Лабораторная работа №6-7
Студент:
Круглов Артем Евгеньевич

Отчет

Прилагаемый проект содержит требования одновременно и 6-й и 7-й Лабораторной работы.

Список файлов

```
AKClasses.cpp
checking.cpp
structs.h
AKClasses.h
checking.h
Makefile
structs.cpp
stud.cpp
t
```

stud.cpp

Основной файл проекта с функцией int main().

```
#include <iostream>
#include <fstream>
#include <string>
#include <list>
#include "checking.h"
#include "structs.h"
#include "AKClasses.h"

using namespace std;

int main(int argc, char** argv) {
    Flags f;
    // Опции из командной строки, переведенные в формат структуры
    sList sl;
    // Список студентов

    if (CheckFlags(argc, argv, &f) != "") {
        cout << CheckFlags(argc, argv, &f) << endl;
        return -1;
        // Если нашли ошибки в параметрах, заканчиваем
    }

    sl = GetStudent(f); // Получаем список студентов

    if (f.toFile) {
        ofstream fl(f.toFileName);
        if (!fl) {
            cout << "can't open file" << endl;
            return 1;
```

```
        }
        for (Student const& s : sl) {

            if (s.university == 1) {
                // В зависимости от университета используем разные
классы
                AKMIEMGen *m = new AKMIEMGen;
                // Генерируем студенческий и пишем в файл
                fl << m->Generate(s) << endl;
            } else {
                AKMGTUUGen *m = new AKMGTUUGen;
                // Генерируем студенческий и пишем в файл
                fl << m->Generate(s) << endl;
            }
        }
    } else {
        for (Student const& s : sl) {

            if (s.university == 1) {
                AKMIEMGen *m = new AKMIEMGen;
                // Генерируем студенческий и пишем в консоль
                cout << m->Generate(s) << endl;
            } else {
                AKMGTUUGen *m = new AKMGTUUGen;
                // Генерируем студенческий и пишем в консоль
                cout << m->Generate(s) << endl;
            }

    }

    }

}
```

Описание:

Программа вызывает функцию разбора параметров. Если она вернула текст ошибки, значит параметры были указаны неверно и мы завершаем работу. Если вернулась пустая строка, вызываем функцию получения списка студентов, которая в зависимости от переданных параметров берет его или из файла или из консоли. Перебираем всех студентов, в зависимости от их университета, генерируем номер студенческого с помощью соответствующего класса.

checking.cpp

Файл вспомогательных функций — разбор параметров, переданных через консоль, и получение списка студентов или из файла или с консоли.

Список функций:

```
string CheckFlags(int argc, char** argv, Flags* f);
```

Разбирает параметры из командной строки. Если что-то не так, возвращает текст ошибки. Если все хорошо, то возвращает пустую строку. Флаги пишет в переменную f.

```
sList GetStudent(Flags f);
```

Получет список студентов или из файла, или из консоли. В зависимости от параметров.

Код:

```
#include <iostream>
#include <fstream>
#include <string>
#include <list>
#include "structs.h"

using std::string;
using namespace std;


string CheckFlags(int argc, char** argv, Flags* f) {
// Рутинный разбор всех параметров
// Учитываем, что параметры --fromFile и --toFile могут идти
// в разном порядке. Если все хорошо, заполняем структуру f
// для более удобного использования дальше.
// Если параметры ошибочные, возвращаем текст ошибки. Иначе пустую
строку

    if (argc != 1 && argc != 3 && argc != 5) {
        return "Wrong number of parameters";
    }

    if (argc == 3 && (string)argv[1] != "--fromFile" &&
(string)argv [1] != "--toFile") {
        return "Not valid flag: " + (string)argv[1];
    }

    if (argc == 5 && (string)argv[3] != "--fromFile" &&
(string)argv [3] != "--toFile") {
        return "Not valid flag: " + (string)argv[3];
    }

    if (argc == 5 && (string)argv[3] == (string)argv[1]) {
        return "Flag is duplicated: " + (string)argv[3];
    }

    if (argc == 1) {
        (*f).fromFile = false;
        (*f).toFile = false;
    }

    if (argc == 3) {
        if ((string)argv[1] == "--fromFile") {
            (*f).fromFile = true;
            (*f).fromFileName = (string)argv[2];
            (*f).toFile = false;
```

```
        }

        if ((string)argv[1] == "--toFile") {
            (*f).toFile = true;
            (*f).toFileName = (string)argv[2];
            (*f).fromFile = false;
        }

    }

    if (argc == 5) {
        (*f).fromFile = true;
        (*f).toFile = true;
        if ((string)argv[1] == "--fromFile") {
            (*f).fromFileName = (string)argv[2];
            (*f).toFileName = (string)argv[4];
        }

        if ((string)argv[1] == "--toFile") {
            (*f).fromFileName = (string)argv[4];
            (*f).toFileName = (string)argv[2];
        }

    }

    return "";
}

sList GetStudent(Flags f) {
// Получаем список студентов из файла или из консоли и возвращаем
его

    list<Student> sl = {};
    Student s;
    s.day = 0;

    if (f.fromFile) {
        ifstream fl;
        fl.open(f.fromFileName);
        if (!fl) {

            cout << endl << "Could not open file: " <<
f.fromFileName << endl;
            return sl; // day == 0 in this case
        }
        while (!fl.eof()) {
            // Читаем очередную строку по словам.
            // Если в какой-то момент очередного слова нет,
            // выдаем ошибку и возвращаем текущий список студентов

            string word;
            if (!(fl >> word)) {
                "Wrong file";
                return sl;
            }

            if (word != "man" && word != "woman") {
```

```
                cout << "Wrong sex";
                return sl;
            }

            if (word == "man") {
                s.sex = 1;
            } else {
                s.sex = 0;
            }

            if (!(fl >> word)) {
                cout << "Wrong file" << endl;
                return sl;
            }

            s.year = stoi(word);

            if (!(fl >> word)) {
                cout << "Wrong file" << endl;
                return sl;
            }

            s.month = stoi(word);
            if (s.month < 1 || s.month > 12) {
                cout << "Wrong month" << endl;
                s.day = 0;
                return sl;
            }
            if (!(fl >> word)) {
                cout << "Wrong file" << endl;
                return sl;
            }

            s.day = stoi(word);
            if (!(fl >> word)) {
                cout << "Wrong file" << endl;
                return sl;
            }
            s.university = stoi(word);

            sl.push_back(s); // Если все удачно, добавляем
студента в список
        }

    }

    if (!f.fromFile) {
        // Если соответствующего параметра нет, читаем из консоли

        string word;
        cout << "Укажите пол студента" << endl;
        cin >> word;
        if (word == "man") {
            s.sex = 1;
        } else {
            if (word == "woman") {
                s.sex = 0;
```

```
            } else {
                cout << "Wrong sex" << endl;
                return sl;
            }
        }
        cout << "Укажите год рождения студента" << endl;
        cin >> word;
        s.year = stoi(word);

        cout << "Укажите месяц рождения студента" << endl;
        cin >> word;
        s.month = stoi(word);

        cout << "Укажите день рождения студента" << endl;
        cin >> word;
        s.day = stoi(word);
        sl.push_back(s);


    }

    return sl;
}
```

