**Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет “Высшая школа экономики”»**

Московский институт электроники и математики им. А.Н.Тихонова НИУ ВШЭ

Департамент компьютерной инженерии

**Курс: Алгоритмизация и программирование**

|  |  |  |
| --- | --- | --- |
| Раздел | Max оценка | Итоговая оценка |
| Работа программы | 1 |  |
| Тесты | 1 |  |
| Правильность алгоритма | 3 |  |
| Ответы на вопросы | 2 |  |
| Дополнительное задание | 3 |  |
| Итого | 10 |  |

**ОТЧЕТ**

**По лабораторной работе № 10**

**Студент: Иванилов Леонид Михайлович**

**Группа: БИВ212**

**Вариант: №143 2-4-4**

**Руководитель: Ерохина Елена Альфредовна**

**Оценка:**

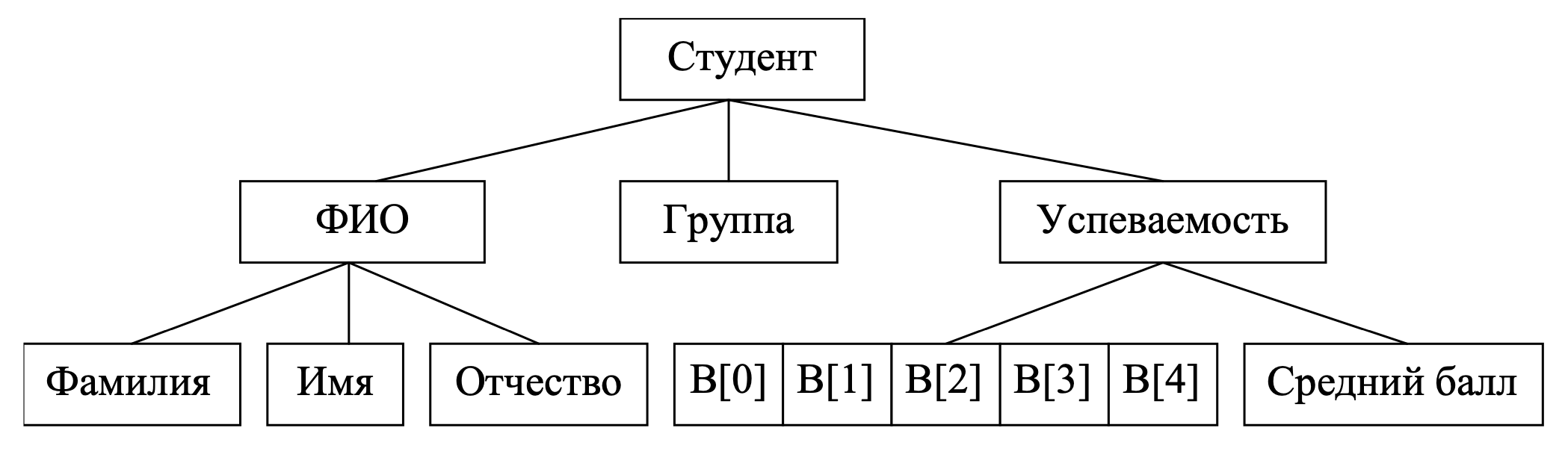
**Дата сдачи: 22.05.2022**

**МОСКВА 2024**

Ввод данных, обработка списка и вывод результатов оформляются как отдельные функции. Анализ существования результата производится в главной функции. Результат (содержимое полученного списка) выводится в новый текстовый файл.

При выделении и освобождении памяти использовать *new* и *delete*. Использовать потоковый ввод и вывод.

