|  |
| --- |
| **МИНОБРНАУКИ РОССИИ**  **САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  **ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  **«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)**  **Кафедра РЭС** |
| **ОТЧЕТ по лабораторной работе № по дисциплине «Информатика» Тема: Разработка консольного приложения для обработки и экспорта телефонной книги.** |
| |  |  | | --- | --- | | Студент гр. 1183 | Чаминов Д. А. | | Преподаватель | Ситников И. Ю. | |

# Содержание

[Содержание 2](#_Toc86833440)

[Спецификация задания 3](#_Toc86833441)

[Описание алгоритма 4](#_Toc86833442)

[Блок-схема алгоритма 5](#_Toc86833443)

[Выбор и обоснование типов переменных 8](#_Toc86833444)

[Вводимые и выводимые параметры и их типы 9](#_Toc86833445)

[Структура проекта 10](#_Toc86833446)

[Текст программы 11](#_Toc86833447)

[Копии экрана при работе программы 12](#_Toc86833448)

[Контрольный пример, сравнение с эталоном 13](#_Toc86833449)

[Результаты 14](#_Toc86833450)

# Спецификация задания

Разрабатываемая программа должна

* Разрабатываться на языке C++ в среде MS Visual Studio
* Работать на операционных системах MS Windows (7 и выше)
* Поддерживать кириллицу
* Производить следующие операции по запросу пользователя:
  + Ввод данных контактов при помощи клавиатуры
  + Ввод данных контактов из файла
  + Импорт данных контактов из файла, использующего Google csv формат записи контактов
  + Поиск контакта
  + Изменение имени контакта
  + Изменение номера контакта
  + Удаление контакта из телефонной книги
  + Вывод данных контактов на экран в виде таблицы
  + Вывод данных контактов в виде таблицы в файл
  + Экспорт контактов в файл в формате Google csv
* Использовать двоичный поиск
* Сортировать данные в телефонной книге по алфавиту
* Включать в себя по крайней мере 2 пользовательских класса

# Описание алгоритма

При старте программа предлагает пользователю выбрать действие.

При выборе действия «Ввести с клавиатуры», программа должна добавлять записи, создаваемые по вводимым пользователем данным, в телефонную книгу, пока в качестве номера телефона не будет введен «0». Затем пользователь должен подтвердить окончание ввода, в случае если пользователь отказывается, ввод продолжается.

При выборе действия «Ввести из файла», программа должна предложить пользователю ввести имя файла. В случае если файл не удается открыть, программа должна предложить пользователю вернуться к выбору действия, открыть другой файл или же завершить выполнение программы. В случае если файл был открыт успешно, программа должна интерпретировать каждые 2 строки этого файла, как пару – имя контакта и его номер.

При выборе действия «Импортировать (Google csv)», программа для открытия файла повторяет действия, указанные выше, а затем, конвертирует данные, которые в файле разделяются запятыми, в записи телефонной книги. Этот формат подразумевает наличие заголовка с перечислением полей, а затем в каждой строчке – данные контакта. Для получения заголовка и этих данных программа проверяет каждый символ строки на соответствие запятой, кавычкам или символу новой строки. Если текущий символ – запятая, а кавычек в ходе исследования текущего поля не встречалось, то следующий символ принадлежит следующему полю. Если текущий символ – символ перевода каретки, то это последнее поле контакта. Для интерпретации данных в качестве контакта телефонной книги программа ищет среди полей заголовка поле «Name» и поле «Phone 1 - Value». В случае, если для контакта было определено несколько номеров, поле телефонного номера содержит «:::», разделяющее телефонные номера. Программа выделяет из поля номера номера (если требуется) и для каждого из номеров создает запись в телефонной книге, добавляя в имя контакта порядковый номер этого телефонного номера.

Примечание: при импорте программа не сохраняет и не использует дополнительные (не имя или номер) поля контактов.

При выборе действия «Найти номер», в случае если телефонная книга пуста, пользователь получает соответствующее сообщение и возвращается к выбору действия. В ином случае программа получает от пользователя имя контакта. После этого программа начинает поиск контакта по имени: по результатам сравнения искомого имени с серединой отрезка, в котором происходит поиск (сравнение происходит без учета регистра), граница этого отрезка (правая, если искомый элемент расположен раньше по алфавиту или элементы совпадают, и левая в ином случае) устанавливается в середину. Сдвиг границ поиска происходит по этому алгоритму пока не сомкнутся. Тогда правая граница и будет искомым элементом, или ближайшим слева (лексикографически) к искомому элементом. Этот же алгоритм поиска используется при вставке записи в телефонную книгу, обеспечивая таким образом алфавитный порядок записей. После этого программа предлагает пользователю выбор – вернуться к выбору действий, изменить имя контакта, изменить телефонный номер или удалить контакт. Затем в соответствии с выбором пользователя программа изменяет соответствующее поле записи, возвращает пользователя к выбору действия или удаляет контакт.

При выборе действия «Вывести на экран», программа для каждой записи получает имя и номер, вычисляет наибольшую длину имени (сравнивая текущий максимум длины с длиной текущего имени); добавляет справа к строке для имени символы разделителей, пока длина строки меньше максимальной длины имени, и 3 разделительных символа; добавляет разделительные символы к строке номера справа, пока длина номера не достигнет 20 символов; аналогично увеличивает длину строки, содержащей, номер записи в телефонной книге до 4х символов (если требуется); выводит полученные строки на экран в следующем порядке: номер записи, имя контакта, телефонный номер контакта.

При выборе действия «Вывести в файл» программа получает имя файла, в случае если файла с таким именем не существует, он будет создан, но если создание или открытие файла не удалось, то пользователю будут предложены те же действия, что и при неудачном открытии файла при выборе «Ввести с клавиатуры». Затем, таблицу, сформированную по тому же алгоритму, что и для пункта «Вывести на экран» программа записывает в указанный файл. (Если файл с таким именем не существовал, он будет создан)

При выборе действия «Экспортировать (Google csv)» программа открывает файл как при действии «Вывести в файл». В файл в качестве заголовка записываются имена полей имени и номера, а затем для каждой записи в телефонной книге – строковые представления соответствующих полей.

# Блок-схема алгоритма

начало

Phonebook pb

конец

recordFind(pb)

нет

да

action == 7

fileOutput(pb)

export(pb)

нет

да

action == 6

нет

да

action == 5

action == 4

pb.print(0,std::cout,’.’)

