**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,
СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)**

_____

Факультет Инфокоммуникационных сетей и систем

Кафедра Программной инженерии и вычислительной техники

Отчет по производственной практике

Предприятие: ООО «НТЦ Севентест»

Период прохождения практики: 28.06.21 – 15.07.21

Выполнил Коваленко Леонид Александрович,
ИКПИ-84
(Ф.И.О., № группы)

Подпись студента_____

Руководитель практики от базы практики:

_____
(Ф.И.О., Подпись)

Оценка _____

Руководитель практики от кафедры:

_____
(кафедра, Ф.И.О.)

_____
(Подпись)

Санкт-Петербург

2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Производственная практика — практическая часть учебного процесса подготовки квалифицированных рабочих и специалистов, проходящая, как правило, на различных предприятиях в условиях реального производства. Является заключительной частью учебной практики, проходящей в учебном заведении. Во время производственной практики происходит закрепление и конкретизация результатов теоретического учебно-практического обучения, приобретение студентами умения и навыков практической работы по присваиваемой квалификации и избранной специальности или профессии.

Перенос учебного процесса в условия, максимально схожие с обстановкой будущей профессиональной деятельностью студента, – обязательный этап на пути получения высшего образования, обусловленный требованиями госстандартов и регламентированный приказом Минобразования и науки РФ № 1383.

ЗАДАНИЯ НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ

1. Разработать клиент-серверное приложение с помощью CORBA.
2. Разработать библиотеку для парсинга файлов выписки плана нумерации ФАС.

ВЫПОЛНЕНИЕ ЗАДАНИЙ

Первое задание

IDL — декларативный язык для формального описания интерфейсов взаимодействия клиентов и серверов в распределенных приложениях. IDL не привязан к какому-либо языку программирования, но для большинства современных языков программирования существуют спецификации, определяющие правила трансляции конструкций IDL в конструкции этих языков.

Описание интерфейса для первого задания представлено в таблице 1.

Таблица 1. Описание интерфейса взаимодействия клиента и сервера

Data.idl

```
module Data
{
    interface ServiceA
    {
        string HelloWorld(in string message);
        boolean Div(in long a, in long b, out long r);
        void CaesarCipher(inout string str, in short k);
    };
};
```

Конструкция *module* определяет пространство имен, в котором будут существовать включенные в нее интерфейсы.

Конструкция *interface* определяет программный интерфейс, с помощью которого объекты общаются друг с другом.

В теле конструкции *interface* объявляются операции, которые должен быть способен выполнять сервер по запросу клиента.

В результате компиляции будут созданы 2 файла: *Data.hh* и *DataSK.cc*. Они содержат описание интерфейса CServiceA и вспомогательных классов.

Серверная часть проекта состоит из 5 файлов:

1. *Server/CServiceA.h* и *Server/CServiceA.cpp* (см. таблицы 2-3) содержат реализацию интерфейса *CServiceA* — класс *CServiceA_i*.
2. *Server/CServiceARunner.h* и *Server/CServiceARunner.cpp* (см. таблицы 4-5) содержат класс, используемый для инициализации брокера объектных запросов и запуска сервисов.
3. *Server/Server.cpp* (см. таблицу 6) описывает работу точки входа.

Таблица 2. Файл Server/CServiceA.h

Server/CServiceA.h

```
#ifndef CSERVICEA_H
#define CSERVICEA_H

#include "../Data.hh"

class CServiceA_i : public POA_Data::ServiceA,
                    public PortableServer::RefCountServantBase
{
public:
    CServiceA_i();
    virtual ~CServiceA_i();
    virtual char *HelloWorld(const char *) override;
    virtual CORBA::Boolean Div(CORBA::Long, CORBA::Long, CORBA::Long &) override;
    virtual void CaesarCipher(char *&, CORBA::Short) override;
};

#endif // CSERVICEA_H
```

Таблица 3. Файл Server/CServiceA.cpp

Server/CServiceA.cpp

```
#include "CServiceA.h"
#include <cstring>
#include <iostream>

CServiceA_i::CServiceA_i()
{
}

CServiceA_i::~CServiceA_i()
{
}

char *CServiceA_i::HelloWorld(const char *)
{
    return CORBA::string_dup("Hello, world!");
}

CORBA::Boolean CServiceA_i::Div(CORBA::Long a, CORBA::Long b, CORBA::Long &r)
{
    r = 0;
    if (b == 0)
    {
        r = 0;
        return false; // error
    }
    r = a / b;
    return true; // success
}

void CServiceA_i::CaesarCipher(char *&str, CORBA::Short k)
{
    for (size_t i = 0, size = strlen(str); i < size; ++i)
    {
        bool lower = ('a' <= str[i] && str[i] <= 'z');
        bool upper = ('A' <= str[i] && str[i] <= 'Z');
        if (lower || upper)
        {
            short t = ('a' ^ (upper ? 32 : 0));
            str[i] = static_cast<char>((str[i] - t + k) % ('z' - 'a' + 1) + t);
        }
    }
}
```

Таблица 4. Файл Server/CServiceARunner.h

Server/CServiceARunner.h

```
#ifndef CSERVICEARUNNER_H
#define CSERVICEARUNNER_H

#include "../Data.hh"
#include "CServiceA.h"
class CServiceARunner
{
public:
    CServiceARunner(int argc, char **argv);
    ~CServiceARunner();
    void run();
    CORBA::String_var get_ior();
private:
    // Брокер объектных запросов (Object Request Broker, ORB)
    CORBA::ORB_var m_orb;
    // Сервант
    PortableServer::Servant_var<CServiceA_i> m_c_service_a;
    // Строковая IOR ссылка на объект-сервант
    CORBA::String_var m_ior;
    // POA
    PortableServer::POA_var m_poa;
    // RootPOA
    CORBA::Object_var m_root_poa_obj;
    // POA менеджер
    PortableServer::POAManager_var poa_manager;
    // ObjectID
    PortableServer::ObjectId_var m_c_service_a_oid;
};

#endif // CSERVICEARUNNER_H
```

Таблица 5. Файл Server/CServiceARunner.cpp

Server/CServiceARunner.cpp

```
#include "CServiceARunner.h"
#include "CServiceA.h"
#include "../Data.hh"

#include <utility>
#include <iostream>

CServiceARunner::CServiceARunner(int argc, char **argv)
{
    // Инициализация брокера объектных запросов
    // (Object Request Broker, ORB)
    m_orb = CORBA::ORB_init(argc, argv);
    if (CORBA::is_nil(m_orb))
    {
        throw std::runtime_error("CORBA::ORB_init failed.");
    }
    // Получение доступа к переносимому объектному адаптеру по умолчанию
    // (Root Portable Object Adapter, RootPOA)
    m_root_poa_obj = m_orb->resolve_initial_references("RootPOA");
    if (CORBA::is_nil(m_root_poa_obj))
    {
        throw std::runtime_error("CORBA::ORB->resolve_initial_references(\"RootPOA\") failed.");
    }
    m_poa = PortableServer::POA::_narrow(m_root_poa_obj);
    if (CORBA::is_nil(m_poa))
    {
        throw std::runtime_error("PortableServer::POA::_narrow failed.");
    }
    PortableServer::POAManager_var poa_manager = m_poa->the_POAManager();
    if (CORBA::is_nil(poa_manager))
    {
        throw std::runtime_error("PortableServer::POA->the_POAManager() failed.");
    }
    // Создание серванта
    m_c_service_a = new CServiceA_i();
    // Активация серванта в RootPOA (переносимом объектном адаптере по умолчанию)
```

Server/CServiceARunner.cpp

```
    m_c_service_a_oid = m_poa->activate_object(m_c_service_a);
    // Запоминание строковой IOR ссылки на объект-сервант
    // (Interoperable Object Reference, IOR)
    m_ior = m_orb->object_to_string(m_c_service_a->_this());
    // Получение корневого контекста именования
    CORBA::Object_var name_service_obj = m_orb->resolve_initial_references("NameService");
    if (CORBA::is_nil(name_service_obj))
    {
        throw std::runtime_error("CORBA::ORB->resolve_initial_references(\"NameService\") failed.");
    }
    CosNaming::NamingContext_var naming_context = CosNaming::NamingContext::_narrow(name_service_obj);
    if (CORBA::is_nil(naming_context))
    {
        throw std::runtime_error("CosNaming::NamingContext::_narrow failed.");
    }
    // Привязка к службе имен CORBA
    CosNaming::Name name;
    name.length(1);
    name[0].id = "DataService";
    name[0].kind = "Context";
    naming_context->rebind(name, m_c_service_a->_this());
    // Активация POA Manager
    poa_manager->activate();
}

CServiceARunner::~CServiceARunner()
{
    try
    {
        m_orb->destroy(); // Уничтожение объекта ORB
    }
    catch (...)
    {
        std::cerr << "CORBA::ORB->destroy exception occurred." << std::endl;
    }
}

void CServiceARunner::run()
{
    m_orb->run();
}

CORBA::String_var CServiceARunner::get_ior()
{
    return m_ior;
}
```

Таблица 6. Файл Server/Server.cpp

Server/Server.cpp

```
#include <iostream>

#include "CServiceARunner.h"
#include "../Data.hh"

int main(int argc, char **argv)
{
    CServiceARunner runner(argc, argv);
    std::cout << runner.get_ior() << std::endl;
    runner.run();
    return 0;
}
```

Клиентская часть проекта состоит из 3 файлов:

1. *Client/CRequestServiceA.h* и *Client/CRequestServiceA.cpp* (см. таблицы 7-8) содержат класс, используемый для инициализации брокера объектных запросов и запуска сервисов.
2. *Client/Client.cpp* (см. таблицу 9) описывает работу точки входа.