**I.**Написать программу, которая вводит из файла структуры вида:



Имя файла задает пользователь. Массив баллов B[0:4] содержит данные о результатах сдачи экзаменов, по 10-балльной шкале. Каждое поле структуры занимает в файле одну строку, а массив оценок размещается на отдельной строке. Средний балл не записан в файле, а вычисляется в процессе чтения данных. Структуры размещаются в:

2) **очереди**

**II.**Реализованной с помощью:

4)  **двунаправленного циклического списка.**

**III.** Для полученного списка программа выполняет следующие действия:

4)  удаляет записи о студентах, имеющих средний балл, равный

минимальному из имеющихся средних баллов;

**Листинг**

#include <iostream>  
#include <fstream>  
#include <string>  
  
using namespace std;  
  
struct student {  
 struct student\* next;  
 struct student\* prev;  
 struct { string second, first, patronymic; } name;  
 string group;  
 struct { int B[4]; float average; } grades;  
};  
  
struct student\* createCircularQueueLinkedList() {  
 struct student\* head = nullptr;  
 struct student\* tail = nullptr;  
  
 string filename\_in;  
 cout << "Введите имя файла с данными: " << endl;  
 getline(cin, filename\_in);  
 cout << endl;  
  
 ifstream fi(filename\_in);  
 if (!fi.is\_open()) {  
 cout << "Файл не найден." << endl;  
 exit(1);  
 }  
  
 string s;  
 while (getline(fi, s)) {  
 struct student\* p = new student;  
 p->name.second = s;  
 getline(fi, p->name.first);  
 getline(fi, p->name.patronymic);  
 getline(fi, p->group);  
 float average = 0;  
 for (int i = 0; i < 4; ++i) {  
 fi >> p->grades.B[i];  
 average += p->grades.B[i];  
 }  
 p->grades.average = average / 4;  
 getline(fi, s); // прочитать пустую строку после оценок  
  
 if (head == nullptr) {  
 head = p;  
 head->prev = p; // Circular: head's prev points to self  
 head->next = p; // Circular: head's next points to self  
 tail = p;  
 } else {  
 tail->next = p;  
 p->prev = tail;  
 tail = p;  
 }  
 }  
  
 // Make the linked list circular  
 if (head != nullptr && tail != nullptr) {  
 head->prev = tail;  
 tail->next = head;  
 }  
  
 fi.close();  
 return head;  
}  
  
void delete\_students\_with\_min\_average(struct student \*&head) {  
 if (head == nullptr) {  
 cout << "Нечего удалять." << endl;  
 return;  
 }  
  
 // Находим минимальный средний балл среди всех студентов  
 struct student \*current = head;  
 float min\_avg = current->grades.average;  
 do {  
 if (current->grades.average < min\_avg) {  
 min\_avg = current->grades.average;  
 }  
 current = current->next;  
 } while (current != head);  
  
 // Удаляем записи о студентах с минимальным средним баллом  
 bool has\_deleted = false; // Флаг для отслеживания удаления элементов  
 do {  
 current = head;  
 bool deleted\_this\_round = false; // Флаг для отслеживания удаления на этой итерации  
  
 do {  
 // Сохраняем указатель на следующий элемент перед удалением текущего  
 struct student \*next\_student = current->next;  
  
 if (current->grades.average == min\_avg) {  
 // Удаление узла current из списка  
 if (current->next == current) {  
 // Единственный элемент в списке  
 delete current;  
 head = nullptr;  
 break;  
 } else {  
 current->prev->next = current->next;  
 current->next->prev = current->prev;  
 if (current == head) {  
 head = current->next;  
 }  
 delete current;  
 has\_deleted = true;  
 deleted\_this\_round = true;  
 }  
 }  
 current = next\_student;  
 } while (current != head);  
  
 // Если на этой итерации ничего не было удалено, выходим из цикла  
 if (!deleted\_this\_round) {  
 break;  
 }  
 } while (head != nullptr);  
  
 // Проверка на пустой список после удаления  
 if (head == nullptr) {  
 cout << "Все записи о студентах удалены." << endl;  
 } else if (has\_deleted) {  
 cout << "Записи о студентах с минимальным средним баллом успешно удалены." << endl;  
 } else {  
 cout << "Нет студентов с минимальным средним баллом." << endl;  
 }  
}  
  
  
void print\_linked\_list(struct student \*head) {  
 if (head == nullptr) {  
 cout << "Список пуст." << endl;  
 exit(1);  
 }  
  
 struct student \*current = head;  
 do {  
 cout << "Second name: " << current->name.second << endl;  
 cout << "First name: " << current->name.first << endl;  
 cout << "Patronymic: " << current->name.patronymic << endl;  
 cout << "Group: " << current->group << endl;  
 cout << "Grades: ";  
 for (int i = 0; i < 4; ++i)  
 cout << current->grades.B[i] << " ";  
 cout << endl;  
 cout << "GPA: " << current->grades.average << endl << endl;  
  
