumber, const std::string& destination) – функция принимает указатель на таблицу, целочисленную переменную номера дня недели, целочисленную переменную номера рейса, строку пункта назначения, после чего определяется объект структуры resultTable по типу Table, его текущий размер приравнивается к нулю. Далее создаётся цикл for, который будет использоваться в качестве итератор, со вложенным условием проверки на равенство целочисленного значения номера дня недели из итерируемого массива data и значения номера дня недели, введённого пользователем, а также проверки на нечётность значения номера рейса из итерируемого массива data и проверки на равенство строки пункта назначения из итерируемого массива data и строки пункта назначения, введённого пользователем. Далее производится проверка на переполнение таблицы resultTable, после чего производится добавление записи в таблицу resultTable, с последующем увеличением текущего размера таблицы на один. Если же проверка не была пройдена, пользователю выводится сообщение об ошибке. Функция возвращает таблицу resultTable.

4)

int countArrivalFlights(const Table& table, int weekday, int flightNumber) – функция принимает указатель на таблицу, целочисленную переменную номера дня недели, целочисленную переменную номера рейса. После создаётся целочисленная переменная счётчика и приравнивается к нулю. Далее создаётся цикл for, который будет использоваться в качестве итератор, со вложенным условием проверки на равенство целочисленного значения номера дня недели из итерируемого массива data и значения номера дня недели, введённого пользователем, а также проверки на чётность значения номера рейса из итерируемого массива. В случае успешного прохождения проверки, к целочисленному значению переменной счётчика прибавляется единица. Функция возвращает значение целочисленной переменной счётчика.

**Код задачи:**

#include <iostream>

#include <fstream>

#include <string>

struct Student

{

unsigned zachotka;

std::string full\_name;

char group[10];

unsigned marks[5];

};

float averageMarkOfStudent(Student& student)

{

float res = 0;

for (unsigned i = 0; i < 5; i++)

{

res += student.marks[i];

}

res /= 5.f;

return res;

}

bool isGoodStudent(Student& student)

{

if (averageMarkOfStudent(student) > 4.f)

{

return true;

}

else

{

return false;

}

}

void inputStudent(Student& student)

{

std::cout << "Enter student's number: ";

std::cin >> student.zachotka;

std::cout << "Enter student's name: ";

std::cin >> student.full\_name;

std::cout << "Enter student's group (10 symbols): ";

for (unsigned i = 0; i < 10; i++)

{

std::cin >> student.group[i];

}

std::cout << "Enter student's marks: ";

for (unsigned i = 0; i < 5; i++)

{

std::cin >> student.marks[i];

}

}

void inputStudentTXT(Student& student, std::ifstream& in)

{

in >> student.zachotka;

in >> student.full\_name;

for (unsigned i = 0; i < 10; i++)

{

in >> student.group[i];

}

for (unsigned i = 0; i < 5; i++)

{

in >> student.marks[i];

}

}

void outputStudent(Student& student)

{

std::cout << student.zachotka << ' ';

std::cout << student.full\_name << ' ';

for (unsigned i = 0; i < 10; i++)

{

std::cout << student.group[i];

}

std::cout << ' ';

for (unsigned i = 0; i < 5; i++)

{

std::cout << student.marks[i] << ' ';

}

std::cout << averageMarkOfStudent(student) << std::endl;

}

struct Table

{

Student students[100];

size\_t n = 0;

const size\_t N = 100;

};

void addStudentsToTableTXT(Table& table, std::ifstream& in)

{

int ch = 0;

while (!in.eof() && table.n != table.N)

{

Student tmp\_student;

inputStudentTXT(tmp\_student, in);

table.students[table.n++] = tmp\_student;

ch++;

}

std::cout << "Introduced: " << ch << " note(-s)." << std::endl;

}

void addOneStudentToTable(Table& table)

{

if (table.n < table.N)

{

Student student;

inputStudent(student);

table.students[table.n++] = Student{ student };

}

else

{

std::cout << "Out of range.";

}

}

void addStudentsToTable(Table& table)

{

int avaibleToAdd = table.N - table.n;

std::cout << "You can add: " << avaibleToAdd << ".";

int value\_of\_students;

std::cout << "Enter the value of students: ";

std::cin >> value\_of\_students;

for (unsigned i = 0; i < value\_of\_students; i++)