Таблица 7. Файл Client/CRequestServiceA.h

Client/CRequestServiceA.h

```
#ifndef CREQUESTSERVICEA_H
#define CREQUESTSERVICEA_H

#include <utility>
#include "../Data.hh"

class CRequestServiceA
{
public:
    CRequestServiceA(int argc, char **argv);
    ~CRequestServiceA();
    CORBA::String_var RequestHelloWorld(CORBA::String_var);
    std::pair<CORBA::Long, bool> RequestDiv(CORBA::Long, CORBA::Long);
    CORBA::String_var RequestCaesarCipher(CORBA::String_var, CORBA::Short);

private:
    // Объект, связанный с брокером объектных запросов (Object Request Broker, ORB)
    CORBA::ORB_var m_orb;
    // Объект посредством которого осуществляются запросы
    Data::ServiceA_var m_obj;
};

#endif // CREQUESTSERVICEA_H
```

Таблица 8. Файл Client/CRequestServiceA.cpp

Client/CRequestServiceA.cpp

```
#include <stdexcept>
#include <iostream>
#include <utility>

#include "CRequestServiceA.h"
#include "../Data.hh"

CRequestServiceA::CRequestServiceA(int argc, char **argv)
{
    // Инициализация брокера объектных запросов (Object Request Broker, ORB)
    m_orb = CORBA::ORB_init(argc, argv);
    if (CORBA::is_nil(m_orb))
    {
        throw std::runtime_error("CORBA::ORB_init failed.");
    }
    // Получение корневого контекста именования
    CORBA::Object_var ns_obj = m_orb->resolve_initial_references("NameService");
    if (CORBA::is_nil(ns_obj))
    {
        throw std::runtime_error("CORBA::ORB->resolve_initial_references failed.");
    }
    CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow(ns_obj);
    if (CORBA::is_nil(nc))
    {
        throw std::runtime_error("CosNaming::NamingContext::_narrow failed.");
    }
    // Получение доступа к серверу
    CosNaming::Name name;
    name.length(1);
    name[0].id = "DataService";
```

Client/CRequestServiceA.cpp

```
    name[0].kind = "Context";
    CORBA::Object_var obj = nc->resolve(name);
    if (CORBA::is_nil(obj))
    {
        throw std::runtime_error("CosNaming::NamingContext->resolve failed.");
    }
    // Получение доступа к запрашиваемому объекту
    m_obj = Data::ServiceA::_narrow(obj);
    if (CORBA::is_nil(obj))
    {
        throw std::runtime_error("Data::ServiceA::_narrow failed.");
    }
}

CRequestServiceA::~CRequestServiceA()
{
    try
    {
        m_orb->destroy(); // Уничтожение объекта ORB
    }
    catch (...)
    {
        std::cerr << "Orb destroy exception occurred." << std::endl;
    }
}

CORBA::String_var CRequestServiceA::RequestHelloWorld(CORBA::String_var message_svar)
{
    return m_obj->HelloWorld(message_svar);
}

std::pair<CORBA::Long, bool> CRequestServiceA::RequestDiv(CORBA::Long a, CORBA::Long b)
{
    CORBA::Long c = 0;
    CORBA::Boolean response = m_obj->Div(a, b, c);
    return {c, response};
}

CORBA::String_var CRequestServiceA::RequestCaesarCipher(CORBA::String_var message_svar, CORBA::Short k)
{
    CORBA::String_INOUT_arg str(message_svar);
    m_obj->CaesarCipher(str, k);
    return CORBA::string_dup(str);
}
```

Таблица 9. Файл Client/Client.cpp

Client/Client.cpp

```
#include "CRequestServiceA.h"
#include <iostream>

int main(int argc, char **argv)
{
    // В конструкторе устанавливается связь с сервером CORBA
    CRequestServiceA requestServiceA(argc, argv);
    // Отправка запросов, получение и вывод ответов
    auto responseHelloWorld = requestServiceA.RequestHelloWorld("Hello");
    auto responseDiv = requestServiceA.RequestDiv(10, 5);
    auto responseCaesarCipher = requestServiceA.RequestCaesarCipher("Hello, world!", 3);
    std::cout << "Response (HelloWorld): " << responseHelloWorld << std::endl;
    std::cout << "Response (Div, 10/2): " << responseDiv.first << " (div error: " << !responseDiv.second
<< ')' << std::endl;
    std::cout << "Response (CaesarCipher): " << responseCaesarCipher << std::endl;
    return 0;
}
```

Сборочный файл представлен в табл. 10.

Таблица 10. Сборочный файл Makefile

Makefile

```
.PHONY : all clean_all clean
CC=g++
CPPFLAGS=-std=c++11 -g -Werror -pedantic-errors -Wredundant-decls \
          -Wall -Wextra -Wpedantic -Wcast-align -Wcast-qual \
          -Wconversion -Wctor-dtor-privacy -Wduplicated-branches \
          -Wduplicated-cond -Wextra-semi -Wfloat-equal -Wlogical-op \
          -Wnon-virtual-dtor -Woverloaded-virtual -Wsign-conversion -Wsign-promo
OMNI_HOME=/opt
INCLUDES=-I$(OMNI_HOME)/include
LIBS=-lomniORB4 -lomnithread -lomniDynamic4
SERVER_SOURCES=$(wildcard ./Server/*.cpp)
CLIENT_SOURCES=$(wildcard ./Client/*.cpp)
SERVER_OBJECTS=$(SERVER_SOURCES:.cpp=.o)
CLIENT_OBJECTS=$(CLIENT_SOURCES:.cpp=.o)
SERVER_RUN=ServerRun
CLIENT_RUN=ClientRun

all: $(SERVER_RUN) $(CLIENT_RUN)

$(SERVER_RUN): DataSK.o $(SERVER_OBJECTS)
          $(CC) -g -o $(SERVER_RUN) -L$(OMNI_HOME)/lib DataSK.o $(SERVER_OBJECTS) $(LIBS)

$(SERVER_OBJECTS): DataSK.o $(SERVER_SOURCES)
          $(foreach src,$(SERVER_SOURCES),$(CC) $(CPPFLAGS) -c $(src) -o $(src:.cpp=.o) &&) true

$(CLIENT_RUN): DataSK.o $(CLIENT_OBJECTS)
          $(CC) -g -o $(CLIENT_RUN) -L$(OMNI_HOME)/lib DataSK.o $(CLIENT_OBJECTS) $(LIBS)

$(CLIENT_OBJECTS): DataSK.o $(CLIENT_SOURCES)
          $(foreach src,$(CLIENT_SOURCES),$(CC) $(CPPFLAGS) -c $(src) -o $(src:.cpp=.o) &&) true

DataSK.o: DataSK.cc Data.hh
          $(CC) -c DataSK.cc

DataSK.cc Data.hh: Data.idl
          omniidl -bcxx Data.idl

clean_all: clean
          rm -f ./$(SERVER_RUN)
          rm -f ./$(CLIENT_RUN)

clean:
          rm -f ./*.o
          rm -f ./*.hh
          rm -f ./*SK.cc
          rm -f ./Server/*.o
          rm -f ./Client/*.o
```

После сборки (выполнение команды *make* в терминале) можно запустить сервис *omniNames*[1], затем сервер и клиент (рисунок 1).

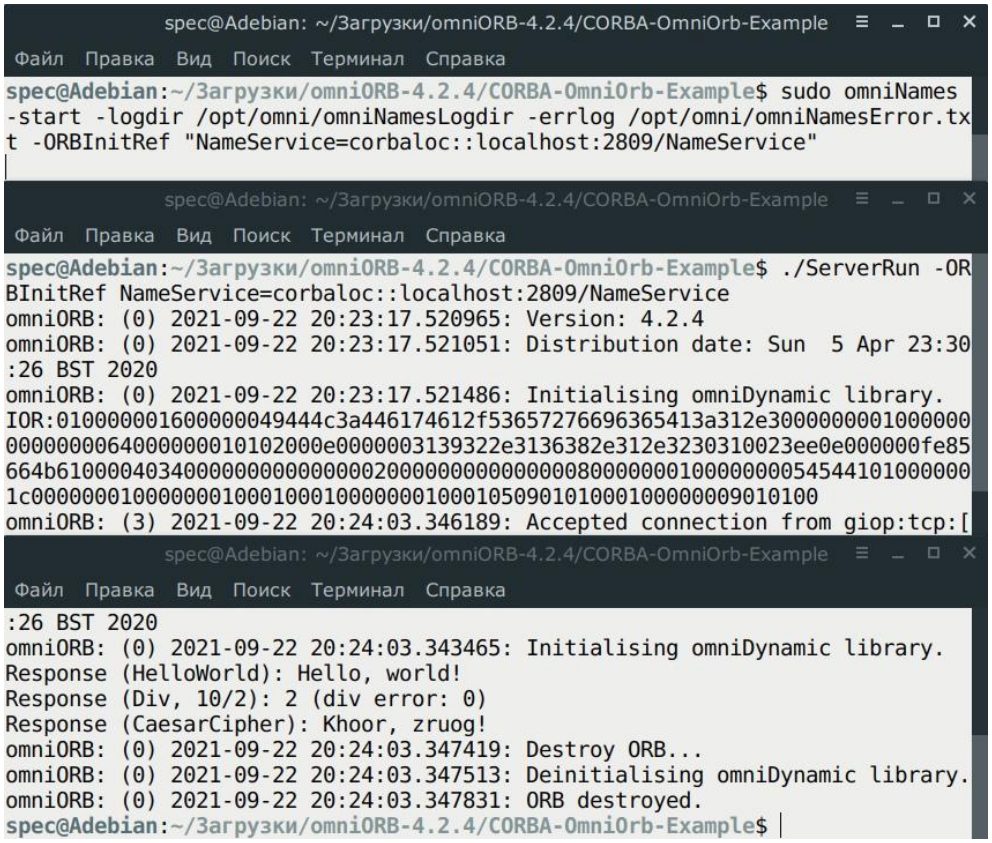

Рисунок 1. Запуск omniNames, сервера и клиента

Response — ответ сервера на запрос клиента:

- *string HelloWorld(in string message)* — клиент отправляет строку message и в ответ получает *string* (строку).
- *boolean Div(in long a, in long b, out long r)* — клиент отправляет числа *a* и *b*, в ответ получает *boolean* (статус деления — есть ошибка или нет) и число *r* (результат деления).
- *void CaesarCipher(inout string str, in short k)* — клиент отправляет строку *str* на редактирование и число *k*. Функция шифрует строку шифром Цезаря на *k* позиций.