 current = current->next;  
 } while (current != head);  
}  
  
void linked\_list\_to\_file(struct student \*head) {  
 if (head == nullptr) {  
 cout << "Список пуст. Нечего записывать в файл." << endl;  
 exit(1);  
 }  
  
 struct student \*current = head;  
 ofstream fo;  
 string filename\_out;  
 cout << "Введите имя файла для записи результатов: " << endl;  
 getline(cin, filename\_out);  
 cout << endl;  
 if (filename\_out.substr(filename\_out.length() - 4) != ".txt") {  
 cout << "Неверное имя файла. Файл должен иметь расширение '.txt'." << endl;  
 exit(1);  
 } else {  
 fo.open(filename\_out);  
 do {  
 fo << current->name.second << endl;  
 fo << current->name.first << endl;  
 fo << current->name.patronymic << endl;  
 fo << current->group << endl;  
 for (int i = 0; i < 4; ++i)  
 fo << current->grades.B[i] << " ";  
 fo << endl << current->grades.average << endl << endl;  
  
 current = current->next;  
 } while (current != head);  
 fo.close();  
 }  
}  
  
void free\_memory(student \*head) {  
 if (head == nullptr) {  
 cout << "Список пуст. Очистка не требуется." << endl;  
 return;  
 }  
  
 student \*current = head;  
 do {  
 student \*temp = current;  
 current = current->next;  
 delete temp;  
 } while (current != head);  
  
 cout << "Память очищена успешно." << endl;  
}  
  
int main() {  
 struct student \*head = nullptr;  
  
 head = createCircularQueueLinkedList(); // Создаем список студентов  
 cout << "Исходный список студентов:" << endl;  
 print\_linked\_list(head); // Выводим исходный список  
  
 delete\_students\_with\_min\_average(head); // Удаляем студентов с минимальным средним баллом  
  
 cout << "Обновленный список студентов:" << endl;  
 print\_linked\_list(head); // Выводим обновленный список  
  
 linked\_list\_to\_file(head); // Записываем список студентов в файл  
  
 free\_memory(head); // Освобождаем память  
  
 return 0;  
}

**ТЕСТЫ**

1. **Пустой файл:**

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание

1. **У всех одинаковый средний бал**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание**

1. **Обычное удаление:**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание**

1. **Очередь из одного элемента:**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание**

**Дополнительное задание**

**Листинг**

#include <iostream>  
#include <fstream>  
#include <string>  
  
using namespace std;  
  
struct student {  
 struct student\* next;  
 struct student\* prev;  
 struct { string second, first, patronymic; } name;  
 string group;  
 struct { int B[4]; float average; } grades;  
};  
  
struct student\* createCircularQueueLinkedList() {  
 struct student\* head = nullptr;  
 struct student\* tail = nullptr;  
  
 string filename\_in;  
 cout << "Введите имя файла с данными: " << endl;  
 getline(cin, filename\_in);  
 cout << endl;  
  
 ifstream fi(filename\_in);  
 if (!fi.is\_open()) {  
 cout << "Файл не найден." << endl;  
 exit(1);  
 }  
  
 string s;  
 while (getline(fi, s)) {  
 struct student\* p = new student;  
 p->name.second = s;  
 getline(fi, p->name.first);  
 getline(fi, p->name.patronymic);  
 getline(fi, p->group);  
 float average = 0;  
 for (int i = 0; i < 4; ++i) {  
 fi >> p->grades.B[i];  
 average += p->grades.B[i];  
 }  
 p->grades.average = average / 4;  
 getline(fi, s); // прочитать пустую строку после оценок  
  
 if (head == nullptr) {  
 head = p;  
 head->prev = p; // Circular: head's prev points to self  
 head->next = p; // Circular: head's next points to self  
 tail = p;  
 } else {  
 tail->next = p;  
 p->prev = tail;  
 tail = p;  
 }  
 }  
  