да

нет

нет

да

нет

да

нет

да

size\_t action = choose\_list(8, “Ввести с клавиатуры\0Ввести из файла\0Импортировать (Google csv)\0Вывести на экран\0Вывести в файл\0Экспортировать (Google csv)\0Найти номер\0Выйти”);

import(pb)

action == 3

inputFromFile(pb)

action == 2

action == 1

pb.input()

Алгоритм inputFromFile:

Алгоритм Phonebook::inputFromFile:

начало

конец

ifstream fin(filename);  
pb.inputFromFile(fin);  
fin.close();

ввод filename

получить Phonebook& pb

нет

да

fin.good()

PhoneRecord r;

r.input(fin);

addRecord(r);

конец

получить std::istream fin

начало

Алгоритм PhoneRecord::input:

начало

конец

setName(name);

setNumber(number);

std::string name, number;

Получить std::istream & input

ввод name, number

Алгоритм PhoneRecord::setName:

Конец

получить std::string name

Начало

this->name=name

Алгоритм PhoneRecord::setNumber:

нет

да

нет

да

начало

конец

number.plus = (s[0]==’+’);

number.num = std::stoll(s);

s.size()==0

number.num=0;

number.plus=0;

s=filter(s)

filterstr

получение std::string s, bool filterstr=true

Алгоритм PhoneRecord::filter:

нет

нет

да

да

конец

вернуть result

i = i+1;

is\_first=false;

result=result+s[i];

(s[i]>=’0’ И s[i]<=’9’) ИЛИ (is\_first И s[i]==’+’)

i<n

bool is\_first=true;

size\_t n = s.size();

std::string result=””;

size\_t i=0;

получение const std::string& s

начало

Алгоритм Phonebook::addRecord:

конец

phbook.emplace(i, r);

std::list<PhoneRecord>::iterator i = findRecord(r);

получить PhoneRecord r

начало

Алгоритм Phonebook:: FindRecord:

нет

да

да

нет

да

нет

m=(left+right)/2;

конец

right=m;

left=m;

phbook[m]>=r

вернуть phbook.begin()+1+left

phbook.size() == 0

вернуть phbook.begin()

size\_t left=-1,

right = phbook.size();

size\_t m=(left+right)/2;

right-left>1

начало

получить const PhonrRecord& r

Алгоритм import:

конец

std::ifstream fin(filename);  
pb.importPhonebook(fin);

fin.close();

ввод std::strring filename

получить Phonebook& pb

начало

Алгоритм Phonebook::importPhonebook:

нет

нет

да

да

std::string buf;  
getline(fin,buf);

std::vector<std::string> f;

size\_t start\_i=0;

получить std::ifstream fin

начало

конец

fin.good()

addCSVRecord(fin,f);

std::pair<std::string,size\_t> tmp = getCSVField(buf,start\_i);;

f.push\_back(tmp.first);

start\_i=tmp.second;

buf.size()>start\_i

Алгоритм Phonebook::addCVSRecord:

нет

нет

нет

да

да

да

конец

addRecord(name,number)

name = f[i];

i++

number = f[i];

f [i] == ” Phone 1 - Value”

f [i] == ”Name”

i<f.size()

std::string name, number;

size\_t i=0;

std::pair<std::string,size\_t> tmp=getCSVField(buf,start\_i);

v.push\_back(tmp.first);

start\_i=tmp.second;

buf.size()>start\_i

std::string buf;

size\_t start\_i=0;

std::vector<std::string> v;

getline(in,buf);

получить std::istream in, const std::vector<std::string>& f

начало

Алгоритм getCSVField:

нет

нет

да

да

конец

вернуть std::pair<std::string, size\_t>(r,i+)

i++

q = НЕ q

r=r+s[i]

s[i]==’ ” ’

(i<s.size()) И (q ИЛИ (s[i]==’,’))

std::string r =””;

bool q=false;

начало

получить std::string s, size\_t i

Алгоритм fileOutput:Алгоритм Phonebook::print:

конец

std::ofstream fout(filename);

pb.print(0,fout,’.’);

fout.close();

ввод std::string filename

получить const Phonebook& pb

начало

нет

да

конец

(\*i).print(name\_length, output, filler)

output: inflate\_string(std::to\_string(n+1),4)

size\_t n =0;

std::list<PhoneRecord>::iterator i = phbook.begin()

i == phbook.end()

начало

Получить size\_t name\_length, std::ostream & output, char filler

Алгоритм PhoneRecord::print:

начало

std::string name = inflate\_string(nickname, name\_length, StrAlign::left, filler);

std::string n = inflate\_string(getNumber(), 20, StrAlign::right, filler);

output: name, n

конец

Получить size\_t name\_length, std::ostream & output, char filler

Алгоритм PhoneRecord::getNumber:

нет

да

конец

вернуть ‘+’ + std::to\_string(number.num)

вернуть std::to\_string(number.num)

number.plus

начало

Алгоритм inflate\_string:

вернуть s

конец

нет

нет

нет

нет

нет

нет

нет

да

да

да

да

да

да

да

s = inflate\_string(s, (w - n) / 2 + n, StrAlign::left, filler);

s = inflate\_string(s, (w - n) / 2 + (w - n) % 2 + s.size(), StrAlign::right, filler);

s.erase(0, (n - w) / 2);

s.erase(w, (n - w) / 2 + (n - w) % 2);

n>w

n>w

n<w

s.erase(w,n-w);

s=s+filler;

n++;

a==left

s.erase(0,n-w);

n>w

s=filler+s;

n++;

n<w

a==right

size\_t n = s.size();

получить std::string s, size\_t w, StrAlign:: a, char filler

начало

Алгоритм Phonebook::export:

Алгоритм export():Алгоритм recordFind:

нет

да

конец

i++

out<<phbook[i].getName();  
out<<phbook.getNumber();

i<phbook.size()

size\_t i=0

out<<”Name, Phone 1 – value”

получить std::ostream out;

начало

std::ofstream fout(filename);

pb.export(fout);

fout.close()

конец

ввод std::string filename

получить const Phonebook& pb

начало

нет

нет

нет

да

да

да

pb.phbook.erase(rp,1)

ввод size\_t action

action == 1

action == 2

ввод std::string name

rp->setName(name)

action == 3

ввод std::string number

rp->setNumber( number)

rp->setNumber( number)

rp->setName(name)

конец

ввод std::string name

auto rp=pb.findRecord(PhoneRecord(name,”0”,false))

начало

получить const Phonebook& pb

# Выбор и обоснование типов переменных

Для представления телефонной книги используется пользовательский класс Phonebook. Каждый объект этого класса содержит std::vector<PhoneRecord>, шаблонный класс std::vector используется в целях обеспечения произвольного доступа к элементам телефонной книги (для двоичного поиска) и обеспечения автоматического выделения и освобождения памяти.