[1] omniNames — это служба именования omniORB для CORBA.

Второе задание

Файлы выписки плана нумерации ФАС имеют CSV-формат. Столбцы:

1. Код оператора: 3 цифры;
2. Стартовый номер абонента: 7 цифр;
3. Конечный номер абонента: 7 цифр;
4. Число номеров в диапазоне: целое число;
5. Оператор: строка;
6. Регион: строка.

Пример строки файла: *301;2110000;2129999;20000;ПАО "Ростелеком";г. Улан-Удэ|Республика Бурятия*.

Два наиболее производительных решения (по поиску диапазона по мобильному номеру):

1. Решение с двусторонней очередью и бинарным поиском, поиск за $O(\log_2(n))$, где n — число диапазонов (результат тестирования производительности представлен на рисунке 2).
2. Решение с деревом цифр и минимальным масками диапазонов, поиск за $O(k)$, где k — количество цифр номера телефона (результат тестирования производительности представлен на рисунке 3).

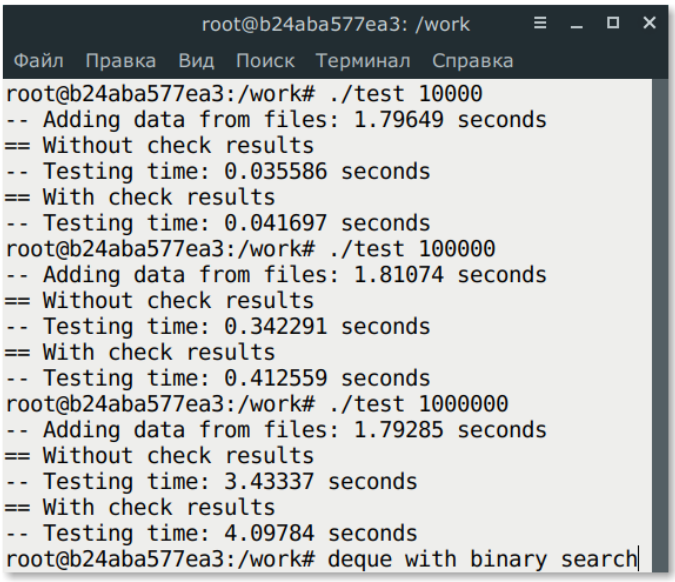

Рисунок 2. Тестирование версии с двусторонней очередью и бинарным поиском

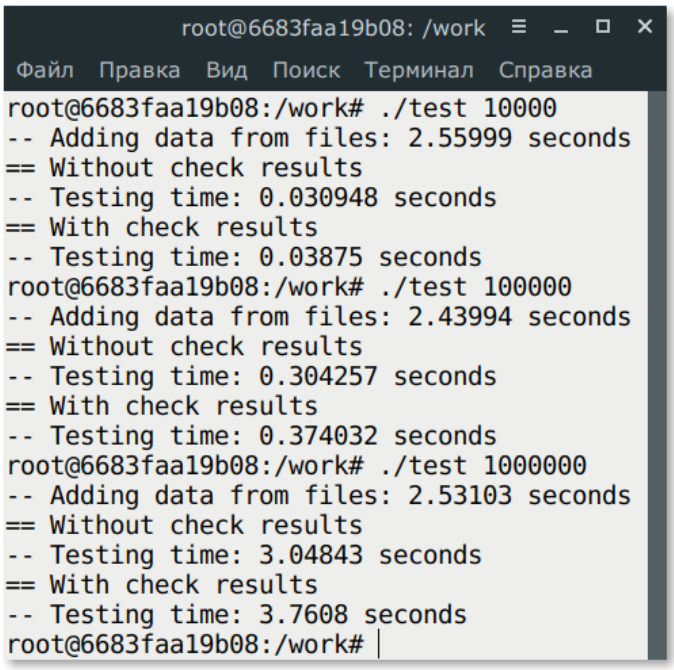

Рисунок 3. Тестирование версии с деревом цифр и минимальным масками диапазонов

Without check results — без проверки номера телефона на корректность.

With check results — с проверкой номера телефона на корректность.

Решение с двусторонней очередью и бинарным поиском состоит из 10 файлов, представленных в таблицах 11-20.

Таблица 11. Файл main.cpp

main.cpp

```cpp
#include <string>
#include <iostream>
#include <fstream>
#include <filesystem>
#include <stdexcept>
#include "lib/mobile_numbers_data.h"

namespace fs = std::filesystem;

std::vector<fs::path> get_files_in_directory(const std::wstring &directory,
                                             std::wstring ext)
{
    std::vector<fs::path> files_vector;
    for (const auto &file : fs::directory_iterator(directory))
    {
        fs::path filepath = file.path();
        if (fs::is_regular_file(filepath) && (ext.empty() || filepath.extension() == ext))
        {
            files_vector.emplace_back(filepath);
        }
    }
    return files_vector;
}

std::wstring replace(const std::wstring &str,
                     const std::wstring &substr,
                     const std::wstring &repl_substr)
{
    size_t pos = str.find(substr);
    if (pos != std::wstring::npos)
```

main.cpp

```
    {
        return str.substr(0, pos) +
               repl_substr +
               str.substr(pos + substr.size(), std::string::npos);
    }
    return str;
}

int main(int argc, char *argv[])
{
    // set locale (required for wstring)
    std::locale::global(std::locale("C.UTF-8"));
    // check args
    if (argc != 2)
    {
        std::wcout << "Use: ./main {mobile_number}" << std::endl;
        return 1;
    }
    // check mobile number
    std::string mobile_number(argv[1]);
    if (!MobileNumbersData::is_mobile_number(mobile_number))
    {
        std::wcout << "Invalid mobile number format." << std::endl;
        return 2;
    }
    // get csv files in current directory
    std::vector<fs::path> files_vector = get_files_in_directory(L"./", L".csv");
    if (files_vector.empty())
    {
        std::wcout << "CSV files not found." << std::endl;
        return 3;
    }
    try
    {
        MobileNumbersData mobile_numbers_data;
        for (const auto &file : files_vector)
        {
            std::wifstream wstream(file);
            if (!wstream.is_open())
            {
                std::wcout << "File \"" << file.wstring() << "\" skipped (open error)." << std::endl;
                continue;
            }
            mobile_numbers_data.add_data(wstream);
        }
        auto range = mobile_numbers_data.find_with_check(mobile_number);
        auto mobile_operator = range.operator_name();
        auto region = range.region();
        region = replace(region, L";", L", ");
        region = replace(region, L"|", L", ");
        std::wcout << mobile_operator << ", " << region << std::endl;
    }
    catch (const std::exception &e)
    {
        std::wcout << e.what() << std::endl;
        return 4;
    }
    return 0;
}
```

Таблица 12. Файл test.cpp

test.cpp

```
#include <string>
#include <iostream>
#include <fstream>
#include <filesystem>
#include <stdexcept>
#include <chrono>
#include "lib/mobile_numbers_data.h"

namespace fs = std::filesystem;
```

test.cpp

```
// Get the current time in seconds
// This function should be used only for measuring time
// (first call, algorithm, second call; then the difference between the values)
#if defined(_WIN32)
#include <Windows.h>
double get_proc_time()
{
    FILETIME createTime, exitTime, kernelTime, userTime;
    if (GetProcessTimes(GetCurrentProcess(), &createTime, &exitTime, &kernelTime, &userTime) != -1)
        return (ULARGE_INTEGER{{userTime.dwLowDateTime, userTime.dwHighDateTime}}).QuadPart /
10000000.L;
    return 0;
}
#else
double get_proc_time()
{
    return static_cast<double>(clock()) / CLOCKS_PER_SEC;
}
#endif

std::vector<fs::path> get_files_in_directory(const std::wstring &directory,
                                             std::wstring ext)
{
    std::vector<fs::path> files_vector;
    for (const auto &file : fs::directory_iterator(directory))
    {
        fs::path filepath = file.path();
        if (fs::is_regular_file(filepath) && (ext.empty() || filepath.extension() == ext))
        {
            files_vector.emplace_back(filepath);
        }
    }
    return files_vector;
}

std::wstring replace(const std::wstring &str,
                     const std::wstring &substr_from,
                     const std::wstring &substr_to)
{
    size_t pos = str.find(substr_from);
    if (pos != std::wstring::npos)
    {
        return str.substr(0, pos) +
               substr_to +
               str.substr(pos + substr_from.size(), std::string::npos);
    }
    return str;
}

int main(int argc, char *argv[])
{
    // set locale (required for wstring)
    std::locale::global(std::locale("C.UTF-8"));
    // check args
    if (argc != 2)
    {
        std::wcout << "Use: ./test {number of tests}" << std::endl;
        return 1;
    }
    int n_tests = 0;
    try
    {
        n_tests = std::atoi(argv[1]);
        if (n_tests <= 0)
        {
            throw std::invalid_argument("Arg {number of tests} is non-positive number.");
        }
    }
    catch (const std::exception &e)
    {
        std::wcout << e.what() << std::endl;
        return 2;
    }
```

test.cpp

```
    // find csv files in current directory
    std::vector<fs::path> files_vector = get_files_in_directory(L"./", L".csv");
    if (files_vector.empty())
    {
        std::wcout << "CSV files not found." << std::endl;
        return 3;
    }
    try
    {
        // start testing
        MobileNumbersData mobile_numbers_data;
        double s = 0, start, finish;
        for (const auto &file : files_vector)
        {
            std::wifstream wstream(file);
            if (!wstream.is_open())
            {
                std::wcout << "File \"" << file.wstring() << "\" skipped (open error)." << std::endl;
                continue;
            }
            start = get_proc_time();
            mobile_numbers_data.add_data(wstream);
            finish = get_proc_time();
            s += finish - start;
        }
        std::wcout << "-- Adding data from files: " << s << " seconds" << std::endl;
        s = 0;
        for (int i = 0; i < n_tests; ++i)
        {
            auto mobile_number = mobile_numbers_data.generate();
            start = get_proc_time();
            auto range = mobile_numbers_data.find_without_check(mobile_number);
            finish = get_proc_time();
            s += finish - start;
        }
        std::wcout << "== Without check results" << std::endl;
        std::wcout << "-- Testing time: " << s << " seconds" << std::endl;
        s = 0;
        for (int i = 0; i < n_tests; ++i)
        {
            auto mobile_number = mobile_numbers_data.generate();
            start = get_proc_time();
            auto range = mobile_numbers_data.find_with_check(mobile_number);
            finish = get_proc_time();
            s += finish - start;
        }
        std::wcout << "== With check results" << std::endl;
        std::wcout << "-- Testing time: " << s << " seconds" << std::endl;
    }
    catch (const std::exception &e)
    {
        std::wcout << e.what() << std::endl;
        return 4;
    }
    return 0;
}
```

