|  |
| --- |
| **МИНОБРНАУКИ РОССИИ**  **САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  **ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  **«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)**  **Кафедра РЭС** |
| **ОТЧЕТ по лабораторной работе №5 по дисциплине «Информатика» Тема: использование классов для создания 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 отвечать следующим требованиям:

* Быть разработана на языке C++ в среде MS Visual Studio.
* Работать в операционных системах Windows 7 и выше.
* Принимать данные контактов, сортировать их и выводить в виде таблицы.
* Содержать хотя бы 2 класса.

# Описание алгоритма

Для хранения данных создается объект класса Phonebook, который содержит в себе всего один объект – контейнер std::list<PhoneRecord> - двусвязный список объектов класса PhoneRecord. Для экземпляров этого класса определена функция поиска места в двусвязном списке для сортировки записей, возвращающая итератор – объект, позволяющий осуществлять доступ к содержимому контейнера std::list<PhoneRecord>, std::list<PhoneRecord>::iterator findPlaceForRecord( const PhoneRecord& r ), функции добавления записи (с созданием по имени и номеру и по самой записи void addRecord), функция позволяющая считать 1 запись с потока ввода с приглашением по потоку вывода и добавить ее в список void inputRecord, функции позволяющие считать произвольное количество записей (до получения записи с номером телефона равным заранее заданному или заданное наперед количество записей) void input, функция распечатки телефонной книги void print.

Двусвязный список – это упорядоченная структура данных, каждый элемент которой обязательно имеет три поля – поле, в котором содержатся хранимые данные, указатель на предыдущий и на следующий элементы. Таким образом std::list является контейнером, обеспечивающим последовательный доступ, и, соответственно, не имеющем реализации случайного доступа.

PhoneRecord является классом, в который инкапсулирована безымянная структура, описывающая телефонный номер. Каждый объект этого класса содержит одну такую структуру number, структура состоит из двух полей: unsigned \_\_int64 number:50 (50 бит хватит для представления 15-значного положительного целого числа) и unsigned \_\_int64 plus:1 – индикатор того был ли в начале номера «+». Также объекты этого класса содержат в себе std::string – объект строки. Для полей записи определены «set’теры» и «get’теры». Также для объектов класса определена функция фильтрации строк std::string filter(const std::string& s), операторы сравнения (которые используют соответствующие операторы сравнение для строк std::string), функция ввода void input(std::istream&, std::ostream&, std::string, std::string), которая инициализирует поля объекта, используя запрос по потоку вывода и поток ввода, конструкторы по умолчанию и по имени и номеру, функция распечатки записи void print.

При старте программа создает объект телефонной книги, вызывает метод ввода для стандартных потоков ввода-вывода через консоль, пока пользователь не введет в качестве телефонного номера «0».

После завершения ввода, при котором, по мере ввода записей, они сортируются, вызывается метод распечатки телефонной книги с максимальной длиной имени контакта 50, через стандартный поток консольного вывода с разделителем ‘.’.

# Блок-схема алгоритма

Конец

system(“pause”);

pb.input(“0”);

pb.print(50,std::cout,’.’)

Phonebook pb;

Начало

Алгоритм Phonebook::input:

да

нет

нет

да

конец

начало

r->input(input, output, prompt\_name. prompt\_number)

PhoneRecord r;

Получить std::string terminator, std::istream & input, std::ostream & output, std::string prompt\_name, std::string prompt\_number

did\_user\_accept()

r->getName() == terminator

addRecord(r)

Алгоритм PhoneRecord::input:

начало

конец

setName(name);

setNumber(number);

std::string name, number;

Получить std::string terminator, std::istream & input, std::ostream & output, std::string prompt\_name, std::string prompt\_number

output: prompt\_name

input: name

output: prompt\_number

input: numbera

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

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 = findPlaceForRecord(r);

получить PhoneRecord r

начало

Алгоритм Phonebook::findPlaceForRecord:

нет

нет

да

да

конец

вернуть i

i ++;

\*i>r

i==phbook.end()

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

начало

получить const PhonrRecord& r

Алгоритм Phonebook::print:

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:

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

# Выбор и обоснование типов переменных

Используется пользовательские классы Phonebook и PhoneRecord.

Phonebook содержит в себе двусвязный список std::list<PhoneRecord>. Использование двусвязного списка обеспечивает быструю вставку в середину телефонной книги, что полезно для используемой сортировки вставками. Случайный доступ к элементам книги не необходим, поэтому двусвязный список подходит для этой задачи.

PhoneRecord содержит в себе строку std::string, которая представляет имя контакта, безымянную структуру, которая содержит два беззнаковых целых – телефонный номер (50 бит достаточно для представления 15-значного числа) и индикатор того, был ли в исходном номере ‘+’ (1 бит).

В качестве счетчиков используется тип «size\_t», являющийся аналогичным «unsigned long long int».

В качестве буферов для считывания с потока ввода и для вывода на поток вывода используются «std::string».

# Вводимые и выводимые параметры и их типы

В программу подаются строки «std::string», содержащие имена и номера контактов телефонной книги, или подтверждение конца ввода.

Программа выводит строки:

* приглашения ввода номера или телефона,
* подтверждение окончания ввода,
* шапку таблицы,
* строки таблицы, состоящие из строки – номера записи, строки – имени контакта и строки – номера телефона.

# Структура проекта

Программа разрабатывается в современной среде разработки Microsoft (r) Visual Studio. Это очень хороший набор программ и вспомогательных средств для профессионального программирования. Процесс написания программы – это написание текста на одном из языков программирования, мы будем использовать языки С и С++. Сам текст является только средством для облегчения восприятия человеком последовательности машинных команд, которые «видит» машина в программе. Поэтому текст с языка С должен быть «переведен» на язык машинных команд, и поскольку этот перевод должен быть не вольным, а точным, процесс называется компиляцией, а выполняет его специальная программа – компилятор. Для нас важно, что тексты в программе – это текстовые файлы на языке С или С++, их имя может быть любым (избегайте использования кириллицы), а расширение .cpp для текста С++ или для классического С .c. Кроме файлов текста, которые обычно содержат тексты выполняемых операций в проекте есть файлы заголовков, которые в основном являются пояснениями компилятору, какие программист использует определения в своих текстах. Файлы заголовков имеют расширение .h от английского headers. Ну, и раз текстов уже несколько, должен быть файл проекта, который содержит в себе как минимум перечень файлов текста, файлов заголовков и еще инструкции в какую форму наш проект должен исходные файлы собрать. Расширение этого файла проекта .vcxproj от Visual C Extended Project. Этап сборки скомпилированных файлов и приводит в случае успеха к созданию исполняемого файла, с расширением .exe (executable). Исполняемый файл не универсален, он строится с учетом особенностей операционной системы и как правило не работает в другой ОС.

# Текст программы

//мой код:

# 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