Для представления записи телефонной книги используется пользовательский класс PhoneRecord. Каждый объект содержит поле типа std::string, в котором записано имя контакта (для поддержки кириллицы используется кодовая страница 1251), анонимную струтуру из двух полей uint64\_t num :50 – числовое представление номера (50 бит хватит для представления 15-значного числа, поскольку ) и uint64\_t plus :1 – индикатор того, содержит ли номер символ “+” в начале.

# Вводимые и выводимые параметры и их типы

Программа получает строки std::string, содержащие число – номер действия, которое выбрал пользователь, в случае ввода с клавиатуры или из файла – std::string (кодовая страница 1251) содержащие имена и номера контактов. Для выбора файла, ввод его имени осуществляется в строку std::string. Во время импорта контактов из файла программа получает строки std::string (кодировка utf-8), содержащие данные полей контактов телефонной книги.

Для вывода телефонной книги используется тот же тип std::string (кодовая страница 1251 для вывода в файл и на экран и utf-8 для экспорта), все сообщения выводимые пользователю являются константными си-строками (const char\*).

# Структура проекта

Данные проекта содержатся в двух директориях.

Содержимое папки import-export

08.12.2021 15:00 7 492 import-export.vcxproj

08.12.2021 15:00 1 570 import-export.vcxproj.filters

01.12.2021 13:26 168 import-export.vcxproj.user

08.12.2021 23:22 10 063 main.cpp

Содержимое папки libs

09.12.2021 00:53 14 502 PhoneNumerCPP.cpp

09.12.2021 01:24 11 603 PhoneNumerCPP.h

08.12.2021 21:30 2 855 utf-8Converting.cpp

08.12.2021 14:54 2 017 utf-8Converting.h

# Текст программы

//main.cpp

#include "../libs/PhoneNumerCPP.h" // for working with phonebook

#include <iostream> // for working with input/output streams

// and console i/o streams

#include <fstream> // for working with file streams

#include <process.h> // for 'system' functions

#include <consoleapi2.h> // for console codepages settings

int main()

{

Phonebook pb; // Phonebook object

SetConsoleCP(1251); // set console input codepage

SetConsoleOutputCP(1251); // set concole output codepage

while (true) { // mainloop (always)

start\_case: // for "go to the start"

std::cout << "Выберите действие:" << std::endl; // prompt for choosing

std::ifstream fin; // file input stream object

std::ofstream fout; // file output stream object

std::string answer; // string for input

switch (choose\_list(8, "Ввести с клавиатуры\0Ввести из файла\0"

"Импортировать (Google csv)\0Вывести на экран\0"

"Вывести в файл\0Экспортировать (Google csv)\0Найти номер\0Выйти")) { // choose action

case 1: // console input

{

std::cout

<< "Для окончания ввода в качестве номера телефона укажите \"0\""

<< std::endl;

// message

pb.input // input phonebook

(

"0",

std::cin, std::cout,

"Введите имя\n>>> ",

"Введите номер\n>>> ",

"Вы действительно хотите закончить ввод?",

"Да\0Нет"

);

std::cout << "Ввод завершен" << std::endl;// action end message

break; // end of action

}

case 2: // file input

{

while (!fin.is\_open()) { // while file is not opened

std::cout << "Введите путь к файлу:" << std::endl

<< ">>> "; // prompt for input

getline(std::cin, answer); // get file name

fin.open(answer); // open file

if (!fin.is\_open()) { // if cannot open

std::cout << "Не удалось открыть файл :(" << std::endl

<< "Выберите действие:" << std::endl;

//message and prompt for choosing

switch (choose\_list(3, "Попробовать снова\0Вернуться к началу\0Завершить выполнение")) {

case 1: // try again

break; // just continue

case 2: // go to the start

goto start\_case; // go to the start

break;

case 3: // exit

return 0; // exit success

}

}

}

pb.inputFromFile(fin); // input phonebook from file

fin.close(); // close file

std::cout << "Ввод завершен" << std::endl; // action end message

break;

}

case 3: // import

{

while (!fin.is\_open()) { // while file is not opened

std::cout << "Введите путь к файлу:" << std::endl

<< ">>> "; // prompt for input

getline(std::cin, answer); // get file name

fin.open(answer); // open file

if (!fin.is\_open()) { // if cannot open

std::cout << "Не удалось открыть файл :(" << std::endl

<< "Выберите действие:" << std::endl;

// message and prompt for choosing action

switch (choose\_list(3, "Попробовать снова\0Вернуться к началу\0Завершить выполнение")) {

case 1: // try again

break; // just continue (loop)

case 2: // go to the start

goto start\_case;// go to the start

break;

case 3: // exit

return 0; // exit success code

break;

}

}

}

pb.importPhonebook(fin); // import phonebook

fin.close(); // close the file

std::cout << "Импорт завершен" << std::endl; // action end message

break; // end of the action

}

case 4: // console output

{

std::cout << "Распечатка телефонной книги:" << std::endl;

//message

pb.print(0, std::cout, '.');// phonebook console output

break; // end of the action

}

case 5: // file output

{

while (!fout.is\_open()) { // while file is not opened

std::cout << "Введите путь к файлу:" << std::endl

<< ">>> "; // prompt for input

getline(std::cin, answer); // input file name

fout.open(answer); // open the file

if (!fout.is\_open()) { // if cannot open the file

std::cout << "Не удалось открыть файл :(" << std::endl

<< "Выберите действие:" << std::endl;

// message and prompt for choosing

switch (choose\_list(3, "Попробовать снова\0Вернуться к началу\0Завершить выполнение")) {

case 1: // try again

break; // just continue (loop)

case 2: // go to the start

goto start\_case;// go to the start

break;

case 3: // exit

return 0; // exit success code

break;

}

}

}

pb.print(0, fout, '.'); // file output

fout.close(); // close the file

std::cout << "Вывод завершен" << std::endl; // action end message

break; // end of the action

}

case 6: // export

{

while (!fout.is\_open()) { // while file is not opened

std::cout << "Введите путь к файлу:" << std::endl

<< ">>> "; // prompt for input

getline(std::cin, answer); // input file name

fout.open(answer); // open the file

if (!fout.is\_open()) { // if cannot open the file

std::cout << "Не удалось открыть файл :(" << std::endl

<< "Выберите действие:" << std::endl;

// message and prompt for choosing

switch (choose\_list(3, "Попробовать снова\0Вернуться к началу\0Завершить выполнение")) {

case 1: // try again

break; // continue (loop)

case 2: // go to the start

goto start\_case; // go to the start

break;

case 3: // exit

return 0; // exit success code

break;

