Практика

а) формирования исходных бинарных файлов на базе исходных текстовых

```cpp
void createBinaryStudents(const std::string& inputFile, const std::string& outputFile) {
    std::ifstream in(inputFile);
    CheckInputFile( &in);
    std::ofstream out(outputFile,  mode: std::ios::binary);
    CheckOutputFile( &out);

    std::string line;
    while (std::getline( &in,  &line)) {
        std::stringstream ss(line);
        std::string inlexStr, lastNameStr, firstNameStr, patronymicStr;

        if (std::getline( &ss,  &inlexStr,  delim: ';') &&
            std::getline( &ss,  &lastNameStr,  delim: ';') &&
            std::getline( &ss,  &firstNameStr,  delim: ';') &&
            std::getline( &ss,  &patronymicStr,  delim: ';')) {

            StudentFull s;

            s.id = stoi(inlexStr);
            std::strncpy(s.lastName,  Source: lastNameStr.c_str(),  Count: sizeof(s.lastName));
            std::strncpy(s.firstName,  Source: firstNameStr.c_str(),  Count: sizeof(s.firstName));
            std::strncpy(s.patronymic,  Source: patronymicStr.c_str(),  Count: sizeof(s.patronymic));

            out.write(reinterpret_cast<char*>(&s),  n: sizeof(StudentFull));
        }
    }

    in.close();
    out.close();
}
```

```cpp
void createBinaryGrades(const std::string& inputFile, const std::string& outputFile) {
    std::ifstream in(inputFile);
    CheckInputFile( &in);
    std::ofstream out(outputFile,  mode: std::ios::binary);
    CheckOutputFile( &out);

    std::string line;
    while (std::getline( &in,  &line)) {
        std::stringstream ss(line);
        std::string group, index,  subj1, grade1, subj2, grade2, subj3, grade3;

        if (std::getline( &ss,  &group,  delim: ';') &&
            std::getline( &ss,  &index,  delim: ';') &&
            std::getline( &ss,  &subj1,  delim: ';') &&
            std::getline( &ss,  &grade1,  delim: ';') &&
            std::getline( &ss,  &subj2,  delim: ';') &&
            std::getline( &ss,  &grade2,  delim: ';') &&
            std::getline( &ss,  &subj3,  delim: ';') &&
            std::getline( &ss,  &grade3,  delim: ';')) {

            StudentFull s;
            s.group = stoi(group);
            s.id = stoi(index);
            s.math = stoi(grade1);
            s.geo = stoi(grade2);
            s.prog = stoi(grade3);

            out.write(reinterpret_cast<char*>(&s),  n: sizeof(StudentFull));
        }
    }

    in.close();
    out.close();
}
```

```cpp
struct StudentFull {
    int id;
    int group;
    char lastName[30];
    char firstName[30];
    char patronymic[30];
    int math;
    int geo;
    int prog;
    double average;

    StudentFull() {
        id = 0;
        group = 0;
        math = 0;
        geo = 0;
        prog = 0;
        average = 0.0;

        lastName[0] = '\0';
        firstName[0] = '\0';
        patronymic[0] = '\0';
    }

};
```

b) подсоединения фамилии студентов к ведомостям оценок с формированием нового бинарного файла;

```
void mergeStudentGrades(const std::string& studentsBin, const std::string& gradesBin, const std::string& outputBin) {

    std::ifstream students(studentsBin, mode: std::ios::binary);
    CheckInputFile([&]students);
    std::ifstream grades(gradesBin, mode: std::ios::binary);
    CheckInputFile([&]grades);
    std::ofstream out(outputBin, mode: std::ios::binary);
    CheckOutputFile([&]out);

    StudentFull grades_s;
    while (grades.read(reinterpret_cast<char*>(&grades_s), n: sizeof(StudentFull))) {

        students.clear();
        students.seekg( 0);

        StudentFull students_s;

        while (students.read(reinterpret_cast<char*>(&students_s), n: sizeof(StudentFull))) {
            if (students_s.id == grades_s.id) {

                strncpy(grades_s.lastName, students_s.lastName, Count: sizeof(grades_s.lastName));
                out.write(reinterpret_cast<char*>(&grades_s), n: sizeof(StudentFull));

                break;
            }
        }
    }
}
```

c) вычисления среднего балла каждого студента с формированием нового бинарного файла;

```
void calculateAverage(const std::string& inputBin, const std::string& outputBin) {
    std::ifstream in(inputBin,  mode: std::ios::binary);
    CheckInputFile( [&] in);
    std::ofstream out(outputBin,  mode: std::ios::binary);
    CheckOutputFile( [&] out);

    StudentFull s;
    while (in.read(reinterpret_cast<char*>(&s),  n: sizeof(StudentFull))) {

        s.average = (s.math + s.geo + s.prog) / 3.0;
        out.write(reinterpret_cast<char*>(&s),  n: sizeof(StudentFull));
    }
}
```

d) формирования списка неуспевающих, состоящего из фамилии, номера группы, номеразачетки;

```
void createFailingList(const std::string& averageBin, const std::string& failingBin) {
    std::ifstream in(averageBin, mode: std::ios::binary);
    CheckInputFile([&]in);
    std::ofstream out(failingBin, mode: std::ios::binary);
    CheckOutputFile([&]out);

    StudentFull s;

    while (in.read(reinterpret_cast<char*>(&s), n: sizeof(StudentFull))) {
        if (s.average < 4.0) {

            StudentFull fs;

            fs.id = s.id;
            fs.group = s.group;
            strncpy(fs.lastName, s.lastName, Count: sizeof(fs.lastName));

            out.write(reinterpret_cast<char*>(&fs), n: sizeof(StudentFull));
        }
    }

    in.close();
    out.close();
}
```

e) сортировки списка неуспевающих по группам, в группе - по фамилиям в алфавитном порядке;

```cpp
template<class Compare>
void sortBin(const std::string& nameBin,Compare comp) {
    int size = countSizeBin(nameBin);

    StudentFull* arr = new StudentFull[size];

    readStudents(nameBin, arr, size);

    std::sort( first: arr,  last: arr + size, comp);

    writeStudentsToFile(nameBin, arr, size);

    delete[] arr;
}
```

