Тампио Илья Сергеевич 3530904/10002 Индивидуальное задание – ООП - Вариант 2.1

Постановка задачи:  
Программа должна обеспечивать:

• Ввод имени входного файла, содержащего информацию для обработки;  
• Ввод имени выходного файла, содержащего результаты работы программы;  
• Проверку корректности информации, содержащейся во входном файле;  
• Обработку информации в соответствии с вариантом задания;  
• Обработку и инициализацию исключительных ситуаций (связанных, например, с  
проверкой значения полей).

Subscriber  
Структура входного файла:  
Первая строка – количество записей файла  
Каждая следующая строка – фамилия, имя, номер телефона, код тарифа.  
Между словами может быть произвольное число пробелов. Другие разделители не допускаются.  
Структура выходного файла:  
Первая строка –количество записей файла  
Каждая следующая строка – фамилия, имя, номер телефона, тариф. Между словами один пробел.

Правила записи входной информации:  
Фамилия записывается латинскими буквами, начинается с большой буквы.  
Внутри фамилии может быть дефис. Фамилия не может состоять менее чем из 3 и более чем 20  
символов.  
Имя записывается одной большой буквой (первая буква имени), за которой следует точка.  
Номер телефона записывается в форме: +x(xxx)xxx-xx-xx  
x- десятичная цифра.  
Тариф – код тарифа – три цифры.

Правила ввода-вывода:  
Имена файлов вводятся из стандартного потока std::cin.  
Строка файла читается целиком, затем проверяется корректность информации.   
Вывод в файл также выполняется построчно.

1. Упорядочить корректно введенную информацию по фамилии, имени, тарифу, номеру  
телефона (сортировка по методу выбора).  
2. Определить сколько различных абонентов, содержится в файле.  
3. Определить число абонентов, имеющих несколько номеров.

Детальные требования и тестплан:

Программа должна быть написана в соответствии со стандартом программирования: C++ Programming Style Guidelines (http://geosoft.no/development/cppstyle.html).

Программа должна быть сделана в двух вариантах:  
1. С использованием класса std::string из STL  
2. С использованием собственного класса String

Собственный класс String должен содержать:  
• Конструктор по умолчанию, создающий пустую строку  
• Конструктор, создающий строку из строки в стиле C  
• Конструкторы копирования и перемещения  
• Деструктор  
• Операторы копирующего и перемещающего присваивания  
• Метод compare для сравнения двух строк в лексикографическом порядке

Перед использованием класса в основном задании его необходимо отладить.

|  |  |  |  |
| --- | --- | --- | --- |
| Требование | Описание | Данные | Результат |
| 1. Входные данные | | | |
| 1.1 Имя файла ввода | Должно указывать на существующий и доступный файл, иначе исключение | 0.txt | Исключение: Ошибка открытия файла ввода и выход из программы |
| 1.2 Размер n массива число, большее чем ноль и не превышающее кол-во элементов | Примечание: если размер массива меньше, чем число введённых элементов, считаются только первые n элементов. | 8copy.txt  7copy.txt | Исключение:  «Размер массива больше введенных данных, завершение программы»  Или  «Размер массив должен быть больше нуля.» |
| 1.3 Данные и их порядок ввода | Фамилия (строка) – Имя (строка) – Телефон (строка) – Код тарифа (строка) | (все файлы) |  |
| 1.4 Файл вывода | Имеет имя output.txt, если недоступен для вывода пишется предупреждение, что записи в файл производиться не будет |  | Сообщение:  Ошибка открытия файла вывода... Запись в файл производиться не будет.\n |
| 1. Тест-план | | | |
| 2.1) Пробелы между данными | В каждой строке случайным образом расставлены пробелы между данными | 1.txt |  |
| 2.2) Постоянность счётчиков | При разных сортировках счётчики уникальных пользователей и тех, у которых несколько номеров должны выдавать тот же результат | (Все файлы, где размер массива больше 2) |  |
| 2.3) Игнорирование одинаковых записей | Одинаковые записи в таблице считываются, но интерпретируются как одна и та же (счётчики не меняются) | 3copy.txt |  |
| 2.4) Неверный формат данных | Выбрасывается исключение, поясняющее что не так |  | Возможные исключения:  "Имя должно быть заглавной буквой с точкой."  "Фамилия должна начинаться с большой буквы и быть словом"  "Телефон должен быть записан в форме +x(xxx)xxx-xx-xx"  "Тариф должен состоять из трёх цифр" |

Код:

OOP.cpp:

#include <iostream>

#include <fstream>

#include <stdexcept>

#include <string>

#include <functional>

#include <utility>

#include "Subscriber.h"

#include "myString.h"

#include "SubscriberList.h"

#include "comparators.cpp"

typedef std::string string;

//typedef myString string;

int main() {

setlocale(LC\_ALL, "ru-RU");

try {

std::ofstream ofs("output.txt");

if (ofs.fail()){

std::cerr << "Ошибка открытия файла вывода... Запись в файл производиться не будет.\n";

}

//Test of Subscriber

//string name = "G.", surname = "Zho-okov", phone = "+7(960)961-11-11", tariff = "303";

//Subscriber sub = Subscriber(name, surname, phone, tariff);

//std::cout << "TestSub\t: " << sub << '\n';

std::cout << "Введите имя тест-файла\n>> ";

string filename = "1.txt";

std::cin >> filename;

std::ifstream ifs(filename);

if (ifs.fail())

{

std::cerr << "Ошибка открытия файла ввода\n";

return 1;

}

int n = 0;

ifs >> n;

if (n <= 0) {

throw std::invalid\_argument("Размер массив должен быть больше нуля.");

}

SubscriberList subList;

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

Subscriber tempSub;

if (ifs.eof()) {

std::cerr << "Размер массива больше введенных данных, завершение программы\n";

return 1;

}

ifs >> tempSub;

subList.add(tempSub);

}

std::cout << "---- Введённый массив ----\n";

subList.printAll(std::cout);

bool exitFlag = false;

while (!exitFlag) {

std::cout << '\n' << "Выберите столбец сортировки:\n1 - Фамилия\n2 - Имя\n3 - Тариф\n4 - НомерТелефона\n5 - Имя+Фамилия\n6 - Выход\n>> ";

int choice = 0;

std::cin >> choice;

switch (choice) {

case 1:

subList.sort(comparators[COMPARATOR\_SURNAME]);

break;

case 2:

subList.sort(comparators[COMPARATOR\_NAME]);

break;

case 3:

subList.sort(comparators[COMPARATOR\_TARIFF]);

break;

case 4:

subList.sort(comparators[COMPARATOR\_PHONE]);

break;

case 5:

subList.sort(comparators[COMPARATOR\_KEY]);

break;

case 6:

// exit

return 0;

default:

throw std::invalid\_argument("Необходимо выбрать число от 1 до 6");

}

ofs << "Отсортированный массив:\n";

ofs << n << '\n';

subList.printAll(ofs);

subList.printAll(std::cout);

// Counters

int uniqueCounter = subList.countUniques();

int severalPhonesCount = subList.countSeverals();

std::cout << "\nUnique count: " << uniqueCounter << '\n';

std::cout << "Many numbers owners: " << severalPhonesCount << '\n';

ofs << "\nUnique count: " << uniqueCounter << '\n';

ofs << "Many numbers owners: " << severalPhonesCount << '\n';

}

}

catch (const std::invalid\_argument& e) {

std::cerr << "invalid\_argument: " << e.what() << '\n';

}

return 0;

}

Subscriber.h:

#pragma once

#include <string>

#include <iostream>

#include "myString.h"

typedef std::string string;

//typedef myString string;

class Subscriber {

public:

Subscriber(string&, string&, string&, string&);

Subscriber();

Subscriber(const Subscriber& src);

const string& getName() const;

const string& getSurname() const;

const string& getPhone() const;

const string& getTariff() const;

void setName(const string& value);

void setSurname(const string& value);

void setPhone(const string& value);

void setTariff(const string& value);

friend std::istream& operator>>(std::istream& is, Subscriber& sub);

friend std::ostream& operator<<(std::ostream& os, const Subscriber& sub);

private:

string name\_;

string surname\_;

string phone\_;

string tariff\_;

};

Subscriber.cpp:

#include "Subscriber.h"

#include <iostream>

#include <cctype>

#include <stdexcept>

string const& Subscriber::getName() const {

return name\_;

}

string const& Subscriber::getSurname() const {

return surname\_;

}

string const& Subscriber::getPhone() const {

return phone\_;

}

string const& Subscriber::getTariff() const {

return tariff\_;

}

inline bool isName(const string& s) {

return std::isupper(s[0]) && s[1] == '.';

}

void Subscriber::setName(const string& value) {

if (!(

value.length() == 2 &&

isName(value)))

{

throw std::invalid\_argument("Имя должно быть заглавной буквой с точкой.");