}

}

}

pb.exportPhonebook(fout); // export phonebook

fout.close(); // close the file

std::cout << "Экспорт завершен" << std::endl;

// end of action message

break; // end of the acion

}

case 7: // find

{

if (pb.isEmpty()) { // if phonebook is empty

std::cout << "Телефонная книга пуста" << std::endl;

//message

break; // end of the action

}

std::cout << "Введите имя контакта" << std::endl << ">>> ";

//prompt for input

getline(std::cin, answer); // get contact name

PhoneRecord to\_find; // Phone record with the same name as got

to\_find.setName(answer); // set this name

auto p\_record = pb.findRecord(to\_find);

// place for iterator to the record

if (pb.isTail(p\_record)) { // if it's the .end() of phoneebok

p\_record--;

// set it to the last element

}

std::cout // output and prompt for action choosing

<< "Имя контакта: \"" << p\_record->getName() << "\""

<< std::endl

<< "Номер: " << p\_record->getNumber() << std::endl

<< "Выберите действие:" << std::endl;

switch (choose\_list(4, "Продолжить\0Переименовать\0Изменить номер\0Удалить")) {

case 1: // continue

break; // continue

case 2: // rename

{

std::cout

<< "Введите новое имя контакта" << std::endl << ">>> ";

//prompt for input

getline(std::cin, answer); // get new name

std::cout // prompt for confirmation

<< "Вы действительно хотите сменить имя контакта с \""

<< p\_record->getName()

<< "\" на \"" << answer << "\"?" << std::endl;

if (choose\_list(2, "Да\0Нет") == 1) {//if confirmed

p\_record->setName(answer); // rename

std::cout << "Имя контакта успешно изменено"

<< std::endl; // message

} else {

std::cout << "Изменения отменены"

<< std::endl; // message

}

break;

}

case 3: // change the number

{

std::cout << "Введите новый номер контакта"

<< std::endl << ">>> ";

// prompt for input

getline(std::cin, answer); // input the number

to\_find.setNumber(answer, true);

// filter and remember new number

std::cout // confirmation

<< "Вы действительно хотите сменить номер контакта с \""

<< p\_record->getNumber()

<< "\" на \"" << to\_find.getNumber() << "\"?"

<< std::endl;

if (choose\_list(2, "Да\0Нет") == 1) {//if confirmed

p\_record->setNumber(to\_find.getNumber(), false);

// set new number

std::cout << "Номер контакта успешно изменен"

<< std::endl; // message

} else {

std::cout << "Изменения отменены" << std::endl;

// message

}

break;

}

case 4: // delete

std::cout // confirmation

<< "Вы действительно хотите удалить контакт с именем \""

<< p\_record->getName() << "\"?" << std::endl;

if (choose\_list(2, "Да\0Нет") == 1) {//if confirmed

pb.deleteRecord(p\_record); // delete record

std::cout << "Контакт удален" << std::endl;

// message

} else {

std::cout << "Удаление отменено" << std::endl;

// message

}

break;

}

break;

}

case 8: // exit

{

return 0; // exit success code

}

}

system("pause"); // wait any key pressed

system("cls"); // clear console

}

}

//PhoneNumberCPP.cpp

#include "PhoneNumerCPP.h" // lib header

// Functions:

std::string inflate\_string(std::string s, size\_t w, StrAlign a, char filler)

{

size\_t n = s.size(); // 's' length

switch (a) { // choose align

case StrAlign::left:

if (n > w) { // if 's' is longer 'w' chars

s.erase(w, n - w); // cut it

}

for (; n < w; n++) { // while 's' is shorter 'w' chars

s = s + filler; // add filler

}

break;

case StrAlign::right:

if (n > w) { // if 's' is longer

s.erase(0, n - w); // cut it

}

for (; n < w; n++) { // while 's' is shorter

s = filler + s; // add filler

}

break;

case StrAlign::middle:

if (n > w) { // if 's' longer

s.erase(0, (n - w) / 2); // cut the head

s.erase(w, (n - w) / 2 + (n - w) % 2); // cut the tail

} else {

s = inflate\_string(s, (w - n) / 2 + n, StrAlign::left, filler);

// add fillers to the right

s = inflate\_string(s, (w - n) / 2 + (w - n) % 2 + s.size(), StrAlign::right, filler); // add to the left

}

}

return s; // return result

}

bool did\_user\_accept(std::string question = "Are you sure?")

{

std::string answer; // place for answer

while (1) { // always

std::cout << question << "(y/n)\n>>> "; // prompt for input

getline(std::cin, answer, '\n'); // input

if (answer == "y") { // if user agree

return true; // return true

} else if (answer == "n") { // if user disagree

return false; // return false

} // else repeat

}

}

size\_t choose\_list(uint16\_t n, const char\* list, std::ostream& out, std::istream& in)

{

while (1) { // always

const char\* s = list; // pointer to the first char of the string

for (size\_t i = 0; i < n; i++) { // for natural numbers from 1 to n

out << '\t' << i + 1 << ". " << s << '\n';

// output option with number

s += strlen(s) + 1; // set pointer to the next option start

}

out << ">>> "; // prompt for input

std::string answer; // string for the answer

getline(in, answer); // get choice

uint16\_t answer\_n = atoi(answer.c\_str());

// convert answer from string to the int

if (answer\_n >= 1 && answer\_n <= n) { // if choice is correct

system("cls"); // clear console

return answer\_n; // return choice

}

}

}

// PhoneRecord methods:

PhoneRecord::PhoneRecord() = default; // default constructor

PhoneRecord::PhoneRecord(std::string name, std::string number, bool checknum)

{

setName(name); // set name

setNumber(number, checknum); // set number

}

std::string PhoneRecord::filter(const std::string & s)

{

bool is\_it\_first = true;

std::string result; // place for result

size\_t n = s.size(); // source string size

for (size\_t i = 0; i < n; i++) { // for each character in source

char c = s[i]; // place for character

if (('0' <= c and c <= '9') or (is\_it\_first and c == '+')) {

// if 'c' is good (digit or '+' as the first)

result += c; // add 'c' to result string

is\_it\_first = false; // it's not first

}

}

return result; // return result

}

void PhoneRecord::setNumber(std::string s, bool filterstr)

{

if (filterstr) { //if we need filter string

s = filter(s); // filter string

}

if (s.size()) {

number.plus = (s[0] == '+'); // set plus if it's necessary

// else set no plus

number.num = std::stoll(s); // set number

} else {

number = { 0,0 }; // fill number with 0

}

}

void PhoneRecord::setName(std::string s) { nickname = s; } // set nickname

std::string PhoneRecord::getNumber()const

{

std::string res; // string for result

if (number.plus) { res = "+"; } else { res = ""; } // if plus add '+' to res

res = res + std::to\_string(number.num);