Таблица 13. Файл lib/mobile_number_defines.h

lib/mobile_number_defines.h

```
#ifndef MOBILE_NUMBER_DEFINES_H
#define MOBILE_NUMBER_DEFINES_H

#include <cmath>
#include <string>
#include <regex>
#include <random>
#include <functional>

namespace MobileNumberDefines
{
    constexpr size_t COUNTRY_CODE_DIGITS = 1;
```

lib/mobile_number_defines.h

```
    constexpr size_t OPERATOR_CODE_DIGITS = 3;
    constexpr size_t CALLER_CODE_DIGITS = 7;
    constexpr int MAX_COUNTRY_CODE = std::pow(10, COUNTRY_CODE_DIGITS) - 1;
    constexpr int MAX_OPERATOR_CODE = std::pow(10, OPERATOR_CODE_DIGITS) - 1;
    constexpr int MAX_CALLER_CODE = std::pow(10, CALLER_CODE_DIGITS) - 1;

    const std::regex mobile_number_regex("^(?:(\\+?\\d)[\\- ]?)?(?:\\(?(\\d{3})\\)?[\\- ]?)(\\d{3})[\\-
]?(\\d{2})[\\- ]?(\\d{2})$");
    /* Can work with next mobile numbers
        +79811742698
        +7(981)1742698
        +7-(981)-174-26-98
        89811742698
        9811742698
        (981) 174 2698
        (981)-174-2698
        981-174-2698
        981 174 2698
        9811742698
        ...
        country_code -- group #1
        operator_code -- group #2
        caller_code -- groups #3, #4 and #5
    */

    const std::basic_regex<wchar_t>
mobile_numbers_range_regex(L"^(\\d{3});(\\d{7});(\\d{7});[^;]*?;([^;]*);(.*)$");
    /* Can work with next mobile numbers range
        ABC/ DEF;От;До;Емкость;Оператор;Регион
        301;2110000;2129999;20000;ПАО "Ростелеком";г. Улан-Удэ|Республика Бурятия
        301;2150000;2169999;20000;ПАО "Ростелеком";г. Улан-Удэ|Республика Бурятия
        301;2180000;2189999;10000;ПАО "Ростелеком";г. Улан-Удэ|Республика Бурятия
        ...
        operator_code -- group #1
        caller_code_from -- group #2
        caller_code_to -- group #3
        mobile_operator -- group #4
        region -- group #5
    */
}

namespace RandomGen
{
    extern std::random_device random_device_mn;
    extern std::mt19937 mt;
    extern std::uniform_int_distribution<size_t> size_t_distribution;
    extern std::uniform_int_distribution<int> int_distribution;
    extern size_t get_random_size_t_number(size_t, size_t);
    extern int get_random_int_number(int, int);
}

#endif // MOBILE_NUMBER_DEFINES_H
```

Таблица 14. Файл lib/mobile_number_defines.cpp

lib/mobile_number_defines.cpp

```
#include "mobile_number_defines.h"

std::random_device RandomGen::random_device_mn;

std::mt19937 RandomGen::mt = std::mt19937(RandomGen::random_device_mn());

std::uniform_int_distribution<size_t> RandomGen::size_t_distribution(0,
std::numeric_limits<size_t>::max());

std::uniform_int_distribution<int> RandomGen::int_distribution(0, std::numeric_limits<int>::max());

size_t RandomGen::get_random_size_t_number(size_t from, size_t to)
{
    return from + RandomGen::size_t_distribution(RandomGen::mt) % (to - from + 1);
}
```

lib/mobile_number_defines.cpp

```
int RandomGen::get_random_int_number(int from, int to)
{
    return from + RandomGen::int_distribution(RandomGen::mt) % (to - from + 1);
}
```

Таблица 15. Файл lib/mobile_numbers_range.h

lib/mobile_numbers_range.h

```
#ifndef MOBILE_NUMBERS_RANGE_H
#define MOBILE_NUMBERS_RANGE_H

#include <list>
#include <cmath>
#include <string>
#include <algorithm>
#include <stdexcept>
#include "mobile_number_defines.h"

class MobileNumbersRange
{
public:
    MobileNumbersRange();
    MobileNumbersRange(std::string, std::string, std::string, std::wstring, std::wstring);
    MobileNumbersRange(int, int, int, std::wstring, std::wstring);
    // getters
    int operator_code_i() const;
    int caller_code_from_i() const;
    int caller_code_to_i() const;
    std::string operator_code() const;
    std::string caller_code_from() const;
    std::string caller_code_to() const;
    std::wstring operator_name() const;
    std::wstring region() const;
    // setters
    void set_operator_code_i(int);
    void set_caller_code_i(int, int);
    void set_operator_code(const std::string &);
    void set_operator_code(const std::wstring &);
    void set_caller_code(const std::string &, const std::string &);
    void set_caller_code(const std::wstring &, const std::wstring &);
    void set_operator_name(const std::wstring &);
    void set_region(const std::wstring &);
    // check
    int cmp_in_range(long) const;
    friend bool operator<(const MobileNumbersRange &, const MobileNumbersRange &);

private:
    int m_operator_code, m_caller_code_from, m_caller_code_to;
    std::wstring m_operator_name, m_region;
};

#endif // MOBILE_NUMBERS_RANGE_H
```

Таблица 16. Файл lib/mobile_numbers_range.cpp

lib/mobile_numbers_range.cpp

```
#include "mobile_numbers_range.h"

MobileNumbersRange::MobileNumbersRange() {}

MobileNumbersRange::MobileNumbersRange(std::string operator_code,
                                       std::string caller_code_from,
                                       std::string caller_code_to,
                                       std::wstring mobile_operator,
                                       std::wstring region)
{
    set_operator_code(operator_code);
```

```
    set_caller_code(caller_code_from, caller_code_to);
    set_operator_name(mobile_operator);
    set_region(region);
}

MobileNumbersRange::MobileNumbersRange(int operator_code,
                                       int caller_code_from,
                                       int caller_code_to,
                                       std::wstring mobile_operator,
                                       std::wstring region)
{
    set_operator_code_i(operator_code);
    set_caller_code_i(caller_code_from, caller_code_to);
    set_operator_name(mobile_operator);
    set_region(region);
}

// getters

int MobileNumbersRange::operator_code_i() const
{
    return m_operator_code;
}

int MobileNumbersRange::caller_code_from_i() const
{
    return m_caller_code_from;
}

int MobileNumbersRange::caller_code_to_i() const
{
    return m_caller_code_to;
}

std::string MobileNumbersRange::operator_code() const
{
    std::string s = std::to_string(m_operator_code);
    if (s.size() < MobileNumberDefines::OPERATOR_CODE_DIGITS)
    {
        return std::string(MobileNumberDefines::OPERATOR_CODE_DIGITS - s.size(), L'0') + s;
    }
    return s;
}

std::string MobileNumbersRange::caller_code_from() const
{
    std::string s = std::to_string(m_caller_code_from);
    if (s.size() < MobileNumberDefines::CALLER_CODE_DIGITS)
    {
        return std::string(MobileNumberDefines::CALLER_CODE_DIGITS - s.size(), L'0') + s;
    }
    return s;
}

std::string MobileNumbersRange::caller_code_to() const
{
    std::string s = std::to_string(m_caller_code_to);
    if (s.size() < MobileNumberDefines::CALLER_CODE_DIGITS)
    {
        return std::string(MobileNumberDefines::CALLER_CODE_DIGITS - s.size(), L'0') + s;
    }
    return s;
}

std::wstring MobileNumbersRange::operator_name() const
{
    return m_operator_name;
}

std::wstring MobileNumbersRange::region() const
{
    return m_region;
}
```

lib/mobile_numbers_range.cpp

```
// setters

void MobileNumbersRange::set_operator_code_i(int operator_code)
{
    if (operator_code < 0 || operator_code > MobileNumberDefines::MAX_OPERATOR_CODE)
    {
        throw std::invalid_argument("set_operator_code_i -> operator_code = " +
                                    std::to_string(operator_code));
    }
    m_operator_code = operator_code;
}

void MobileNumbersRange::set_caller_code_i(int caller_code_from, int caller_code_to)
{
    if (caller_code_from < 0 || caller_code_from > MobileNumberDefines::MAX_CALLER_CODE)
    {
        throw std::invalid_argument("set_caller_code_i -> caller_code_from = " +
                                    std::to_string(caller_code_from));
    }
    if (caller_code_to < 0 || caller_code_to > MobileNumberDefines::MAX_CALLER_CODE)
    {
        throw std::invalid_argument("set_caller_code_i -> caller_code_to = " +
                                    std::to_string(caller_code_to));
    }
    m_caller_code_from = caller_code_from;
    m_caller_code_to = caller_code_to;
}

void MobileNumbersRange::set_operator_code(const std::wstring &operator_code)
{
    set_operator_code(std::string(operator_code.begin(), operator_code.end()));
}

void MobileNumbersRange::set_operator_code(const std::string &operator_code)
{
    if (operator_code.size() != MobileNumberDefines::OPERATOR_CODE_DIGITS)
    {
        throw std::invalid_argument("set_operator_code -> operator_code.size() != " +
                                    std::to_string(MobileNumberDefines::OPERATOR_CODE_DIGITS));
    }
    auto is_operator_code = std::all_of(operator_code.begin(), operator_code.end(), static_cast<int
(*)(int)>(std::isdigit));
    if (!is_operator_code)
    {
        throw std::invalid_argument("set_operator_code -> operator_code -- invalid argument: " +
                                    std::string(operator_code.begin(), operator_code.end()));
    }
    m_operator_code = std::stoi(operator_code);
}

void MobileNumbersRange::set_caller_code(const std::wstring &caller_code_from, const std::wstring
&caller_code_to)
{
    set_caller_code(std::string(caller_code_from.begin(), caller_code_from.end()),
                    std::string(caller_code_to.begin(), caller_code_to.end()));
}

void MobileNumbersRange::set_caller_code(const std::string &caller_code_from, const std::string
&caller_code_to)
{
    if (caller_code_from.size() != MobileNumberDefines::CALLER_CODE_DIGITS)
    {
        throw std::invalid_argument("set_caller_code -> caller_code_from.size() != " +
                                    std::to_string(MobileNumberDefines::CALLER_CODE_DIGITS));
    }
    if (caller_code_to.size() != MobileNumberDefines::CALLER_CODE_DIGITS)
    {
        throw std::invalid_argument("set_caller_code -> caller_code_to.size() != " +
                                    std::to_string(MobileNumberDefines::CALLER_CODE_DIGITS));
    }
    auto is_caller_code_from = std::all_of(caller_code_from.begin(), caller_code_from.end(),
static_cast<int (*)(int)>(std::isdigit));
    if (!is_caller_code_from)
    {
```