{

addOneStudentToTable(table);

}

}

void removeStudentFromTable(Table& table, float average)

{

for (unsigned i = 0; i < table.n; i++)

{

if (averageMarkOfStudent(table.students[i]) < average)

{

for (unsigned j = i; j < table.n; j++)

{

std::swap(table.students[j], table.students[j + 1]);

}

table.n--;

i--;

}

}

}

void insert(Table& table, Student& student, unsigned index)

{

if (table.N <= table.n)

{

return;

}

for (unsigned i = table.n; i > index; i--)

{

std::swap(table.students[i], table.students[i - 1]);

}

table.students[index] = student;

table.n++;

}

Table makeGoodTable(Table& table)

{

Table goodTable;

for (unsigned i = 0; i < table.n; i++)

{

if (averageMarkOfStudent(table.students[i]) > 4.f)

{

bool flag = false;

for (unsigned index = 0; index < goodTable.n; index++)

{

if (averageMarkOfStudent(goodTable.students[index]) < averageMarkOfStudent(table.students[i]))

{

flag = true;

insert(goodTable, table.students[i], index);

break;

}

}

if (!flag)

{

insert(goodTable, table.students[i], goodTable.n);

}

}

}

return goodTable;

}

void outputTable(Table table)

{

for (unsigned i = 0; i < table.n; i++)

{

outputStudent(table.students[i]);

}

}

void outputStudentZach(Student& student)

{

std::cout << student.zachotka << ' ';

}

void outputTableZach(Table table)

{

for (unsigned i = 0; i < table.n; i++)

{

outputStudentZach(table.students[i]);

std::cout << '\n';

}

}

void Act(Table& studentTable)

{

while (true)

{

std::cout << "Enter the code of action that you want" << std::endl;

std::cout << "1 - Fill table by keyboard" << std::endl;

std::cout << "2 - Fill table by .txt file" << std::endl;

std::cout << "3 - Add note to table" << std::endl;

std::cout << "4 - Make list of students (their numbers) which average mark more than 4.0" << std::endl;

std::cout << "5 - Remove notes about students which average mark lower than given" << std::endl;

std::cout << "6 - Write a table" << std::endl;

std::cout << "0 - Finish" << std::endl;

std::ifstream in("input.txt");

unsigned ans;

std::cin >> ans;

if (ans == 1)

{

addStudentsToTable(studentTable);

}

if (ans == 2)

{

addStudentsToTableTXT(studentTable, in);

}

if (ans == 3)

{

addOneStudentToTable(studentTable);

}

if (ans == 4)

{

outputTableZach(makeGoodTable(studentTable));

}

if (ans == 5)

{

float avg;

std::cout << "Enter average mark: ";

std::cin >> avg;

removeStudentFromTable(studentTable, avg);

}

if (ans == 6)

{

outputTable(studentTable);

}

if (ans == 0)

{

return;

}

}

}

int main()

{

Table studentTable;

Act(studentTable);

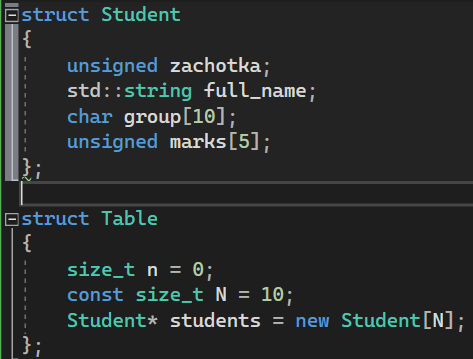
return 0;

}

**Условие задания №2:**

Задание №2: Разработать набор операций для управления таблицей, созданной на основе динамического массива.