// convert number to string and add it to res

return res; // return res

}

std::string PhoneRecord::getName()const { return nickname; } //return nickname

void PhoneRecord::input(std::istream & input, std::ostream & output, std::string prompt\_name, std::string prompt\_number)

{

std::string name = "", number = ""; // variables for name and number

output << prompt\_name; // prompt for input

while (name == "") {

getline(input, name, '\n'); // input

}

output << prompt\_number; // prompt for input

while (number == "") {

getline(input, number, '\n'); // input

}

setName(name); // set name

setNumber(number); // set number

}

void PhoneRecord::print(size\_t name\_length, std::ostream & output, char filler) const

{

std::string name = inflate\_string(nickname, name\_length, StrAlign::left, filler); // format name

std::string num = inflate\_string(getNumber(), 20, StrAlign::right, filler); // format number

output << name << num << '\n'; // output

}

// There is definition of comparison operators below

// comparison by nickname with standard std::string comparison operators

// NOTA BENE: case-sensitive

bool operator<(const PhoneRecord & a, const PhoneRecord & b) { return a.nickname < b.nickname; }

bool operator>(const PhoneRecord & a, const PhoneRecord & b) { return a.nickname > b.nickname; }

bool operator==(const PhoneRecord & a, const PhoneRecord & b) { return a.nickname == b.nickname; }

bool operator<=(const PhoneRecord & a, const PhoneRecord & b) { return a.nickname <= b.nickname; }

bool operator>=(const PhoneRecord & a, const PhoneRecord & b) { return a.nickname >= b.nickname; }

int8\_t compair(const PhoneRecord & a, const PhoneRecord & b)

{

size\_t al = a.nickname.size(), bl = b.nickname.size();

// characters amount

for (size\_t ai = 0, bi = 0; ai < al && bi < bl; ai++, bi++) {

// for each char

char ac = tolower(a.nickname[ai]), bc = tolower(b.nickname[bi]);

// changing case if it needs

if (ac > bc) {// if current a char is more than current b char

return 1;

} else if (ac < bc) {// if current b char is more than current a char

return -1;

}

}

if (al == bl) { return 0; } // equal strings

if (al < bl) { return -1; } // a is substr of b

return 1; // b is substr of a

}

// Phonebook methods:

void Phonebook::addRecord(std::string name, std::string number, bool checknum)

{

PhoneRecord r(name, number); // create record

std::vector<PhoneRecord>::iterator i = findRecord(r);

//find place for record in the book

phbook.emplace(i, r); // adds new record to the vector of records

}

void Phonebook::addRecord(PhoneRecord r)

{

std::vector<PhoneRecord>::iterator i = findRecord(r);

//find place for record in the book

phbook.emplace(i, r);

// adds new record to the vector of records

}

void Phonebook::inputRecord(std::istream & input, std::ostream & output, std::string prompt\_name, std::string prompt\_number)

{

PhoneRecord r; // place for input

r.input(); // input

addRecord(r); // adding result

}

void Phonebook::input(std::string terminator, std::istream & input, std::ostream & output, std::string prompt\_name, std::string prompt\_number, std::string exit\_q, const char\* yes\_0\_no)

{

PhoneRecord r; // place for result

do {

while (true) {// always

r.input(input, output, prompt\_name, prompt\_number); // input record

if (r.getNumber() == terminator) { break; }

// if it's end od the input ask user if he really wanto to exit

addRecord(r);

// add record to the vector

}

output << exit\_q << std::endl;

} while (choose\_list(2, yes\_0\_no, output) == 2);

// continue if user don't want to exit

}

void Phonebook::inputFromFile(std::ifstream & fin)

{

PhoneRecord r; // buffer PhoneRecord

while (fin.good()) { // while there is no EOF

r.input(fin, std::cout, "", ""); // input record

addRecord(r); // add record to the phonebook

}

}

void Phonebook::input(size\_t amount, std::istream & input, std::ostream & output, std::string prompt\_name, std::string prompt\_number)

{

PhoneRecord r; // place for input

for (size\_t i = 0; i < amount; i++) { // for 'amount' records

r.input(input, output, prompt\_name, prompt\_number); // input

addRecord(r); // add record to the vector

}

}

void Phonebook::print(size\_t name\_length, std::ostream & output, char filler)

{

if (!name\_length) { // if zero

name\_length = maxNameLength() + 3; // use max name leght

}

output << " # " << " name" << inflate\_string("number ", name\_length + 16, StrAlign::right) << '\n'; // table header

size\_t n = 0; // counter

for (std::vector<PhoneRecord>::const\_iterator i = phbook.cbegin(); i != phbook.cend(); i++) { // for each record in records

output << ' ' << inflate\_string(std::to\_string(++n), 4); // print # of element

(\*i).print(name\_length, output, filler); // print table row

}

}

std::vector<PhoneRecord>::iterator Phonebook::findRecord(const PhoneRecord & rec)

{

if (!phbook.size()) { return phbook.begin(); }

// if phonebook is empty - push first

size\_t l = static\_cast<size\_t>(-1ll), r = phbook.size();

// l=0xFFF...F=-1 - left border, r - right border

size\_t m; // middle

while (r - l > 1) { // while element is not found

m = r + l; //

m /= 2; // middle index calculated

if (compair(phbook[m], rec) != -1) { // if searching placed before

r = m; // move right border

} else { // else

l = m; // move left border

}

}

return phbook.begin() + 1 + l; // return iterator to this element

}

size\_t Phonebook::maxNameLength()

{

size\_t mnl = 0; // current max, min string size is 0

for (auto record : phbook) { // for each record in phbook

size\_t tmp = record.getName().size(); // get name length

if (mnl < tmp) { // if current max is less then the record name length

mnl = tmp; // make the current max equal to the record name length

}

}

return mnl; // return result

}

std::pair<std::string, size\_t> getCSVField(const std::string & s, size\_t from, char sep)

{

std::string result = ""; // string for result

bool is\_in\_quotes = false; // is there quote before

size\_t i = from; // index of the first checking char

for (; i < s.size() && (is\_in\_quotes || s[i] != sep); i++) {

// for each char in quotes or in the string or before 'sep' char

if (s[i] == '"') { // if there is the quote here

is\_in\_quotes = !is\_in\_quotes; // switch the flag

continue; // continue

} //else

result = result + s[i]; // add char to result string

}

return std::pair<std::string, size\_t>(result, i + 1);

// pair of the csv field and index of the next one

}

void Phonebook::addCSVRecord(std::istream & stream, const std::vector<std::string>&fields)