}

this->name\_ = value;

}

inline bool isWordDashed(const string& s) {

for (int i = 0; s[i] != '\0'; i++) {

if (!(std::isalpha(s[i]) || (s[i] == '-' && i != 0 && i != s.length() - 1))) {

return false;

}

}

return true;

}

void Subscriber::setSurname(const string& value) {

if (!(

value.length() >= 3 &&

value.length() <= 20 &&

std::isupper(value[0]) &&

isWordDashed(value)))

{

throw std::invalid\_argument("Фамилия должна начинаться с большой буквы и быть словом");

}

this->surname\_ = value;

}

inline bool isPhone(const string& s) {

//+x(xxx)xxx-xx-xx

for (int i = 0; s[i] != '\0'; i++) {

if (!(

(std::isdigit(s[i]) && i != 0 && i != 2 && i != 6 && i != 10 && i != 13) ||

(s[i] == '+' && i == 0) ||

(s[i] == '-' && (i == 13 || i == 10)) ||

(s[i] == '(' && i == 2) ||

(s[i] == ')' && i == 6)

))

{

std::cout << "i:\t" << i << "--> '" << s[i] << "'\n";

return false;

}

}

return true;

}

void Subscriber::setPhone(const string& value) {

if (!(

value.length() == 16 &&

isPhone(value)

))

{

throw std::invalid\_argument("Телефон должен быть записан в форме +x(xxx)xxx-xx-xx");

}

this->phone\_ = value;

}

inline bool isTariff(const string& s) {

for (int i = 0; s[i] != '\0'; i++) {

if (!std::isdigit(s[i])) {

return false;

}

}

return true;

}

void Subscriber::setTariff(const string& value) {

if (!(

value.length() == 3 &&

isTariff(value)

))

{

throw std::invalid\_argument("Тариф должен состоять из трёх цифр");

}

this->tariff\_ = value;

}

Subscriber::Subscriber(string& name, string& surname, string& phone, string& tariff) {

this->setName(name);

this->setSurname(surname);

this->setPhone(phone);

this->setTariff(tariff);

}

Subscriber::Subscriber() {

name\_ = "X.";

surname\_ = "X-XXXXXXXXXXXXXXXXXX";

phone\_ = "+X(XXX)XXX-XX-XX";

tariff\_ = "XXX";

}

std::istream& operator>>(std::istream& is, Subscriber& sub) {

string name = "", surname = "", phone = "", tariff = "";

is >> surname >> name >> phone >> tariff;

sub = Subscriber(name, surname, phone, tariff);

return is;

}

std::ostream& operator<<(std::ostream& os, const Subscriber& sub) {

os << "[ " << sub.getName() << ", " << sub.getSurname() << ", " << sub.getPhone() << ", " << sub.getTariff() << " ]\n";

return os;

}

Subscriber::Subscriber(const Subscriber& src) {

name\_ = src.name\_;

surname\_ = src.surname\_;

phone\_ = src.phone\_;

tariff\_ = src.tariff\_;

}

SubscriberList.h:

#pragma once

#include <iostream>

#include <functional>

#include "Subscriber.h"

class SubscriberList {

public:

SubscriberList();

SubscriberList(const SubscriberList&);

SubscriberList& operator=(const SubscriberList&);

~SubscriberList();

void add(Subscriber s);

void sort(std::function<bool(const Subscriber&, const Subscriber&)> compare);

void printAll(std::ostream&) const;

Subscriber& operator[](std::size\_t pos) const;

int countUniques() const;

int countSeverals() const;

private:

static const int MULTIPLIER = 2;

std::size\_t size\_;

int capacity\_;

Subscriber\*\* array\_;

};

myString.h:

#pragma once

#include <iostream>

class myString

{

public:

myString();

myString(const char\* str);

myString(const myString& s);

~myString();

myString& operator=(const myString& other);

myString operator+(const myString) const;

void push\_back(char c);

bool empty() const;

//char\* getString() const;

size\_t length() const;

bool operator>(const myString&) const;

bool operator<(const myString&) const;

bool operator==(const myString&) const;

bool operator!=(const myString&) const;

bool operator>=(const myString&) const;

bool operator<=(const myString&) const;

friend std::ostream& operator<<(std::ostream&, const myString&);

friend std::istream& operator>>(std::istream&, myString&);

operator const char\* () const;

private:

static const int CAPACITY\_MULTIPLIER\_ = 2;

char\* str\_;

std::size\_t size\_, capacity\_;