 // Make the linked list circular  
 if (head != nullptr && tail != nullptr) {  
 head->prev = tail;  
 tail->next = head;  
 }  
  
 fi.close();  
 return head;  
}

void input\_student\_data(student \*new\_student) {  
 cout << "Введите фамилию: ";  
 getline(cin, new\_student->name.second);  
 cout << "Введите имя: ";  
 getline(cin, new\_student->name.first);  
 cout << "Введите отчество: ";  
 getline(cin, new\_student->name.patronymic);  
 cout << "Введите группу: ";  
 getline(cin, new\_student->group);  
 cout << "Введите оценки (4 шт.): ";  
 float average = 0;  
 for (int i = 0; i < 4; ++i) {  
 cin >> new\_student->grades.B[i];  
 average += new\_student->grades.B[i];  
 }  
 new\_student->grades.average = average / 4;  
 cin.ignore(); // очищаем буфер после ввода оценок  
}  
  
void insert\_before\_surname(student \*&head, const string &surname) {  
 if (head == nullptr) {  
 cout << "Список пуст." << endl;  
 return;  
 }  
  
 student \*current = head;  
 bool found = false;  
 bool first\_iteration = true;  
  
 // Создание нового студента  
 student \*new\_student = new student;  
 input\_student\_data(new\_student);  
  
 do {  
 if (current->name.second == surname) {  
 student \*inserted\_student = new student(\*new\_student);  
  
 inserted\_student->next = current;  
 inserted\_student->prev = current->prev;  
 current->prev->next = inserted\_student;  
 current->prev = inserted\_student;  
  
 if (current == head && first\_iteration) {  
 head = inserted\_student;  
 }  
  
 found = true;  
 }  
  
 current = current->next;  
 first\_iteration = false;  
 } while (current != head);  
  
 delete new\_student; // удаляем шаблон нового студента  
  
 if (!found) {  
 cout << "Студенты с фамилией " << surname << " не найдены." << endl;  
 exit(1);  
 }  
}  
  
void print\_linked\_list(struct student \*head) {  
 if (head == nullptr) {  
 cout << "Список пуст." << endl;  
 return;  
 }  
  
 struct student \*current = head;  
  
 // Вывод от начала к концу  
 cout << "Вывод от начала к концу:" << endl;  
 do {  
 cout << "Second name: " << current->name.second << endl;  
 cout << "First name: " << current->name.first << endl;  
 cout << "Patronymic: " << current->name.patronymic << endl;  
 cout << "Group: " << current->group << endl;  
 cout << "Grades: ";  
 for (int i = 0; i < 4; ++i)  
 cout << current->grades.B[i] << " ";  
 cout << endl;  
 cout << "GPA: " << current->grades.average << endl << endl;  
  
 current = current->next;  
 } while (current != head);  
  
 // Переход к последнему элементу для вывода от конца к началу  
 current = head->prev;  
  
 // Вывод от конца к началу  
 cout << "Вывод от конца к началу:" << endl;  
 do {  
 cout << "Second name: " << current->name.second << endl;  
 cout << "First name: " << current->name.first << endl;  
 cout << "Patronymic: " << current->name.patronymic << endl;  
 cout << "Group: " << current->group << endl;  
 cout << "Grades: ";  
 for (int i = 0; i < 4; ++i)  
 cout << current->grades.B[i] << " ";  
 cout << endl;  
 cout << "GPA: " << current->grades.average << endl << endl;  
  
 current = current->prev;  
 } while (current != head->prev);  
}  
  
void linked\_list\_to\_file(struct student \*head) {  
 if (head == nullptr) {  
 cout << "Список пуст. Нечего записывать в файл." << endl;  
 exit(1);  
 }  
  
 struct student \*current = head;  
 ofstream fo;  
 string filename\_out;  
 cout << "Введите имя файла для записи результатов: " << endl;  
 getline(cin, filename\_out);  
 cout << endl;  
 if (filename\_out.substr(filename\_out.length() - 4) != ".txt") {  
 cout << "Неверное имя файла. Файл должен иметь расширение '.txt'." << endl;  
 exit(1);  
 } else {  
 fo.open(filename\_out);  
 do {  
 fo << current->name.second << endl;  
 fo << current->name.first << endl;  
 fo << current->name.patronymic << endl;  
 fo << current->group << endl;  
 for (int i = 0; i < 4; ++i)  
 fo << current->grades.B[i] << " ";  
 fo << endl << current->grades.average << endl << endl;  
  