{

std::string buf; // buffer string

size\_t start\_i = 0; // index of the current field start

std::vector<std::string> contact\_values; // fields values vector

getline(stream, buf, '\n'); // get contact fields to the buffer

while (buf.size() > start\_i) { // while end of the line is not reached

std::pair<std::string, size\_t> tmp = getCSVField(buf, start\_i);

// get field

contact\_values.push\_back(tmp.first); // add the field

start\_i = tmp.second; // set next field start index

}

if (!contact\_values.size()) { return; } // exit if it is the empty line

std::string name = "", number = ""; // strings for name and number

for (size\_t i = 0; i < fields.size(); i++) { // for each field

if (fields[i] == "Name") { // if it's name field

name = contact\_values[i]; // copy to name string

} else if (fields[i] == "Phone 1 - Value") {//if it's number field

number = contact\_values[i]; // copy to the number string

}

}

size\_t start = 0; // start index in the number string

name = UTFtoString(name); // convert name string according to 1251 codepage

if (number.find(":::") != std::string::npos) { // if more than 1 number defined

while (1) { // always

size\_t i = 1;

size\_t n = number.find(":::", start);// look for the second number

if (n == std::string::npos) { // if there is not more numbers

addRecord(name+" ("+std::to\_string(i++)+")", number.substr(start)); // add record with this number

break; // exit

} // else

addRecord(name, number.substr(start, n));

//split the string and add record with this number

start = n + 3; // next time start with next number

}

} else {

addRecord(name, number); // add record

}

}

void Phonebook::importPhonebook(std::istream & stream)

{

std::string buf; // string bufer

getline(stream, buf, '\n'); // get string

std::vector<std::string> fields; // vector for csv fields

size\_t start\_i = 0; // the field strart index

while (buf.size() > start\_i) { // while it's not the end of the string

std::pair<std::string, size\_t> tmp = getCSVField(buf, start\_i);

// get this field and index of the next one

fields.push\_back(tmp.first); // add the field name to the fields vector

start\_i = tmp.second; // set next field start index

}

while (!stream.eof()) { // while it's not the end of the file

addCSVRecord(stream, fields);

// add record with fields names from vector and

// field values from the .csv stream

}

}

void Phonebook::exportPhonebook(std::ostream & stream)

{

stream << StringToUTF("Name,Phone 1 - Value") << std::endl; // csv header

for (auto r : phbook) { // for each record in phonebook

stream << StringToUTF('"' + r.getName() + "\"," + r.getNumber()) << std::endl; // make csv representation of the record,

// convert it from the 1251 codepage to the UTF-8

// and write it to the stream

}

}

void Phonebook::deleteRecord(std::vector<PhoneRecord>::iterator p\_record)

{

phbook.erase(p\_record); // delete record from the vector by iterator

}

bool Phonebook::isTail(std::vector<PhoneRecord>::iterator p\_record)

{

return p\_record == phbook.end(); // is the iterator the end iterator

}

bool Phonebook::isHead(std::vector<PhoneRecord>::iterator p\_record)

{

return p\_record == phbook.begin();// is it the first element

}

bool Phonebook::isEmpty()

{

return phbook.empty();

}

//PhoneNumberCPP.h

#ifndef PHONE\_NUMBER\_CPP\_INCLUDED

#define PHONE\_NUMBER\_CPP\_INCLUDED

#include <string> // for std::string class and string functions

#include <iostream> // input-output streams lib

#include <vector> // for vector container

#include <fstream> // for file stream

#include <process.h> // for 'system' function

#include "utf-8Converting.h"

// for codepages converting

/// <summary>

/// Types of string align

/// </summary>

enum StrAlign { left, middle, right };

/// <summary>

/// Inflates string. If string is longer than 'w' characters, cuts the string

/// </summary>

/// <param name="s">the string</param>

/// <param name="w">the width of the string</param>

/// <param name="a">- string align</param>

/// <param name="filler">- character for filling space</param>

/// <returns>inflated string</returns>

std::string inflate\_string(std::string s, size\_t w, StrAlign a=StrAlign::left, char filler=' ');

/// <summary>

/// Asks user.

/// Filters input

/// </summary>

/// <param name="question">- the question</param>

/// <returns>Returns true if user agree, else - false</returns>

bool did\_user\_accept(std::string question);

/// <summary>

/// Prints indexed strings list and wait for answer (number from 1 to n)

/// </summary>

/// <param name="n">- number of the list elements.

/// N.B. IF AMOUNT OF LIST ELEMENTS IS LESS THEN n YOU FACE TO UNDEFINED BEHAVIOUR</param>

/// <param name="list">- char\*, elements of the list must be separated by '\0'</param>

/// <param name="out">output stream</param>

/// <param name="in">input stream</param>

/// <returns>choice number</returns>

size\_t choose\_list(uint16\_t n, const char \* list, std::ostream& out=std::cout,std::istream& in = std::cin);

/// <summary>

/// Extracts exactly 1 field from the string

/// </summary>

/// <param name="s">- the string with fields</param>

/// <param name="from">- index of the start of the field</param>

/// <param name="sep">- separation char</param>

/// <returns>pair of the field and index of the next one</returns>

std::pair<std::string, size\_t> getCSVField(const std::string& s, size\_t from, char sep=',');

/// <summary>

/// Class explaning phonebook record

/// </summary>

class PhoneRecord {

protected:

// There is "friending" of comparison operators below

// NOTA BENE: case-sensitive

friend bool operator<(const PhoneRecord& a, const PhoneRecord& b);

friend bool operator>(const PhoneRecord& a, const PhoneRecord& b);

friend bool operator==(const PhoneRecord& a, const PhoneRecord& b);

friend bool operator<=(const PhoneRecord& a, const PhoneRecord& b);

friend bool operator>=(const PhoneRecord& a, const PhoneRecord& b);

/// <summary>

/// case-insensitive record names compairing

/// </summary>

/// <param name="a"></param>

/// <param name="b"></param>

/// <returns>

/// -1 if first record name is more than second,

/// 0 if records names are equal,

/// 1 if second record name is more than first.

/// </returns>

friend int8\_t compair(const PhoneRecord& a, const PhoneRecord& b);

/// <summary>

/// Filters string (excludes all symbols except digits and '+' if it's the first one)

/// </summary>

/// <param name="s">- filtering string</param>

/// <returns>filtered string</returns>

std::string filter(const std::string& s);

/// <summary>

/// String for nickname

/// </summary>

std::string nickname;

/// <summary>

/// struct that explains phone number

/// </summary>

struct {

/// <summary>

/// phone number

/// </summary>

uint64\_t num : 50;

/\*

According to E.164 (standard phone number recording format)

phone numbers should be not longer than 15 digits.