Разделение ответственности

Универсальность через шаблон

Гибкость и масштабируемость

```cpp
void writeStudentsToBin(const std::string& filename, StudentFull* arr, int size) {
    std::ofstream out(filename,  mode: std::ios::binary);
    CheckOutputFile( [&] out);

    for (int i = 0; i < size; ++i) {
        out.write(reinterpret_cast<char*>(&arr[i]),  n: sizeof(StudentFull));
    }

    out.close();
}
```

```cpp
void readStudents(const std::string& filename, StudentFull* arr, int count) {
    std::ifstream in(filename,  mode: std::ios::binary);
    CheckInputFile( [&] in);

    for (int i = 0; i < count; ++i) {
        if (!in.read(reinterpret_cast<char*>(&arr[i]),  n: sizeof(StudentFull)))
            throw std::runtime_error("Error read Students");
    }

    in.close();
}
```

```cpp
int countSizeStudentBin(const std::string& filename) {
    std::ifstream in(filename,  mode: std::ios::binary | std::ios::ate);
    CheckInputFile( [&] in);

    std::streamsize size  = in.tellg();
    return static_cast<int>(size / sizeof(StudentFull));
}
```

f) распечатки бинарного файла до сортировки и после;

```
void printStudentsList(const std::string& failingBin) {
    std::ifstream in(failingBin, mode: std::ios::binary);
    CheckInputFile([&]in);

    StudentFull fs;
    while (in.read(reinterpret_cast<char*>(&fs), n: sizeof(StudentFull))) {
        std::cout << "lastName: " << fs.lastName;
        std::cout << " firstName: " << fs.firstName;
        std::cout << " patronymic: " << fs.patronymic;
        std::cout << " group: " << fs.group;
        std::cout << " id: " << fs.id ;
        std::cout << " math: " << fs.math;
        std::cout << " geo: " << fs.geo;
        std::cout << " prog: " << fs.prog;
        std::cout << " average: " << std::fixed << std::setprecision( n: 4) << fs.average << std::endl;
    }
}
```

g) формирования ведомости оценок для заданной группы, упорядоченной по алфавиту;h) формирования ведомости оценок для заданной группы, упорядоченной по убыванию среднего балла;

```cpp
template<class Compare>
void GroupSorted(const std::string& mergedBin, int targetGroup, Compare comp) {
    int size{};
    StudentFull* arr = readGroupFromFile(mergedBin, targetGroup, [&] size);

    std::sort( first: arr, last: arr + size, comp);
    writeStudentsToFile(mergedBin, arr, size);

    delete[] arr;
}
```

```cpp
StudentFull* readGroupFromFile(const std::string& filename, int targetGroup, int& outSize) {
    std::ifstream in(filename, mode: std::ios::binary);
    CheckInputFile([&] in);

    StudentFull* arr = nullptr;
    outSize = 0;
    StudentFull s;

    while (in.read(reinterpret_cast<char*>(&s), n: sizeof(StudentFull))) {
        if (s.group == targetGroup) {
            addElementToArray([&] arr, [&] outSize, s);
        }
    }

    in.close();
    return arr;
}
```

```cpp
template<class T>
void copyArray(T* arr, T*& newArr, int oldSize) {
    newArr = new T[oldSize + 1];
    for (int i = 0; i < oldSize; ++i) {
        newArr[i] = arr[i];
    }
}

template<class T>
void addElementToArray(T*& arr, int& size, T element) {
    T* tempArr = nullptr;
    copyArray(arr, newArr: [&] tempArr, size);
    tempArr[size] = element;
    delete[] arr;
    arr = tempArr;
    ++size;
}
```

Гибкость через шаблоны

Динамическое расширение массива

Модульность и читаемость

i) формирования списка отличников, состоящего из фамилии, номера группы, номера зачетки;

```
void createGoodList(const std::string& inBin, const std::string& outBin) {
    std::ifstream in(inBin, mode: std::ios::binary);
    CheckInputFile([&]in);
    std::ofstream out(outBin, mode: std::ios::binary);
    CheckOutputFile([&]out);

    StudentFull s;
    while (in.read(reinterpret_cast<char*>(&s), n: sizeof(StudentFull))) {
        if (s.average >= 8.0) {

            StudentFull gs;

            strncpy(gs.lastName, s.lastName, Count: sizeof(gs.lastName));
            gs.id = s.id;
            gs.group = s.group;

            out.write(reinterpret_cast<char*>(&gs), n: sizeof(StudentFull));
        }
    }
}
```

Запись из .bin ы .txt

```
void binaryToText(const std::string& inputBin, const std::string& outputText) {
    std::ifstream in(inputBin, mode: std::ios::binary);
    CheckInputFile([&] in);

    std::ofstream out(outputText);
    CheckOutputFile([&] out);

    StudentFull student;

    while (in.read(reinterpret_cast<char*>(&student), n: sizeof(StudentFull))) {
        out << student.group << ';'
            << student.id << ';'
            << student.lastName << ';'
            << student.firstName << ';'
            << student.patronymic << ';'
            << "MA" << ';'
            << student.math << ';'
            << "GEO" << ';'
            << student.geo << ';'
            << "PROG" << ';'
            << student.prog << ';'
            << std::fixed << std::setprecision( n: 2) << student.average << std::endl;
    }

    in.close();
    out.close();
}
```