lib/mobile_numbers_range.cpp

```
        throw std::invalid_argument("set_caller_code -> caller_code_from -- invalid argument: " +
                                    std::string(caller_code_from.begin(), caller_code_from.end()));
    }

    auto is_caller_code_to = std::all_of(caller_code_to.begin(), caller_code_to.end(), static_cast<int
(*)(int)>(std::isdigit));
    if (!is_caller_code_to)
    {
        throw std::invalid_argument("set_caller_code -> caller_code_to -- invalid argument: " +
                                    std::string(caller_code_to.begin(), caller_code_to.end()));
    }
    m_caller_code_from = std::stoi(caller_code_from);
    m_caller_code_to = std::stoi(caller_code_to);
}

void MobileNumbersRange::set_operator_name(const std::wstring &mobile_operator)
{
    m_operator_name = mobile_operator;
}

void MobileNumbersRange::set_region(const std::wstring &region)
{
    m_region = region;
}

// check
int MobileNumbersRange::cmp_in_range(long caller_code) const
{
    long t = static_cast<long>(m_operator_code) * (MobileNumberDefines::MAX_CALLER_CODE + 1);
    long from = t + m_caller_code_from;
    if (caller_code < from)
    {
        return -1;
    }
    long to = t + m_caller_code_to;
    if (caller_code > to)
    {
        return 1;
    }
    return 0;
}

bool operator<(const MobileNumbersRange &range1, const MobileNumbersRange &range2)
{
    return range1.m_operator_code < range2.m_operator_code ||
           (range1.m_operator_code == range2.m_operator_code &&
            range1.m_caller_code_from < range2.m_caller_code_from);
}
```

Таблица 17. Файл lib/mobile_numbers_data.h

lib/mobile_numbers_data.h

```
#ifndef MOBILE_NUMBERS_DATA_H
#define MOBILE_NUMBERS_DATA_H

#include <istream>
#include <string>
#include <regex>
#include <deque>
#include <algorithm>
#include <stdexcept>
#include "mobile_numbers_range.h"
#include "mobile_number_defines.h"

class MobileNumbersData
{
public:
    MobileNumbersData();
    void add_data(std::wistream &);
    MobileNumbersRange find_with_check(std::string) const;
    MobileNumbersRange find_without_check(std::string) const;
    std::string generate() const;
```

lib/mobile_numbers_data.h

```
    static bool is_mobile_number(const std::string &);

private:
    std::deque<MobileNumbersRange> m_data;
};

#endif // MOBILE_NUMBERS_DATA_H
```

Таблица 18. Файл lib/mobile_numbers_data.cpp

lib/mobile_numbers_data.cpp

```
#include "mobile_numbers_data.h"

MobileNumbersData::MobileNumbersData()
{
}

void MobileNumbersData::add_data(std::wistream &stream)
{
    bool maybe_header = true;
    MobileNumbersRange range;
    size_t m_data_size_from = m_data.size();
    std::match_results<std::wstring::const_iterator> sm;
    for (std::wstring line; std::getline(stream, line);)
    {
        if (std::regex_match(line, sm, MobileNumberDefines::mobile_numbers_range_regex))
        {
            range.set_operator_code(sm.str(1));
            range.set_caller_code(sm.str(2), sm.str(3));
            range.set_operator_name(sm.str(4));
            range.set_region(sm.str(5));
            m_data.emplace_back(range);
        }
        else
        {
            if (maybe_header && !(L'0' <= line[0] && line[0] <= L'9'))
            {
                maybe_header = false;
            }
            else
            {
                throw std::invalid_argument(
                    std::string("Argument is invalid: ") +
                    std::string(line.begin(), line.end()));
            }
        }
    }
    auto it = m_data.begin();
    std::advance(it, m_data_size_from);
    if (!std::is_sorted(it, m_data.end()))
    {
        std::sort(it, m_data.end());
    }
    std::inplace_merge(m_data.begin(), it, m_data.end());
}

MobileNumbersRange MobileNumbersData::find_with_check(std::string mobile_number) const
{
    std::smatch sm;
    if (!std::regex_match(mobile_number, sm, MobileNumberDefines::mobile_number_regex))
    {
        throw std::out_of_range("Invalid mobile number.");
    }
    return find_without_check(sm.str(2) + sm.str(3) + sm.str(4) + sm.str(5));
}

MobileNumbersRange MobileNumbersData::find_without_check(std::string mobile_number) const
{
    if (m_data.empty())
    {
```

```
        throw std::out_of_range("Data is empty. Use method 'add' before 'find'.");
    }
    size_t left = 0, right = m_data.size() - 1, mid;
    long target = std::stol(mobile_number);
    while (left <= right && right != std::wstring::npos)
    {
        mid = left + ((right - left) >> 1);
        int cmp = m_data[mid].cmp_in_range(target);
        if (cmp == -1)
        {
            right = mid - 1;
        }
        else if (cmp == 1)
        {
            left = mid + 1;
        }
        else
        {
            return m_data[mid];
        }
    }
    throw std::range_error("Not found.");
}

std::string MobileNumbersData::generate() const
{
    MobileNumbersRange range = m_data[RandomGen::get_random_size_t_number(0, m_data.size() - 1)];
    std::string operator_code = range.operator_code();
    std::string caller_code =
std::to_string(RandomGen::get_random_int_number(range.caller_code_from_i(), range.caller_code_to_i()));
    return std::string(MobileNumberDefines::OPERATOR_CODE_DIGITS - operator_code.size(), '0') +
           operator_code +
           std::string(MobileNumberDefines::CALLER_CODE_DIGITS - caller_code.size(), '0') +
           caller_code;
}

bool MobileNumbersData::is_mobile_number(const std::string &arg)
{
    return std::regex_match(arg, MobileNumberDefines::mobile_number_regex);
}
```

Таблица 19. Сборочный файл lib/Makefile

lib/Makefile

```
.PHONY : all clean
CXX=g++
CXXFLAGS=-std=c++17 -O4 -Werror -pedantic-errors -Wredundant-decls \
          -Wall -Wextra -Wpedantic -Wcast-align -Wcast-qual \
          -Wconversion -Wctor-dtor-privacy -Wduplicated-branches \
          -Wduplicated-cond -Wextra-semi -Wfloat-equal -Wlogical-op \
          -Wnon-virtual-dtor -Woverloaded-virtual -Wsign-conversion -Wsign-promo

all: $(patsubst %.cpp, %.o, $(wildcard *.cpp))

%.o: %.cpp %.h Makefile
          $(CXX) $(CXXFLAGS) -c -o $@ $<

clean:
          rm -f *.o
```

Таблица 20. Сборочный файл Makefile

Makefile

```
.PHONY : all clean
CXX=g++
CXXFLAGS=-std=c++17 -O4 -Werror -pedantic-errors -Wredundant-decls \
          -Wall -Wextra -Wpedantic -Wcast-align -Wcast-qual \
          -Wconversion -Wctor-dtor-privacy -Wduplicated-branches \
          -Wduplicated-cond -Wextra-semi -Wfloat-equal -Wlogical-op \
          -Wnon-virtual-dtor -Woverloaded-virtual -Wsign-conversion -Wsign-promo
LIB_HEADERS=$(wildcard lib/*.h)
LIB_SOURCES=$(wildcard lib/*.cpp)
LIB_OBJECTS=$(patsubst %.cpp, %.o, $(LIB_SOURCES))

all: main test

main: $(LIB_OBJECTS) main.o
          $(CXX) $(CXXFLAGS) -o main $(LIB_OBJECTS) main.o

test: $(LIB_OBJECTS) test.o
          $(CXX) $(CXXFLAGS) -o test $(LIB_OBJECTS) test.o

$(LIB_OBJECTS): $(LIB_SOURCES) $(LIB_HEADERS)
          @cd lib && $(MAKE) && cd ../

%.o: %.cpp Makefile
          $(CXX) $(CXXFLAGS) -c -o $@ $<

clean:
          rm -f *.o
```

Решение с деревом цифр и минимальным масками диапазонов состоит из 11 файлов, представленных в таблицах 21-31.