So greatest phone number is: 999 999 999 999 999.

Then 50 bits are enough for recording this number,

because 2^50 = 1 125 899 906 842 624.

\*/

/// <summary>

/// Does number includes plus flag

/// </summary>

uint64\_t plus : 1;

} number; // object of the nameless struct

public:

struct Field {

std::string name;

std::string value;

};

/// <summary>

/// Number getter

/// </summary>

/// <returns>phone number as an std::string</returns>

std::string getNumber() const;

/// <summary>

/// Name getter

/// </summary>

/// <returns>nickname</returns>

std::string getName() const;

/// <summary>

/// Constructor with no parameters

/// </summary>

PhoneRecord();

/// <summary>

/// Constructor with initialization

/// </summary>

/// <param name="name">- nickname</param>

/// <param name="number">- phone number as a std::string</param>

/// <param name="checknum">= true, flag if we need filter string</param>

PhoneRecord(std::string name, std::string number, bool checknum=true);

/// <summary>

/// Inputs phone record from the stream

/// </summary>

/// <param name="input">- input stream</param>

/// <param name="output">- output stream for prompt</param>

/// <param name="prompt\_name">- prompt for input</param>

/// <param name="prompt\_number">- prompt for input</param>

void input(

std::istream& input = std::cin,

std::ostream& output = std::cout,

std::string prompt\_name = "Input nickname\n>>> ",

std::string prompt\_number = "Input number\n>>> ");

/// <summary>

/// Number setter with filtering

/// </summary>

/// <param name="number">- std::string, that includes phone number an any format with misprints</param>

/// <param name="filterstr">= true, flag if we need filter string</param>

void setNumber(std::string number, bool filterstr = true);

/// <summary>

/// Name setter

/// </summary>

/// <param name="name">- new nickname</param>

void setName(std::string name);

/// <summary>

/// Prints formated record

/// </summary>

/// <param name="name\_length">- length of the nickname</param>

/// <param name="output">- output stream</param>

/// <param name="filler">fills space with it</param>

void print(size\_t name\_length, std::ostream& output, char filler = ' ') const;

};

// Compairing operators deprications

[[deprecated("Use int8\_t compair(const PhoneRecord&, const PhoneRecord&) instead of this")]]

bool operator<(const PhoneRecord& a, const PhoneRecord& b);

[[deprecated("Use int8\_t compair(const PhoneRecord&, const PhoneRecord&) instead of this")]]

bool operator>(const PhoneRecord& a, const PhoneRecord& b);

[[deprecated("Use int8\_t compair(const PhoneRecord&, const PhoneRecord&) instead of this")]]

bool operator==(const PhoneRecord& a, const PhoneRecord& b);

[[deprecated("Use int8\_t compair(const PhoneRecord&, const PhoneRecord&) instead of this")]]

bool operator<=(const PhoneRecord& a, const PhoneRecord& b);

[[deprecated("Use int8\_t compair(const PhoneRecord&, const PhoneRecord&) instead of this")]]

bool operator>=(const PhoneRecord& a, const PhoneRecord& b);

/// <summary>

/// Class explaining phonedook

/// </summary>

class Phonebook {

protected:

/// <summary>

/// The member of Standard Template Library,

/// that storages my phone records

/// </summary>

std::vector<PhoneRecord> phbook;

public:

/// <summary>

/// adds record by name and number

/// </summary>

/// <param name="name"></param>

/// <param name="number"></param>

/// <param name="checknum">- should we filter number</param>

void addRecord(std::string name, std::string number, bool checknum = true);

/// <summary>

/// adds the record to the phonebook

/// </summary>

/// <param name="r"></param>

void addRecord(PhoneRecord record);

/// <summary>

/// Uses streams to create and add a new record

/// </summary>

/// <param name="input">stream</param>

/// <param name="output">stream</param>

/// <param name="prompt\_name">prompt for name input</param>

/// <param name="prompt\_number">prompt for number input</param>

void inputRecord(

std::istream& input = std::cin,

std::ostream& output = std::cout,

std::string prompt\_name = "Input nickname\n>>> ",

std::string prompt\_number = "Input number\n>>> ");

/// <summary>

/// Adds elements from inputs stream, until

/// reaches record with number "terminator"

/// </summary>

/// <param name="terminator"></param>

/// <param name="input">stream</param>

/// <param name="output">stream</param>

/// <param name="prompt\_name">prompt for name input</param>

/// <param name="prompt\_number">prompt for number input</param>

void input(

std::string terminator = "0",

std::istream& input = std::cin,

std::ostream& output = std::cout,

std::string prompt\_name = "Input nickname\n>>> ",

std::string prompt\_number = "Input number\n>>> ",

std::string exit\_q="Do you really want to exit?",

const char \* yes\_0\_no = "Yes\0No");

/// <summary>

/// Adds "amount" elements from inputs stream

/// </summary>

/// <param name="amount"></param>

/// <param name="input">stream</param>

/// <param name="output">stream</param>

/// <param name="prompt\_name">prompt for name input</param>

/// <param name="prompt\_number">prompt for number input</param>

void input(

size\_t amount,

std::istream& input = std::cin,

std::ostream& output = std::cout,

std::string prompt\_name = "Input nickname\n>>> ",

std::string prompt\_number = "Input number\n>>> ");

/// <summary>

/// Input records from file till the end of it

/// </summary>

/// <param name="fin">- input file stream</param>

void inputFromFile(std::ifstream & fin);

/// <summary>

/// prints phonebook in the table

/// </summary>

/// <param name="name\_length">- length of the name column, if '0' got, programm use (max\_length\_in\_phonebook + 3)</param>

/// <param name="output">stream</param>

/// <param name="filler">stream</param>

void print(size\_t name\_length = 0, std::ostream& output = std::cout, char filler = ' ');

/// <summary>

/// Binary search

/// Find nearest record (lexicographically) in the book

/// </summary>

/// <param name="r">- record</param>

/// <returns>iterator to the record</returns>

std::vector<PhoneRecord>::iterator findRecord(const PhoneRecord& r);

/// <summary>

/// Finds max name length

/// </summary>

/// <returns>max name lenght</returns>

size\_t maxNameLength();

/// <summary>

/// Adds record with fields name (there must be "Name" and "Phone 1 - Value")

/// with field values from the stream.