**Структура записи и структура таблицы, определённые средствами языка программирования C++:**



**Код задачи:**

#include <iostream>

#include <iomanip>

#include <string>

#include <windows.h>

struct Flight

{

std::string destination;

int flightNumber;

int weekday;

std::string time;

double delay;

};

struct Table

{

int n;

int N = 100;

Flight\* data = new Flight[N];

};

void printTable(const Table& table)

{

std::cout << "-------------------------------------------------------------------" << std::endl;

std::cout << "| Пункт назначения | Номер рейса | День недели | Время | Задержка |" << std::endl;

std::cout << "-------------------------------------------------------------------" << std::endl;

for (int i = 0; i < table.n; i++) {

std::cout << "| " << std::setw(17) << std::left << table.data[i].destination

<< "| " << std::setw(12) << table.data[i].flightNumber

<< "| " << std::setw(11) << table.data[i].weekday

<< "| " << std::setw(6) << table.data[i].time

<< "| " << std::setw(9) << table.data[i].delay << "|" << std::endl;

}

std::cout << "-------------------------------------------------------------------" << std::endl;

}

Flight getFlightFromInput()

{

Flight flight;

std::cout << "Введите пункт назначения: ";

std::cin >> flight.destination;

std::cout << "Введите номер рейса: ";

std::cin >> flight.flightNumber;

std::cout << "Введите день недели: ";

std::cin >> flight.weekday;

std::cout << "Введите время вылета/прилета: ";

std::cin >> flight.time;

std::cout << "Введите информацию о задержке вылета/прилета в часах: ";

std::cin >> flight.delay;

return flight;

}

void addFlight(Table& table, const Flight& flight)

{

if (table.n < table.N)

{

table.data[table.n] = flight;

table.n++;

}

else

{

Flight\* oldData = table.data;

table.data = new Flight[table.N + 1];

for (unsigned i = 0; i < table.N; i++)

{

table.data[i] = oldData[i];

}

table.data[table.n] = flight;

table.n++;

}

}

Table deleteFlight(Table& table, int flightNumber)

{

Table archive;

archive.n = 0;

bool found = false;

for (int i = 0; i < table.n; i++)

{

if (table.data[i].flightNumber == flightNumber)

{

archive.data[archive.n] = table.data[i];

archive.n++;

found = true;

for (int j = i; j < table.n - 1; j++)

{

table.data[j] = table.data[j + 1];

}

table.n--;

break;

}

}

if (!found)

{

std::cout << "Рейс с номером " << flightNumber << " не найден!" << std::endl;

}

return archive;

}

Table findFlights(Table& table, int weekday, int flightNumber, std::string& destination)

{

Table resultTable;

resultTable.n = 0;

for (int i = 0; i < table.n; i++)

{

if (table.data[i].weekday == weekday && table.data[i].flightNumber % 2 != 0 && table.data[i].destination == destination)

{

if (resultTable.n < resultTable.N)

{

resultTable.data[resultTable.n] = table.data[i];

resultTable.n++;

}

else

{

Flight\* oldTable = resultTable.data;

resultTable.data = new Flight[table.N + 1];

for (i = 0; i < table.N; i++)

{

resultTable.data[i] = oldTable[i];

}

resultTable.data[resultTable.n] = table.data[i];

resultTable.n++;

}

}

}

return resultTable;

}

int countArrivalFlights(const Table& table, int weekday, int flightNumber)

{

int count = 0;

for (int i = 0; i < table.n; i++) {

if (table.data[i].weekday == weekday && table.data[i].flightNumber % 2 == 0)

{

count++;

}

}

return count;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "RUS");

Table table = { 0 };

Flight flight;

std::string dest;

int day;

int choice;

do {

std::cout << "Выберите операцию:" << std::endl;

std::cout << "1. Добавить расписание по новому рейсу." << std::endl;

std::cout << "2. Удалить информацию об отмененном рейсе и сохранить ее в архивной таблице." << std::endl;

std::cout << "3. Сформировать список рейсов, вылетающих из аэропорта в заданный день недели и в заданный пункт назначения." << std::endl;

std::cout << "4. Подсчитать количество рейсов, прилетающих в аэропорт в заданный день недели." << std::endl;

std::cout << "5. Вывести таблицу рейсов." << std::endl;

std::cout << "6. Выход." << std::endl;

std::cin >> choice;

switch (choice) {

case 1:

addFlight(table, getFlightFromInput());

break;

case 2:

std::cout << "Введите номер рейса, который хотите удалить:" << std::endl;

int num;

std::cin >> num;

std::cout << "Архивная таблица удалённых рейсов:" << std::endl;

printTable(deleteFlight(table, num));

std::cout << "Обновлённая таблица:" << std::endl;

printTable(table);

break;

case 3:

std::cout << "Введите день:" << std::endl;

std::cin >> day;

std::cout << "Введите пункт назначения:" << std::endl;

std::cin >> dest;

std::cout << "Таблица по вашему запросу:" << std::endl;

printTable(findFlights(table, day, flight.flightNumber, dest));

break;

case 4:

std::cout << "Введите день:" << std::endl;

std::cin >> day;

std::cout << "Количество рейсов: " << countArrivalFlights(table, day, flight.flightNumber) << std::endl;

break;

case 5:

printTable(table);

break;

case 6:

std::cout << "Программа завершена." << std::endl;

break;

default:

std::cout << "Неверный выбор операции. Попробуйте еще раз." << std::endl;

break;