Таблица 21. Файл main.cpp

main.cpp

```
#include <string>
#include <iostream>
#include <fstream>
#include <filesystem>
#include <stdexcept>
#include "lib/mobile_numbers_data.h"

namespace fs = std::filesystem;

std::vector<fs::path> get_files_in_directory(const std::wstring &directory,
                                             std::wstring ext)
{
    std::vector<fs::path> files_vector;
    for (const auto &file : fs::directory_iterator(directory))
    {
        fs::path filepath = file.path();
        if (fs::is_regular_file(filepath) && (ext.empty() || filepath.extension() == ext))
        {
            files_vector.emplace_back(filepath);
        }
    }
    return files_vector;
}

std::wstring replace(const std::wstring &str,
                     const std::wstring &substr,
                     const std::wstring &repl_substr)
{
    size_t pos = str.find(substr);
    if (pos != std::wstring::npos)
    {
        return str.substr(0, pos) +
               repl_substr +
               str.substr(pos + substr.size(), std::string::npos);
```

main.cpp

```
    }
    return str;
}

int main(int argc, char *argv[])
{
    // set locale (required for wstring)
    std::locale::global(std::locale("C.UTF-8"));
    // check args
    if (argc != 2)
    {
        std::wcout << "Use: ./main {mobile_number}" << std::endl;
        return 1;
    }
    // check mobile number
    std::string mobile_number(argv[1]);
    if (!MobileNumbersData::is_mobile_number(mobile_number))
    {
        std::wcout << "Invalid mobile number format." << std::endl;
        return 2;
    }
    // get csv files in current directory
    std::vector<fs::path> files_vector = get_files_in_directory(L"./", L".csv");
    if (files_vector.empty())
    {
        std::wcout << "CSV files not found." << std::endl;
        return 3;
    }
    try
    {
        MobileNumbersData mobile_numbers_data;
        for (const auto &file : files_vector)
        {
            std::wifstream wstream(file);
            if (!wstream.is_open())
            {
                std::wcout << "File \"" << file.wstring() << "\" skipped (open error)." << std::endl;
                continue;
            }
            mobile_numbers_data.add_data(wstream);
        }
        auto range = mobile_numbers_data.find_with_check(mobile_number);
        auto mobile_operator = range.operator_name();
        auto region = range.region();
        region = replace(region, L";", L", ");
        region = replace(region, L"|", L", ");
        std::wcout << mobile_operator << ", " << region << std::endl;
    }
    catch (const std::exception &e)
    {
        std::wcout << e.what() << std::endl;
        return 4;
    }
    return 0;
}
```

Таблица 22. Файл test.cpp

test.cpp

```
#include <string>
#include <iostream>
#include <fstream>
#include <filesystem>
#include <stdexcept>
#include <chrono>
#include "lib/mobile_numbers_data.h"

namespace fs = std::filesystem;

// Get the current time in seconds
// This function should be used only for measuring time
// (first call, algorithm, second call; then the difference between the values)
```

test.cpp

```
#if defined(_WIN32)
#include <Windows.h>
double get_proc_time()
{
    FILETIME createTime, exitTime, kernelTime, userTime;
    if (GetProcessTimes(GetCurrentProcess(), &createTime, &exitTime, &kernelTime, &userTime) != -1)
        return (ULARGE_INTEGER{{userTime.dwLowDateTime, userTime.dwHighDateTime}}).QuadPart /
10000000.L;
    return 0;
}
#else
double get_proc_time()
{
    return static_cast<double>(clock()) / CLOCKS_PER_SEC;
}
#endif

std::vector<fs::path> get_files_in_directory(const std::wstring &directory,
                                             std::wstring ext)
{
    std::vector<fs::path> files_vector;
    for (const auto &file : fs::directory_iterator(directory))
    {
        fs::path filepath = file.path();
        if (fs::is_regular_file(filepath) && (ext.empty() || filepath.extension() == ext))
        {
            files_vector.emplace_back(filepath);
        }
    }
    return files_vector;
}

std::wstring replace(const std::wstring &str,
                     const std::wstring &substr_from,
                     const std::wstring &substr_to)
{
    size_t pos = str.find(substr_from);
    if (pos != std::wstring::npos)
    {
        return str.substr(0, pos) +
               substr_to +
               str.substr(pos + substr_from.size(), std::string::npos);
    }
    return str;
}

int main(int argc, char *argv[])
{
    // set locale (required for wstring)
    std::locale::global(std::locale("C.UTF-8"));
    // check args
    if (argc != 2)
    {
        std::wcout << "Use: ./test {number of tests}" << std::endl;
        return 1;
    }
    int n_tests = 0;
    try
    {
        n_tests = std::atoi(argv[1]);
        if (n_tests <= 0)
        {
            throw std::invalid_argument("Arg {number of tests} is non-positive number.");
        }
    }
    catch (const std::exception &e)
    {
        std::wcout << e.what() << std::endl;
        return 2;
    }
    // find csv files in current directory
    std::vector<fs::path> files_vector = get_files_in_directory(L"./", L".csv");
    if (files_vector.empty())
    {
```

test.cpp

```
        std::wcout << "CSV files not found." << std::endl;
        return 3;
    }
    try
    {
        // start testing
        MobileNumbersData mobile_numbers_data;
        double s = 0, start, finish;
        for (const auto &file : files_vector)
        {
            std::wifstream wstream(file);
            if (!wstream.is_open())
            {
                std::wcout << "File \"" << file.wstring() << "\" skipped (open error)." << std::endl;
                continue;
            }
            start = get_proc_time();
            mobile_numbers_data.add_data(wstream);
            finish = get_proc_time();
            s += finish - start;
        }
        std::wcout << "-- Adding data from files: " << s << " seconds" << std::endl;
        s = 0;
        for (int i = 0; i < n_tests; ++i)
        {
            auto mobile_number = mobile_numbers_data.generate();
            start = get_proc_time();
            auto range = mobile_numbers_data.find_without_check(mobile_number);
            finish = get_proc_time();
            s += finish - start;
        }
        std::wcout << "== Without check results" << std::endl;
        std::wcout << "-- Testing time: " << s << " seconds" << std::endl;
        s = 0;
        for (int i = 0; i < n_tests; ++i)
        {
            auto mobile_number = mobile_numbers_data.generate();
            start = get_proc_time();
            auto range = mobile_numbers_data.find_with_check(mobile_number);
            finish = get_proc_time();
            s += finish - start;
        }
        std::wcout << "== With check results" << std::endl;
        std::wcout << "-- Testing time: " << s << " seconds" << std::endl;
    }
    catch (const std::exception &e)
    {
        std::wcout << e.what() << std::endl;
        return 4;
    }
    return 0;
}
```

Таблица 23. Файл lib/digits_tree.h

lib/digits_tree.h

```
#ifndef DIGITS_TREE_H
#define DIGITS_TREE_H

#include <cstdint>
#include <memory>
#include <cstdlib>
#include <stdexcept>
#include <deque>
#include "mobile_number_defines.h"
#include "mobile_numbers_range.h"

template <size_t D = 10>
class DigitsTree
{
public:
    DigitsTree() : m_size(0)
```

```
    {
    }

    void push(const std::string &str, const MobileNumbersRange &obj)
    {
        ++m_size;
        if (str.empty())
        {
            if (!m_current)
            {
                m_current = std::make_unique<MobileNumbersRange>(obj);
                return;
            }
            else
            {
                throw std::invalid_argument("push -> obj already exists");
            }
            m_current = std::make_unique<MobileNumbersRange>(obj);
        }
        char first_digit = str[0];
        if ('0' <= first_digit && first_digit <= '9')
        {
            size_t index = static_cast<size_t>(first_digit - '0');
            if (!m_ranges[index])
            {
                m_ranges[index] = std::make_unique<DigitsTree>();
            }
            m_ranges[index]->push(str.substr(1), obj);
        }
        else
        {
            if (!m_current)
            {
                m_current = std::make_unique<MobileNumbersRange>(obj);
            }
            else
            {
                throw std::invalid_argument("push -> obj already exists");
            }
        }
    }

    MobileNumbersRange find(const std::string &str, size_t shift_ = 0) const
    {
        if (empty())
        {
            std::out_of_range("Data is empty. Use method 'push' before 'find'.");
        }
        if (m_current)
        {
            return *m_current;
        }
        if (str.empty())
        {
            throw std::invalid_argument("find -> str is empty");
        }
        if (shift_ >= str.size())
        {
            throw std::invalid_argument("find -> shift >= str.size()");
        }
        if (!('0' <= str[shift_] && str[shift_] <= '9'))
        {
            throw std::invalid_argument("find -> str[shift_] not in [0, 9]");
        }
        size_t index = static_cast<size_t>(str[shift_] - '0');
        if (m_ranges[index])
        {
            return m_ranges[index]->find(str, shift_ + 1);
        }
        throw std::out_of_range("Not found.");
    }

    inline bool empty() const
    {
```

lib/digits_tree.h

```
        return m_size == 0;
    }

    MobileNumbersRange get_random() const
    {
        if (empty())
        {
            std::out_of_range("Data is empty. Use method 'add' before 'find'.");
        }
        if (!m_current)
        {
            std::deque<size_t> d;
            for (size_t i = 0; i < D; ++i)
            {
                if (m_ranges[i])
                {
                    d.push_back(i);
                }
            }
            if (!d.empty())
            {
                return m_ranges[d[RandomGen::get_random_size_t_number(0, d.size() - 1)]]->get_random();
            }
            else
            {
                std::invalid_argument("get_random -> something is wrong");
            }
        }
        return *m_current;
    }

    size_t size() const
    {
        return m_size;
    }

private:
    std::unique_ptr<DigitsTree> m_ranges[D];
    std::unique_ptr<MobileNumbersRange> m_current;
    size_t m_size;
};

#endif // DIGITS_TREE_H
```