/// </summary>

/// <param name="stream"></param>

/// <param name="fields"></param>

void addCSVRecord(std::istream& stream, const std::vector<std::string>& fields);

/// <summary>

/// imports Phonebook from Google csv format using stream

/// </summary>

/// <param name="stream"></param>

void importPhonebook(std::istream& stream);

/// <summary>

/// imports Phonebook to \*idk\* format using stream

/// </summary>

/// <param name="stream"></param>

void exportPhonebook(std::ostream& stream);

/// <summary>

/// Deletes record

/// </summary>

/// <param name="p\_record"> iterator to the record</param>

void deleteRecord(std::vector<PhoneRecord>::iterator p\_record);

/// <summary>

/// Checks if it's the end iterator

/// </summary>

/// <param name="p\_record"></param>

/// <returns></returns>

bool isTail(std::vector<PhoneRecord>::iterator p\_record);

/// <summary>

/// Checks if it's the first element

/// </summary>

/// <param name="p\_record"></param>

/// <returns></returns>

bool isHead(std::vector<PhoneRecord>::iterator p\_record);

/// <summary>

/// Returns if phonebook is empty

/// </summary>

/// <returns></returns>

bool isEmpty();

};

#endif

//utf-8Converting.h

#ifndef MY\_UTF\_8\_CONVERTING\_LIB\_INCLUDED

#define MY\_UTF\_8\_CONVERTING\_LIB\_INCLUDED

#include <Windows.h>

#include <iostream>

/// <summary>

/// Converts from the utf-8 to the 1251 codepage

/// </summary>

/// <param name="str">- input string</param>

/// <param name="n">- input string size, 0 if it's NULL-terminated string</param>

/// <param name="out">- a pointer to the output c-string</param>

/// <param name="fill\_unknown\_with">- char, unknown charc will be filled with in result string</param>

void UTFconvert(const char str[], size\_t n, char\* out, char fill\_unknown\_with = '@');

/// <summary>

/// Convert std::string from utf-8 to cp 1251

/// </summary>

/// <param name="str">- input string</param>

/// <param name="out">- a pointer to the output c-string</param>

/// <param name="fill\_unknown\_with">- char, unknown charc will be filled with in result string</param>

void UTFconvert(const std::string& str, char\* out, char fill\_unknown\_with = '@');

/// <summary>

/// Converts from utf-8 to the cp 1251

/// </summary>

/// <param name="str">input c-string</param>

/// <param name="n">- input string size, 0 if it's NULL-terminated string</param>

/// <param name="fill\_unknown\_with">- char, unknown charc will be filled with in result string</param>

/// <returns>std::string - result</returns>

std::string UTFtoString(const char str[], size\_t n, char fill\_unknown\_with = '@');

/// <summary>

/// Converts from utf-8 to the cp 1251

/// </summary>

/// <param name="str">input string</param>

/// <param name="fill\_unknown\_with">- char, unknown charc will be filled with in result string</param>

/// <returns>std::string - result</returns>

std::string UTFtoString(const std::string& str, char fill\_unknown\_with = '@');

/// <summary>

/// Convert string from cp 1251 to utf-8

/// </summary>

/// <param name="s">input string</param>

/// <returns>std::string as utf-8 string</returns>

std::string StringToUTF(std::string s);

#endif // !MY\_UTF-8\_CONVERTING\_LIB\_INCLUDED

//utf-8Converting.cpp

#include "utf-8Converting.h" //lib header

void UTFconvert(const char str[], size\_t n, char\* out, char fill\_unknown\_with)

{

if (!n) { n = strlen(str); } // calculate string length if n is a NULL

wchar\_t\* wide\_buffer = nullptr; // buffer declaration and allocation

size\_t wn =MultiByteToWideChar(CP\_UTF8, MB\_PRECOMPOSED, str, n, wide\_buffer, 0);

// calculate wide char c-string size

wide\_buffer = new wchar\_t[wn]; // buffer allocate

size\_t wn = MultiByteToWideChar(CP\_UTF8, MB\_PRECOMPOSED, str, n, wide\_buffer, wn);

// convert string from utf-8 to utf-16 and write it to the buffer

WideCharToMultiByte(1251, NULL, wide\_buffer, -1, out, n, &fill\_unknown\_with, NULL);

// convert string from utf-16 to cp1251 and write to the \*out

delete[] wide\_buffer; // free buffer

}

void UTFconvert(const std::string& str, char\* out, char fill\_unknown\_with)

{

UTFconvert(str.c\_str(), str.size(), out, fill\_unknown\_with);

// convert string to the c-string, convert it to the cp1251 and write to the \*out

}

std::string UTFtoString(const char str[], size\_t n, char fill\_unknown\_with)

{

if (!n) { n = strlen(str); } // if n is a NULL, calculate string lenght

char\* buf = new char[n]; // buffer declaration and allocation

UTFconvert(str, n, buf, fill\_unknown\_with);

// convert to the cp 1251 and write it to the buffer

std::string result = buf; // result string

delete[] buf; // free the buffer

return result; // return result

}

std::string UTFtoString(const std::string& str, char fill\_unknown\_with)

{

return UTFtoString(str.c\_str(), str.size(), fill\_unknown\_with);

// convert string to the c-string, convert it to cp 1251,

// create std::string and return it

}

std::string StringToUTF(std::string s)

{

wchar\_t\* wide\_buffer=nullptr; // wchar pointer declaration

size\_t n = MultiByteToWideChar(1251, NULL, s.c\_str(), -1, wide\_buffer, 0);

// calculate size of the c-wstring

wide\_buffer = new wchar\_t[n]; // buffer memory allocation

MultiByteToWideChar(1251, NULL, s.c\_str(), -1, wide\_buffer, n);

// converting from codepage 1251 to utf 16 and writing it

// to the buffer with saving amount of wchars

char\* buffer = nullptr; // char pointer declaration

n=WideCharToMultiByte(CP\_UTF8, NULL, wide\_buffer, n, buffer, 0, NULL, NULL);

buffer = new char[n]; // buffer allocation

WideCharToMultiByte(CP\_UTF8, NULL, wide\_buffer, n, buffer, n, NULL, NULL);

// convert string from utf-16 buffer, to the cp 1251

// and write it to the second buffer

delete[] wide\_buffer; // free utf-16 buffer

std::string result = buffer;

// create std::string and write result c-string to it

delete[] buffer; // free result buffer

return result; // return result std::string

}

# Копии экрана при работе программы

\*скриншоты (лучше win+shift+s в режиме окна)\*

# Контрольный пример, сравнение с эталоном

Ну тут типа информация по прохождению теста / тестов

# Результаты

К результатам работы относятся характеристики разработанной программы, такие, как: • тип ОС, для которой разработано приложение • тип приложения (консольное, оконное…) • путь, имя и размер исполняемого .exe файла • ограничения на исходные данные • результаты проверки выходных значений по независимому источнику • достоинства и недостатки программы