}

} while (choice != 6);

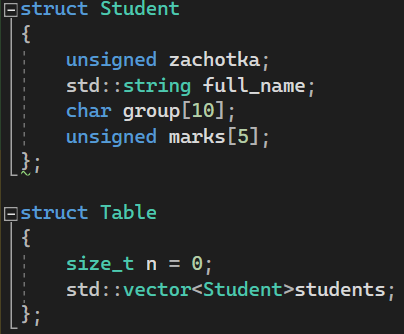
return 0;

}

**Условие задания №3:**

Задание №3: Разработать набор операций для управления таблицей, созданной на основе контейнера <vector>.

**Структура записи и структура таблицы, определённые средствами языка программирования C++:**

****

**Код задачи:**

#include <iostream>

#include <iomanip>

#include <string>

#include <windows.h>

#include <vector>

struct Flight

{

std::string destination;

int flightNumber;

int weekday;

std::string time;

double delay;

};

struct Table

{

int n;

std::vector <Flight> data;

};

void printTable(const Table& table)

{

std::cout << "-------------------------------------------------------------------" << std::endl;

std::cout << "| Пункт назначения | Номер рейса | День недели | Время | Задержка |" << std::endl;

std::cout << "-------------------------------------------------------------------" << std::endl;

for (int i = 0; i < table.n; i++)

{

std::cout << "| " << std::setw(17) << std::left << table.data[i].destination

<< "| " << std::setw(12) << table.data[i].flightNumber

<< "| " << std::setw(11) << table.data[i].weekday

<< "| " << std::setw(6) << table.data[i].time

<< "| " << std::setw(9) << table.data[i].delay << "|" << std::endl;

}

std::cout << "-------------------------------------------------------------------" << std::endl;

}

Flight getFlightFromInput()

{

Flight flight;

std::cout << "Введите пункт назначения: ";

std::cin >> flight.destination;

std::cout << "Введите номер рейса: ";

std::cin >> flight.flightNumber;

std::cout << "Введите день недели: ";

std::cin >> flight.weekday;

std::cout << "Введите время вылета/прилета: ";

std::cin >> flight.time;

std::cout << "Введите информацию о задержке вылета/прилета в часах: ";

std::cin >> flight.delay;

return flight;

}

void addFlight(Table& table, const Flight& flight)

{

table.data.push\_back(flight);

table.n++;

}

Table deleteFlight(Table& table, int flightNumber)

{

Table archive;

archive.n = 0;

bool found = false;

for (int i = 0; i < table.n; i++)

{

if (table.data[i].flightNumber == flightNumber)

{

archive.data.push\_back(table.data[i]);

archive.n++;

found = true;

table.data.erase(table.data.begin() + i);

table.n--;

break;

}

}

if (!found) {

std::cout << "Рейс с номером " << flightNumber << " не найден!" << std::endl;

}

return archive;

}

Table findFlights(const Table& table, int weekday, int flightNumber, const std::string& destination)

{

Table resultTable;

resultTable.n = 0;

for (int i = 0; i < table.n; i++)

{

if (table.data[i].weekday == weekday && table.data[i].flightNumber % 2 != 0 && table.data[i].destination == destination)

{

resultTable.data.push\_back(table.data[i]);

resultTable.n++;

}

}

return resultTable;

}

int countArrivalFlights(const Table& table, int weekday, int flightNumber)

{

int count = 0;

for (int i = 0; i < table.n; i++) {

if (table.data[i].weekday == weekday && table.data[i].flightNumber % 2 == 0)

{

count++;

}

}

return count;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "RUS");

Table table = { 0 };