Таблица 24. Файл lib/mobile_number_defines.h

lib/mobile_number_defines.h

```
#ifndef MOBILE_NUMBER_DEFINES_H
#define MOBILE_NUMBER_DEFINES_H

#include <cmath>
#include <string>
#include <regex>
#include <random>
#include <functional>

namespace MobileNumberDefines
{
    constexpr size_t COUNTRY_CODE_DIGITS = 1;
    constexpr size_t OPERATOR_CODE_DIGITS = 3;
    constexpr size_t CALLER_CODE_DIGITS = 7;
    constexpr int MAX_COUNTRY_CODE = std::pow(10, COUNTRY_CODE_DIGITS) - 1;
    constexpr int MAX_OPERATOR_CODE = std::pow(10, OPERATOR_CODE_DIGITS) - 1;
    constexpr int MAX_CALLER_CODE = std::pow(10, CALLER_CODE_DIGITS) - 1;

    const std::regex mobile_number_regex("^(?:(\\+?\\d)[\\- ]?)?(?:\\(?(\\d{3})\\)?[\\- ]?)(\\d{3})[\\-
]?(\\d{2})[\\- ]?(\\d{2})$");
    /* Can work with next mobile numbers
        +79811742698
        +7(981)1742698
        +7-(981)-174-26-98
        89811742698
```

lib/mobile_number_defines.h

```
        9811742698
        (981) 174 2698
        (981)-174-2698
        981-174-2698
        981 174 2698
        9811742698
        ...
        country_code -- group #1
        operator_code -- group #2
        caller_code -- groups #3, #4 and #5
    */

    const std::basic_regex<wchar_t>
mobile_numbers_range_regex(L"^(\\d{3});(\\d{7});(\\d{7});[^;]*?;([^;]*);(.*)$");
    /* Can work with next mobile numbers range
        ABC/ DEF;От;До;Емкость;Оператор;Регион
        301;2110000;2129999;20000;ПАО "Ростелеком";г. Улан-Удэ|Республика Бурятия
        301;2150000;2169999;20000;ПАО "Ростелеком";г. Улан-Удэ|Республика Бурятия
        301;2180000;2189999;10000;ПАО "Ростелеком";г. Улан-Удэ|Республика Бурятия
        ...
        operator_code -- group #1
        caller_code_from -- group #2
        caller_code_to -- group #3
        mobile_operator -- group #4
        region -- group #5
    */
}

namespace RandomGen
{
    extern std::random_device random_device_mn;
    extern std::mt19937 mt;
    extern std::uniform_int_distribution<size_t> size_t_distribution;
    extern std::uniform_int_distribution<int> int_distribution;
    extern size_t get_random_size_t_number(size_t, size_t);
    extern int get_random_int_number(int, int);
}

#endif // MOBILE_NUMBER_DEFINES_H
```

Таблица 25. Файл lib/mobile_number_defines.cpp

lib/mobile_number_defines.cpp

```
#include "mobile_number_defines.h"

std::random_device RandomGen::random_device_mn;

std::mt19937 RandomGen::mt = std::mt19937(RandomGen::random_device_mn());

std::uniform_int_distribution<size_t> RandomGen::size_t_distribution(0,
std::numeric_limits<size_t>::max());

std::uniform_int_distribution<int> RandomGen::int_distribution(0, std::numeric_limits<int>::max());

size_t RandomGen::get_random_size_t_number(size_t from, size_t to)
{
    return from + RandomGen::size_t_distribution(RandomGen::mt) % (to - from + 1);
}

int RandomGen::get_random_int_number(int from, int to)
{
    return from + RandomGen::int_distribution(RandomGen::mt) % (to - from + 1);
}
```

Таблица 26. Файл lib/mobile_numbers_range.h

lib/mobile_numbers_range.h

```
#ifndef MOBILE_NUMBERS_RANGE_H
#define MOBILE_NUMBERS_RANGE_H

#include <list>
#include <cmath>
#include <string>
#include <algorithm>
#include <stdexcept>
#include "mobile_number_defines.h"

class MobileNumbersRange
{
public:
    MobileNumbersRange();
    MobileNumbersRange(std::string, std::string, std::string, std::wstring, std::wstring);
    MobileNumbersRange(int, int, int, std::wstring, std::wstring);
    // getters
    int operator_code_i() const;
    int caller_code_from_i() const;
    int caller_code_to_i() const;
    std::string operator_code() const;
    std::string caller_code_from() const;
    std::string caller_code_to() const;
    std::wstring operator_name() const;
    std::wstring region() const;
    // setters
    void set_operator_code_i(int);
    void set_caller_code_i(int, int);
    void set_operator_code(const std::string &);
    void set_operator_code(const std::wstring &);
    void set_caller_code(const std::string &, const std::string &);
    void set_caller_code(const std::wstring &, const std::wstring &);
    void set_operator_name(const std::wstring &);
    void set_region(const std::wstring &);
    std::list<std::string> split();

private:
    int m_operator_code, m_caller_code_from, m_caller_code_to;
    std::wstring m_operator_name, m_region;
};

#endif // MOBILE_NUMBERS_RANGE_H
```

Таблица 27. Файл lib/mobile_numbers_range.cpp

lib/mobile_numbers_range.cpp

```
#include "mobile_numbers_range.h"

MobileNumbersRange::MobileNumbersRange() {}

MobileNumbersRange::MobileNumbersRange(std::string operator_code,
                                       std::string caller_code_from,
                                       std::string caller_code_to,
                                       std::wstring mobile_operator,
                                       std::wstring region)
{
    set_operator_code(operator_code);
    set_caller_code(caller_code_from, caller_code_to);
    set_operator_name(mobile_operator);
    set_region(region);
}

MobileNumbersRange::MobileNumbersRange(int operator_code,
                                       int caller_code_from,
                                       int caller_code_to,
                                       std::wstring mobile_operator,
                                       std::wstring region)
{
    set_operator_code_i(operator_code);
    set_caller_code_i(caller_code_from, caller_code_to);
```

```
    set_operator_name(mobile_operator);
    set_region(region);
}

// getters

int MobileNumbersRange::operator_code_i() const
{
    return m_operator_code;
}

int MobileNumbersRange::caller_code_from_i() const
{
    return m_caller_code_from;
}

int MobileNumbersRange::caller_code_to_i() const
{
    return m_caller_code_to;
}

std::string MobileNumbersRange::operator_code() const
{
    std::string s = std::to_string(m_operator_code);
    if (s.size() < MobileNumberDefines::OPERATOR_CODE_DIGITS)
    {
        return std::string(MobileNumberDefines::OPERATOR_CODE_DIGITS - s.size(), L'0') + s;
    }
    return s;
}

std::string MobileNumbersRange::caller_code_from() const
{
    std::string s = std::to_string(m_caller_code_from);
    if (s.size() < MobileNumberDefines::CALLER_CODE_DIGITS)
    {
        return std::string(MobileNumberDefines::CALLER_CODE_DIGITS - s.size(), L'0') + s;
    }
    return s;
}

std::string MobileNumbersRange::caller_code_to() const
{
    std::string s = std::to_string(m_caller_code_to);
    if (s.size() < MobileNumberDefines::CALLER_CODE_DIGITS)
    {
        return std::string(MobileNumberDefines::CALLER_CODE_DIGITS - s.size(), L'0') + s;
    }
    return s;
}

std::wstring MobileNumbersRange::operator_name() const
{
    return m_operator_name;
}

std::wstring MobileNumbersRange::region() const
{
    return m_region;
}

// setters

void MobileNumbersRange::set_operator_code_i(int operator_code)
{
    if (operator_code < 0 || operator_code > MobileNumberDefines::MAX_OPERATOR_CODE)
    {
        throw std::invalid_argument("set_operator_code_i -> operator_code = " +
                                    std::to_string(operator_code));
    }
    m_operator_code = operator_code;
}

void MobileNumbersRange::set_caller_code_i(int caller_code_from, int caller_code_to)
```

```
{
    if (caller_code_from < 0 || caller_code_from > MobileNumberDefines::MAX_CALLER_CODE)
    {
        throw std::invalid_argument("set_caller_code_i -> caller_code_from = " +
                                    std::to_string(caller_code_from));
    }
    if (caller_code_to < 0 || caller_code_to > MobileNumberDefines::MAX_CALLER_CODE)
    {
        throw std::invalid_argument("set_caller_code_i -> caller_code_to = " +
                                    std::to_string(caller_code_to));
    }
    m_caller_code_from = caller_code_from;
    m_caller_code_to = caller_code_to;
}

void MobileNumbersRange::set_operator_code(const std::wstring &operator_code)
{
    set_operator_code(std::string(operator_code.begin(), operator_code.end()));
}

void MobileNumbersRange::set_operator_code(const std::string &operator_code)
{
    if (operator_code.size() != MobileNumberDefines::OPERATOR_CODE_DIGITS)
    {
        throw std::invalid_argument("set_operator_code -> operator_code.size() != " +
                                    std::to_string(MobileNumberDefines::OPERATOR_CODE_DIGITS));
    }
    auto is_operator_code = std::all_of(operator_code.begin(), operator_code.end(), static_cast<int
(*)(int)>(std::isdigit));
    if (!is_operator_code)
    {
        throw std::invalid_argument("set_operator_code -> operator_code -- invalid argument: " +
                                    std::string(operator_code.begin(), operator_code.end()));
    }
    m_operator_code = std::stoi(operator_code);
}

void MobileNumbersRange::set_caller_code(const std::wstring &caller_code_from, const std::wstring
&caller_code_to)
{
    set_caller_code(std::string(caller_code_from.begin(), caller_code_from.end()),
                    std::string(caller_code_to.begin(), caller_code_to.end()));
}

void MobileNumbersRange::set_caller_code(const std::string &caller_code_from, const std::string
&caller_code_to)
{
    if (caller_code_from.size() != MobileNumberDefines::CALLER_CODE_DIGITS)
    {
        throw std::invalid_argument("set_caller_code -> caller_code_from.size() != " +
                                    std::to_string(MobileNumberDefines::CALLER_CODE_DIGITS));
    }
    if (caller_code_to.size() != MobileNumberDefines::CALLER_CODE_DIGITS)
    {
        throw std::invalid_argument("set_caller_code -> caller_code_to.size() != " +
                                    std::to_string(MobileNumberDefines::CALLER_CODE_DIGITS));
    }
    auto is_caller_code_from = std::all_of(caller_code_from.begin(), caller_code_from.end(),
static_cast<int (*)(int)>(std::isdigit));
    if (!is_caller_code_from)
    {
        throw std::invalid_argument("set_caller_code -> caller_code_from -- invalid argument: " +
                                    std::string(caller_code_from.begin(), caller_code_from.end()));
    }

    auto is_caller_code_to = std::all_of(caller_code_to.begin(), caller_code_to.end(), static_cast<int
(*)(int)>(std::isdigit));
    if (!is_caller_code_to)
    {
        throw std::invalid_argument("set_caller_code -> caller_code_to -- invalid argument: " +
                                    std::string(caller_code_to.begin(), caller_code_to.end()));
    }
    m_caller_code_from = std::stoi(caller_code_from);
    m_caller_code_to = std::stoi(caller_code_to);
```

```
}

void MobileNumbersRange::set_operator_name(const std::wstring &mobile_operator)
{
    m_operator_name = mobile_operator;
}

void MobileNumbersRange::set_region(const std::wstring &region)
{
    m_region = region;
}

std::list<std::string> MobileNumbersRange::split()
{
    auto get_number_of_digits = [](int x) -> int
    {
        return static_cast<int>(std::log10(x) + 1);
    };
    std::list<std::string> range_splitted;
    int cct = m_caller_code_to;
    while (cct >= m_caller_code_from)
    {
        int digits_init = get_number_of_digits(cct);
        if (cct % 10 == 9 && cct - m_caller_code_from >= 9)
        {
            int digits = get_number_of_digits(cct - m_caller_code_from + 10) - 1;
            int pw = static_cast<int>(std::pow(10, digits));
            for (int i = 0; i <= digits; ++i, pw /= 10)
            {
                if (cct % pw == pw - 1)
                {
                    int t = cct / pw;
                    std::string t1;
                    if (t != 0)
                    {
                        t1 = std::to_string(t);
                    }
                    std::string t2 = std::string(static_cast<size_t>(digits_init) - t1.size(), '?');
                    range_splitted.push_back(t1 + t2);
                    break;
                }
            }
            cct -= pw;
        }
        else
        {
            if (range_splitted.empty() || range_splitted.back() != "?" || cct != 0)
            {
                range_splitted.push_back(std::to_string(cct));
                --cct;
            }
            else
            {
                break;
            }
        }
    }
    for (auto &x : range_splitted)
    {
        if (x.size() < MobileNumberDefines::CALLER_CODE_DIGITS)
        {
            x = std::string(MobileNumberDefines::CALLER_CODE_DIGITS - x.size(), '0') + x;
        }
    }
    return range_splitted;
}
```