};

myString.cpp:

#include "myString.h"

#include <cstdlib>

myString::myString()

{

str\_ = new char[1];

\*str\_ = 0;

size\_ = 0;

capacity\_ = 1;

}

myString::myString(const char\* str)

{

std::size\_t temp = 0;

for (temp; str[temp] != '\0'; temp++) {}

temp++;

str\_ = new char[temp];

for (std::size\_t i = 0; i < temp; ++i)

\*(str\_ + i) = \*(str + i);

size\_ = temp-1;

capacity\_ = temp;

}

myString::myString(const myString& s) {

size\_ = s.size\_;

capacity\_ = s.capacity\_;

str\_ = new char[capacity\_];

for (std::size\_t i = 0; i < size\_+1; ++i)

{

\*(str\_ + i) = \*(s.str\_ + i);

}

}

myString& myString::operator=(const myString& s)

{

this->~myString();

size\_ = s.size\_;

capacity\_ = s.capacity\_;

str\_ = new char[capacity\_];

for (std::size\_t i = 0; i < capacity\_; ++i)

{

\*(str\_ + i) = \*(s.str\_ + i);

}

return \*this;

}

myString myString::operator+(const myString rhs) const

{

//std::cout << '\n' << \*this << " + " << rhs << " = ";

myString temp = \*this;

//std::cout << "( " << rhs.size\_ << ' ' << size\_ << "temp:" << temp << " )";

for (size\_t i = 0; i < rhs.size\_; i++) {

temp.push\_back(rhs[i]);

}

return myString(temp);

}

myString::~myString()

{

delete[] str\_;

}

bool myString::empty() const

{

return !size\_;

}

//char\* myString::getString() const

//{

// return str\_;

//}

size\_t myString::length() const

{

return size\_;

}

void myString::push\_back(char c)

{

if (size\_ == capacity\_ - 1)

{

capacity\_ \*= CAPACITY\_MULTIPLIER\_;

char\* temp = new char[capacity\_];

std::memcpy(temp, str\_, size\_+1);

delete[] str\_;

str\_ = temp;

}

str\_[size\_] = c;

size\_++;

str\_[size\_] = 0;

}

bool myString::operator<(const myString& s) const

{

const myString& s1 = \*this;

const myString& s2 = s;

size\_t temp = (s1.size\_ < s2.size\_) ? s1.size\_ : s2.size\_;

for (size\_t i = 0; i < temp; ++i)

{

if (s1.str\_[i] < s2.str\_[i])

{

return true;

}

else if (s1.str\_[i] > s2.str\_[i])

{

return false;

}

}

if (s1.size\_ == s2.size\_)

return false;

else if (temp == s1.size\_)

return true;

return false;

}

bool myString::operator==(const myString& s) const

{

const myString& s1 = \*this;

const myString& s2 = s;

size\_t temp = (s1.size\_ < s2.size\_) ? s1.size\_ : s2.size\_;

for (size\_t i = 0; i < temp; ++i)

{

if (s1.str\_[i] != s2.str\_[i])

{

return false;

}

}

return true;

}

bool myString::operator!=(const myString& s) const {

return !(\*this == s);

}

bool myString::operator>(const myString& s) const

{

const myString& s1 = \*this;

const myString& s2 = s;

return (s1 != s2) && !(s1 < s2);

}

bool myString::operator>=(const myString& s) const

{

return (\*this > s) || (\*this == s);

}

bool myString::operator<=(const myString& s) const

{

return (\*this < s) || (\*this == s);

}

std::ostream& operator<<(std::ostream& os, const myString& s)

{

os << s.str\_;

return os;

}

std::istream& operator>>(std::istream& is, myString& s)

{

char c = 0;

delete[] s.str\_;

s.str\_ = new char[1];

\*s.str\_ = 0;

s.capacity\_ = 1;

s.size\_ = 0;

c = is.peek();

while (c == ' ' || c == '\n' || c == '\t' || c == '\r')

{

is.get(c);

c = is.peek();

}

is.get(c);

while (!(c == ' ' || c == '\n' || c == '\t' || c == '\r'))

{

s.push\_back(c);

is.get(c);

if (is.eof()) break;

}

return is;

}

myString::operator const char\* () const {

return this->str\_;

}

Comparators.cpp:

#include <functional>

#include "Subscriber.h"

enum comparatorTags { COMPARATOR\_SURNAME = 0, COMPARATOR\_NAME, COMPARATOR\_TARIFF, COMPARATOR\_PHONE, COMPARATOR\_KEY };

static std::function<bool(const Subscriber&, const Subscriber&)> compareNames = [](const Subscriber& a, const Subscriber& b)->bool {return a.getName() < b.getName(); };

static std::function<bool(const Subscriber&, const Subscriber&)> compareSurnames = [](const Subscriber& a, const Subscriber& b)->bool {return a.getSurname() < b.getSurname(); };

static std::function<bool(const Subscriber&, const Subscriber&)> compareTariffs = [](const Subscriber& a, const Subscriber& b)->bool {return a.getTariff() < b.getTariff(); };

static std::function<bool(const Subscriber&, const Subscriber&)> comparePhones = [](const Subscriber& a, const Subscriber& b)->bool {return a.getPhone() < b.getPhone(); };

static std::function<bool(const Subscriber&, const Subscriber&)> compareKeys = [](const Subscriber& a, const Subscriber& b)->bool {return (a.getName() + a.getSurname()) < (b.getName() + b.getSurname()); };

static std::function<bool(const Subscriber&, const Subscriber&)> comparators[5] = {compareSurnames, compareNames, compareTariffs, comparePhones, compareKeys};

Файлы тестов:

1.txt:

10

Bhookov C. +1(960)961-11-11 123

Bhookov B. +3(960)961-11-11 124

Chookov A. +2(960)961-11-11 125

Ahookov D. +3(960)961-11-11 025

Bhookov B. +3(960)961-22-22 124

Ehookov D. +5(960)961-11-11 125

Ahookov B. +3(960)961-22-22 025

Bhookov B. +3(960)961-33-33 124

Ohookov K. +7(707)707-77-77 707

Nnn U. +0(010)000-00-00 337

2copy.txt:

13

Angle G. +7(999)255-55-31 102

Michael F. +7(999)255-55-31 103

Artem L. +7(999)255-55-31 101

Lexa G. +7(999)255-55-31 098

Angle G. +7(799)255-55-31 345

Lupa G. +7(999)235-55-31 102

Gari P. +7(989)255-55-31 122

Angle G. +7(999)255-55-11 102

Dichael F. +7(999)255-55-31 103

Aichael F. +7(999)255-55-31 103

Bichael F. +7(999)255-55-31 103

Cichael F. +7(999)255-55-31 103

Eichael F. +7(999)255-55-31 103

3copy.txt:

13

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Mmm U. +0(000)000-00-00 432

Nnn U. +0(009)000-00-00 333

Nnn U. +0(010)000-00-00 337

Nnn U. +0(010)000-00-00 337

4copy.txt:

19

Abb E. +9(909)000-00-00 284

Bab A. +9(000)000-00-00 002

Aab C. +4(000)000-00-00 064

Bba B. +6(000)000-00-00 087

Abb E. +1(000)000-00-00 093

Baa E. +8(000)000-00-00 085

Bab A. +0(000)000-00-00 054

Aaa J. +3(000)000-00-00 009

Bbb D. +2(000)000-00-00 005

Aba I. +7(000)000-00-00 003

Abb E. +5(000)000-00-00 001

Abb E. +8(100)000-00-00 219

Bab A. +4(000)000-00-00 315

Mmm U. +0(000)000-00-00 432

Bab A. +7(908)000-00-00 201

Abb E. +5(000)000-00-00 090

Abb E. +5(000)000-00-00 111

Bab A. +9(000)000-00-00 023

Eee U. +8(234)000-00-00 293

5copy.txt:

10

Bab A. +9(000)000-00-00 002

Aab C. +4(000)000-00-00 064

Bba B. +6(000)000-00-00 087

Aba H. +1(000)000-00-00 093

Baa E. +8(000)000-00-00 085

Bab F. +0(000)000-00-00 054

Aaa J. +3(000)000-00-00 009

Bbb D. +2(000)000-00-00 005

Aba I. +7(000)000-00-00 003

Abb E. +5(000)000-00-00 001

6copy.txt:

3

Angle G. +7(999)255-55-31 102

Michael F. +7(999)245-55-31 103

Artem L. +7(123)222-33-33 303

7copy.txt:

4

Angle G. +7(999)255-55-31 102

Michael F. +7(999)245-55-31 103

Artem L. +7(123)222-33-33 303

8copy.txt:

4

Angle G. +7(999)255-55-31 102

Michael F. +7(999)245-55-31 103

Artem L. +7(123)222-33-33 303

9copy.txt:

1

Angle G. +7(999)255-55-31 102