Flight flight;

std::string dest;

int day;

int choice;

do {

std::cout << "Выберите операцию:" << std::endl;

std::cout << "1. Добавить расписание по новому рейсу." << std::endl;

std::cout << "2. Удалить информацию об отмененном рейсе и сохранить ее в архивной таблице." << std::endl;

std::cout << "3. Сформировать список рейсов, вылетающих из аэропорта в заданный день недели и в заданный пункт назначения." << std::endl;

std::cout << "4. Подсчитать количество рейсов, прилетающих в аэропорт в заданный день недели." << std::endl;

std::cout << "5. Вывести таблицу рейсов." << std::endl;

std::cout << "6. Выход." << std::endl;

std::cin >> choice;

switch (choice) {

case 1:

addFlight(table, getFlightFromInput());

break;

case 2:

std::cout << "Введите номер рейса, который хотите удалить:" << std::endl;

int num;

std::cin >> num;

std::cout << "Архивная таблица удалённых рейсов:" << std::endl;

printTable(deleteFlight(table, num));

std::cout << "Обновлённая таблица:" << std::endl;

printTable(table);

break;

case 3:

std::cout << "Введите день:" << std::endl;

std::cin >> day;

std::cout << "Введите пункт назначения:" << std::endl;

std::cin >> dest;

std::cout << "Таблица по вашему запросу:" << std::endl;

printTable(findFlights(table, day, flight.flightNumber, dest));

break;

case 4:

std::cout << "Введите день:" << std::endl;

std::cin >> day;

std::cout << "Количество рейсов: " << countArrivalFlights(table, day, flight.flightNumber) << std::endl;

break;

case 5:

printTable(table);

break;

case 6:

std::cout << "Программа завершена." << std::endl;

break;

default:

std::cout << "Неверный выбор операции. Попробуйте еще раз." << std::endl;

break;

}

} while (choice != 6);

return 0;

}

**4 БЛОК-СХЕМА ОДНОЙ ИЗ ОПЕРАЦИЙ ВАРИАНТА**

Блок схема операции: Добавить запись в таблицу (рис. 3).