Таблица 28. Файл lib/mobile_numbers_data.h

lib/mobile_numbers_data.h

```
#ifndef MOBILE_NUMBERS_DATA_H
#define MOBILE_NUMBERS_DATA_H

#include <istream>
#include <string>
#include <regex>
#include <algorithm>
#include <stdexcept>
#include "mobile_numbers_range.h"
#include "digits_tree.h"
#include "mobile_number_defines.h"

class MobileNumbersData
{
public:
    MobileNumbersData();
    void add_data(std::wistream &);
    MobileNumbersRange find_with_check(std::string) const;
    MobileNumbersRange find_without_check(std::string) const;
    std::string generate() const;
    static bool is_mobile_number(const std::string &);

private:
    DigitsTree<> m_data;
};

#endif // MOBILE_NUMBERS_DATA_H
```

Таблица 29. Файл lib/mobile_numbers_data.cpp

lib/mobile_numbers_data.cpp

```
#include "mobile_numbers_data.h"

MobileNumbersData::MobileNumbersData()
{
}

void MobileNumbersData::add_data(std::wistream &stream)
{
    bool maybe_header = true;
    MobileNumbersRange range;
    std::match_results<std::wstring::const_iterator> sm;
    for (std::wstring line; std::getline(stream, line);)
    {
        if (std::regex_match(line, sm, MobileNumberDefines::mobile_numbers_range_regex))
        {
            range.set_operator_code(sm.str(1));
            range.set_caller_code(sm.str(2), sm.str(3));
            range.set_operator_name(sm.str(4));
            range.set_region(sm.str(5));
            std::string operator_code = range.operator_code();
            auto range_splitted = range.split();
            for (const auto &x : range_splitted)
            {
                m_data.push(operator_code + x, range);
            }
        }
        else
        {
            if (maybe_header && !(L'0' <= line[0] && line[0] <= L'9'))
            {
                maybe_header = false;
            }
            else
            {
                throw std::invalid_argument(
                    std::string("Argument is invalid: ") +
                    std::string(line.begin(), line.end()));
            }
        }
```

lib/mobile_numbers_data.cpp

```
    }
}

MobileNumbersRange MobileNumbersData::find_with_check(std::string mobile_number) const
{
    std::smatch sm;
    if (!std::regex_match(mobile_number, sm, MobileNumberDefines::mobile_number_regex))
    {
        throw std::out_of_range("Invalid mobile number.");
    }
    return m_data.find(sm.str(2) + sm.str(3) + sm.str(4) + sm.str(5));
}

MobileNumbersRange MobileNumbersData::find_without_check(std::string mobile_number) const
{
    return m_data.find(mobile_number);
}

std::string MobileNumbersData::generate() const
{
    MobileNumbersRange range = m_data.get_random();
    std::string operator_code = range.operator_code();
    std::string caller_code =
std::to_string(RandomGen::get_random_int_number(range.caller_code_from_i(), range.caller_code_to_i()));
    return std::string(MobileNumberDefines::OPERATOR_CODE_DIGITS - operator_code.size(), '0') +
           operator_code +
           std::string(MobileNumberDefines::CALLER_CODE_DIGITS - caller_code.size(), '0') +
           caller_code;
}

bool MobileNumbersData::is_mobile_number(const std::string &arg)
{
    return std::regex_match(arg, MobileNumberDefines::mobile_number_regex);
}
```

Таблица 30. Сборочный файл lib/Makefile

lib/Makefile

```
.PHONY : all clean
CXX=g++
CXXFLAGS=-std=c++17 -O4 -Werror -pedantic-errors -Wredundant-decls \
          -Wall -Wextra -Wpedantic -Wcast-align -Wcast-qual \
          -Wconversion -Wctor-dtor-privacy -Wduplicated-branches \
          -Wduplicated-cond -Wextra-semi -Wfloat-equal -Wlogical-op \
          -Wnon-virtual-dtor -Woverloaded-virtual -Wsign-conversion -Wsign-promo

all: $(patsubst %.cpp, %.o, $(wildcard *.cpp))

%.o: %.cpp %.h Makefile
          $(CXX) $(CXXFLAGS) -c -o $@ $<

clean:
          rm -f *.o
```

Таблица 31. Сборочный файл Makefile

Makefile

```
.PHONY : all clean
CXX=g++
CXXFLAGS=-std=c++17 -O4 -Werror -pedantic-errors -Wredundant-decls \
          -Wall -Wextra -Wpedantic -Wcast-align -Wcast-qual \
          -Wconversion -Wctor-dtor-privacy -Wduplicated-branches \
          -Wduplicated-cond -Wextra-semi -Wfloat-equal -Wlogical-op \
          -Wnon-virtual-dtor -Woverloaded-virtual -Wsign-conversion -Wsign-promo
LIB_HEADERS=$(wildcard lib/*.h)
LIB_SOURCES=$(wildcard lib/*.cpp)
LIB_OBJECTS=$(patsubst %.cpp, %.o, $(LIB_SOURCES))

all: main test
```

Makefile

```
main: $(LIB_OBJECTS) main.o
	$(CXX) $(CXXFLAGS) -o main $(LIB_OBJECTS) main.o

test: $(LIB_OBJECTS) test.o
	$(CXX) $(CXXFLAGS) -o test $(LIB_OBJECTS) test.o

$(LIB_OBJECTS): $(LIB_SOURCES) $(LIB_HEADERS)
	@cd lib && $(MAKE) && cd ../

%.o: %.cpp Makefile
	$(CXX) $(CXXFLAGS) -c -o $@ $<

clean:
	rm -f *.o
```

ДНЕВНИК ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ

Даты начала и окончания	Рабочее место	Содержание выполненных работ	Подпись руководителя
28.06.21	ООО «НТЦ Севентест»	Получение первого задания на практику, связанного с освоением основных возможностей CORBA	
29.06.21	ООО «НТЦ Севентест»	Поиск и чтение информации о CORBA.	
30.06.21	ООО «НТЦ Севентест»	Чтение документации omniORB, реализующего спецификацию CORBA 2.1.	
01.06.21	ООО «НТЦ Севентест»	Создание клиент-серверного приложения с помощью omniORB 4.2.4.	
02.06.21	ООО «НТЦ Севентест»	Попытка использования omniORB для взаимодействия программ, работающих в двух разных сетях.	
05.06.21	ООО «НТЦ Севентест»	Поиск и чтение информации о CORBA, чтение документации omniORB.	
06.06.21	ООО «НТЦ Севентест»	Сдача первого задания.	
07.06.21	ООО «НТЦ Севентест»	Получение второго задания на практику, связанного с созданием библиотеки для парсинга файлов выписки плана нумерации ФАС.	
08.06.21	ООО «НТЦ Севентест»	Создание программы для поиска запрашиваемого номера в CSV файлах текущей директории.	
09.06.21	ООО «НТЦ Севентест»	Создание программы для загрузки данных ФАС из CSV файлов текущей директории в память и поиска запрашиваемого номера. Использование двусторонней очереди с бинарным поиском.	
12.06.21	ООО «НТЦ Севентест»	Использование хэш-таблицы с упорядоченным множеством. Оценка производительности решения.	
13.06.21	ООО «НТЦ Севентест»	Использование хэш-таблицы без уникальных ключей с упорядоченным множеством. Оценка производительности решения.	

Даты начала и окончания	Рабочее место	Содержание выполненных работ	Подпись руководителя
14.06.21	ООО «НТЦ Севентест»	Использование хэш-таблицы с минимальными масками диапазонов. Оценка производительности решения.	
15.06.21	ООО «НТЦ Севентест»	Использование дерева цифр с минимальным масками диапазонов. Оценка производительности решения.	

ЗАКЛЮЧЕНИЕ

В результате прохождения производственной практики в ООО «НТЦ Севентест» мои профессиональные навыки, необходимые выпускнику направления «программная инженерия», были значительно расширены. В ходе практики мной были приобретены и усвоены навыки по работе с языком программирования C++.

СПИСОК ЛИТЕРАТУРЫ

- Герберт Шилдт. C++. Руководство для начинающих. 2005.