AKClasses.h

Объявление трех основных классов.

```
class AKGenerator
```

Абстрактный класс-генератор

```
class  AKMIEMGen: public virtual AKGenerator
```

Класс-генератор для МИЭМ

```
class  AKMGTUUGen: public virtual AKGenerator
```

Класс-генератор для МГТУУ

Код:

```
#ifndef AKGENERATOR
#define AKGENERATOR
#include <iostream>
#include <string>
#include <map>
#include "structs.h"

using std::string;

class AKGenerator {
// Абстрактный класс-генератор
```

```
    public:
        string Generate(Student st);
        // Генерирует номер студенческого билета

    protected:
        map<string, string> codes;
        // Словарь, в котором хранятся все уже сгенерированные
коды
        // для встретившихся дат рождений. Нужно для обеспечения
уникальности

        int PseudoMax(int maxValue, int dobavka);
        // Возвращает псевдослучайное число в диапазоне от 0 до
maxValue
        // Для генерации случайных чисел используется добавка при
инициализации
        // генератора псевдослучайных чисел.

        string completeString(string s, int n);
        // Дополняет строку нулями слева. Используется при
добавлении в номер билета
        // коротких месяцев или дней рождения, а также при
получении маленьких
        // псевдослучайных чисел.

        virtual string sex(int n) const = 0;
        // Возвращает код для пола в номер студенческого билета
        // Каждый класс-потомок должен переопределить этот метод
по-своему

        virtual string Pseudo(string date) = 0;
        // Возвращает псевдослучайное число для добавления в номер
студ.билета
        // Каждый класс-потомок должен переопределить этот метод
по-своему

        virtual void MakeBetter(string*) {};
        // Поскольку для некоторых сгенерированных номеров
студ.билетов невозможно
        // правильным образом подобрать последний символ, метод
изменяет сгенерированную
        // псевдослучайную часть номера так, чтобы последний
символ можно было подобрать
        // Каждый класс-потомок должен переопределить метод,
поскольку алгоритм формирования
        // последнего символа у каждого университета свой

        virtual void Finalize(string*) = 0;
        // Добавляет последний символ в номер студ.билета

        int Calc(string);
        // Вычисляет сумму произведений цифры на ее порядковый
номер


};
```

```
class  AKMIEMGen: public virtual AKGenerator {
// Класс-генератор для МИЭМ

    public:
    protected:
        string sex(int n) const override;
        string Pseudo(string) override;
        void Finalize(string*) override;
        void MakeBetter(string*) override;
};

class  AKMGTUUGen: public virtual AKGenerator {
// Класс-генератор для МГТУУ

    public:
    protected:
    string sex(int n) const override;
    string Pseudo(string) override;
    void Finalize(string*) override;
    void MakeBetter(string*) override;
};
#endif
```