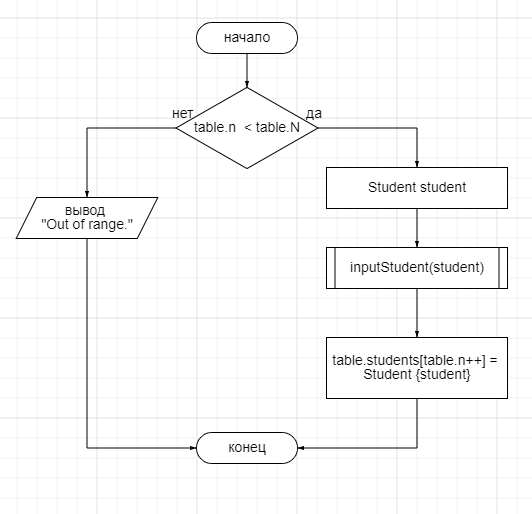


Рисунок 3 – Блок-схема функции deleteFlight

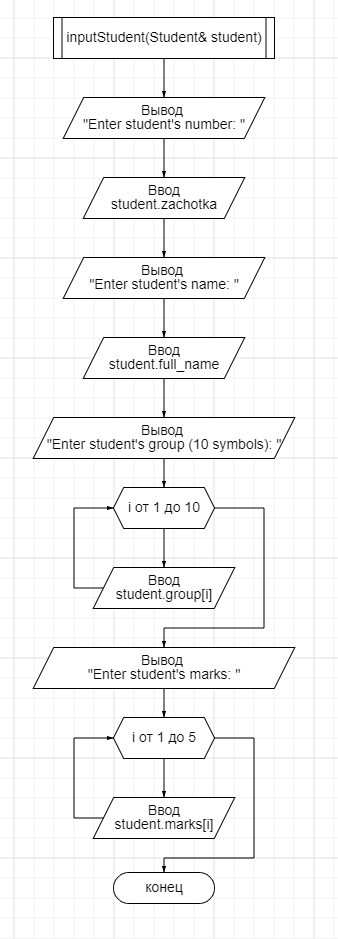
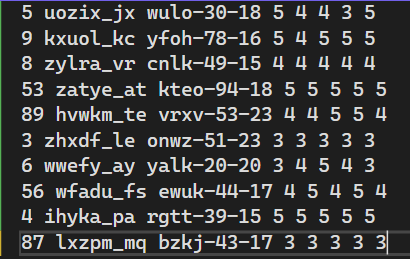
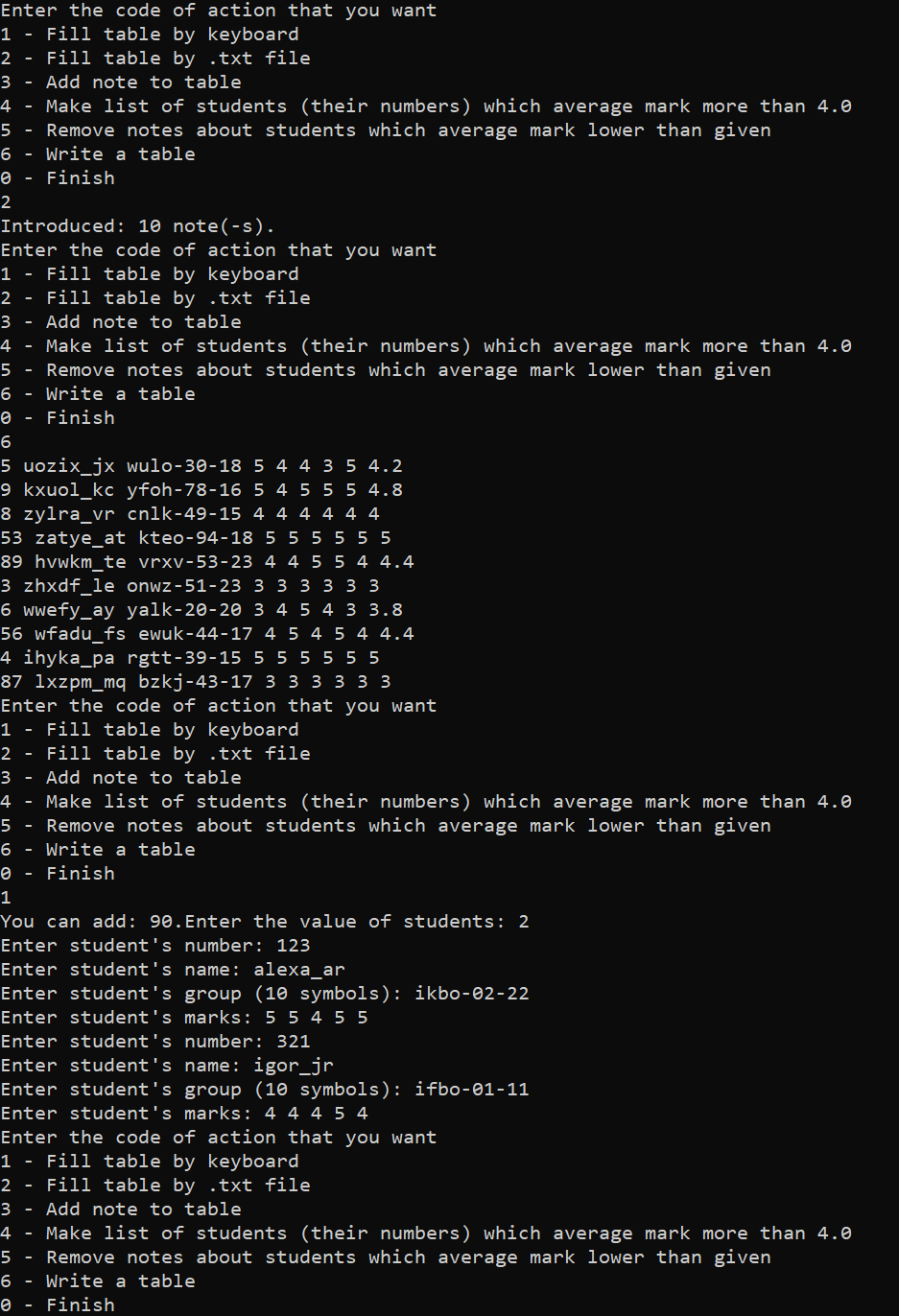
****