 current = current->next;  
 } while (current != head);  
 fo.close();  
 }  
}  
  
void free\_memory(student \*head) {  
 if (head == nullptr) {  
 cout << "Список пуст. Очистка не требуется." << endl;  
 return;  
 }  
  
 student \*current = head;  
 do {  
 student \*temp = current;  
 current = current->next;  
 delete temp;  
 } while (current != head);  
  
 cout << "Память очищена успешно." << endl;  
}  
  
int main() {  
 struct student \*head = nullptr;  
  
 head = createCircularQueueLinkedList(); // Создаем список студентов  
 cout << "Исходный список студентов:" << endl;  
 print\_linked\_list(head);// Выводим исходный список  
  
 string surname;  
 cout << "Введите фамилию, перед которой нужно вставить нового студента: ";  
 getline(cin, surname);  
  
 insert\_before\_surname(head, surname); // Удаляем студентов с минимальным средним баллом  
  
 cout << "Обновленный список студентов:" << endl;  
 print\_linked\_list(head); // Выводим обновленный список  
  
 linked\_list\_to\_file(head); // Записываем список студентов в файл  
  
 free\_memory(head); // Освобождаем память  
  
 return 0;  
}

**Тесты**

**Введите имя файла с данными:**

**file1.txt**

**Исходный список студентов:**

**Вывод от начала к концу:**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: Sergeev2**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 7 10**

**GPA: 7.25**

**Second name: Sergeev3**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 6 6**

**GPA: 6**

**Вывод от конца к началу:**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 6 6**

**GPA: 6**

**Second name: Sergeev3**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: Sergeev2**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 7 10**

**GPA: 7.25**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Введите фамилию, перед которой нужно вставить нового студента: Sergeev1**

**Введите фамилию: klfj**

**Введите имя: aklfj**

**Введите отчество: asfkj**

**Введите группу: lasfkj**

**Введите оценки (4 шт.): 1 1 1 1**

**Обновленный список студентов:**

**Вывод от начала к концу:**

**Second name: klfj**

**First name: aklfj**

**Patronymic: asfkj**

**Group: lasfkj**

**Grades: 1 1 1 1**

**GPA: 1**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: Sergeev2**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 7 10**

**GPA: 7.25**

**Second name: Sergeev3**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: klfj**

**First name: aklfj**

**Patronymic: asfkj**

**Group: lasfkj**

**Grades: 1 1 1 1**

**GPA: 1**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 6 6**

**GPA: 6**

**Вывод от конца к началу:**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 6 6**

**GPA: 6**

**Second name: klfj**

**First name: aklfj**

**Patronymic: asfkj**

**Group: lasfkj**

**Grades: 1 1 1 1**

**GPA: 1**

**Second name: Sergeev3**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: Sergeev2**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 7 10**

**GPA: 7.25**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: klfj**

**First name: aklfj**

**Patronymic: asfkj**

**Group: lasfkj**

**Grades: 1 1 1 1**

**GPA: 1**

1. **Случай при отсутствии заданной фамилии в очереди**

**Введите имя файла с данными:**

**file1.txt**

**Исходный список студентов:**

**Вывод от начала к концу:**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: Sergeev2**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 7 10**

**GPA: 7.25**

**Second name: Sergeev3**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 6 6**

**GPA: 6**

**Вывод от конца к началу:**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 6 6**

**GPA: 6**

**Second name: Sergeev3**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Second name: Sergeev2**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 6 6 7 10**

**GPA: 7.25**

**Second name: Sergeev1**

**First name: Sergey**

**Patronymic: Sergeevich**

**Group: BIV211**

**Grades: 7 8 9 10**

**GPA: 8.5**

**Введите фамилию, перед которой нужно вставить нового студента: sergo**

**Введите фамилию: aslkfj**

**Введите имя: safm**

**Введите отчество: asfljh**

**Введите группу: asjfkh**

**Введите оценки (4 шт.): 1 1 1 1**

**Студенты с фамилией sergo не найдены.**