AKClasses.cpp

Реализация всех методов классов.

```
#include <iostream>
#include <string>
#include <random>
#include <ctime>
#include "structs.h"
#include "AKClasses.h"

using std::string;
using namespace std;

int AKGenerator::PseudoMax(int maxValue, int dobavka) {
    int r;
    int t = (int)time(nullptr);
    srand(t + dobavka); // Добавляем к времени добавку, чтобы
получать разные последовательности
    r = rand();

    r = ((double)r) / RAND_MAX * maxValue; // Изменяем r чтобы
попасть в диапазон

    return r;
}

string AKGenerator::completeString(string s, int n) {
    int k = s.length();
    if (k < n) {
        // Добавляем нужное число нулей
```

```
        for (int i = 0; i < n - k; i++) {
            s = "0" + s;
        }
    }
    return s;
}

string AKGenerator::Generate(Student st) {
    string id = "";
    id += this->sex(st.sex);
    id += this->completeString(to_string(st.year), 4);
    id += this->completeString(to_string(st.month), 2);
    id += this->completeString(to_string(st.day), 2);

    string date = to_string(st.year) + to_string(st.month) +
to_string(st.day);

    string code; // Строка со случайным числом

    map <string,string> :: iterator it;
    it = this->codes.find(date); // Ищем дату рождения в словаре
    if (it == this->codes.end()) {
        // Если не нашли, генерим новый код
        code = this->Pseudo(date);
        id += code; // Добавляем его в студ.билет
        this->codes[date] = code; // Добавляем в словарь
        this->MakeBetter(&id); // Решаем проблему заполняемости
последнего символа
    } else {
        // Если нашли, берем то, что уже было
        code = it->second;
        id += code;
        this->MakeBetter(&id);
        // Улучшать все равно приходится, поскольку в зависимости
от пола
        // в МГТУУ один и тот же код со случайным числом может не
подходить
        // для генерации последнего символа
    }

    this->Finalize(&id); // Добавляем последний символ

    return id;
}

int AKGenerator::Calc(string s) {
    int n = 0;
    for (int i = 0; i < s.length(); i++) {
        n += (i + 1) * (s[i] - '0');
    }
    return n;
}

string AKMIEMGen::sex(int n) const {
    return to_string(4*(n+1));
}
```

```
string AKMIEMGen::Pseudo(string date) {
    int r;
    int d = stoi(date);
    r = this->PseudoMax(99999, d);
    // Если получили короткое число, дополняем до 5 символов
нулями
    return this->completeString(to_string(r), 5);
}

string AKMGTUUGen::sex(int n) const {
    return to_string(n+1);
}

string AKMGTUUGen::Pseudo(string date) {
    int r;
    int d = stoi(date);
    r = this->PseudoMax(9999, d);
    // Если получили короткое число, дополняем до 4 символов
нулями
    return this->completeString(to_string(r), 4);
}

void AKMIEMGen::Finalize(string* s) {

    int n = this->Calc(*s);
    // Перебираем все цифры, чтобы найти ту, которая подойдет под
    // правило заполнения последнего символа.
    for (int i = 0; i < 10; i++) {
        if ((n + 15*i) % 11 == 0) {
            (*s) += i + '0';
            break;
        }
    }
}

void AKMIEMGen::MakeBetter(string* s) {
    int n = this->Calc(*s);
    int len = (*s).length();
    // Последний символ будет стоять на 15-й позиции.
    // При его подборе мы фактически решаем уравнение в целых
числах
    // X + 15*C = 0 (mod 11), где X - выражение от первых 14
символов,
    // C - нужный нам последний символ. Видно, что если C
получится равным 10,
    // то мы не сможем добавить это как один символ. 10
получается, если X дает
    // остаток 4 при делении на 11. В этом случае поменяем
последний символ кода
    // на 1 и тогда остаток будет другим.

    if (n % 11 == 4) {
        if ((*s)[len-1] == '9') {
            (*s)[len-1] = '8';
        } else {
            (*s)[len-1] = '0' + ((*s)[len-1] - '0' + 1);
        }
```

```
    }

}

void AKMGTUUGen::MakeBetter(string* s) {
    int n = this->Calc(*s);
    int len = (*s).length();
    // Последний символ будет стоять на 14-й позиции.
    // При его подборе мы фактически решаем уравнение в целых
числах
    // X + 14*C = 0 (mod 10), где X - выражение от первых 13
символов,
    // C - нужный нам последний символ. Видно, что если X -
нечетное,
    // то подобрать C не получится. В этом случае поменяем
последний символ кода
    // на 1 и тогда поскольку последний символ стоит на нечетной
позиции, то
    // новый X станет четным.

    if (n % 2 == 1) {
        if ((*s)[len-1] == '9') {
            (*s)[len-1] = '8';
        } else {
            (*s)[len-1] = '0' + ((*s)[len-1] - '0' + 1);
        }
    }

}

void AKMGTUUGen::Finalize(string* s) {
    int n = this->Calc(*s);
    // Перебираем все цифры, чтобы найти ту, которая подойдет под
    // правило заполнения последнего символа.

    for (int i = 0; i < 10; i++) {
        if ((n + 14*i) % 10 == 0) {
            (*s) += i + '0';
            break;
        }
    }

}
```

Для проверки работоспособности подготовлен тестовый файл `t` с параметрами студентов