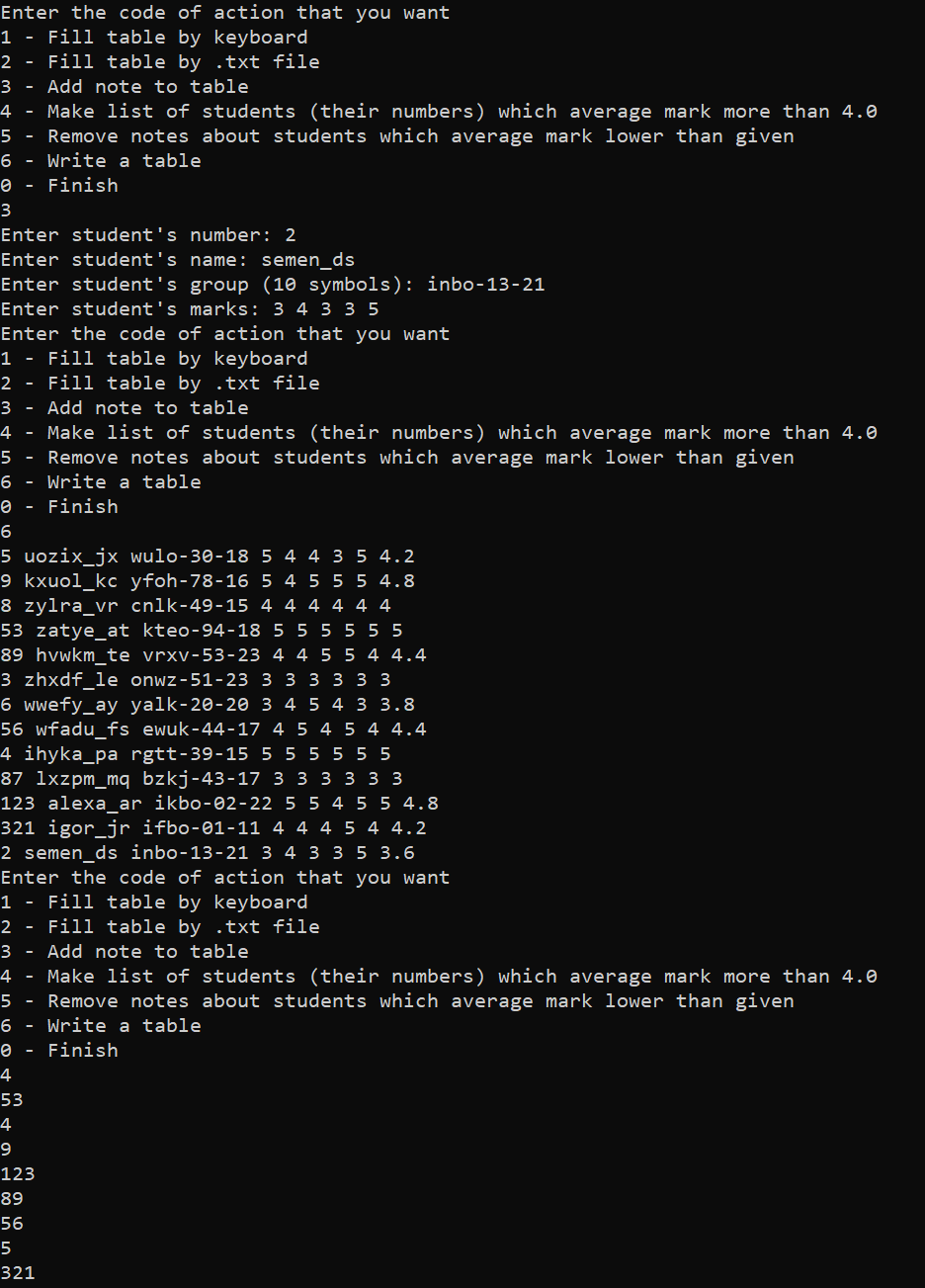
Рисунок 4 – Блок-схема подфункции inputStudent(Student& student)

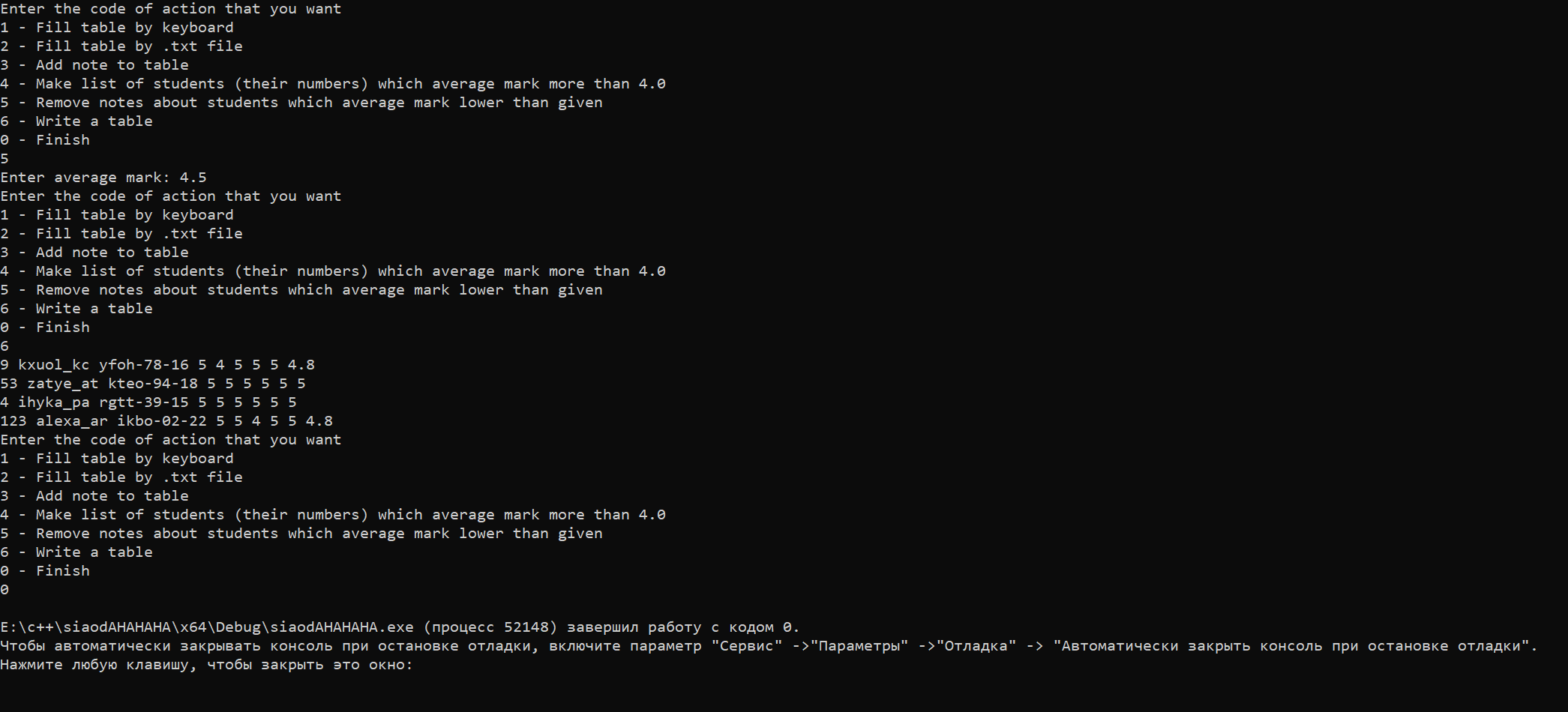
**5 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

****

input.txt – исходный текстовый файл для ввода в таблицу



****

****

**6 ВЫВОД**

Мной был разработан набор операций для управления таблицей, созданной на основе статического/динамического массива и контейнера <vector>.

В ходе работы над таблицей, созданной на основе статического/динамического массива, мной были разработаны функции, реализующие удаление элементов таблиц, а также операция, позволяющая отсортировать и вывести список рейсов по заданным критериям, а также подсчитать количество рейсов по заданным критериям. В ходе выполнения работы я прибегал к основным функциям и операциям массивов C++.

В ходе работы над таблицей, созданной на основе контейнера <vector, мной были разработаны функции, реализующие удаление элементов таблиц, а также операция, позволяющая отсортировать и вывести список рейсов по заданным критериям, а также подсчитать количество рейсов по заданным критериям. В ходе выполнения работы я прибегал к использованию функций, непосредственно относящихся к библиотеке <vector>, в связи с чем я существенно облегчил себе задачу по реализации вышеперечисленных функций.